aws

Developer Guide

Amazon OpenSearch Service

Amazon OpenSearch Service: Developer Guide

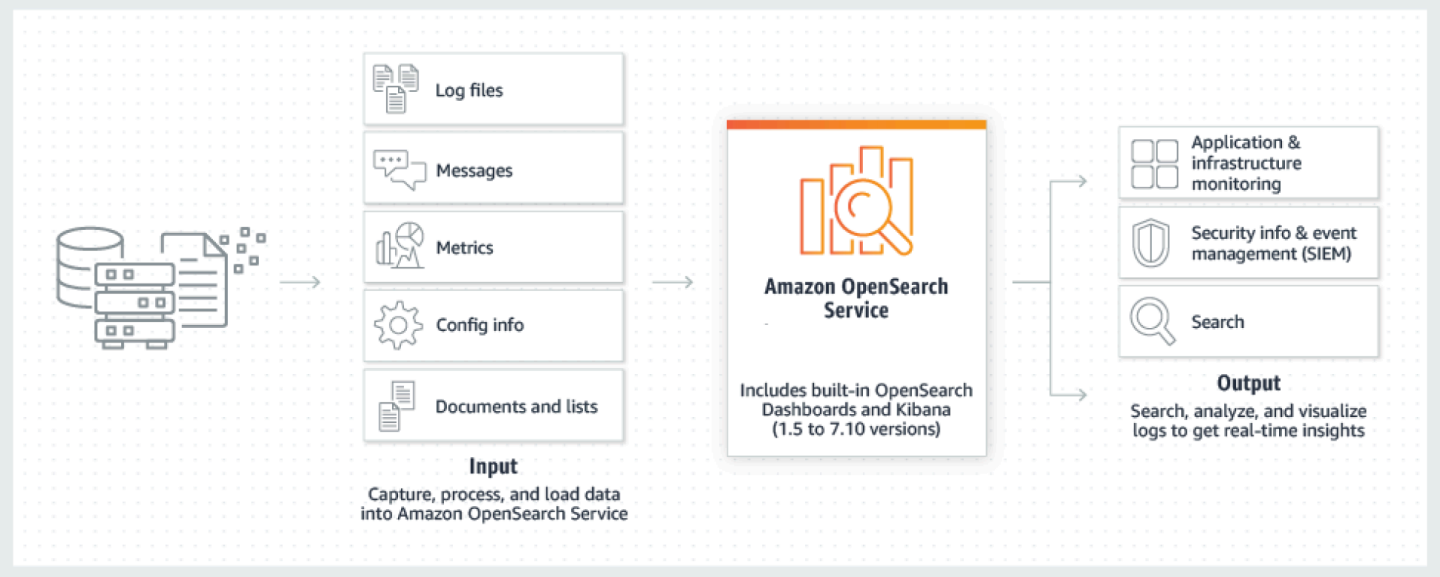
Table of Contents

What is Amazon OpenSearch Service?

Amazon OpenSearch Service is a managed service that makes it easy to deploy, operate, and scale OpenSearch clusters in the AWS Cloud. An OpenSearch Service domain is synonymous with an OpenSearch cluster. Domains are clusters with the settings, instance types, instance counts, and storage resources that you specify. Amazon OpenSearch Service supports OpenSearch and legacy Elasticsearch OSS (up to 7.10, the final open source version of the software). When you create a domain, you have the option of which search engine to use.

OpenSearch is a fully open-source search and analytics engine for use cases such as log analytics, real-time application monitoring, and clickstream analysis. For more information, see the OpenSearch documentation.

Amazon OpenSearch Service provisions all the resources for your OpenSearch cluster and launches it. It also automatically detects and replaces failed OpenSearch Service nodes, reducing the overhead associated with self-managed infrastructures. You can scale your cluster with a single API call or a few clicks in the console.



To get started using OpenSearch Service, you create an OpenSearch Service *domain*, which is equivalent to an OpenSearch *cluster*. Each EC2 instance in the cluster acts as one OpenSearch Service node.

You can use the OpenSearch Service console to set up and configure a domain in minutes. If you prefer programmatic access, you can use the AWS CLI, the AWS SDKs, or Terraform.

Features of Amazon OpenSearch Service

OpenSearch Service includes the following features:

Scale

- Numerous configurations of CPU, memory, and storage capacity known as *instance types*, including cost-effective Graviton instances
- Supports up to 1002 data nodes
- Up to 25 PB of attached storage
- Cost-effective UltraWarm and cold storage for read-only data

Security

- AWS Identity and Access Management (IAM) access control
- Easy integration with Amazon VPC and VPC security groups
- Encryption of data at rest and node-to-node encryption
- Amazon Cognito, HTTP basic, or SAML authentication for OpenSearch Dashboards
- Index-level, document-level, and field-level security
- Audit logs
- Dashboards multi-tenancy

Stability

- Numerous geographical locations for your resources, known as *Regions* and *Availability Zones*
- Node allocation across two or three Availability Zones in the same AWS Region, known as *Multi-AZ*
- Dedicated master nodes to offload cluster management tasks
- Automated snapshots to back up and restore OpenSearch Service domains

Flexibility

- SQL support for integration with business intelligence (BI) applications
- Custom packages to improve search results

Integration with popular services

- Data visualization using OpenSearch Dashboards
- Integration with Amazon CloudWatch for monitoring OpenSearch Service domain metrics and setting alarms
- Integration with AWS CloudTrail for auditing configuration API calls to OpenSearch Service domains
- Integration with Amazon S3, Amazon Kinesis, and Amazon DynamoDB for loading streaming data into OpenSearch Service
- Alerts from Amazon SNS when your data exceeds certain thresholds

When to use OpenSearch versus Amazon OpenSearch Service

Use the following table to help you decide whether provisioned Amazon OpenSearch Service or self-managed OpenSearch is the correct choice for you.

OpenSearch	Amazon OpenSearch Service
• Your organization is willing to, and has people with the correct skills to, manually monitor and maintain self-provisioned clusters. • You want full, compile-level control of your code. • Your organization prefers, or uniquely uses, open source software. • You have a multi-cloud strategy, requiring technologies that aren't vendor-specific. • Your team is capable of addressing any critical production issues. • You want the flexibility to use, modify, and extend the product however you want.	• You don’t want to manually manage, monitor, and maintain your infrastructure. • You want simple ways to manage growing analytics costs by layering your data across storage tiers, taking advantage of the durability and low cost of Amazon S3. • You want to take advantage of integrations with other AWS services such as DynamoDB, Amazon DocumentDB (with MongoDB compatibility), IAM, CloudWatch, and CloudFormation. • You want easy access to assistance from Support for preventative maintenance and during production issues. • You want to take advantage of features such as self-healing, proactive maintenance, resiliency, and backups.

OpenSearch	Amazon OpenSearch Service
• You want immediate access to new features as soon as they're released.	

Supported versions of OpenSearch and Elasticsearch

OpenSearch Service supports multiple versions of OpenSearch and legacy open-source Elasticsearch versions. For some versions, we have already published end of standard support and extended support date. It is recommended you upgrade to the latest available OpenSearch version to get the best use of OpenSearch Service, in terms of price-performance, feature richness, and secuity improvements. Please see below table for a list of versions and their support schedule:

The end of support schedule for Elasticsearch versions is as follows:

Software Version	End of Standard Support	End of Extended Support
Elasticsearch versions 1.5 and 2.3	November 7, 2025	November 7, 2026
Elasticsearch versions 5.1 to 5.5	November 7, 2025	November 7, 2026
Elasticsearch versions 5.6	November 7, 2025	November 7, 2028
Elasticsearch versions 6.0 to 6.7	November 7, 2025	November 7, 2026

Software Version	End of Standard Support	End of Extended Support
Elasticsearch versions 6.8	Not announced	Not announced
Elasticsearch versions 7.1 to 7.8	November 7, 2025	November 7, 2026
Elasticsearch versions 7.9	Not announced	Not announced
Elasticsearch versions 7.10	Not announced	Not announced

The end of support schedule for OpenSearch versions is as follows:

Software Version	End of Standard Support	End of Extended Support
OpenSearch versions 1.0 and 1.2	November 7, 2025	November 7, 2026
OpenSearch versions 1.3	Not announced	Not announced
OpenSearch versions 2.3 to 2.9	November 7, 2025	November 7, 2026

Software Version	End of Standard Support	End of Extended Support
OpenSearch versions 2.11 and higher versions	Not announced	Not announced

Standard support and extended support of OpenSearch and Elasticsearch

AWS provides regular bug fixes and security updates for versions covered under Standard Support. For versions under Extended Support, AWS provides critical security fixes for a period of at least 12 months after end of standard support, for an additional flat fee each Normalized Instance Hour (NIH). NIH is computed as a factor of the instance size (e.g. medium, large), and number of instance hours (see calculating extended support charges section below for an example). Extended support charges are applied automatically when a domain is running a version for which standard support has ended. You can upgrade to a recent version that is still covered under standard support to avoid extended support charges. For more information on extended support charges, please see the Extended support costs. For general information about extended support, please see the Extended Support.

Calculating extended support charges

Domains running versions under extended support will be charged a flat additional fee/Normalized Instance Hour (NIH), for example, $0.0065 in the US East (North Virginia) Region. NIH is computed as a factor of the instance size (e.g., medium, large), and the number of instance hours. For example, if you are running an m7g.medium.search instance for 24 hours in the US East (North Virginia) Region, which is priced at $0.068/Instance hour (on-demand), you will typically pay $1.632 ($0.068x24). If you are running a version that is in extended support, you will pay an additional $0.0065/NIH, which is computed as $0.0065 x 24 (number of instance hours) x 2 (size normalization factor; 2 for medium-sized instances), which comes to $0.312 for extended support for 24 hours. The total amount you will pay for 24 hours will be a sum of the standard instance

usage cost and the extended support cost, which is $1.944 ($1.632+$0.312). The below table shows the normalization factor for various instance sizes in OpenSearch Service.

Instance size	Normalization Factor	
nano	0.25	
micro	0.5	
small	1	
medium	2	
large	4	
xlarge	8	
2xlarge	16	
4xlarge	32	
8xlarge	64	
9xlarge	72	
10xlarge	80	
12xlarge	96	
16xlarge	128	
18xlarge	144	
24xlarge	192	
32xlarge	256	

Pricing for Amazon OpenSearch Service

For OpenSearch Service, you pay for each hour of use of an EC2 instance and for the cumulative size of any EBS storage volumes attached to your instances. Standard AWS data transfer charges also apply.

However, some notable data transfer exceptions exist. If a domain uses multiple Availability Zones, OpenSearch Service does not bill for traffic between the Availability Zones. Significant data transfer occurs within a domain during shard allocation and rebalancing. OpenSearch Service neither meters nor bills for this traffic. Similarly, OpenSearch Service does not bill for data transfer between UltraWarm/cold nodes and Amazon S3.

For full pricing details, see Amazon OpenSearch Service pricing. For information about charges incurred during configuration changes, see the section called "Charges for configuration changes".

Related services

OpenSearch Service commonly is used with the following services:

Amazon CloudWatch

OpenSearch Service domains automatically send metrics to CloudWatch so that you can monitor domain health and performance. For more information, see Monitoring OpenSearch cluster metrics with Amazon CloudWatch.

CloudWatch Logs can also go the other direction. You might configure CloudWatch Logs to stream data to OpenSearch Service for analysis. To learn more, see the section called "Loading streaming data from Amazon CloudWatch".

AWS CloudTrail

Use AWS CloudTrail to get a history of the OpenSearch Service configuration API calls and related events for your account. For more information, see Monitoring Amazon OpenSearch Service API calls with AWS CloudTrail.

Amazon Kinesis

Kinesis is a managed service for real-time processing of streaming data at a massive scale. For more information, see the section called "Loading streaming data from Amazon Kinesis Data Streams" and the section called "Loading streaming data from Amazon Data Firehose".

Amazon S3

Amazon Simple Storage Service (Amazon S3) provides storage for the internet. This guide provides Lambda sample code for integration with Amazon S3. For more information, see the section called "Loading streaming data from Amazon S3".

AWS IAM

AWS Identity and Access Management (IAM) is a web service that you can use to manage access to your OpenSearch Service domains. For more information, see the section called "Identity and Access Management".

AWS Lambda

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. This guide provides Lambda sample code to stream data from DynamoDB, Amazon S3, and Kinesis. For more information, see the section called "Loading streaming data into OpenSearch Service".

Amazon DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. To learn more about streaming data to OpenSearch Service, see the section called "Loading streaming data from Amazon DynamoDB".

Amazon QuickSight

You can visualize data from OpenSearch Service using Amazon QuickSight dashboards. For more information, see Using Amazon OpenSearch Service with Amazon QuickSight in the *Amazon QuickSight User Guide*.

Note

OpenSearch includes certain Apache-licensed Elasticsearch code from Elasticsearch B.V. and other source code. Elasticsearch B.V. is not the source of that other source code. ELASTICSEARCH is a registered trademark of Elasticsearch B.V.

Setting up Amazon OpenSearch Service

Grant permissions

In production environments, we recommend that you use finer-grained policies. To learn more about access management, see Access management for AWS resources in the IAM User Guide.

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.
- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Create a role for a third-party identity provider (federation) in the *IAM User Guide*.
- IAM users:
  - Create a role that your user can assume. Follow the instructions in Create a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

Grant programmatic access

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
Workforce identity	Use temporary credentials to sign programmatic requests	Following the instructions for the interface that you want to use.

Which user needs programmatic access?	To	By
(Users managed in IAM Identity Center)	to the AWS CLI, AWS SDKs, or AWS APIs.	• For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*. • For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the *IAM User Guide*.

Which user needs programmatic access?	To	By
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

Install and configure the AWS CLI

If you want to use OpenSearch Service APIs, you must install the latest version of the AWS Command Line Interface (AWS CLI). You don't need the AWS CLI to use OpenSearch Service from the console, and you can get started without the CLI by following the steps in Getting started with Amazon OpenSearch Service.

To set up the AWS CLI

1. To install the latest version of the AWS CLI for macOS, Linux, or Windows, see Installing or updating the latest version of the AWS CLI.
2. To configure the AWS CLI and secure setup of your access to AWS services, including OpenSearch Service, see Quick configuration with `aws configure`.
3. To verify the setup, enter the following DataBrew command at the command prompt.

```
aws opensearch help
```

AWS CLI commands use the default AWS Region from your configuration, unless you set it with a parameter or a profile. To set your AWS Region with a parameter, you can add the `--region` parameter to each command.

To set your AWS Region with a profile, first add a named profile in the `~/.aws/config` file or the `%UserProfile%/.aws/config` file (for Microsoft Windows). Follow the steps in Named profiles for the AWS CLI. Next, set your AWS Region and other settings with a command similar to the one in the following example.

```
[profile opensearch]
aws_access_key_id = ACCESS-KEY-ID-OF-IAM-USER
aws_secret_access_key = SECRET-ACCESS-KEY-ID-OF-IAM-USER
region = us-east-1
output = text
```

Open the console

Most of the console-oriented topics in this section start from the OpenSearch Service console. If you aren't already signed in to your AWS account, sign in, then open the OpenSearch Service console and continue to the next section to continue getting started with OpenSearch Service.

Getting started with Amazon OpenSearch Service

To get started, sign up for an AWS account if you don't already have one. After you are set up with an account, complete the getting started tutorial for Amazon OpenSearch Service. Consult the following introductory topics if you need more information while learning about the service:

- Create a domain.
- Size the domain appropriately for your workload.
- Control access to your domain using a domain access policy or fine-grained access control.
- Index data manually or from other AWS services.
- Use OpenSearch Dashboards to search your data and create visualizations.
- Learn about more advanced options for creating a domain. For more information, see *Creating and managing domains*.
- Discover how to manage the indices in your domain. For more information, see *Managing indexes*.
- Try out one of the tutorials for working with Amazon OpenSearch Service. For more information, see *Tutorials*.

For information on migrating to OpenSearch Service from a self-managed OpenSearch cluster, see the section called “Migrating to OpenSearch Service”.

For more detailed information, see *Creating and managing domains* and the other topics within this guide. For information on migrating to OpenSearch Service from a self-managed OpenSearch cluster, see the section called “Migrating to OpenSearch Service”.

You can complete the following steps by using the OpenSearch Service console, the AWS CLI, or the AWS SDK. For information about installing and setting up the AWS CLI, see the AWS Command Line Interface User Guide.

Create an Amazon OpenSearch Service domain

> **Important**
>
> This is a concise tutorial for configuring a *test* Amazon OpenSearch Service domain. Do not use this process to create production domains. For a comprehensive version of the same process, see *Creating and managing domains*.

An OpenSearch Service domain is synonymous with an OpenSearch cluster. Domains are clusters with the settings, instance types, instance counts, and storage resources that you specify. You can create an OpenSearch Service domain by using the console, the AWS CLI, or the AWS SDKs.

To create an OpenSearch Service domain using the console

1. Go to https://aws.amazon.com and choose **Sign In to the Console**.
2. Under **Analytics**, choose **Amazon OpenSearch Service**.
3. Choose **Create domain**.
4. Provide a name for the domain. The examples in this tutorial use the name *movies*.
5. For the domain creation method, choose **Standard create**.

   > **Note**
   >
   > To quickly configure a production domain with best practices, you can choose **Easy create**. For the development and testing purposes of this tutorial, we'll use **Standard create**.

6. For templates, choose **Dev/test**.
7. For the deployment option, choose **Domain with standby**.
8. For **Version**, choose the latest version.
9. For now, ignore the **Data nodes**, **Warm and cold data storage**, **Dedicated master nodes**, **Snapshot configuration**, and **Custom endpoint** sections.
10. For simplicity in this tutorial, use a public access domain. Under **Network**, choose **Public access**.
11. In the fine-grained access control settings, keep the **Enable fine-grained access control** check box selected. Select **Create master user** and provide a username and password.

12. For now, ignore the **SAML authentication** and **Amazon Cognito authentication** sections.
13. For **Access policy**, choose **Only use fine-grained access control**. In this tutorial, fine-grained access control handles authentication, not the domain access policy.
14. Ignore the rest of the settings and choose **Create**. New domains typically take 15–30 minutes to initialize, but can take longer depending on the configuration. After your domain initializes, select it to open its configuration pane. Note the domain endpoint under **General information** (for example, `https://search-my-domain.us-east-1.es.amazonaws.com`), which you'll use in the next step.

Next: Upload data to an OpenSearch Service domain for indexing

Upload data to Amazon OpenSearch Service for indexing

> **Important**
>
> This is a concise tutorial for uploading a small amount of test data to Amazon OpenSearch Service. For more about uploading data in a production domain, see *Indexing data*.

You can upload data to an OpenSearch Service domain using the command line or most programming languages.

The following example requests use curl (a common HTTP client) for brevity and convenience. Clients like curl can't perform the request signing that's required if your access policies specify IAM users or roles. To successfully complete this process, you must use fine-grained access control with a primary username and password like you configured in Step 1.

You can install curl on Windows and use it from the command prompt, but we recommend a tool like Cygwin or the Windows Subsystem for Linux. macOS and most Linux distributions come with curl preinstalled.

Option 1: Upload a single document

Run the following command to add a single document to the *movies* domain:

```
curl -XPUT -u 'master-user:master-user-password' 'domain-endpoint/movies/_doc/1' -d
 '{"director": "Burton, Tim", "genre": ["Comedy","Sci-Fi"], "year": 1996, "actor":
```

```
["Jack Nicholson","Pierce Brosnan","Sarah Jessica Parker"], "title": "Mars Attacks!"}'
-H 'Content-Type: application/json'
```

In the command, provide the username and password that you created in Step 1.

For a detailed explanation of this command and how to make signed requests to OpenSearch Service, see *Indexing data*.

Option 2: Upload multiple documents

To upload a JSON file that contains multiple documents to an OpenSearch Service domain

1. Create a local file called `bulk_movies.json`. Paste the following content into the file and add a trailing newline:

```
{ "index" : { "_index": "movies", "_id" : "2" } }
{"director": "Frankenheimer, John", "genre": ["Drama", "Mystery", "Thriller",
 "Crime"], "year": 1962, "actor": ["Lansbury, Angela", "Sinatra, Frank", "Leigh,
 Janet", "Harvey, Laurence", "Silva, Henry", "Frees, Paul", "Gregory, James",
 "Bissell, Whit", "McGiver, John", "Parrish, Leslie", "Edwards, James", "Flowers,
 Bess", "Dhiegh, Khigh", "Payne, Julie", "Kleeb, Helen", "Gray, Joe", "Nalder,
 Reggie", "Stevens, Bert", "Masters, Michael", "Lowell, Tom"], "title": "The
 Manchurian Candidate"}
{ "index" : { "_index": "movies", "_id" : "3" } }
{"director": "Baird, Stuart", "genre": ["Action", "Crime", "Thriller"], "year":
 1998, "actor": ["Downey Jr., Robert", "Jones, Tommy Lee", "Snipes, Wesley",
 "Pantoliano, Joe", "Jacob, Ir\u00e8ne", "Nelligan, Kate", "Roebuck, Daniel",
 "Malahide, Patrick", "Richardson, LaTanya", "Wood, Tom", "Kosik, Thomas",
 "Stellate, Nick", "Minkoff, Robert", "Brown, Spitfire", "Foster, Reese",
 "Spielbauer, Bruce", "Mukherji, Kevin", "Cray, Ed", "Fordham, David", "Jett,
 Charlie"], "title": "U.S. Marshals"}
{ "index" : { "_index": "movies", "_id" : "4" } }
{"director": "Ray, Nicholas", "genre": ["Drama", "Romance"], "year": 1955, "actor":
 ["Hopper, Dennis", "Wood, Natalie", "Dean, James", "Mineo, Sal", "Backus, Jim",
 "Platt, Edward", "Ray, Nicholas", "Hopper, William", "Allen, Corey", "Birch,
 Paul", "Hudson, Rochelle", "Doran, Ann", "Hicks, Chuck", "Leigh, Nelson",
 "Williams, Robert", "Wessel, Dick", "Bryar, Paul", "Sessions, Almira", "McMahon,
 David", "Peters Jr., House"], "title": "Rebel Without a Cause"}
```

2. Run the following command in the local directory where the file is stored to upload it to the *movies* domain:

```
curl -XPOST -u 'master-user:master-user-password' 'domain-endpoint/movies/_bulk' --
data-binary @bulk_movies.json -H 'Content-Type: application/x-ndjson'
```

For more information about the bulk file format, see *Indexing data*.

Next: Search documents

Search documents in Amazon OpenSearch Service

To search documents in an Amazon OpenSearch Service domain, use the OpenSearch search API. Alternatively, you can use OpenSearch Dashboards to search documents in the domain.

Search documents from the command line

Run the following command to search the *movies* domain for the word *mars*:

```
curl -XGET -u 'master-user:master-user-password' 'domain-endpoint/movies/_search?
q=mars&pretty=true'
```

If you used the bulk data on the previous page, try searching for *rebel* instead.

You should see a response similar to the following:

```
{
  "took" : 5,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 0.2876821,
```

```
    "hits" : [
      {
        "_index" : "movies",
        "_type" : "_doc",
        "_id" : "1",
        "_score" : 0.2876821,
        "_source" : {
          "director" : "Burton, Tim",
          "genre" : [
            "Comedy",
            "Sci-Fi"
          ],
          "year" : 1996,
          "actor" : [
            "Jack Nicholson",
            "Pierce Brosnan",
            "Sarah Jessica Parker"
          ],
          "title" : "Mars Attacks!"
        }
      }
    ]
  }
}
```

Search documents using OpenSearch Dashboards

OpenSearch Dashboards is a popular open source visualization tool designed to work with OpenSearch. It provides a helpful user interface for you to search and monitor your indices.

To search documents from an OpenSearch Service domain using Dashboards

1. Navigate to the OpenSearch Dashboards URL for your domain. You can find the URL on the domain's dashboard in the OpenSearch Service console. The URL follows this format:

   ```
   domain-endpoint/_dashboards/
   ```

2. Log in using your primary username and password.
3. To use Dashboards, you need to create at least one index pattern. Dashboards uses these patterns to identify which indexes you want to analyze. Open the left navigation panel, choose **Stack Management**, choose **Index Patterns**, and then choose **Create index pattern**. For this tutorial, enter *movies*.

4. Choose **Next step** and then choose **Create index pattern**. After the pattern is created, you can view the various document fields such as `actor` and `director`.
5. Go back to the **Index Patterns** page and make sure that `movies` is set as the default. If it's not, select the pattern and choose the star icon to make it the default.
6. To begin searching your data, open the left navigation panel again and choose **Discover**.
7. In the search bar, enter *mars* if you uploaded a single document, or *rebel* if you uploaded multiple documents, and then press **Enter**. You can try searching other terms, such as actor or director names.

Next: Delete a domain

Delete an Amazon OpenSearch Service domain

Because the *movies* domain from this tutorial is for test purposes, make sure to delete it when you're done experimenting to avoid incurring charges.

To delete an OpenSearch Service domain from the console

1. Sign in to the **Amazon OpenSearch Service** console.
2. Under **Domains**, select the **movies** domain.
3. Choose **Delete** and confirm deletion.

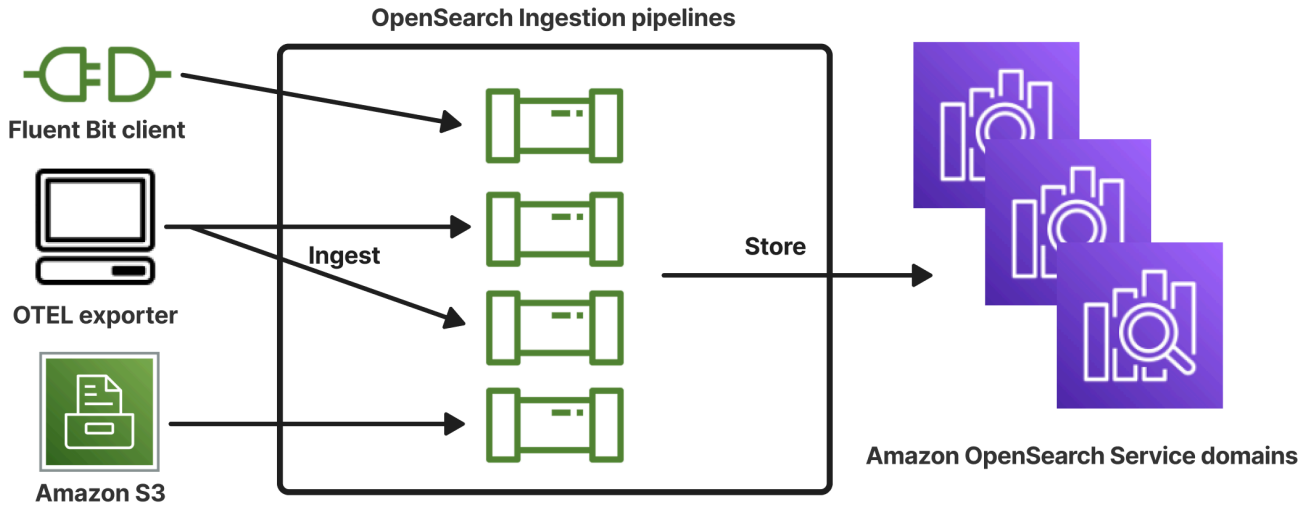
Amazon OpenSearch Ingestion

Amazon OpenSearch Ingestion is a fully managed, serverless data collector that delivers real-time log, metric, and trace data to Amazon OpenSearch Service domains and OpenSearch Serverless collections.

With OpenSearch Ingestion, you no longer need to use third-party solutions like Logstash or Jaeger to ingest data into your OpenSearch Service domains and OpenSearch Serverless collections. You configure your data producers to send data to OpenSearch Ingestion. Then, it automatically delivers the data to the domain or collection that you specify. You can also configure OpenSearch Ingestion to transform your data before delivering it.

Also, with OpenSearch Ingestion, you don't need to worry about provisioning servers, managing and patching software, or scaling your cluster of servers. You provision ingestion *pipelines* directly within the AWS Management Console, and OpenSearch Ingestion takes care of managing and scaling them.

OpenSearch Ingestion is a subset of Amazon OpenSearch Service. It's powered by Data Prepper, which is an open source data collector that can filter, enrich, transform, normalize, and aggregate data for downstream analysis and visualization.



Key concepts

As you get started with OpenSearch Ingestion, you can benefit from understanding the following concepts:

Pipeline

From an OpenSearch Ingestion perspective, a *pipeline* refers to a single provisioned data collector that you create within OpenSearch Service. You can think of it as the entire YAML configuration file, which includes one or more sub-pipelines. For steps to create an ingestion pipeline, see the section called “Creating pipelines”.

Sub-pipeline

You define sub-pipelines *within* a YAML configuration file. Each sub-pipeline is a combination of a source, a buffer, zero or more processors, and one or more sinks. You can define multiple sub-pipelines in a single YAML file, each with unique sources, processors, and sinks. To aid in monitoring with CloudWatch and other services, we recommend that you specify a pipeline name that's distinct from all of its sub-pipelines.

You can string multiple sub-pipelines together within a single YAML file, so that the source for one sub-pipeline is another sub-pipeline, and its sink is a third sub-pipeline. For an example, see the section called “OpenTelemetry Collector”.

Source

The input component of a sub-pipeline. It defines the mechanism through which a pipeline consumes records. The source can consume events either by receiving them over HTTPS, or by reading from external endpoints such as Amazon S3. There are two types of sources: *push-based* and *pull-based*. Push-based sources, such as HTTP and OTel logs, stream records to ingestion endpoints. Pull-based sources, such as OTel trace and S3, pull data from the source.

Processors

Intermediate processing units that can filter, transform, and enrich records into a desired format before publishing them to the sink. The processor is an optional component of a pipeline. If you don't define a processor, records are published in the format defined in the source. You can have more than one processor. A pipeline runs processors in the order that you define them.

Sink

The output component of a sub-pipeline. It defines one or more destinations that a sub-pipeline publishes records to. OpenSearch Ingestion supports OpenSearch Service domains as sinks. It also supports sub-pipelines as sinks. This means that you can string together multiple sub-pipelines within a single OpenSearch Ingestion pipeline (YAML file). Self-managed OpenSearch clusters aren't supported as sinks.

Buffer

The part of a processor that acts as the layer between the source and the sink. You can't manually configure a buffer within your pipeline. OpenSearch Ingestion uses a default buffer configuration.

Route

The part of a processor that allows pipeline authors to only send events that match certain conditions to different sinks.

A valid sub-pipeline definition must contain a source and a sink. For more information about each of these pipeline elements, see the configuration reference.

Benefits of OpenSearch Ingestion

OpenSearch Ingestion has the following main benefits:

- Eliminates the need for you to manually manage a self-provisioned pipeline.
- Automatically scales your pipelines based on capacity limits that you define.
- Keeps your pipeline up to date with security and bug patches.
- Provides the option to connect pipelines to your virtual private cloud (VPC) for an added layer of security.
- Allows you to stop and start pipelines in order to control costs.
- Provides pipeline configuration blueprints for popular use cases to help you get up and running faster.
- Allows you to interact programmatically with your pipelines through the various AWS SDKs and the OpenSearch Ingestion API.
- Supports performance monitoring in Amazon CloudWatch and error logging in CloudWatch Logs.

Limitations

OpenSearch Ingestion has the following limitations:

- You can only ingest data into domains running OpenSearch 1.0 or later, or Elasticsearch 6.8 or later. If you're using the OTel trace source, we recommend using Elasticsearch 7.9 or later so that you can use the OpenSearch Dashboards plugin.
- If a pipeline is writing to an OpenSearch Service domain that's within a VPC, the pipeline must be created in the same AWS Region as the domain.
- You can only configure a single data source within a pipeline definition.
- You can't specify self-managed OpenSearch clusters as sinks.
- You can't specify a custom endpoint as a sink. You can still write to a domain that has custom endpoints enabled, but you must specify its standard endpoint.
- You can't specify resources within opt-in Regions as sources or sinks.
- There are some constraints on the parameters that you can include in a pipeline configuration. For more information, see the section called “Configuration requirements and constraints”.

Supported Data Prepper versions

OpenSearch Ingestion currently supports the following major versions of Data Prepper:

- 2.x

When you create a pipeline, use the required `version` option to specify the major version of Data Prepper to use. For example, `version: "2"`. OpenSearch Ingestion retrieves the latest supported *minor* version of that major version and provisions the pipeline with that version. For more information, see the section called “Specifying the pipeline version”.

OpenSearch Ingestion pipelines are always provisioned with the latest version of Data Prepper. For information about the features and bug fixes that are in each version of Data Prepper, see the Releases page.

When you update a pipeline's YAML configuration file, if there's support for a new minor version of Data Prepper, OpenSearch Ingestion automatically upgrades the pipeline to the latest supported minor version of the major version that's specified in the pipeline configuration. For example, you might have `version: "2"` in your pipeline configuration, and OpenSearch Ingestion initially provisioned the pipeline with version 2.6.0. When support for version 2.7.0 is added, and you make a change to the pipeline configuration, OpenSearch Ingestion upgrades the pipeline to version 2.7.0. This process keeps your pipeline up to date with the latest bug fixes and performance

improvements. OpenSearch Ingestion can't update the major version of your pipeline unless you manually change the `version` option within the pipeline configuration. For more information, see the section called “Updating pipelines”.

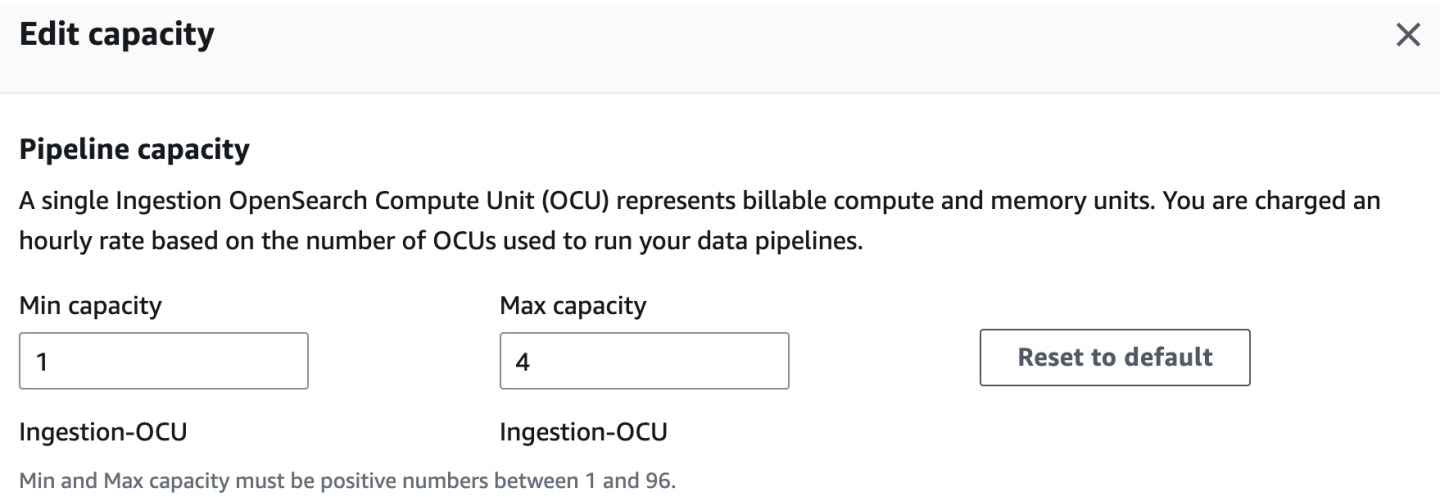
Scaling pipelines

You don't need to provision and manage pipeline capacity yourself. OpenSearch Ingestion automatically scales your pipeline capacity according to your estimated workload, based on the minimum and maximum *Ingestion OpenSearch Compute Units* (Ingestion OCUs) that you specify.

Each Ingestion OCU is a combination of approximately 8 GiB of memory and 2 vCPUs. You can specify the minimum and maximum OCU values for a pipeline, and OpenSearch Ingestion automatically scales your pipeline capacity based on these limits.

You can specify the following values:

- **Minimum capacity** – The pipeline can reduce capacity down to this number of Ingestion OCUs. The specified minimum capacity is also the starting capacity for a pipeline.
- **Maximum capacity** – The pipeline can increase capacity up to this number of Ingestion OCUs.



Make sure that the maximum capacity for a pipeline is high enough to handle spikes in workload, and the minimum capacity is low enough to minimize costs when the pipeline isn't busy. Based on your settings, OpenSearch Ingestion automatically scales the number of Ingestion OCUs for your pipeline to process the ingest workload. At any specific time, you're charged only for the Ingestion OCUs that are being actively used by your pipeline.

The capacity allocated to your OpenSearch Ingestion pipeline scales up and down based on the processing requirements of your pipeline and the load generated by your client application. When capacity is constrained, OpenSearch Ingestion scales up by allocating more compute units (GiB of memory). When your pipeline is processing smaller workloads, or not processing data at all, it can scale down to the minimum configured Ingestion OCUs.

You can specify a minimum of 1 Ingestion OCU, a maximum of 96 Ingestion OCUs for stateless pipelines, and a maximum of 48 Ingestion OCUs for stateful pipelines. We recommend a minimum of at least 2 Ingestion OCUs for push-based sources. When persistent buffering is enabled, you can specify a minimum of 2 and maximum of 384 Ingestion OCUs.

Given a standard log pipeline with a single source, a simple grok pattern, and a sink, each compute unit can support up to 2 MiB per second. For more complex log pipelines with multiple processors, each compute unit might support less ingest load. Based on pipeline capacity and resource utilization, the OpenSearch Ingestion scaling process kicks in.

To ensure high availability, Ingestion OCUs are distributed across Availability Zones (AZs). The number of AZs depends on the minimum capacity that you specify.

For example, if you specify a minimum of 2 compute units, the Ingestion OCUs that are in use at any given time are evenly distributed across 2 AZs. If you specify a minimum of 3 or more compute units, the Ingestion OCUs are evenly distributed across 3 AZs. We recommend that you provision *at least two* Ingestion OCUs to ensure 99.9% availability for your ingest pipelines.

You're not billed for Ingestion OCUs when a pipeline is in the `Create failed`, `Creating`, `Deleting`, and `Stopped` states.

For instructions to configure and retrieve capacity settings for a pipeline, see the section called “Creating pipelines”.

OpenSearch Ingestion pricing

At any specific time, you only pay for the number of Ingestion OCUs that are allocated to a pipeline, regardless of whether there's data flowing through the pipeline. OpenSearch Ingestion immediately accommodates your workloads by scaling pipeline capacity up or down based on usage.

For full pricing details, see Amazon OpenSearch Service pricing.

Supported AWS Regions

OpenSearch Ingestion is available in a subset of AWS Regions that OpenSearch Service is available in. For a list of supported Regions, see Amazon OpenSearch Service endpoints and quotas in the *AWS General Reference*.

Setting up roles and users in Amazon OpenSearch Ingestion

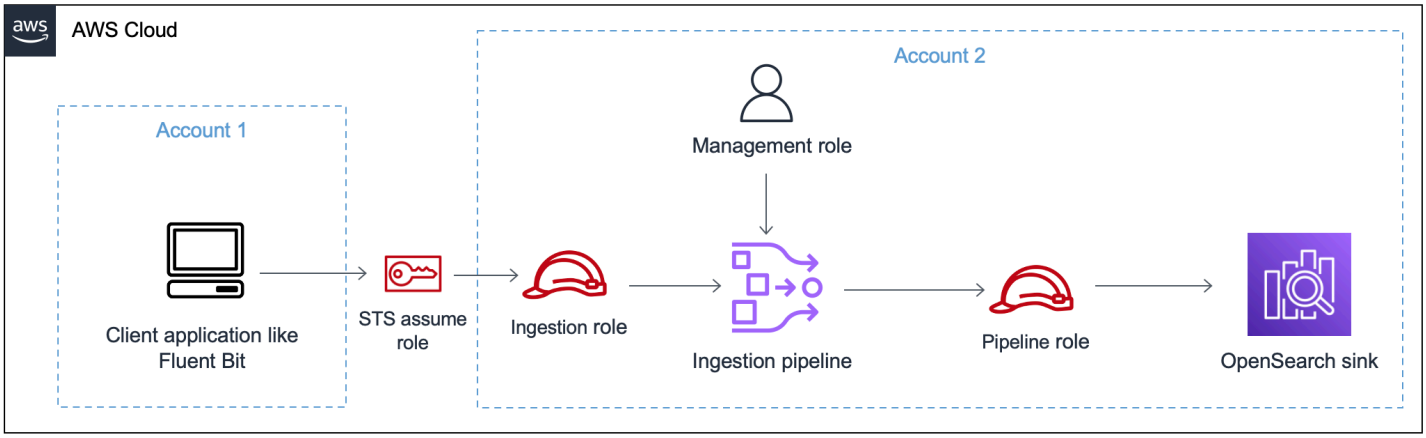
Amazon OpenSearch Ingestion uses a variety of permissions models and IAM roles to allow source applications to write to pipelines, and to allow pipelines to write to sinks. Before you can start ingesting data, you must create one or more IAM roles with specific permissions based on your use case.

At minimum, you require the following roles to set up a successful pipeline.

Name	Description
Management role	Any principal that's managing pipelines (generally a "pipeline admin") needs management access, which includes permissions like `osis:CreatePipeline` and `osis:UpdatePipeline`. These permissions allow a user to administer pipelines but not necessarily write data to them.
Pipeline role	The pipeline role, which you specify within the pipeline's YAML configuration, provides the required permissions for a pipeline to write to the domain or collection sink and read from pull-based sources. For more information, see the following topics: • the section called “Granting pipelines access to domains” • the section called “Granting pipelines access to collections”
Ingestion role	The ingestion role contains the `osis:Ingest` permission for the pipeline resource. This permission allows push-based sources to ingest data into a pipeline.

The following image demonstrates a typical pipeline setup, where a data source such as Amazon S3 or Fluent Bit is writing to a pipeline in a different account. In this case, the client needs to assume

the ingestion role in order to access the pipeline. For more information, see the section called "Cross-account ingestion".



For a simple setup guide, see the section called "Tutorial: Ingest data into a domain".

Management role

In addition to the basic `osis:*` permissions needed to create and modify a pipeline, you also need the `iam:PassRole` permission for the pipeline role resource. Any AWS service that accepts a role must use this permission. OpenSearch Ingestion assumes the role every time it needs to write data to a sink. This helps administrators ensure that only approved users can configure OpenSearch Ingestion with a role that grants permissions. For more information, see Granting a user permissions to pass a role to an AWS service.

If you're using the AWS Management Console (using blueprints and later checking on your pipeline), you need the following permissions to create and update a pipeline:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Resource":"*",
         "Action":[
            "osis:CreatePipeline",
            "osis:GetPipelineBlueprint",
            "osis:ListPipelineBlueprints",
            "osis:GetPipeline",
            "osis:ListPipelines",
            "osis:GetPipelineChangeProgress",
```

```
            "osis:ValidatePipeline",
            "osis:UpdatePipeline"
         ]
      },
      {
         "Resource":[
            "arn:aws:iam::your-account-id:role/pipeline-role"
         ],
         "Effect":"Allow",
         "Action":[
            "iam:PassRole"
         ]
      }
   ]
}
```

If you're using the AWS CLI (not prevalidating your pipeline or using blueprints), you need the following permissions to create and update a pipeline:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Resource":"*",
         "Action":[
            "osis:CreatePipeline",
            "osis:UpdatePipeline"
         ]
      },
      {
         "Resource":[
            "arn:aws:iam::your-account-id:role/pipeline-role"
         ],
         "Effect":"Allow",
         "Action":[
            "iam:PassRole"
         ]
      }
   ]
}
```

Pipeline role

A pipeline needs certain permissions to write to its sink. These permissions depend on whether the sink is an OpenSearch Service domain or an OpenSearch Serverless collection.

In addition, a pipeline might need permissions to pull from the source application (if the source is a pull-based plugin), and permissions to write to an S3 dead letter queue, if configured.

Topics

Writing to a domain sink

An OpenSearch Ingestion pipeline needs permission to write to an OpenSearch Service domain that is configured as its sink. These permissions include the ability to describe the domain and send HTTP requests to it.

To provide your pipeline with the required permissions to write to a sink, first create an AWS Identity and Access Management (IAM) role with the required permissions. These permissions are the same for public and VPC pipelines. Then, specify the pipeline role in the domain access policy so that the domain can accept write requests from the pipeline.

Finally, specify the role ARN as the value of the **sts_role_arn** option within the pipeline configuration:

```
version: "2"
source:
  http:
    ...
processor:
  ...
sink:
  - opensearch:
      ...
      aws:
        sts_role_arn: arn:aws:iam::your-account-id:role/pipeline-role
```

For instructions to complete each of these steps, see Allowing pipelines to access domains.

Writing to a collection sink

An OpenSearch Ingestion pipeline needs permission to write to an OpenSearch Serverless collection that is configured as its sink. These permissions include the ability to describe the collection and send HTTP requests to it.

First, create an IAM role that has the `aoss:BatchGetCollection` permission against all resources (*). Then, include this role in a data access policy and provide it permissions to create indexes, update indexes, describe indexes, and write documents within the collection. Finally, specify the role ARN as the value of the **sts_role_arn** option within the pipeline configuration.

For instructions to complete each of these steps, see Allowing pipelines to access collections.

Writing to a dead-letter queue

If you configure your pipeline to write to a dead-letter queue (DLQ), you must include the `sts_role_arn` option within the DLQ configuration. The permissions included in this role allow the pipeline to access the S3 bucket that you specify as the destination for DLQ events.

You must use the same `sts_role_arn` in all pipeline components. Therefore, you must attach a separate permissions policy to your pipeline role that provides DLQ access. At minimum, the role must be allowed the `S3:PutObject` action on the bucket resource:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "WriteToS3DLQ",
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::my-dlq-bucket/*"
    }
  ]
}
```

You can then specify the role within the pipeline's DLQ configuration:

```
  ...
  sink:
    opensearch:
```

```
      dlq:
        s3:
          bucket: "my-dlq-bucket"
          key_path_prefix: "dlq-files"
          region: "us-west-2"
          sts_role_arn: "arn:aws:iam::your-account-id:role/pipeline-role"
```

Ingestion role

All source plugins that OpenSearch Ingestion currently supports, with the exception of S3, use a push-based architecture. This means that the source application *pushes* the data to the pipeline, rather than the pipeline *pulling* the data from the source.

Therefore, you must grant your source applications the required permissions to ingest data into an OpenSearch Ingestion pipeline. At minimum, the role that signs the request must be granted permission for the `osis:Ingest` action, which allows it to send data to a pipeline. The same permissions are required for public and VPC pipeline endpoints.

The following example policy allows the associated principal to ingest data into a single pipeline called `my-pipeline`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PermitsWriteAccessToPipeline",
      "Effect": "Allow",
      "Action": "osis:Ingest",
      "Resource": "arn:aws:osis:region:your-account-id:pipeline/pipeline-name"
    }
  ]
}
```

For more information, see the section called “Integrating pipelines”.

Cross-account ingestion

You might need to ingest data into a pipeline from a different AWS account, such as an application account. To configure cross-account ingestion, define an ingestion role within the same account as the pipeline and establish a trust relationship between the ingestion role and the application account:

```
{
  "Version": "2012-10-17",
  "Statement": [{
     "Effect": "Allow",
     "Principal": {
       "AWS": "arn:aws:iam::external-account-id:root"
      },
     "Action": "sts:AssumeRole"
  }]
}
```

Then, configure your application to assume the ingestion role. The application account must grant the application role AssumeRole permissions for the ingestion role in the pipeline account.

For detailed steps and example IAM policies, see the section called "Providing cross-account ingestion access".

Granting Amazon OpenSearch Ingestion pipelines access to domains

An Amazon OpenSearch Ingestion pipeline needs permission to write to the OpenSearch Service domain that is configured as its sink. To provide access, you configure an AWS Identity and Access Management (IAM) role with a restrictive permissions policy that limits access to the domain that a pipeline is sending data to. For example, you might want to limit an ingestion pipeline to only the domain and indexes that are required to support its use case.

Before you specify the role in your pipeline configuration, you must configure it with an appropriate trust relationship, and then grant it access to the domain within the domain access policy.

Topics

Step 1: Create a pipeline role

The role that you specify in the **sts_role_arn** parameter of a pipeline configuration must have an attached permissions policy that allows it to send data to the domain sink. It must also have a

trust relationship that allows OpenSearch Ingestion to assume the role. For instructions on how to attach a policy to a role, see Adding IAM identity permissions in the *IAM User Guide*.

The following sample policy demonstrates the least privilege that you can provide in a pipeline configuration's **sts_role_arn** role for it to write to a single domain:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "es:DescribeDomain",
            "Resource": "arn:aws:es:*:your-account-id:domain/*"
        },
        {
            "Effect": "Allow",
            "Action": "es:ESHttp*",
            "Resource": "arn:aws:es:*:your-account-id:domain/domain-name/*"
        }
    ]
}
```

If you plan to reuse the role to write to multiple domains, you can make the policy more broad by replacing the domain name with a wildcard character (*).

The role must have the following trust relationship, which allows OpenSearch Ingestion to assume the pipeline role:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"osis-pipelines.amazonaws.com"
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

In addition, we recommend that you add the `aws:SourceAccount` and `aws:SourceArn` condition keys to the policy to protect yourself against the confused deputy problem. The source account is the owner of the pipeline.

For example, you could add the following condition block to the policy:

```
"Condition": {
    "StringEquals": {
        "aws:SourceAccount": "your-account-id"
    },
    "ArnLike": {
        "aws:SourceArn": "arn:aws:osis:region:your-account-id:pipeline/*"
    }
}
```

Step 2: Include the pipeline role in the domain access policy

In order for a pipeline to write data to a domain, the domain must have a domain-level access policy that allows the **sts_role_arn** pipeline role to access it.

The following sample domain access policy allows the pipeline role named `pipeline-role`, which you created in the previous step, to write data to the domain named `ingestion-domain`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::your-account-id:role/pipeline-role"
      },
      "Action": ["es:DescribeDomain", "es:ESHttp*"],
      "Resource": "arn:aws:es:region:your-account-id:domain/domain-name/*"
    }
  ]
}
```

Step 3: Map the pipeline role (only for domains that use fine-grained access control)

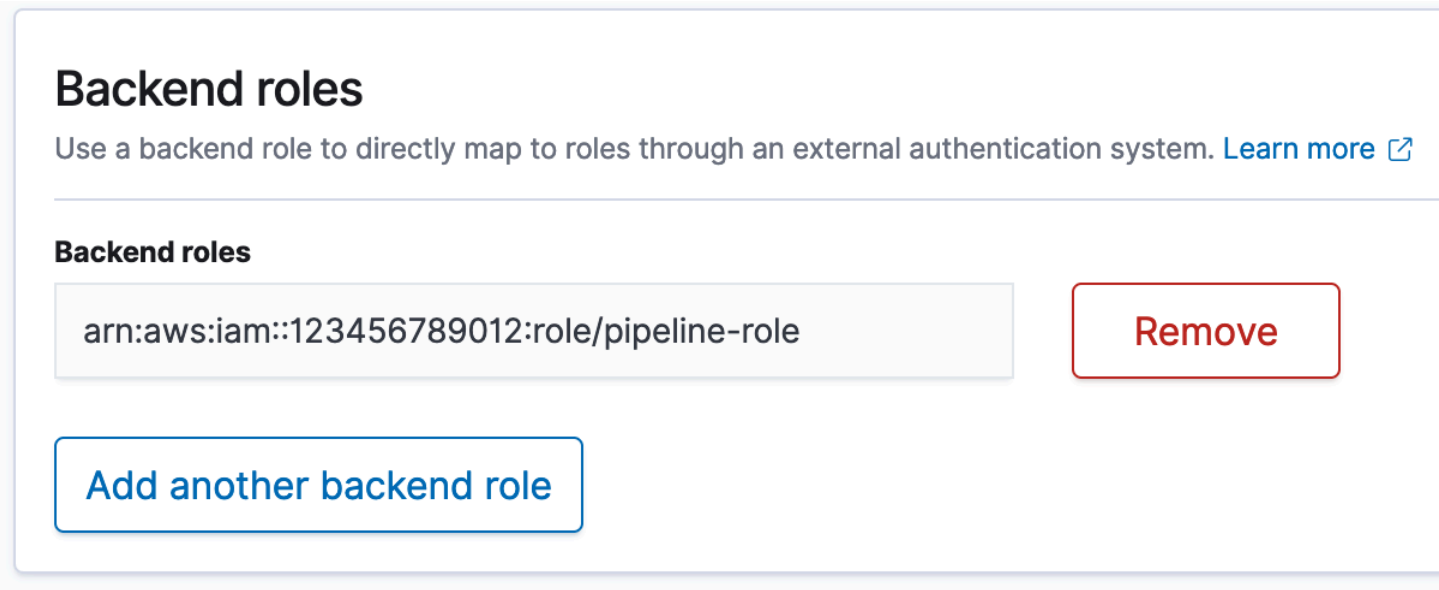
If your domain uses fine-grained access control for authentication, there are extra steps you need to take to provide your pipeline access to a domain. The steps differ depending on your domain configuration:

Scenario 1: Different master role and pipeline role – If you're using an IAM Amazon Resource Name (ARN) as the master user and it's *different* than the pipeline role (`sts_role_arn`), you need to map the pipeline role to the OpenSearch `all_access` backend role. This essentially adds the pipeline role as an additional master user. For more information, see Additional master users.

Scenario 2: Master user in the internal user database – If your domain uses a master user in the internal user database and HTTP basic authentication for OpenSearch Dashboards, you can't pass the master username and password directly into the pipeline configuration. Instead, you need to map the pipeline role (`sts_role_arn`) to the OpenSearch `all_access` backend role. This essentially adds the pipeline role as an additional master user. For more information, see Additional master users.

Scenario 3: Same master role and pipeline role (uncommon) – If you're using an IAM ARN as the master user, and it's the same ARN that you're using as the pipeline role (`sts_role_arn`), you don't need to take any further action. The pipeline has the required permissions to write to the domain. This scenario is uncommon because most environments use an admin role or some other role as the master role.

The following image shows how to map the pipeline role to a backend role:



Step 4: Specify the role in the pipeline configuration

In order to successfully create a pipeline, you must specify the pipeline role that you created in step 1 as the **sts_role_arn** parameter in your pipeline configuration. The pipeline assumes this role in order to sign requests to the OpenSearch Service domain sink.

In the `sts_role_arn` field, specify the ARN of the IAM pipeline role:

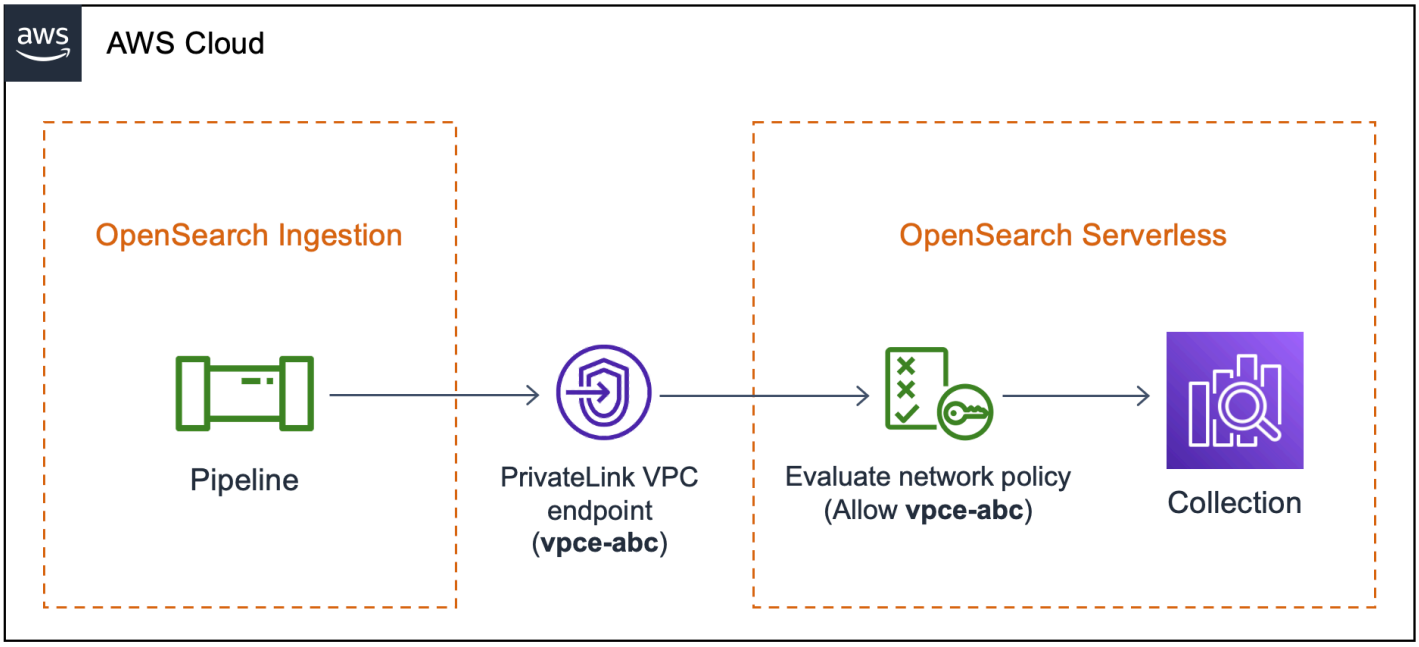
```
version: "2"
log-pipeline:
  source:
    http:
      path: "/${pipelineName}/logs"
  processor:
    - grok:
        match:
          log: [ "%{COMMONAPACHELOG}" ]
  sink:
    - opensearch:
        hosts: [ "https://search-domain-name.us-east-1.es.amazonaws.com" ]
        index: "my-index"
        aws:
          region: "region"
          sts_role_arn: "arn:aws:iam::your-account-id:role/pipeline-role"
```

For a full reference of required and unsupported parameters, see the section called “Supported plugins and options”.

Granting Amazon OpenSearch Ingestion pipelines access to collections

An Amazon OpenSearch Ingestion pipeline can write to an OpenSearch Serverless public collection or VPC collection. To provide access to the collection, you configure an AWS Identity and Access Management (IAM) pipeline role with a permissions policy that grants access to the collection. Before you specify the role in your pipeline configuration, you must configure it with an appropriate trust relationship, and then grant it data access permissions through a data access policy.

During pipeline creation, OpenSearch Ingestion creates an AWS PrivateLink connection between the pipeline and the OpenSearch Serverless collection. All traffic from the pipeline goes through this VPC endpoint and is routed to the collection. In order to reach the collection, the endpoint must be granted access to the collection through a network access policy.



Topics

Providing network access to pipelines

Each collection that you create in OpenSearch Serverless has at least one network access policy associated with it. Network access policies determine whether the collection is accessible over the internet from public networks, or whether it must be accessed privately. For more information about network policies, see the section called "Network access".

Within a network access policy, you can only specify OpenSearch Serverless-managed VPC endpoints. For more information, see the section called "VPC endpoints". However, in order for the pipeline to write to the collection, the policy must also grant access to the VPC endpoint that OpenSearch Ingestion automatically creates between the pipeline and the collection. Therefore, when you create a pipeline that has an OpenSearch Serverless collection sink, you must provide the name of the associated network policy using the `network_policy_name` option.

For example:

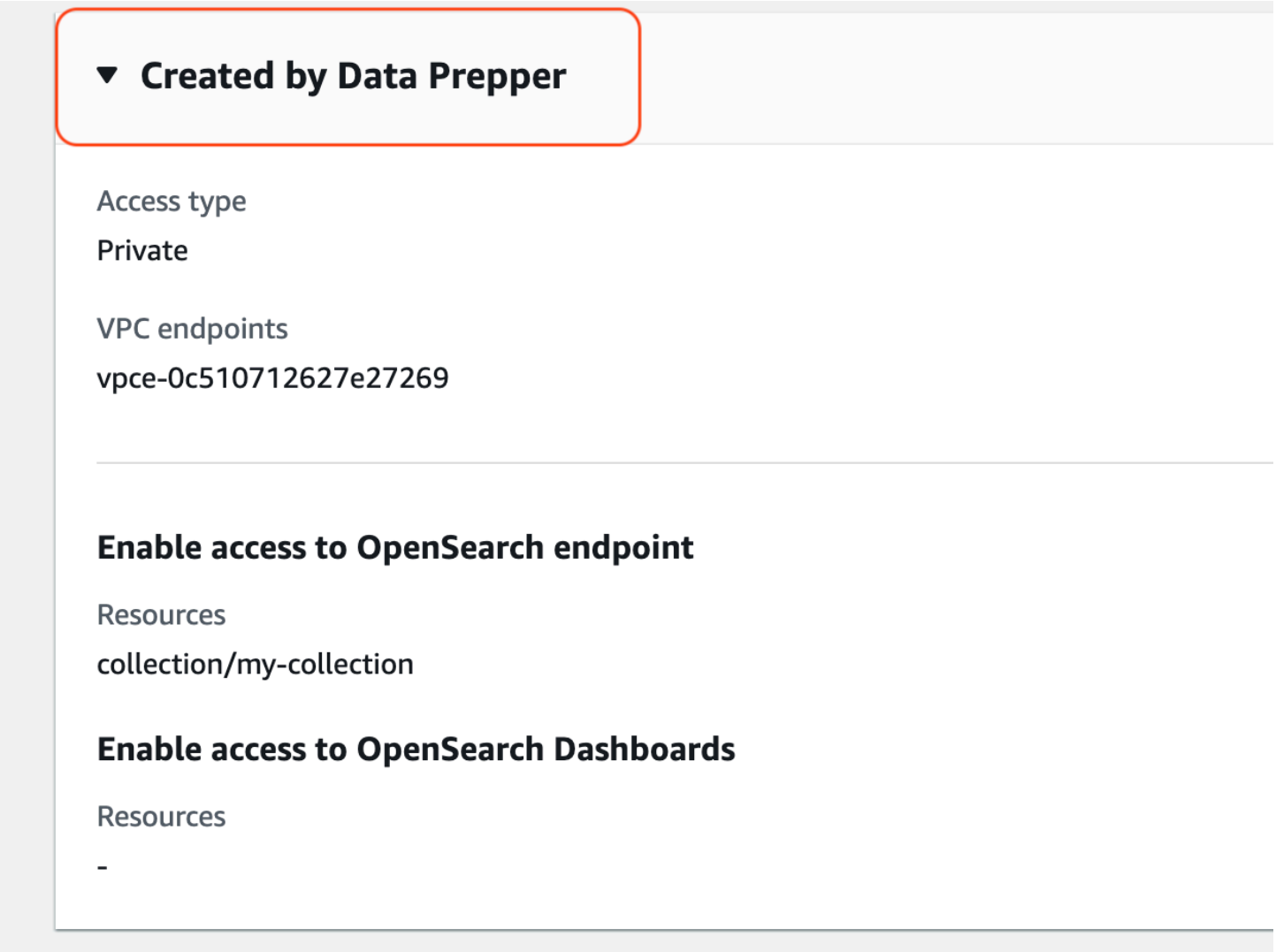
```
...
sink:
   - opensearch:
       hosts: [ "https://collection-id.region.aoss.amazonaws.com" ]
       index: "my-index"
        aws:
          serverless: true
          serverless_options:
            network_policy_name: "network-policy-name"
```

During pipeline creation, OpenSearch Ingestion checks for the existence of the specified network policy. If it doesn't exist, OpenSearch Ingestion creates it. If it does exist, OpenSearch Ingestion updates it by adding a new rule to it. The rule grants access to the VPC endpoint that connects the pipeline and the collection.

For example:

```
{
   "Rules":[
      {
         "Resource":[
            "collection/my-collection"
         ],
         "ResourceType":"collection"
      }
   ],
   "SourceVPCEs":[
      "vpce-0c510712627e27269" # The ID of the VPC endpoint that OpenSearch Ingestion
 creates between the pipeline and collection
   ],
   "Description":"Created by Data Prepper"
}
```

In the console, any rules that OpenSearch Ingestion adds to your network policies are named **Created by Data Prepper**:



Note

In general, a rule that specifies public access for a collection overrides a rule that specifies private access. Therefore, if the policy already had *public* access configured, this new rule that OpenSearch Ingestion adds doesn't actually change the behavior of the policy. For more information, see the section called “Policy precedence”.

If you stop or delete the pipeline, OpenSearch Ingestion deletes the VPC endpoint between the pipeline and the collection. It also modifies the network policy to remove the VPC endpoint from the list of allowed endpoints. If you restart the pipeline, it recreates the VPC endpoint and re-updates the network policy with the endpoint ID.

Step 1: Create a pipeline role

The role that you specify in the **sts_role_arn** parameter of a pipeline configuration must have an attached permissions policy that allows it to send data to the collection sink. It must also have a trust relationship that allows OpenSearch Ingestion to assume the role. For instructions on how to attach a policy to a role, see Adding IAM identity permissions in the *IAM User Guide*.

The following sample policy demonstrates the least privilege that you can provide in a pipeline configuration's **sts_role_arn** role for it to write to collections:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "aoss:APIAccessAll",
                "aoss:BatchGetCollection",
                "aoss:CreateSecurityPolicy",
                "aoss:GetSecurityPolicy",
                "aoss:UpdateSecurityPolicy"
            ],
            "Resource": "*"
        }
    ]
}
```

The role must have the following trust relationship, which allows OpenSearch Ingestion to assume it:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "osis-pipelines.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
```

```
}
```

Step 2: Create a collection

Create an OpenSearch Serverless collection with the following settings. For instructions to create a collection, see the section called “Creating collections”.

Data access policy

Create a data access policy for the collection that grants the required permissions to the pipeline role. For example:

```
[
  {
    "Rules": [
      {
        "Resource": [
          "index/collection-name/*"
        ],
        "Permission": [
          "aoss:CreateIndex",
          "aoss:UpdateIndex",
          "aoss:DescribeIndex",
          "aoss:WriteDocument"
        ],
        "ResourceType": "index"
      }
    ],
    "Principal": [
      "arn:aws:iam::account-id:role/pipeline-role"
    ],
    "Description": "Pipeline role access"
  }
]
```

Note

In the `Principal` element, specify the Amazon Resource Name (ARN) of the pipeline role that you created in the previous step.

Network access policy

Create a network access policy for the collection. You can ingest data into a public collection or a VPC collection. For example, the following policy provides access to a single OpenSearch Serverless-managed VPC endpoint:

```
[
   {
      "Description":"Rule 1",
      "Rules":[
         {
            "ResourceType":"collection",
            "Resource":[
               "collection/collection-name"
            ]
         }
      ],
      "AllowFromPublic": false,
      "SourceVPCEs":[
         "vpce-050f79086ee71ac05"
      ]
   }
]
```

Important

You must specify the name of the network policy within the `network_policy_name` option in the pipeline configuration. At the time of pipeline creation, OpenSearch Ingestion updates this network policy to allow access to the VPC endpoint that it automatically creates between the pipeline and the collection. See step 3 for an example pipeline configuration. For more information, see the section called "Providing network access to pipelines".

Step 3: Create a pipeline

Finally, create a pipeline in which you specify the pipeline role and collection details. The pipeline assumes this role in order to sign requests to the OpenSearch Serverless collection sink.

Make sure to do the following:

- For the `hosts` option, specify the endpoint of the collection that you created in step 2.
- For the `sts_role_arn` option, specify the Amazon Resource Name (ARN) of the pipeline role that you created in step 1.
- Set the `serverless` option to `true`.
- Set the `network_policy_name` option to the name of the network policy attached to the collection. OpenSearch Ingestion automatically updates this network policy to allow access from the VPC that it creates between the pipeline and the collection. For more information, see the section called “Providing network access to pipelines”.

```
version: "2"
log-pipeline:
  source:
    http:
        path: "/log/ingest"
  processor:
    - date:
        from_time_received: true
        destination: "@timestamp"
  sink:
    - opensearch:
        hosts: [ "https://collection-id.region.aoss.amazonaws.com" ]
        index: "my-index"
        aws:
          serverless: true
          serverless_options:
            network_policy_name: "network-policy-name" # If the policy doesn't exist, a
 new policy is created.
          region: "us-east-1"
          sts_role_arn: "arn:aws:iam::account-id:role/pipeline-role"
```

For a full reference of required and unsupported parameters, see the section called “Supported plugins and options”.

Getting started with Amazon OpenSearch Ingestion

Amazon OpenSearch Ingestion supports ingesting data into managed OpenSearch Service domains and OpenSearch Serverless collections. The following tutorials walk you through the basic steps to get a pipeline up and running.

The first tutorial shows you how to use Amazon OpenSearch Ingestion to configure a simple pipeline and ingest data into an Amazon OpenSearch Service domain.

The second tutorial shows you how to use Amazon OpenSearch Ingestion to configure a simple pipeline and ingest data into an Amazon OpenSearch Serverless collection.

> **Note**
>
> Pipeline creation will fail if you don't set up the correct permissions. See the section called "Setting up roles and users" for a better understanding of the required roles before you create a pipeline.

Topics

Tutorial: Ingesting data into a domain using Amazon OpenSearch Ingestion

This tutorial shows you how to use Amazon OpenSearch Ingestion to configure a simple pipeline and ingest data into an Amazon OpenSearch Service domain. A *pipeline* is a resource that OpenSearch Ingestion provisions and manages. You can use a pipeline to filter, enrich, transform, normalize, and aggregate data for downstream analytics and visualization in OpenSearch Service.

This tutorial walks you through the basic steps to get a pipeline up and running quickly. For more detailed information, see the section called "Creating pipelines".

You'll complete the following steps in this tutorial:

1. Create the pipeline role.
2. Create a domain.
3. Create a pipeline.
4. Ingest some sample data.

Within the tutorial, you'll create the following resources:

- A pipeline named `ingestion-pipeline`
- A domain named `ingestion-domain` that the pipeline will write to
- An IAM role named `PipelineRole` that the pipeline will assume in order to write to the domain

Required permissions

To complete this tutorial, you must have the correct IAM permissions. Your user or role must have an attached identity-based policy with the following minimum permissions. These permissions allow you to create a pipeline role (`iam:Create`), create or modify a domain (`es:*`), and work with pipelines (`osis:*`).

In addition, the `iam:PassRole` permission is required on the pipeline role resource. This permission allows you to pass the pipeline role to OpenSearch Ingestion so that it can write data to the domain.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Resource":"*",
         "Action":[
            "osis:*",
            "iam:Create*",
            "es:*"
         ]
      },
      {
         "Resource":[
            "arn:aws:iam::your-account-id:role/PipelineRole"
         ],
         "Effect":"Allow",
         "Action":[
            "iam:PassRole"
         ]
      }
   ]
}
```

Step 1: Create the pipeline role

First, create a role that the pipeline will assume in order to access the OpenSearch Service domain sink. You'll include this role within the pipeline configuration later in this tutorial.

To create the pipeline role

1. Open the AWS Identity and Access Management console at https://console.aws.amazon.com/iamv2/.
2. Choose **Policies**, and then choose **Create policy**.
3. In this tutorial, you'll ingest data into a domain called `ingestion-domain`, which you'll create in the next step. Select **JSON** and paste the following policy into the editor. Replace `{your-account-id}` with your account ID, and modify the Region if necessary.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "es:DescribeDomain",
            "Resource": "arn:aws:es:us-east-1:your-account-id:domain/ingestion-
domain"
        },
        {
            "Effect": "Allow",
            "Action": "es:ESHttp*",
            "Resource": "arn:aws:es:us-east-1:your-account-id:domain/ingestion-
domain/*"
        }
    ]
}
```

If you want to write data to an *existing* domain, replace `ingestion-domain` with the name of your domain.

> **Note**
>
> For simplicity in this tutorial, we use a fairly broad access policy. In production environments, however, we recommend that you apply a more restrictive access policy

to your pipeline role. For an example policy that provides the minimum required permissions, see the section called "Granting pipelines access to domains".

4. Choose **Next**, choose **Next**, and name your policy **pipeline-policy**.
5. Choose **Create policy**.
6. Next, create a role and attach the policy to it. Choose **Roles**, and then choose **Create role**.
7. Choose **Custom trust policy** and paste the following policy into the editor:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"osis-pipelines.amazonaws.com"
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

8. Choose **Next**. Then search for and select **pipeline-policy** (which you just created).
9. Choose **Next** and name the role **PipelineRole**.
10. Choose **Create role**.

Remember the Amazon Resource Name (ARN) of the role (for example, `arn:aws:iam::`*`your-account-id`*`:role/PipelineRole`). You'll need it when you create your pipeline.

Step 2: Create a domain

Next, create a domain named `ingestion-domain` to ingest data into.

Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home and create a domain that meets the following requirements:

- Is running OpenSearch 1.0 or later, or Elasticsearch 7.4 or later
- Uses public access
- Does not use fine-grained access control

> **Note**
>
> These requirements are meant to ensure simplicity in this tutorial. In production environments, you can configure a domain with VPC access and/or use fine-grained access control. To use fine-grained access control, see Map the pipeline role.

The domain must have an access policy that grants permission to `PipelineRole`, which you created in the previous step. The pipeline will assume this role (named **sts_role_arn** in the pipeline configuration) in order to send data to the OpenSearch Service domain sink.

Make sure that the domain has the following domain-level access policy, which grants `PipelineRole` access to the domain. Replace the Region and account ID with your own:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::your-account-id:role/PipelineRole"
      },
      "Action": "es:*",
      "Resource": "arn:aws:es:us-east-1:your-account-id:domain/ingestion-domain/*"
    }
  ]
}
```

For more information about creating domain-level access policies, see Resource-based access policies.

If you already have a domain created, modify its existing access policy to provide the above permissions to `PipelineRole`.

> **Note**
>
> Remember the domain endpoint (for example, `https://search-ingestion-domain.us-east-1.es.amazonaws.com`). You'll use it in the next step to configure your pipeline.

Step 3: Create a pipeline

Now that you have a domain and a role with the appropriate access rights, you can create a pipeline.

To create a pipeline

1. Within the Amazon OpenSearch Service console, choose **Pipelines** from the left navigation pane.
2. Choose **Create pipeline**.
3. Select the blank pipeline, then choose **Select blueprint**.
4. Name the pipeline **ingestion-pipeline** and keep the capacity settings as their defaults.
5. In this tutorial, you'll create a simple sub-pipeline called `log-pipeline` that uses the Http source plugin. This plugin accepts log data in a JSON array format. You'll specify a single OpenSearch Service domain as the sink, and ingest all data into the `application_logs` index.

   Under **Pipeline configuration**, paste the following YAML configuration into the editor:

```
version: "2"
log-pipeline:
  source:
    http:
      path: "/${pipelineName}/test_ingestion_path"
  processor:
    - date:
        from_time_received: true
        destination: "@timestamp"
  sink:
    - opensearch:
        hosts: [ "https://search-ingestion-domain.us-east-1.es.amazonaws.com" ]
        index: "application_logs"
        aws:
          sts_role_arn: "arn:aws:iam::your-account-id:role/PipelineRole"
          region: "us-east-1"
```

> **Note**
>
> The `path` option specifies the URI path for ingestion. This option is required for pull-based sources. For more information, see the section called "Specifying the ingestion path".

6. Replace the `hosts` URL with the endpoint of the domain that you created (or modified) in the previous section. Replace the `sts_role_arn` parameter with the ARN of `PipelineRole`.
7. Choose **Validate pipeline** and make sure that the validation succeeds.
8. For simplicity in this tutorial, configure public access for the pipeline. Under **Network**, choose **Public access**.

   For information about configuring VPC access, see the section called "Configuring VPC access for pipelines".
9. Keep log publishing enabled in case you encounter any issues while completing this tutorial. For more information, see the section called "Monitoring pipeline logs".

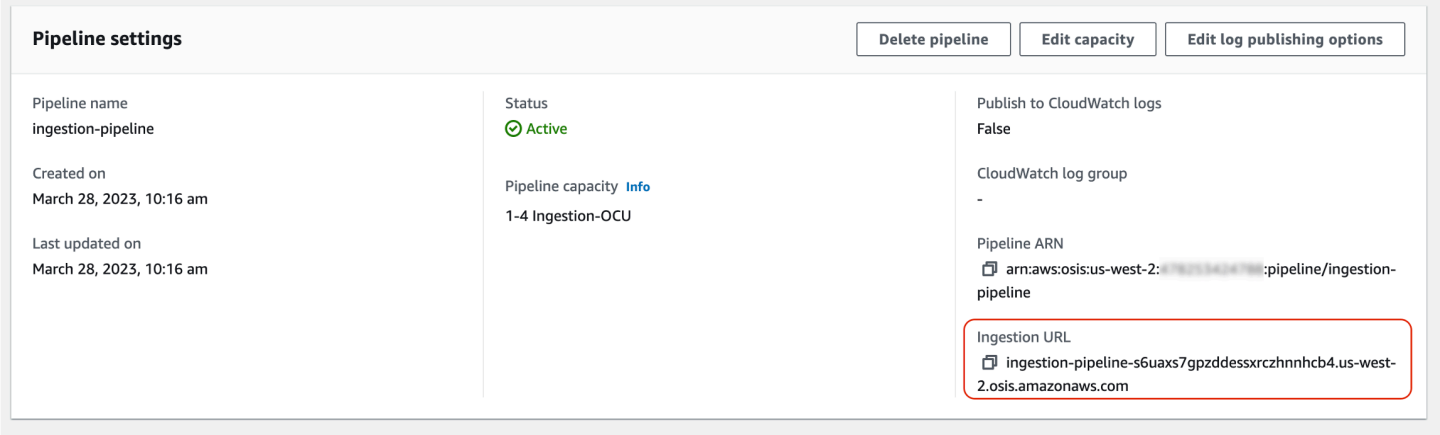
   Specify the following log group name: `/aws/vendedlogs/OpenSearchIngestion/ingestion-pipeline/audit-logs`
10. Choose **Next**. Review your pipeline configuration and choose **Create pipeline**. The pipeline takes 5–10 minutes to become active.

Step 4: Ingest some sample data

When the pipeline status is `Active`, you can start ingesting data into it. You must sign all HTTP requests to the pipeline using Signature Version 4. Use an HTTP tool such as Postman or awscurl to send some data to the pipeline. As with indexing data directly to a domain, ingesting data into a pipeline always requires either an IAM role or an IAM access key and secret key.

> **Note**
>
> The principal signing the request must have the `osis:Ingest` IAM permission.

First, get the ingestion URL from the **Pipeline settings** page:



Then, ingest some sample data. The following request uses awscurl to send a single log file to the `application_logs` index:

```
awscurl --service osis --region us-east-1 \
    -X POST \
    -H "Content-Type: application/json" \
    -d
 '[{"time":"2014-08-11T11:40:13+00:00","remote_addr":"122.226.223.69","status":"404","request":
 http://www.k2proxy.com//hello.html HTTP/1.1","http_user_agent":"Mozilla/4.0
 (compatible; WOW64; SLCC2;)"}]' \
    https://pipeline-endpoint.us-east-1.osis.amazonaws.com/log-pipeline/
test_ingestion_path
```

You should see a `200 OK` response. If you get an authentication error, it might be because you're ingesting data from a separate account than the pipeline is in. See the section called "Fixing permissions issues".

Now, query the `application_logs` index to ensure that your log entry was successfully ingested:

```
awscurl --service es --region us-east-1 \
     -X GET \
     https://search-ingestion-domain.us-east-1.es.amazonaws.com/application_logs/
_search | json_pp
```

Sample response:

```
{
   "took":984,
   "timed_out":false,
   "_shards":{
```

```
      "total":1,
      "successful":5,
      "skipped":0,
      "failed":0
   },
   "hits":{
      "total":{
         "value":1,
         "relation":"eq"
      },
      "max_score":1.0,
      "hits":[
         {
            "_index":"application_logs",
            "_type":"_doc",
            "_id":"z6VY_IMBRpceX-DU6V4O",
            "_score":1.0,
            "_source":{
               "time":"2014-08-11T11:40:13+00:00",
               "remote_addr":"122.226.223.69",
               "status":"404",
               "request":"GET http://www.k2proxy.com//hello.html HTTP/1.1",
               "http_user_agent":"Mozilla/4.0 (compatible; WOW64; SLCC2;)",
               "@timestamp":"2022-10-21T21:00:25.502Z"
            }
         }
      ]
   }
}
```


Fixing permissions issues

If you followed the steps in the tutorial and you still see authentication errors when you try to ingest data, it might be because the role that is writing to a pipeline is in a different AWS account than the pipeline itself. In this case, you need to create and assume a role that specifically enables you to ingest data. For instructions, see the section called “Providing cross-account ingestion access”.

Related resources

This tutorial presented a simple use case of ingesting a single document over HTTP. In production scenarios, you'll configure your client applications (such as Fluent Bit, Kubernetes, or the

OpenTelemetry Collector) to send data to one or more pipelines. Your pipelines will likely be more complex than the simple example in this tutorial.

To get started configuring your clients and ingesting data, see the following resources:

- Creating and managing pipelines
- Configuring your clients to send data to OpenSearch Ingestion
- Data Prepper documentation

Tutorial: Ingesting data into a collection using Amazon OpenSearch Ingestion

This tutorial shows you how to use Amazon OpenSearch Ingestion to configure a simple pipeline and ingest data into an Amazon OpenSearch Serverless collection. A *pipeline* is a resource that OpenSearch Ingestion provisions and manages. You can use a pipeline to filter, enrich, transform, normalize, and aggregate data for downstream analytics and visualization in OpenSearch Service.

For a tutorial that demonstrates how to ingest data into a provisioned OpenSearch Service *domain*, see the section called “Tutorial: Ingest data into a domain”.

You'll complete the following steps in this tutorial:

1. Create the pipeline role.
2. Create a collection.
3. Create a pipeline.
4. Ingest some sample data.

Within the tutorial, you'll create the following resources:

- A pipeline named `ingestion-pipeline-serverless`
- A collection named `ingestion-collection` that the pipeline will write to
- An IAM role named `PipelineRole` that the pipeline will assume in order to write to the collection

Required permissions

To complete this tutorial, you must have the correct IAM permissions. Your user or role must have an attached identity-based policy with the following minimum permissions. These permissions allow you to create a pipeline role (`iam:Create*`), create or modify a collection (`aoss:*`), and work with pipelines (`osis:*`).

In addition, the `iam:PassRole` permission is required on the pipeline role resource. This permission allows you to pass the pipeline role to OpenSearch Ingestion so that it can write data to the collection.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Resource":"*",
         "Action":[
            "osis:*",
            "iam:Create*",
            "aoss:*"
         ]
      },
      {
         "Resource":[
            "arn:aws:iam::your-account-id:role/PipelineRole"
         ],
         "Effect":"Allow",
         "Action":[
            "iam:PassRole"
         ]
      }
   ]
}
```

Step 1: Create the pipeline role

First, create a role that the pipeline will assume in order to access the OpenSearch Serverless collection sink. You'll include this role within the pipeline configuration later in this tutorial.

To create the pipeline role

1. Open the AWS Identity and Access Management console at https://console.aws.amazon.com/iamv2/.
2. Choose **Policies**, and then choose **Create policy**.
3. Select **JSON** and paste the following policy into the editor. Modify the collection ARN and name accordingly.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "aoss:BatchGetCollection",
        "aoss:APIAccessAll"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:aoss:us-east-1:your-account-id:collection/collection-id"
    },
    {
      "Action": [
        "aoss:CreateSecurityPolicy",
        "aoss:GetSecurityPolicy",
        "aoss:UpdateSecurityPolicy"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aoss:collection": "collection-name"
        }
      }
    }
  ]
}
```

4. Choose **Next**, choose **Next**, and name your policy **collection-pipeline-policy**.
5. Choose **Create policy**.
6. Next, create a role and attach the policy to it. Choose **Roles**, and then choose **Create role**.
7. Choose **Custom trust policy** and paste the following policy into the editor:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"osis-pipelines.amazonaws.com"
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

8. Choose **Next**. Then search for and select **collection-pipeline-policy** (which you just created).
9. Choose **Next** and name the role **PipelineRole**.
10. Choose **Create role**.

Remember the Amazon Resource Name (ARN) of the role (for example, `arn:aws:iam::`*`your-account-id`*`:role/PipelineRole`). You'll need it when you create your pipeline.

Step 2: Create a collection

Next, create a collection to ingest data into. We'll name the collection `ingestion-collection`.

1. Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Choose **Collections** from the left navigation and choose **Create collection**.
3. Name the collection **ingestion-collection**.
4. For **Security**, choose **Standard create**.
5. Under **Network access settings**, change the access type to **Public**.
6. Keep all other settings as their defaults and choose **Next**.
7. Now, configure a data acces policy for the collection. For **Definition method**, choose **JSON** and paste the following policy into the editor. This policy does two things:

   - Allows the pipeline role to write to the collection.

- Allows you to *read* from the collection. Later, after you ingest some sample data into the pipeline, you'll query the collection to ensure that the data was successfully ingested and written to the index.

```
[
  {
    "Rules": [
      {
        "Resource": [
          "index/ingestion-collection/*"
        ],
        "Permission": [
          "aoss:CreateIndex",
          "aoss:UpdateIndex",
          "aoss:DescribeIndex",
          "aoss:ReadDocument",
          "aoss:WriteDocument"
        ],
        "ResourceType": "index"
      }
    ],
    "Principal": [
      "arn:aws:iam::your-account-id:role/PipelineRole",
      "arn:aws:iam::your-account-id:role/Admin"
    ],
    "Description": "Rule 1"
  }
]
```

8. Replace the `Principal` elements. The first principal should specify the pipeline role that you created. The second should specify a user or role that you can use to query the collection later.
9. Choose **Next**. Name the access policy **pipeline-domain-access** and choose **Next** again.
10. Review your collection configuration and choose **Submit**.

When the collection is active, note the OpenSearch endpoint under **Endpoint** (for example, `https://{collection-id}.us-east-1.aoss.amazonaws.com`). You'll need it when you create your pipeline.

Step 3: Create a pipeline

Now that you have a collection and a role with the appropriate access rights, you can create a pipeline.

To create a pipeline

1. Within the Amazon OpenSearch Service console, choose **Pipelines** from the left navigation pane.
2. Choose **Create pipeline**.
3. Select the blank pipeline, then choose **Select blueprint**.
4. Name the pipeline **serverless-ingestion** and keep the capacity settings as their defaults.
5. In this tutorial, we'll create a simple sub-pipeline called `log-pipeline` that uses the HTTP source plugin. The plugin accepts log data in a JSON array format. We'll specify a single OpenSearch Serverless collection as the sink, and ingest all data into the `my_logs` index.

   Under **Pipeline configuration**, paste the following YAML configuration into the editor:

   ```
   version: "2"
   log-pipeline:
     source:
       http:
         path: "/${pipelineName}/test_ingestion_path"
     processor:
       - date:
           from_time_received: true
           destination: "@timestamp"
     sink:
       - opensearch:
           hosts: [ "https://collection-id.us-east-1.aoss.amazonaws.com" ]
           index: "my_logs"
           aws:
             sts_role_arn: "arn:aws:iam::your-account-id:role/PipelineRole"
             region: "us-east-1"
             serverless: true
   ```

6. Replace the `hosts` URL with the endpoint of the collection that you created in the previous section. Replace the `sts_role_arn` parameter with the ARN of `PipelineRole`. Optionally, modify the `region`.
7. Choose **Validate pipeline** and make sure that the validation succeeds.

8. For simplicity in this tutorial, we'll configure public access for the pipeline. Under **Network**, choose **Public access**.

   For information about configuring VPC access, see the section called "Configuring VPC access for pipelines".

9. Keep log publishing enabled in case you encounter any issues while completing this tutorial. For more information, see the section called "Monitoring pipeline logs".

   Specify the following log group name: `/aws/vendedlogs/OpenSearchIngestion/serverless-ingestion/audit-logs`

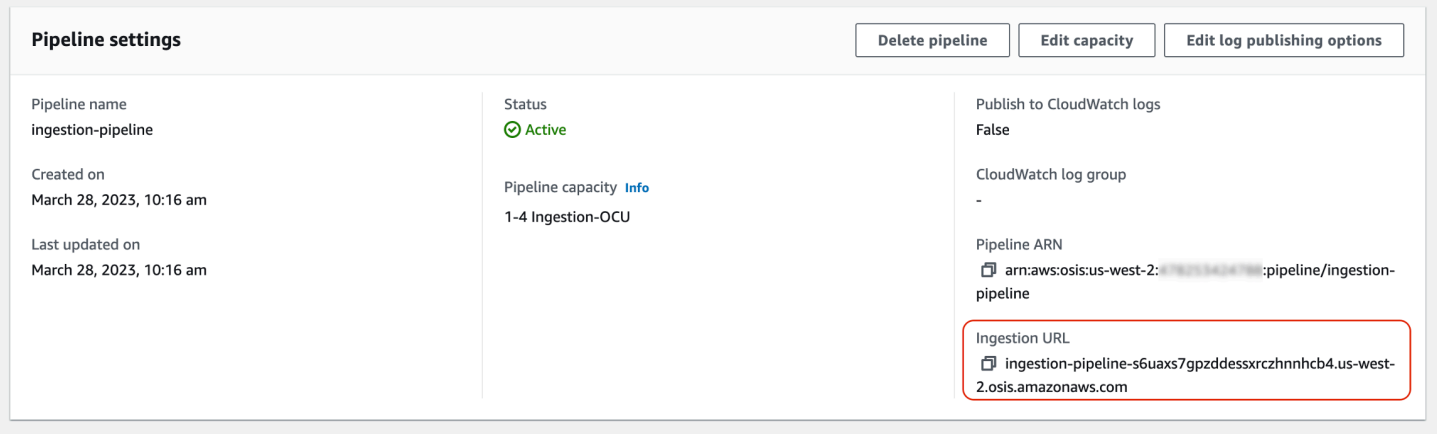
10. Choose **Next**. Review your pipeline configuration and choose **Create pipeline**. The pipeline takes 5–10 minutes to become active.

Step 4: Ingest some sample data

When the pipeline status is `Active`, you can start ingesting data into it. You must sign all HTTP requests to the pipeline using Signature Version 4. Use an HTTP tool such as Postman or awscurl to send some data to the pipeline. As with indexing data directly to a collection, ingesting data into a pipeline always requires either an IAM role or an IAM access key and secret key.

> **Note**
>
> The principal signing the request must have the `osis:Ingest` IAM permission.

First, get the ingestion URL from the **Pipeline settings** page:



Then, ingest some sample data. The following sample request uses awscurl to send a single log file to the `my_logs` index:

```
awscurl --service osis --region us-east-1 \
    -X POST \
    -H "Content-Type: application/json" \
    -d
 '[{"time":"2014-08-11T11:40:13+00:00","remote_addr":"122.226.223.69","status":"404","request":
 http://www.k2proxy.com//hello.html HTTP/1.1","http_user_agent":"Mozilla/4.0
 (compatible; WOW64; SLCC2;)"}]' \
    https://pipeline-endpoint.us-east-1.osis.amazonaws.com/log-pipeline/
test_ingestion_path
```

You should see a `200 OK` response.

Now, query the `my_logs` index to ensure that the log entry was successfully ingested:

```
awscurl --service aoss --region us-east-1 \
     -X GET \
     https://collection-id.us-east-1.aoss.amazonaws.com/my_logs/_search | json_pp
```

Sample response:

```
{
   "took":348,
   "timed_out":false,
   "_shards":{
      "total":0,
      "successful":0,
      "skipped":0,
      "failed":0
   },
   "hits":{
      "total":{
         "value":1,
         "relation":"eq"
      },
      "max_score":1.0,
      "hits":[
         {
            "_index":"my_logs",
            "_id":"1%3A0%3ARJgDvIcBTy5m12xrKE-y",
            "_score":1.0,
```

```
            "_source":{
               "time":"2014-08-11T11:40:13+00:00",
               "remote_addr":"122.226.223.69",
               "status":"404",
               "request":"GET http://www.k2proxy.com//hello.html HTTP/1.1",
               "http_user_agent":"Mozilla/4.0 (compatible; WOW64; SLCC2;)",
               "@timestamp":"2023-04-26T05:22:16.204Z"
            }
         }
      ]
   }
}
```

Related resources

This tutorial presented a simple use case of ingesting a single document over HTTP. In production scenarios, you'll configure your client applications (such as Fluent Bit, Kubernetes, or the OpenTelemetry Collector) to send data to one or more pipelines. Your pipelines will likely be more complex than the simple example in this tutorial.

To get started configuring your clients and ingesting data, see the following resources:

- Creating and managing pipelines
- Configuring your clients to send data to OpenSearch Ingestion
- Data Prepper documentation

Overview of pipeline features in Amazon OpenSearch Ingestion

Amazon OpenSearch Ingestion provisions *pipelines*, which consist of a source, a buffer, zero or more processors, and one or more sinks. Ingestion pipelines are powered by Data Prepper as the data engine. For an overview of the various components of a pipeline, see the section called “Key concepts”.

The following sections provide an overview of some of the most commonly used features in Amazon OpenSearch Ingestion.

> **Note**
>
> This is not an exhaustive list of features that are available for pipelines. For comprehensive documentation of all available pipeline functionality, see the Data Prepper documentation.

Note that OpenSearch Ingestion places some constraints on the plugins and options that you can use. For more information, see the section called “Supported plugins and options”.

Persistent buffering

A persistent buffer stores your data in a disk-based buffer across multiple Availability Zones to add durability to your data. You can use persistent buffering to ingest data for all supported push-based sources without the need to set up a standalone buffer. These include HTTP and OpenTelemetry sources for logs, traces, and metrics.

To enable persistent buffering, choose **Enable persistent buffer** when creating or updating a pipeline. For more information, see the section called “Creating pipelines”. OpenSearch Ingestion automatically determines the required buffering capacity based on the Ingestion OpenSearch Compute Units (Ingestion OCUs) that you specify for the pipeline.

If you enable persistent buffering for a pipeline, the default maximum request payload size is 10 MB for HTTP sources, and 4 MB for OpenTelemetry sources. For HTTP sources, you can increase the maximum request payload size to 20 MB. The request payload size is the size of the entire HTTP request which typically contains multiple events. Any given event may not exceed 3.5 MB. If you don’t enable persistent buffering, you will not not be able to modify the maximum payload size to 20 MB.

For pipelines with persistent buffering enabled, the configured pipeline units are split between compute units and buffer units. If a pipeline is using a CPU intensive processor (grok, key value and/or split string) then the specified units are split in a 1:1 ratio of buffer to compute otherwise they are split in a 3:1 ratio. There is a bias towards provisioning more compute units with each of these ratios.

For example:

- Pipeline with grok and 2 max units - 1 compute unit and 1 buffer units

- Pipeline with grok and 5 max units - 3 compute units and 2 buffer units
- Pipeline with no processors and 2 max units - 1 compute unit and 1 buffer units
- Pipeline with no processors and 4 max units - 1 compute unit and 3 buffer units
- Pipeline with grok and 5 max units - 2 compute units and 3 buffer units

By default, pipelines use an AWS owned key to encrypt buffer data. These pipelines don't need any additional permissions for the pipeline role. Alternately, you can specify a customer managed key and add the following IAM permissions to the pipeline role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "KeyAccess",
            "Effect": "Allow",
            "Action": [
              "kms:Decrypt",
              "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "arn:aws:kms:{region}:{aws-account-
id}:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ]
}
```

For more information, see Customer managed keys in the *AWS Key Management Service Developer Guide*.

> **Note**
>
> If you disable persistent buffering, your pipeline will be updated to run entirely on in-memory buffering.

Provisioning persistent buffering

Amazon OpenSearch Ingestion tracks data written into a sink and automatically resumes writing from the last successful check point should there be an outage in the sink or other issues that prevents data from being successfully written. There are no additional services or components needed for persistent buffers other than minimum and maximum OpenSearch compute Units (OCU) set for the pipeline. When persistent buffering is turned on, each Ingestion OCU is now capable of providing persistent buffering along with its existing ability to ingest, transform, and route data. After you enable persistent buffering, data is retained in the buffer for 72 hours. Amazon OpenSearch Ingestion dynamically allocates the buffer from the minimum and maximum allocation of OCUs that you define for the pipelines.

The number of Ingestion OCUs used for persistent buffering is dynamically calculated based on the data source, the transformations on the streaming data, and the sink that the data is written to. Because a portion of the Ingestion OCUs now applies to persistent buffering, in order to maintain the same ingestion throughput for your pipeline, you need to increase the minimum and maximum Ingestion OCUs when turning on persistent buffering. This amount of OCUs that you need with persistent buffering depends on the source that you are ingesting data from and also on the type of processing that you are performing on the data. The following table shows the number of OCUs that you need with persistent buffering with different sources and processors.

Splitting

You can configure an OpenSearch Ingestion pipeline to *split* incoming events into a sub-pipeline, allowing you to perform different types of processing on the same incoming event.

The following example pipeline splits incoming events into two sub-pipelines. Each sub-pipeline uses its own processor to enrich and manipulate the data, and then sends the data to different OpenSearch indexes.

```
version: "2"
log-pipeline:
  source:
    http:
    ...
  sink:
    - pipeline:
        name: "logs_enriched_one_pipeline"
    - pipeline:
        name: "logs_enriched_two_pipeline"

logs_enriched_one_pipeline:
  source:
    log-pipeline
  processor:
   ...
  sink:
    - opensearch:
        # Provide a domain or collection endpoint
        # Enable the 'serverless' flag if the sink is an OpenSearch Serverless
 collection
        aws:
          ...
        index: "enriched_one_logs"
```

```
logs_enriched_two_pipeline:
  source:
    log-pipeline
  processor:
   ...
  sink:
    - opensearch:
        # Provide a domain or collection endpoint
        # Enable the 'serverless' flag if the sink is an OpenSearch Serverless
 collection
        aws:
          ...
          index: "enriched_two_logs"
```

Chaining

You can *chain* multiple sub-pipelines together in order to perform data processing and enrichment in chunks. In other words, you can enrich an incoming event with certain processing capabilities in one sub-pipeline, then send it to another sub-pipeline for additional enrichment with a different processor, and finally send it to its OpenSearch sink.

In the following example, the `log_pipeline` sub-pipeline enriches an incoming log event with a set of processors, then sends the event to an OpenSearch index named `enriched_logs`. The pipeline sends the same event to the `log_advanced_pipeline` sub-pipeline, which processes it and sends it to a different OpenSearch index named `enriched_advanced_logs`.

```
version: "2"
log-pipeline:
  source:
    http:
    ...
  processor:
    ...
  sink:
    - opensearch:
        # Provide a domain or collection endpoint
        # Enable the 'serverless' flag if the sink is an OpenSearch Serverless
 collection
        aws:
          ...
          index: "enriched_logs"
```

```
    - pipeline:
        name: "log_advanced_pipeline"

log_advanced_pipeline:
  source:
    log-pipeline
  processor:
   ...
  sink:
    - opensearch:
        # Provide a domain or collection endpoint
        # Enable the 'serverless' flag if the sink is an OpenSearch Serverless
 collection
        aws:
          ...
          index: "enriched_advanced_logs"
```

Dead-letter queues

Dead-letter queues (DLQs) are destinations for events that a pipeline fails to write to a sink. In OpenSearch Ingestion, you must specify a Amazon S3 bucket with appropriate write permissions to be used as the DLQ. You can add a DLQ configuration to every sink within a pipeline. When a pipeline encounters write errors, it creates DLQ objects in the configured S3 bucket. DLQ objects exist within a JSON file as an array of failed events.

A pipeline writes events to the DLQ when either of the following conditions are met:

- The `max_retries` for the OpenSearch sink have been exhausted. OpenSearch Ingestion requires a minimum of 16 for this option.
- Events are rejected by the sink due to an error condition.

Configuration

To configure a dead-letter queue for a sub-pipeline, specify the `dlq` option within the `opensearch` sink configuration:

```
apache-log-pipeline:
  ...
  sink:
    opensearch:
      dlq:
```

```
        s3:
          bucket: "my-dlq-bucket"
          key_path_prefix: "dlq-files"
          region: "us-west-2"
          sts_role_arn: "arn:aws:iam::123456789012:role/dlq-role"
```

Files written to this S3 DLQ will have the following naming pattern:

```
dlq-v${version}-${pipelineName}-${pluginId}-${timestampIso8601}-${uniqueId}
```

For more information, see Dead-Letter Queues (DLQ).

For instructions to configure the `sts_role_arn` role, see the section called "Writing to a dead-letter queue".

Example

Consider the following example DLQ file:

```
dlq-v2-apache-log-pipeline-opensearch-2023-04-05T15:26:19.152938Z-e7eb675a-
f558-4048-8566-dac15a4f8343
```

Here's an example of data that failed to be written to the sink, and is sent to the DLQ S3 bucket for further analysis:

```
Record_0
pluginId            "opensearch"
pluginName          "opensearch"
pipelineName        "apache-log-pipeline"
failedData
index    "logs"
indexId   null
status    0
message  "Number of retries reached the limit of max retries (configured value 15)"
document
log      "sample log"
timestamp     "2023-04-14T10:36:01.070Z"

Record_1
pluginId            "opensearch"
pluginName          "opensearch"
pipelineName        "apache-log-pipeline"
```

```
failedData
index               "logs"
indexId   null
status    0
message  "Number of retries reached the limit of max retries (configured value 15)"
document
log                 "another sample log"
timestamp           "2023-04-14T10:36:01.071Z"
```

Index management

Amazon OpenSearch Ingestion has many index management capabilities, including the following.

Creating indexes

You can specify an index name in a pipeline sink and OpenSearch Ingestion creates the index when it provisions the pipeline. If an index already exists, the pipeline uses it to index incoming events. If you stop and restart a pipeline, or if you update its YAML configuration, the pipeline attempts to create new indexes if they don't already exist. A pipeline can never delete an index.

The following example sinks create two indexes when the pipeline is provisioned:

```
sink:
  - opensearch:
      index: apache_logs
  - opensearch:
      index: nginx_logs
```

Generating index names and patterns

You can generate dynamic index names by using variables from the fields of incoming events. In the sink configuration, use the format `string${}` to signal string interpolation, and use a JSON pointer to extract fields from events. The options for `index_type` are `custom` or `management_disabled`. Because `index_type` defaults to `custom` for OpenSearch domains and `management_disabled` for OpenSearch Serverless collections, it can be left unset.

For example, the following pipeline selects the `metadataType` field from incoming events to generate index names.

```
pipeline:
  ...
```

```
  sink:
    opensearch:
      index: "metadata-${metadataType}"
```

The following configuration continues to generate a new index every day or every hour.

```
pipeline:
  ...
  sink:
    opensearch:
      index: "metadata-${metadataType}-%{yyyy.MM.dd}"

pipeline:
  ...
  sink:
    opensearch:
      index: "metadata-${metadataType}-%{yyyy.MM.dd.HH}"
```

The index name can also be a plain string with a date-time pattern as a suffix, such as `my-index-%{yyyy.MM.dd}`. When the sink sends data to OpenSearch, it replaces the date-time pattern with UTC time and creates a new index for each day, such as `my-index-2022.01.25`. For more information, see the DateTimeFormatter class.

This index name can also be a formatted string (with or without a date-time pattern suffix), such as `my-${index}-name`. When the sink sends data to OpenSearch, it replaces the `"${index}"` portion with the value in the event being processed. If the format is `"${index1/index2/index3}"`, it replaces the field `index1/index2/index3` with its value in the event.

Generating document IDs

A pipeline can generate a document ID while indexing documents to OpenSearch. It can infer these document IDs from the fields within incoming events.

This example uses the `uuid` field from an incoming event to generate a document ID.

```
pipeline:
  ...
  sink:
    opensearch:
      index_type: custom
      index: "metadata-${metadataType}-%{yyyy.MM.dd}"
```

```
      "document_id": "uuid"
```

In the following example, the Add entries processor merges the fields `uuid` and `other_field` from the incoming event to generate a document ID.

The `create` action ensures that documents with identical IDs aren't overwritten. The pipeline drops duplicate documents without any retry or DLQ event. This is a reasonable expectation for pipeline authors who use this action, because the goals is to avoid updating existing documents.

```
pipeline:
  ...
  processor:
   - add_entries:
      entries:
        - key: "my_doc_id_field"
          format: "${uuid}-${other_field}"
  sink:
    - opensearch:
        ...
        action: "create"
        document_id: "my_doc_id"
```

You might want to set an event's document ID to a field from a sub-object. In the following example, the OpenSearch sink plugin uses the sub-object `info/id` to generate a document ID.

```
sink:
  - opensearch:
        ...
        document_id: info/id
```

Given the following event, the pipeline will generate a document with the `_id` field set to `json001`:

```
{
   "fieldA":"arbitrary value",
   "info":{
      "id":"json001",
      "fieldA":"xyz",
      "fieldB":"def"
   }
}
```

Generating routing IDs

You can use the `routing_field` option within the OpenSearch sink plugin to set the value of a document routing property (`_routing`) to a value from an incoming event.

Routing supports JSON pointer syntax, so nested fields also are available, not just top-level fields.

```
sink:
  - opensearch:
      ...
      routing_field: metadata/id
      document_id: id
```

Given the following event, the plugin generates a document with the `_routing` field set to `abcd`:

```
{
   "id":"123",
   "metadata":{
      "id":"abcd",
      "fieldA":"valueA"
   },
   "fieldB":"valueB"
}
```

For instructions to create index templates that pipelines can use during index creation, see Index templates.

End-to-end acknowledgement

OpenSearch Ingestion ensures the durability and reliability of data by tracking its delivery from source to sinks in stateless pipelines using *end-to-end acknowledgement*. Currently, only the S3 source plugin supports end-to-end acknowledgement.

With end-to-end acknowledgement, the pipeline source plugin creates an *acknowledgement set* to monitor a batch of events. It receives a positive acknowledgement when those events are successfully sent to their sinks, or a negative acknowledgement when any of the events could not be sent to their sinks.

In the event of a failure or crash of a pipeline component, or if a source fails to receive an acknowledgement, the source times out and takes necessary actions such as retrying or logging

the failure. If the pipeline has multiple sinks or multiple sub-pipelines configured, event-level acknowledgements are sent only after the event is sent to *all* sinks in *all* sub-pipelines. If a sink has a DLQ configured, end-to-end acknowledgements also tracks events written to the DLQ.

To enable end-to-end acknowledgement, include the `acknowledgments` option within the source configuration:

```
s3-pipeline:
  source:
    s3:
      acknowledgments: true
...
```

Source back pressure

A pipeline can experience back pressure when it's busy processing data, or if its sinks are temporarily down or slow to ingest data. OpenSearch Ingestion has different ways of handling back pressure depending on the source plugin that a pipeline is using.

HTTP source

Pipelines that use the HTTP source plugin handle back pressure differently depending on which pipeline component is congested:

- **Buffers** – When buffers are full, the pipeline starts returning HTTP status `REQUEST_TIMEOUT` with error code 408 back to the source endpoint. As buffers are freed up, the pipeline starts processing HTTP events again.
- **Source threads** – When all HTTP source threads are busy executing requests and the unprocessed request queue size has exceeded the maximum allowed number of requests, the pipeline starts to return HTTP status `TOO_MANY_REQUESTS` with error code 429 back to the source endpoint. When the request queue drops below the maximum allowed queue size, the pipeline starts processing requests again.

OTel source

When buffers are full for pipelines that use OpenTelemetry sources (OTel logs, OTel metrics, and OTel trace), the pipeline starts to return HTTP status `REQUEST_TIMEOUT` with error code 408 to the source endpoint. As buffers are freed up, the pipeline starts processing events again.

S3 source

When buffers are full for pipelines with an S3 source, the pipelines stop processing SQS notifications. As the buffers are freed up, the pipelines start processing notifications again.

If a sink is down or unable to ingest data and end-to-end acknowledgement is enabled for the source, the pipeline stops processing SQS notifications until it receives a successful acknowledgement from all sinks.

Creating Amazon OpenSearch Ingestion pipelines

A *pipeline* is the mechanism that Amazon OpenSearch Ingestion uses to move data from its *source* (where the data comes from) to its *sink* (where the data goes). In OpenSearch Ingestion, the sink will always be a single Amazon OpenSearch Service domain, while the source of your data could be clients like Amazon S3, Fluent Bit, or the OpenTelemetry Collector.

For more information, see Pipelines in the OpenSearch documentation.

Prerequisites and required IAM role

To create an OpenSearch Ingestion pipeline, you must have the following resources:

- An IAM role that OpenSearch Ingestion will assume to write to the sink. You will include this role ARN in your pipeline configuration.
- An OpenSearch Service domain or OpenSearch Serverless collection to act as the sink. If you're writing to a domain, it must be running OpenSearch 1.0 or later, or Elasticsearch 7.4 or later. The sink must have an access policy that grants the appropriate permissions to your IAM pipeline role.

For instructions to create these resources, see the following topics:

- the section called “Granting pipelines access to domains”
- the section called “Granting pipelines access to collections”

> **Note**
>
> If you're writing to a domain that uses fine-grained access control, there are extra steps you need to complete. See the section called “Step 3: Map the pipeline role (only for domains that use fine-grained access control)”.

Required IAM permissions

OpenSearch Ingestion uses the following IAM permissions for creating pipelines:

- `osis:CreatePipeline` – Create a pipeline.
- `osis:ValidatePipeline` – Check whether a pipeline configuration is valid.
- `iam:PassRole` – Pass the pipeline role to OpenSearch Ingestion so that it can write data to the domain. This permission must be on the pipeline role resource (the ARN that you specify for the `sts_role_arn` option in the pipeline configuration), or simply `*` if you plan to use different roles in each pipeline.

For example, the following policy grants permission to create a pipeline:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Resource":"*",
         "Action":[
            "osis:CreatePipeline",
            "osis:ListPipelineBlueprints",
            "osis:ValidatePipeline"
         ]
      },
      {
         "Resource":[
            "arn:aws:iam::your-account-id:role/pipeline-role"
         ],
         "Effect":"Allow",
         "Action":[
            "iam:PassRole"
         ]
```

```
      }
   ]
}
```

OpenSearch Ingestion also includes a permission called `osis:Ingest`, which is required in order to send signed requests to the pipeline using Signature Version 4. For more information, see the section called "Creating an ingestion role".

> **Note**
>
> In addition, the first user to create a pipeline in an account must have permissions for the `iam:CreateServiceLinkedRole` action. For more information, see pipeline role resource.

For more information about each permission, see Actions, resources, and condition keys for OpenSearch Ingestion in the *Service Authorization Reference*.

Specifying the pipeline version

When you configure a pipeline, you must specify the major version of Data Prepper that the pipeline will run. To specify the version, include the `version` option in your pipeline configuration:

```
version: "2"
log-pipeline:
  source:
    ...
```

When you choose **Create**, OpenSearch Ingestion determines the latest available *minor* version of the major version that you specify, and provisions the pipeline with that version. For example, if you specify `version: "2"`, and the latest supported version of Data Prepper is 2.1.1, OpenSearch Ingestion provisions your pipeline with version 2.1.1. We don't publicly display the minor version that your pipeline is running.

In order to upgrade your pipeline when a new major version of Data Prepper is available, edit the pipeline configuration and specify the new version. You can't downgrade a pipeline to an earlier version.

> **Note**
>
> OpenSearch Ingestion doesn't immediately support new versions of Data Prepper as soon as they're released. There will be some lag between when a new version is publicly available and when it's supported in OpenSearch Ingestion. In addition, OpenSearch Ingestion might explicitly not support certain major or minor versions altogether. For a comprehensive list, see the section called "Supported Data Prepper versions".

Any time you make a change to your pipeline that initiates a blue/green deployment, OpenSearch Ingestion can upgrade it to the latest minor version of the major version that's currently configured in the pipeline YAML file. For more information, see the section called "Blue/green deployments for pipeline updates". OpenSearch Ingestion can't change the major version of your pipeline unless you explicitly update the `version` option within the pipeline configuration.

Specifying the ingestion path

For pull-based sources like OTel trace and OTel metrics, OpenSearch Ingestion requires the additional `path` option in your source configuration. The path is a string such as `/log/ingest`, which represents the URI path for ingestion. This path defines the URI that you use to send data to the pipeline.

For example, say you specify the following entry sub-pipeline for an ingestion pipeline named `logs`:

```
entry-pipeline:
  source:
    http:
      path: "/my/test_path"
```

When you ingest data into the pipeline, you must specify the following endpoint in your client configuration: `https://logs-abcdefgh.us-west-2.osis.amazonaws.com/my/test_path`.

The path must start with a slash (/) and can contain the special characters '-', '_', '.', and '/', as well as the `${pipelineName}` placeholder. If you use `${pipelineName}` (such as `path: "/${pipelineName}/test_path"`), the variable is replaced with the name of the associated sub-pipeline. In this example, it would be `https://logs.us-west-2.osis.amazonaws.com/entry-pipeline/test_path`.

Creating pipelines

This section describes how to create OpenSearch Ingestion pipelines using the OpenSearch Service console and the AWS CLI.

Console

To create a pipeline

1. Sign in to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Choose **Pipelines** in the left navigation pane and choose **Create pipeline**.
3. Either select a blank pipeline, or choose a configuration blueprint. Blueprints include a preconfigured YAML and JSON configuration file for a variety of common use cases. For more information, see the section called "Using blueprints to create a pipeline".

   Choose **Select blueprint**.
4. Enter a name for the pipeline.
5. (Optional) Choose **Enable persistent buffer**. A persistent buffer stores your data in a disk-based buffer across multiple AZs. For more information, see Persistent buffering. If you enable persistent buffer, select the AWS Key Management Service key to encrypt the buffer data.
6. Configure the minimum and maximum pipeline capacity in Ingestion OpenSearch Compute Units (OCUs). For more information, see the section called "Scaling pipelines".
7. Under **Pipeline configuration**, provide your pipeline configuration in YAML format. If you're using a blueprint, the configuration is already pre-populated, but you must make some modifications.

   A single pipeline configuration file can contain 1-10 sub-pipelines. Each sub-pipeline is a combination of a single source, zero or more processors, and a single sink. For OpenSearch Ingestion, the sink must always be an OpenSearch Service domain. For a list of supported options, see the section called "Supported plugins and options".

   > **Note**
   >
   > You must include the `sts_role_arn` option in each sub-pipeline. The pipeline assumes the role defined in `sts_role_arn` to sign requests to the domain. For more information, see the section called "Granting pipelines access to domains".

The following sample configuration file uses the HTTP source and Grok plugins to process unstructured log data and send it to an OpenSearch Service domain. The sub-pipeline is named `log-pipeline`.

```
version: "2"
log-pipeline:
  source:
    http:
      path: "/log/ingest"
  processor:
    - grok:
        match:
          log: [ '%{COMMONAPACHELOG}' ]
    - date:
        from_time_received: true
        destination: "@timestamp"
  sink:
    - opensearch:
        hosts: [ "https://search-my-domain.us-east-1.es.amazonaws.com" ]
        index: "apache_logs"
        aws:
          sts_role_arn: "arn:aws:iam::123456789012:role/{pipeline-role}"
          region: "us-east-1"
```

You can build your own pipeline configuration, or choose **Upload file** and import an existing configuration for a self-managed Data Prepper pipeline. Alternatively, you can use a configuration blueprint.

8. After you configure your pipeline, choose **Validate pipeline** to confirm that your configuration is correct. If the validation fails, fix the errors and re-run the validation.
9. Under **Network configuration**, choose either **VPC access** or **Public access**. If you choose **Public access**, skip to the next step. If you choose **VPC access**, configure the following settings:

Setting	Description
Endpoint management	Choose whether you want to create your VPC endpoints yourself, or have OpenSearch Ingestion create them for you. Endpoint management defaults to endpoints managed by OpenSearch Ingestion.

Setting	Description
VPC	Choose the ID of the virtual private cloud (VPC) that you want to use. The VPC and pipeline must be in the same AWS Region.
Subnets	Choose one or more subnets. OpenSearch Service will place a VPC endpoint and *elastic network interfaces* in the subnets.
Security groups	Choose one or more VPC security groups that allow your required application to reach the OpenSearch Ingestion pipeline on the ports (80 or 443) and protocols (HTTP or HTTPs) exposed by the pipeline.
VPC attachment options	If your source is a self-managed endpoint, attach your pipeline to a VPC. Choose one of the default CIDR options provided, or use a custom CIDR.

For more information, see the section called “Configuring VPC access for pipelines”.

10. (Optional) Under **Tags**, add one or more tags (key-value pairs) to your pipeline. For more information, see the section called “Tagging pipelines”.
11. (Optional) Under **Log publishing options**, turn on pipeline log publishing to Amazon CloudWatch Logs. We recommend that you enable log publishing so that you can more easily troubleshoot pipeline issues. For more information, see the section called “Monitoring pipeline logs”.
12. Choose **Next**.
13. Review your pipeline configuration and choose **Create**.

OpenSearch Ingestion runs an asynchronous process to build the pipeline. Once the pipeline status is `Active`, you can start ingesting data.

AWS CLI

The create-pipeline command accepts the pipeline configuration as a string or within a .yaml file. If you provide the configuration as a string, each new line must be escaped with `\n`. For example, `"log-pipeline:\n source:\n http:\n processor:\n - grok:\n ...`

The following sample command creates a pipeline with the following configuration:

- Minimum of 4 Ingestion OCUs, maximum of 10 Ingestion OCUs

- Provisioned within a virtual private cloud (VPC)
- Log publishing enabled

```
aws osis create-pipeline \
  --pipeline-name my-pipeline \
  --min-units 4 \
  --max-units 10 \
  --log-publishing-options
 IsLoggingEnabled=true,CloudWatchLogDestination={LogGroup="MyLogGroup"} \
  --vpc-options
 SecurityGroupIds={sg-12345678,sg-9012345},SubnetIds=subnet-1212234567834asdf \
  --pipeline-configuration-body "file://pipeline-config.yaml"
```

OpenSearch Ingestion runs an asynchronous process to build the pipeline. Once the pipeline status is `Active`, you can start ingesting data. To check the status of the pipeline, use the GetPipeline command.

OpenSearch Ingestion API

To create an OpenSearch Ingestion pipeline using the OpenSearch Ingestion API, call the CreatePipeline operation.

After your pipeline is successfully created, you can configure your client and start ingesting data into your OpenSearch Service domain. For more information, see the section called “Integrating pipelines”.

Tracking the status of pipeline creation

You can track the status of a pipeline as OpenSearch Ingestion provisions it and prepares it to ingest data.

Console

After you initially create a pipeline, it goes through multiple stages as OpenSearch Ingestion prepares it to ingest data. To view the various stages of pipeline creation, choose the pipeline name to see its **Pipeline settings** page. Under **Status**, choose **View details**.

A pipeline goes through the following stages before it's available to ingest data:

- **Validation** – Validating pipeline configuration. When this stage is complete, all validations have succeeded.

- **Create environment** – Preparing and provisioning resources. When this stage is complete, the new pipeline environment has been created.
- **Deploy pipeline** – Deploying the pipeline. When this stage is complete, the pipeline has been successfully deployed.
- **Check pipeline health** – Checking the health of the pipeline. When this stage is complete, all health checks have passed.
- **Enable traffic** – Enabling the pipeline to ingest data. When this stage is complete, you can start ingesting data into the pipeline.

CLI

Use the get-pipeline-change-progress command to check the status of a pipeline. The following AWS CLI request checks the status of a pipeline named `my-pipeline`:

```
aws osis get-pipeline-change-progress \
    --pipeline-name my-pipeline
```

Response:

```
{
   "ChangeProgressStatuses": {
      "ChangeProgressStages": [
         {
            "Description": "Validating pipeline configuration",
            "LastUpdated": 1.671055851E9,
            "Name": "VALIDATION",
            "Status": "PENDING"
         }
      ],
      "StartTime": 1.671055851E9,
      "Status": "PROCESSING",
      "TotalNumberOfStages": 5
   }
}
```

OpenSearch Ingestion API

To track the status of pipeline creation using the OpenSearch Ingestion API, call the GetPipelineChangeProgress operation.

Using blueprints to create a pipeline

Rather than creating a pipeline definition from scratch, you can use *configuration blueprints*, which are preconfigured YAML templates for common ingestion scenarios such as Trace Analytics or Apache logs. Configuration blueprints help you easily provision pipelines without having to author a configuration from scratch.

Console

To use a pipeline blueprint

1. Sign in to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Choose **Pipelines** in the left navigation pane and choose **Create pipeline**.
3. Select a blueprint from the list of use cases, then choose **Select blueprint**. The pipeline configuration populates with a sub-pipeline for the use case you selected.
4. Review the commented-out text which guides you through configuring the blueprint.

> **Important**
>
> The pipeline blueprint isn't valid as-is. You need to make some modifications, such as providing the AWS Region and the role ARN to use for authentication, otherwise pipeline validation will fail.

CLI

To get a list of all available blueprints using the AWS CLI, send a list-pipeline-blueprints request.

```
aws osis list-pipeline-blueprints
```

The request returns a list of all available blueprints.

To get more detailed information about a specific blueprint, use the get-pipeline-blueprint command:

```
aws osis get-pipeline-blueprint --blueprint-name AWS-ApacheLogPipeline
```

This request returns the contents of the Apache log pipeline blueprint:

```
{
   "Blueprint":{
      "PipelineConfigurationBody":"###\n  # Limitations: https://docs.aws.amazon.com/
opensearch-service/latest/ingestion/ingestion.html#ingestion-limitations\n###\n###\n
 # apache-log-pipeline:\n    # This pipeline receives logs via http (e.g. FluentBit),
 extracts important values from the logs by matching\n    # the value in the 'log' key
 against the grok common Apache log pattern. The grokked logs are then sent\n    # to
 OpenSearch to an index named 'logs'\n###\n\nversion: \"2\"\napache-log-pipeline:\n
 source:\n    http:\n      # Provide the path for ingestion. ${pipelineName} will be
 replaced with pipeline name configured for this pipeline.\n      # In this case it
 would be \"/apache-log-pipeline/logs\". This will be the FluentBit output URI value.
\n      path: \"/${pipelineName}/logs\"\n  processor:\n    - grok:\n        match:\n
         log: [ \"%{COMMONAPACHELOG_DATATYPED}\" ]\n  sink:\n    - opensearch:\n
   # Provide an AWS OpenSearch Service domain endpoint\n        # hosts: [ \"https://
search-mydomain-1a2a3a4a5a6a7a8a9a0a9a8a7a.us-east-1.es.amazonaws.com\" ]\n
 aws:\n          # Provide a Role ARN with access to the domain. This role should have
 a trust relationship with osis-pipelines.amazonaws.com\n          # sts_role_arn:
 \"arn:aws:iam::123456789012:role/Example-Role\"\n          # Provide the region of the
 domain.\n          # region: \"us-east-1\"\n          # Enable the 'serverless' flag
 if the sink is an Amazon OpenSearch Serverless collection\n          # serverless:
 true\n        index: \"logs\"\n        # Enable the S3 DLQ to capture any failed
 requests in an S3 bucket\n        # dlq:\n          # s3:\n            # Provide an
 S3 bucket\n            # bucket: \"your-dlq-bucket-name\"\n            # Provide a key
 path prefix for the failed requests\n            # key_path_prefix: \"${pipelineName}/
logs/dlq\"\n            # Provide the region of the bucket.\n            # region:
 \"us-east-1\"\n            # Provide a Role ARN with access to the bucket. This role
 should have a trust relationship with osis-pipelines.amazonaws.com\n            #
 sts_role_arn: \"arn:aws:iam::123456789012:role/Example-Role\"\n",
      "BlueprintName":"AWS-ApacheLogPipeline"
   }
}
```

OpenSearch Ingestion API

To get information about pipeline blueprints using the OpenSearch Ingestion API, use the the ListPipelineBlueprints and GetPipelineBlueprint operations.

Viewing Amazon OpenSearch Ingestion pipelines

You can view the details about an Amazon OpenSearch Ingestion pipeline using the AWS Management Console, the AWS CLI, or the OpenSearch Ingestion API.

Console

To view a pipeline

1. Sign in to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Choose **Pipelines** in the left navigation pane.
3. (Optional) To view pipelines with a particular status, choose **Any status** and select a status to filter by.

   A pipeline can have the following statuses:

   - `Creating` – The pipeline is being created.
   - `Active` – The pipeline is active and ready to ingest data.
   - `Updating` – The pipeline is being updated.
   - `Deleting` – The pipeline is being deleted.
   - `Create failed` – The pipeline could not be created.
   - `Update failed` – The pipeline could not be updated.
   - `Starting` – The pipeline is starting.
   - `Start failed` – The pipeline could not be started.
   - `Stopping` – The pipeline is being stopped.
   - `Stopped` – The pipeline is stopped and can be restarted at any time.

You're not billed for Ingestion OCUs when a pipeline is in the `Create failed`, `Creating`, `Deleting`, and `Stopped` states.

CLI

To view pipelines using the AWS CLI, send a list-pipelines request:

```
aws osis list-pipelines
```

The request returns a list of all existing pipelines:

```
{
    "NextToken": null,
    "Pipelines": [
```

```
        {,
            "CreatedAt": 1.671055851E9,
            "LastUpdatedAt": 1.671055851E9,
            "MaxUnits": 4,
            "MinUnits": 2,
            "PipelineArn": "arn:aws:osis:us-west-2:123456789012:pipeline/log-pipeline",
            "PipelineName": "log-pipeline",
            "Status": "ACTIVE",
            "StatusReason": {
                "Description": "The pipeline is ready to ingest data."
            }
        },
            "CreatedAt": 1.671055851E9,
            "LastUpdatedAt": 1.671055851E9,
            "MaxUnits": 2,
            "MinUnits": 8,
            "PipelineArn": "arn:aws:osis:us-west-2:123456789012:pipeline/another-
pipeline",
            "PipelineName": "another-pipeline",
            "Status": "CREATING",
            "StatusReason": {
                "Description": "The pipeline is being created. It is not able to ingest
 data."
            }
        }
    ]
}
```

To get information about a single pipeline, use the get-pipeline command:

```
aws osis get-pipeline --pipeline-name "my-pipeline"
```

The request returns configuration information for the specified pipeline:

```
{
    "Pipeline": {
        "PipelineName": "my-pipeline",
        "PipelineArn": "arn:aws:osis:us-east-1:123456789012:pipeline/my-pipeline",
        "MinUnits": 9,
        "MaxUnits": 10,
        "Status": "ACTIVE",
        "StatusReason": {
            "Description": "The pipeline is ready to ingest data."
```

```
        },
        "PipelineConfigurationBody": "log-pipeline:\n source:\n http:\n processor:\n
 - grok:\n match:\nlog: [ '%{COMMONAPACHELOG}' ]\n - date:\n from_time_received: true
\n destination: \"@timestamp\"\n  sink:\n - opensearch:\n hosts: [ \"https://search-
mdp-performance-test-duxkb4qnycd63rpy6svmvyvfpi.us-east-1.es.amazonaws.com\" ]\n index:
 \"apache_logs\"\n aws_sts_role_arn: \"arn:aws:iam::123456789012:role/my-domain-role
\"\n  aws_region: \"us-east-1\"\n  aws_sigv4: true",,
        "CreatedAt": "2022-10-01T15:28:05+00:00",
        "LastUpdatedAt": "2022-10-21T21:41:08+00:00",
        "IngestEndpointUrls": [
            "my-pipeline-123456789012.us-east-1.osis.amazonaws.com"
        ]
    }
}
```

OpenSearch Ingestion API

To view OpenSearch Ingestion pipelines using the OpenSearch Ingestion API, call the ListPipelines and GetPipeline operations.

Updating Amazon OpenSearch Ingestion pipelines

You can update Amazon OpenSearch Ingestion pipelines using the AWS Management Console, the AWS CLI, or the OpenSearch Ingestion API. OpenSearch Ingestion initiates a blue/green deployment when you update a pipeline's YAML configuration. For more information, see the section called “Blue/green deployments for pipeline updates”.

Topics

Considerations

Consider the following when you update a pipeline:

- You can edit a pipeline's capacity limits, log publishing options, and YAML configuration. You can't edit its name or network settings.

- If your pipeline writes to a VPC domain sink, you can't go back and change the sink to a different VPC domain after the pipeline is created. You must delete and recreate the pipeline with the new sink. You can still switch the sink from a VPC domain to a public domain, from a public domain to a VPC domain, or from a public domain to another public domain.
- You can switch the pipeline sink at any time between a public OpenSearch Service domain and an OpenSearch Serverless collection.
- When you update a pipeline's YAML configuration, OpenSearch Ingestion initiates a blue/green deployment. For more information, see the section called “Blue/green deployments for pipeline updates”.
- When you update a pipeline's YAML configuration, OpenSearch Ingestion automatically upgrades your pipeline to the latest supported minor version of the major version of Data Prepper that's specified in the pipeline configuration. This process keeps your pipeline up to date with the latest bug fixes and performance improvements.
- You can still make updates to your pipeline when it's stopped.

Permissions required

OpenSearch Ingestion uses the following IAM permissions for updating pipelines:

- `osis:UpdatePipeline` – Update a pipeline.
- `osis:ValidatePipeline` – Check whether a pipeline configuration is valid.
- `iam:PassRole` – Pass the pipeline role to OpenSearch Ingestion so that it can write data to the domain. This permission is only required if you're updating the pipeline YAML configuration, not if you're modifying other settings such as log publishing or capacity limits.

For example, the following policy grants permission to update a pipeline:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Resource":"*",
         "Action":[
            "osis:UpdatePipeline",
            "osis:ValidatePipeline"
         ]
```

```
      },
      {
         "Resource":[
            "arn:aws:iam::{your-account-id}:role/{pipeline-role}"
         ],
         "Effect":"Allow",
         "Action":[
            "iam:PassRole"
         ]
      }
   ]
}
```

Updating pipelines

You can update Amazon OpenSearch Ingestion pipelines using the AWS Management Console, the AWS CLI, or the OpenSearch Ingestion API.

Console

To update a pipeline

1. Sign in to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Choose **Pipelines** in the left navigation pane.
3. Choose a pipeline to open its settings. You can edit a pipeline's capacity limits, log publishing options, and YAML configuration. You can't edit its name or network settings.
4. When you're done making changes, choose **Save**.

CLI

To update a pipeline using the AWS CLI, send an update-pipeline request. The following sample request uploads a new configuration file and updates the minimum and maximum capacity values:

```
aws osis update-pipeline \
  --pipeline-name "my-pipeline" \
  --pipline-configuration-body "file://new-pipeline-config.yaml" \
  --min-units 11 \
  --max-units 18
```

OpenSearch Ingestion API

To update an OpenSearch Ingestion pipeline using the OpenSearch Ingestion API, call the UpdatePipeline operation.

Blue/green deployments for pipeline updates

OpenSearch Ingestion initiates a *blue/green* deployment process when you update a pipeline's YAML configuration.

Blue/green refers to the practice of creating a new environment for pipeline updates and routing traffic to the new environment after those updates are complete. The practice minimizes downtime and maintains the original environment in the event that deployment to the new environment is unsuccessful. Blue/green deployments themselves don't have any performance impact, but performance might change if your pipeline configuration changes in a way that alters performance.

OpenSearch Ingestion blocks auto-scaling during blue/green deployments. You continue to be charged only for traffic to the old pipeline until it's redirected to the new pipeline. Once traffic has been redirected, you're only charged for the new pipeline. You're never charged for two pipelines simultaneously.

When you update a pipeline's YAML configuration file, OpenSearch Ingestion can automatically upgrade your pipeline to the latest supported minor version of the major version of Data Prepper that's specified in the pipeline configuration. For example, you might have `version: "2"` in your pipeline configuration, and OpenSearch Ingestion initially provisioned the pipeline with version 2.1.0. When support for version 2.1.1 is added, and you make a change to your pipeline configuration, OpenSearch Ingestion upgrades your pipeline to version 2.1.1.

This process keeps your pipeline up to date with the latest bug fixes and performance improvements. OpenSearch Ingestion can't update the major version of your pipeline unless you manually change the `version` option within the pipeline configuration.

Managing Amazon OpenSearch Ingestion pipeline costs

Stopping and starting Amazon OpenSearch Ingestion pipelines helps you manage costs for development and test environments. You can temporarily stop a pipeline instead of setting it up and tearing it down each time that you use the pipeline.

Overview of stopping and starting an OpenSearch Ingestion pipeline

You can stop a pipeline during periods where you don't need to ingest data into it. You can start the pipeline again anytime you need to use it. Starting and stopping simplifies the setup and teardown processes for pipelines used for development, testing, or similar activities that don't require continuous availability.

While your pipeline is stopped, you aren't charged for any Ingestion OCU hours. You can still update stopped pipelines, and they receive automatic minor version updates and security patches.

Don't use starting and stopping if you need to keep your pipeline running but it has more capacity than you need. If your pipeline is too costly or not very busy, consider reducing its maximum capacity limits. For more information, see the section called “Scaling pipelines”.

Stopping an OpenSearch Ingestion pipeline

To use an OpenSearch Ingestion pipeline or perform administration, you always begin with an active pipeline, then stop the pipeline, and then start the pipeline again. While your pipeline is stopped, you're not charged for Ingestion OCU hours.

Console

To stop a pipeline

1. Sign in to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. In the navigation pane, choose **Pipelines**, and then choose a pipeline. You can perform the stop operation from this page, or navigate to the details page for the pipeline that you want to stop.
3. For **Actions**, choose **Stop pipeline**.

   If a pipeline can't be stopped and started, the **Stop pipeline** action isn't available.

AWS CLI

To stop a pipeline using the AWS CLI, call the stop-pipeline command with the following parameters:

- `--pipeline-name` – the name of the pipeline.

Example

```
aws osis stop-pipeline --pipeline-name my-pipeline
```

OpenSearch Ingestion API

To stop a pipeline using the OpenSearch Ingestion API, call the StopPipeline operation with the following parameter:

- `PipelineName` – the name of the pipeline.

Starting an OpenSearch Ingestion pipeline

You always start an OpenSearch Ingestion pipeline beginning with a pipeline that's already in the stopped state. The pipeline keeps its configuration settings such as capacity limits, network settings, and log publishing options.

Restarting a pipeline usually takes several minutes.

Console

To start a pipeline

1. Sign in to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. In the navigation pane, choose **Pipelines**, and then choose a pipeline. You can perform the start operation from this page, or navigate to the details page for the pipeline that you want to start.
3. For **Actions**, choose **Start pipeline**.

AWS CLI

To start a pipeline by using the AWS CLI, call the start-pipeline command with the following parameters:

- `--pipeline-name` – the name of the pipeline.

Example

```
aws osis start-pipeline --pipeline-name my-pipeline
```

OpenSearch Ingestion API

To start an OpenSearch Ingestion pipeline using the OpenSearch Ingestion API, call the StartPipeline operation with the following parameter:

- `PipelineName` – the name of the pipeline.

Deleting Amazon OpenSearch Ingestion pipelines

You can delete an Amazon OpenSearch Ingestion pipeline using the AWS Management Console, the AWS CLI, or the OpenSearch Ingestion API. You can't delete a pipeline when has a status of `Creating` or `Updating`.

Console

To delete a pipeline

1. Sign in to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Choose **Pipelines** in the left navigation pane.
3. Select the pipeline that you want to delete and choose **Delete**.
4. Confirm deletion and choose **Delete**.

CLI

To delete a pipeline using the AWS CLI, send a delete-pipeline request:

```
aws osis delete-pipeline --pipeline-name "my-pipeline"
```

OpenSearch Ingestion API

To delete an OpenSearch Ingestion pipeline using the OpenSearch Ingestion API, call the DeletePipeline operation with the following parameter:

- `PipelineName` – the name of the pipeline.

Supported plugins and options for Amazon OpenSearch Ingestion pipelines

Amazon OpenSearch Ingestion supports a subset of sources, processors, and sinks compared to open source Data Prepper. In addition, there are some constraints that OpenSearch Ingestion places on the available options for each supported plugin. The following sections describe the plugins and associated options that OpenSearch Ingestion supports.

> **Note**
>
> OpenSearch Ingestion doesn't support any buffer plugins because it automatically configures a default buffer. You receive a validation error if you include a buffer in your pipeline configuration.

Topics

Supported plugins

OpenSearch Ingestion supports the following Data Prepper plugins:

Sources:

- Amazon DocumentDB
- DynamoDB
- OpenSearch

- HTTP
- Kafka
- OTel logs

- OTel metrics
- OTel trace
- S3
- Amazon Kinesis Data Streams

Processors:

- AWS Lambda
- Add_entries
- Delay
- Delete Entries
- Convert_entry_type
- Copy_Values
- List_to_Map
- Lower_case_string
- Rename_Keys
- Routes
- Split_String
- String_converter
- Subsitute-string
- translate
- trim-string
- uppercase-string
- Write JSON
- Flatten
- Split Event
- Aggregate
- Anomaly detector
- CSV
- Date

- Decompress
- Dissect
- Drop events
- Geo IP
- Grok
- Key value
- Map to list
- Mutate event (series of processors)
- Mutate string (series of processors)
- Obfuscate
- OTel metrics
- OTel trace group
- OTel trace
- Parse Ion
- Parse JSON
- Parse XML
- Select entries
- Service-map
- Trace peer forwarder
- Truncate
- User agent

Sinks:

- OpenSearch (supports OpenSearch Service, OpenSearch Serverless, and Elasticsearch 6.8 or later)
- S3

Sink codecs:

- Avro

- NDJSON
- JSON
- Parquet

Stateless versus stateful processors

Stateless processors perform operations like transformations and filtering, while *stateful* processors perform operations like aggregations, which remember the result of the previous run. OpenSearch Ingestion supports the stateful processors Aggregate and Service-map. All other supported processors are stateless.

For pipelines that contain only stateless processors, the maximum capacity limit is 96 Ingestion OCUs. If a pipeline contains any stateful processors, the maximum capacity limit is 48 Ingestion OCUs. However, if a pipeline has persistent buffering enabled, it can have a maximum of 384 Ingestion OCUs with only stateless processors, or 192 Ingestion OCUs if it contains any stateful processors. For more information, see the section called “Scaling pipelines”.

End-to-end acknowledgment is only supported for stateless processors. For more information, see the section called “End-to-end acknowledgement”.

Configuration requirements and constraints

Unless otherwise specified below, all options described in the Data Prepper configuration reference for the supported plugins listed above are allowed in OpenSearch Ingestion pipelines. The following sections explain the constraints that OpenSearch Ingestion places on certain plugin options.

> **Note**
>
> OpenSearch Ingestion doesn't support any buffer plugins because it automatically configures a default buffer. You receive a validation error if you include a buffer in your pipeline configuration.

Many options are configured and managed internally by OpenSearch Ingestion, such as `authentication` and `acm_certificate_arn`. Other options, such as `thread_count` and `request_timeout`, have performance impacts if changed manually. Therefore, these values are set internally to ensure optimal performance of your pipelines.

Lastly, some options can't be passed to OpenSearch Ingestion, such as `ism_policy_file` and `sink_template`, because they're local files when run in open source Data Prepper. These values aren't supported.

Topics

General pipeline options

The following general pipeline options are set by OpenSearch Ingestion and aren't supported in pipeline configurations:

- `workers`
- `delay`

Grok processor

The following Grok processor options aren't supported:

- `patterns_directories`
- `patterns_files_glob`

HTTP source

The HTTP source plugin has the following requirements and constraints:

- The `path` option is *required*. The path is a string such as `/log/ingest`, which represents the URI path for log ingestion. This path defines the URI that you use to send data to the pipeline. For example, `https://log-pipeline.us-west-2.osis.amazonaws.com/log/ingest`. The path must start with a slash (/), and can contain the special characters '-', '_', '.', and '/', as well as the `${pipelineName}` placeholder.
- The following HTTP source options are set by OpenSearch Ingestion and aren't supported in pipeline configurations:
  - `port`
  - `ssl`
  - `ssl_key_file`
  - `ssl_certificate_file`
  - `aws_region`
  - `authentication`
  - `unauthenticated_health_check`
  - `use_acm_certificate_for_ssl`
  - `thread_count`
  - `request_timeout`
  - `max_connection_count`
  - `max_pending_requests`
  - `health_check_service`
  - `acm_private_key_password`
  - `acm_certificate_timeout_millis`
  - `acm_certificate_arn`

OpenSearch sink

The OpenSearch sink plugin has the following requirements and limitations.

- The `aws` option is *required*, and must contain the following options:
  - `sts_role_arn`
  - `region`
  - `hosts`

  - `serverless` (if the sink is an OpenSearch Serverless collection)
- The `sts_role_arn` option must point to the same role for each sink within a YAML definition file.
- The `hosts` option must specify an OpenSearch Service domain endpoint or an OpenSearch Serverless collection endpoint. You can't specify a custom endpoint for a domain; it must be the standard endpoint.
- If the `hosts` option is a serverless collection endpoint, you must set the `serverless` option to `true`. In addition, if your YAML definition file contains the `index_type` option, it must be set to `management_disabled`, otherwise validation fails.
- The following options aren't supported:
  - `username`
  - `password`
  - `cert`
  - `proxy`
  - `dlq_file` - If you want to offload failed events to a dead letter queue (DLQ), you must use the `dlq` option and specify an S3 bucket.
  - `ism_policy_file`
  - `socket_timeout`
  - `template_file`
  - `insecure`

OTel metrics source, OTel trace source, and OTel logs source

The OTel metrics source, OTel trace source, and OTel logs source plugins have the following requirements and limitations:

- The `path` option is *required*. The path is a string such as `/log/ingest`, which represents the URI path for log ingestion. This path defines the URI that you use to send data to the pipeline. For example, `https://log-pipeline.us-west-2.osis.amazonaws.com/log/ingest`. The path must start with a slash (/), and can contain the special characters '-', '_', '.', and '/', as well as the `${pipelineName}` placeholder.
- The following options are set by OpenSearch Ingestion and aren't supported in pipeline configurations:

- `port`
- `ssl`
- `sslKeyFile`
- `sslKeyCertChainFile`
- `authentication`
- `unauthenticated_health_check`
- `useAcmCertForSSL`
- `unframed_requests`
- `proto_reflection_service`
- `thread_count`
- `request_timeout`
- `max_connection_count`
- `acmPrivateKeyPassword`
- `acmCertIssueTimeOutMillis`
- `health_check_service`
- `acmCertificateArn`
- `awsRegion`

OTel trace group processor

The OTel trace group processor has the following requirements and limitations:

- The `aws` option is *required*, and must contain the following options:
  - `sts_role_arn`
  - `region`
  - `hosts`
- The `sts_role_arn` option specify the same role as the pipeline role that you specify in the OpenSearch sink configuration.
- The `username`, `password`, `cert`, and `insecure` options aren't supported.
- The `aws_sigv4` option is required and must be set to true.
- The `serverless` option within the OpenSearch sink plugin isn't supported. The Otel trace group processor doesn't currently work with OpenSearch Serverless collections.

- The number of `otel_trace_group` processors within the pipeline configuration body can't exceed 8.

OTel trace processor

The OTel trace processor has the following requirements and limitations:

- The value of the `trace_flush_interval` option can't exceed 300 seconds.

Service-map processor

The Service-map processor has the following requirements and limitations:

- The value of the `window_duration` option can't exceed 300 seconds.

S3 source

The S3 source plugin has the following requirements and limitations:

- The `aws` option is *required*, and must contain `region` and `sts_role_arn` options.
- The value of the `records_to_accumulate` option can't exceed 200.
- The value of the `maximum_messages` option can't exceed 10.
- If specified, the `disable_bucket_ownership_validation` option must be set to false.
- If specified, the `input_serialization` option must be set to `parquet`.

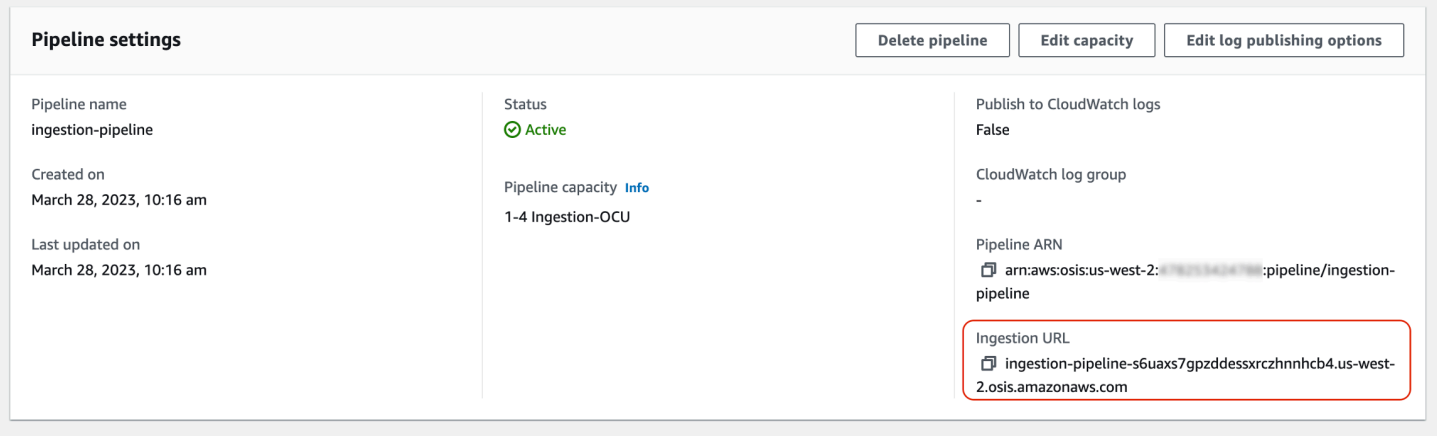
Integrating Amazon OpenSearch Ingestion pipelines with other services and applications

To successfully ingest data into an Amazon OpenSearch Ingestion pipeline, you must configure your client application (the *source*) to send data to the pipeline endpoint. Your source might be clients like Fluent Bit logs, the OpenTelemetry Collector, or a simple S3 bucket. The exact configuration differs for each client.

The important differences during source configuration (compared to sending data directly to an OpenSearch Service domain or OpenSearch Serverless collection) are the AWS service name (`osis`) and the host endpoint, which must be the pipeline endpoint.

Constructing the ingestion endpoint

To ingest data into a pipeline, send it to the ingestion endpoint. To locate the ingestion URL, navigate to the **Pipeline settings** page and copy the **Ingestion URL**:



To construct the full ingestion endpoint for pull-based sources like OTel trace and OTel metrics, add the ingestion path from your pipeline configuration to the ingestion URL.

For example, say that your pipeline configuration has the following ingestion path:

```
entry-pipeline:
  source:
    http:
      path: "/my/test_path"
```

The full ingestion endpoint, which you specify in your client configuration, will take the following format: `https://ingestion-pipeline-abcdefg.us-west-2.osis.amazonaws.com/my/test_path`.

For more information, see the section called “Specifying the ingestion path”.

Creating an ingestion role

All requests to OpenSearch Ingestion must be signed with Signature Version 4. At minimum, the role that signs the request must be granted permission for the `osis:Ingest` action, which allows it to send data to an OpenSearch Ingestion pipeline.

For example, the following AWS Identity and Access Management (IAM) policy allows the corresponding role to send data to a single pipeline:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "osis:Ingest",
      "Resource": "arn:aws:osis:us-east-1:{account-id}:pipeline/pipeline-name"
    }
  ]
}
```

Note

To use the role for *all* pipelines, replace the ARN in the `Resource` element with a wildcard (*).

Providing cross-account ingestion access

Note

You can only provide cross-account ingestion access for public pipelines, not VPC pipelines.

You might need to ingest data into a pipeline from a different AWS account, such as an account that houses your source application. If the principal that is writing to a pipeline is in a different account than the pipeline itself, you need to configure the principal to trust another IAM role to ingest data into the pipeline.

To configure cross-account ingestion permissions

1. Create the ingestion role with `osis:Ingest` permission (described in the previous section) within the same AWS account as the pipeline. For instructions, see Creating IAM roles.
2. Attach a trust policy to the ingestion role that allows a principal in another account to assume it:

```
{
  "Version": "2012-10-17",
  "Statement": [{
```

```
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{external-account-id}:root"
       },
      "Action": "sts:AssumeRole"
  }]
}
```

3. In the other account, configure your client application (for example, Fluent Bit) to assume the ingestion role. In order for this to work, the application account must grant permissions to the application user or role to assume the ingestion role.

   The following example identity-based policy allows the attached principal to assume `ingestion-role` from the pipeline account:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::{account-id}:role/ingestion-role"
    }
  ]
}
```

The client application can then use the AssumeRole operation to assume `ingestion-role` and ingest data into the associated pipeline.

Using an OpenSearch Ingestion pipeline with Amazon DynamoDB

You can use an OpenSearch Ingestion pipeline with DynamoDB to stream DynamoDB table events (such as create, update, and delete) to Amazon OpenSearch Service domains and collections. The OpenSearch Ingestion pipeline incorporates change data capture (CDC) infrastructure to provide a high-scale, low-latency way to continuously stream data from a DynamoDB table.

There are two ways that you can use DynamoDB as a source to process data—with and without a *full initial snapshot*.

A full initial snapshot is a backup of a table that DynamoDB takes with the point-in-time recovery (PITR) feature. DynamoDB uploads this snapshot to Amazon S3. From there, an OpenSearch

Ingestion pipeline sends it to one index in a domain, or partitions it to multiple indexes in a domain. To keep the data in DynamoDB and OpenSearch consistent, the pipeline syncs all of the create, update, and delete events in the DynamoDB table with the documents saved in the OpenSearch index or indexes.

When you use a full initial snapshot, your OpenSearch Ingestion pipeline first ingests the snapshot and then starts reading data from DynamoDB Streams. It eventually catches up and maintains near real-time data consistency between DynamoDB and OpenSearch. When you choose this option, you must enable both PITR and a DynamoDB stream on your table.

You can also use the OpenSearch Ingestion integration with DynamoDB to stream events without a snapshot. Choose this option if you already have a full snapshot from some other mechanism, or if you just want to stream current events from a DynamoDB table with DynamoDB Streams. When you choose this option, you only need to enable a DynamoDB stream on your table.

For more information about this integration, see DynamoDB zero-ETL integration with Amazon OpenSearch Service in the *Amazon DynamoDB Developer Guide*.

Topics

Prerequisites

To set up your pipeline, you must have a DynamoDB table with DynamoDB Streams enabled. Your stream should use the `NEW_IMAGE` stream view type. However, OpenSearch Ingestion pipelines can also stream events with `NEW_AND_OLD_IMAGES` if this stream view type fits your use case.

If you're using snapshots, you must also enable point-in-time recovery on your table. For more information, see Creating a table, Enabling point-in-time recovery, and Enabling a stream in the *Amazon DynamoDB Developer Guide*.

Step 1: Configure the pipeline role

After you have your DynamoDB table set up, set up the pipeline role that you want to use in your pipeline configuration, and add the following DynamoDB permissions in the role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "allowRunExportJob",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:DescribeContinuousBackups",
                "dynamodb:ExportTableToPointInTime"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:{account-id}:table/my-table"
            ]
        },
        {
            "Sid": "allowCheckExportjob",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeExport"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:{account-id}:table/my-table/export/*"
            ]
        },
        {
            "Sid": "allowReadFromStream",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:{account-id}:table/my-table/stream/*"
            ]
        },
        {
```

```
            "Sid": "allowReadAndWriteToS3ForExport",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo- /{exportPath}/*"
            ]
        }
    ]
}
```

You can also use an AWS KMS customer managed key to encrypt the export data files. To decrypt the exported objects, specify `s3_sse_kms_key_id` for the key ID in the export configuration of the pipeline with the following format: `arn:aws:kms:us-west-2:{account-id}:key/my-key-id`. The following policy includes the required permissions for using a customer managed key:

```
{
    "Sid": "allowUseOfCustomManagedKey",
    "Effect": "Allow",
    "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
    ],
    "Resource": arn:aws:kms:us-west-2:{account-id}:key/my-key-id
}
```

Step 2: Create the pipeline

You can then configure an OpenSearch Ingestion pipeline like the following, which specifies DynamoDB as the source. This sample pipeline ingests data from `table-a` with the PITR snapshot, followed by events from DynamoDB Streams. A start position of `LATEST` indicates that the pipeline should read the latest data from DynamoDB Streams.

```
version: "2"
cdc-pipeline:
  source:
    dynamodb:
```

```
      tables:
      - table_arn: "arn:aws:dynamodb:us-west-2:{account-id}:table/table-a"
        export:
          s3_bucket: "amzn-s3-demo-"
          s3_prefix: "export/"
        stream:
          start_position: "LATEST"
      aws:
        region: "us-west-2"
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
  sink:
  - opensearch:
      hosts: ["https://search-mydomain.us-east-1.es.amazonaws.com"]
      index: "${getMetadata(\"table_name\")}"
      index_type: custom
      normalize_index: true
      document_id: "${getMetadata(\"primary_key\")}"
      action: "${getMetadata(\"opensearch_action\")}"
      document_version: "${getMetadata(\"document_version\")}"
      document_version_type: "external"
```

You can use a preconfigured DynamoDB blueprint to create this pipeline. For more information, see the section called “Using blueprints to create a pipeline”.

Data consistency

OpenSearch Ingestion supports end-to-end acknowledgement to ensure data durability. When a pipeline reads snapshots or streams, it dynamically creates partitions for parallel processing. The pipeline marks a partition as complete when it receives an acknowledgement after ingesting all records in the OpenSearch domain or collection.

If you want to ingest into an OpenSearch Serverless *search* collection, you can generate a document ID in the pipeline. If you want to ingest into an OpenSearch Serverless *time series* collection, note that the pipeline doesn't generate a document ID.

An OpenSearch Ingestion pipeline also maps incoming event actions into corresponding bulk indexing actions to help ingest documents. This keeps data consistent, so that every data change in DynamoDB is reconciled with the corresponding document changes in OpenSearch.

Mapping data types

OpenSearch Service dynamically maps data types in each incoming document to the corresponding data type in DynamoDB. The following table shows how OpenSearch Service automatically maps various data types.

Data type	OpenSearch	DynamoDB
Number	OpenSearch automatically maps numeric data. If the number is a whole number, OpenSearch maps it as a long value. If the number is fractional, then OpenSearch maps it as a float value. OpenSearch dynamically maps various attributes based on the first sent document. If you have a mix of data types for the same attribute in DynamoDB, such as both a whole number and a fractional number, mapping might fail. For example, if your first document has an attribute that is a whole number, and a later document has that same attribute as a fractional number, OpenSearch fails to ingest the second document. In these cases, you should provide an explicit mapping template, such as the following: `{ "template": { "mappings": { "properties": { "MixedNumberAttribute": { "type": "float" } }`	DynamoDB supports numbers.

Data type	OpenSearch	DynamoDB
	<pre> } } }</pre> If you need double precision, use string-type field mapping. There is no equivalent numeric type that supports 38 digits of precision in OpenSearch.	
Number set	OpenSearch automatically maps a number set into an array of either long values or float values. As with the scalar numbers, this depends on whether the first number ingested is a whole number or a fractional number. You can provide mappings for number sets the same way that you map scalar strings.	DynamoDB supports types that represent sets of numbers.

Data type	OpenSearch	DynamoDB
String	OpenSearch automatically maps string values as text. In some situations, such as enumerated values, you can map to the keyword type. The following example shows how to map a DynamoDB attribute named `PartType` to an OpenSearch keyword. `{ "template": { "mappings": { "properties": { "PartType": { "type": "keyword" } } } } }`	DynamoDB supports strings.
String set	OpenSearch automatically maps a string set into an array of strings. You can provide mappings for string sets the same way that you map scalar strings.	DynamoDB supports types that represent sets of strings.

Data type	OpenSearch	DynamoDB
Binary	OpenSearch automatically maps binary data as text. You can provide a mapping to write these as binary fields in OpenSearch. The following example shows how to map a DynamoDB attribute named `ImageData` to an OpenSearch binary field. `{ "template": { "mappings": { "properties": { "ImageData": { "type": "binary" } } } } }`	DynamoDB supports binary type attributes.
Binary set	OpenSearch automatically maps a binary set into an array of binary data as text. You can provide mappings for number sets the same way that you map scalar binary.	DynamoDB supports types that represent sets of binary values.
Boolean	OpenSearch maps a DynamoDB Boolean type into an OpenSearch Boolean type.	DynamoDB supports Boolean type attributes.

Data type	OpenSearch	DynamoDB
Null	OpenSearch can ingest documents with the DynamoDB null type. It saves the value as a null value in the document. There is no mapping for this type, and this field is not indexed or searchable. If the same attribute name is used for a null type and then later changes to different type such as string, OpenSearch creates a dynamic mapping for the first non-null value. Subsequent values can still be DynamoDB null values.	DynamoDB supports null type attributes.
Map	OpenSearch maps DynamoDB map attributes to nested fields. The same mappings apply within a nested field. The following example maps a string in a nested field to a keyword type in OpenSearch: `{ "template": { "mappings": { "properties": { "AdditionalDescriptions": { "properties": { "PartType": { "type": "keyword" } } } } } } }`	DynamoDB supports map type attributes.

Data type	OpenSearch	DynamoDB
List	OpenSearch provides different results for DynamoDB lists, depending on what is in the list. When a list contains all of the same type of scalar types (for example, a list of all strings), then OpenSearch ingests the list as an array of that type. This works for string, number, Boolean, and null types. The restrictions for each of these types are the same as restrictions for a scalar of that type. You can also provide mappings for lists of maps by using the same mapping as you would use for a map. You can't provide a list of mixed types.	DynamoDB supports list type attributes.

Data type	OpenSearch	DynamoDB
Set	OpenSearch provides different results for DynamoDB sets depending on what is in the set. When a set contains all of the same type of scalar types (for example, a set of all strings), then OpenSearch ingests the set as an array of that type. This works for string, number, Boolean, and null types. The restrictions for each of these types are the same as the restricti ons for a scalar of that type. You can also provide mappings for sets of maps by using the same mapping as you would use for a map. You can't provide a set of mixed types.	DynamoDB supports types that represent sets.

We recommend that you configure the dead-letter queue (DLQ) in your OpenSearch Ingestion pipeline. If you've configured the queue, OpenSearch Service sends all failed documents that can't be ingested due to dynamic mapping failures to the queue.

In case automatic mappings fail, you can use `template_type` and `template_content` in your pipeline configuration to define explicit mapping rules. Alternatively, you can create mapping templates directly in your search domain or collection before you start the pipeline.

Limitations

Consider the following limitations when you set up an OpenSearch Ingestion pipeline for DynamoDB:

- The OpenSearch Ingestion integration with DynamoDB currently doesn't support cross-Region ingestion. Your DynamoDB table and OpenSearch Ingestion pipeline must be in the same AWS Region.
- Your DynamoDB table and OpenSearch Ingestion pipeline must be in the same AWS account.

- An OpenSearch Ingestion pipeline supports only one DynamoDB table as its source.
- DynamoDB Streams only stores data in a log for up to 24 hours. If ingestion from an initial snapshot of a large table takes 24 hours or more, there will be some initial data loss. To mitigate this data loss, estimate the size of the table and configure appropriate compute units of OpenSearch Ingestion pipelines.

Recommended CloudWatch Alarms for DynamoDB

The following CloudWatch metrics are recommended for monitioring the performance of your ingestion pipeline. These metrics can help you identify the amount of data processed from exports, the amount of events processed from streams, the errors in processing exports and stream events, and the number of documents written to the destination. You can setup CloudWatch alarms to perform an action when one of these metrics exceed a specified value for a specified amount of time.

Metric	Description
`dynamodb-pipeline.BlockingBuffer.bufferUsage.value`	Indicates how much of the buffer is being utilized.
`dynamodb-pipeline.dynamodb.activeExportS3ObjectConsumers.value`	Shows the total number of OCUs that are actively processing Amazon S3 objects for the export.
`dynamodb-pipeline.dynamodb.bytesProcessed.count`	Count of bytes processed from DynamoDB source.
`dynamodb-pipeline.dynamodb.changeEventsProcessed.count`	Number of change events processed from DynamoDB stream.
`dynamodb-pipeline.dynamodb.changeEventsProcessingErrors.count`	Number of errors from change events processed from DynamoDB.
`dynamodb-pipeline.dynamodb.exportJobFailure.count`	Number of export job submission attempts that have failed.

Metric	Description
dynamodb-pipeline.dynamodb.exportJobSuccess.count	Number of export jobs that have been submitted successfully.
dynamodb-pipeline.dynamodb.exportRecordsProcessed.count	Total number of records processed from the export.
dynamodb-pipeline.dynamodb.exportRecordsTotal.count	Total number of records exported from DynamoDB, essential for tracking data export volumes.
dynamodb-pipeline.dynamodb.exportS3ObjectsProcessed.count	Total number of export data files that have been processed successfully from Amazon S3.
dynamodb-pipeline.opensearch.bulkBadRequestErrors.count	Count of errors during bulk requests due to malformed request.
dynamodb-pipeline.opensearch.bulkRequestLatency.avg	Average latency for bulk write requests made to OpenSearch.
dynamodb-pipeline.opensearch.bulkRequestNotFoundErrors.count	Number of bulk requests that failed because the target data could not be found.
dynamodb-pipeline.opensearch.bulkRequestNumberOfRetries.count	Number of retries by OpenSearch Ingestion pipelines to write OpenSearch cluster.
dynamodb-pipeline.opensearch.bulkRequestSizeBytes.sum	Total size in bytes of all bulk requests made to OpenSearch.
dynamodb-pipeline.opensearch.documentErrors.count	Number of errors when sending documents to OpenSearch. The documents causing the errors witll be sent to DLQ.
dynamodb-pipeline.opensearch.documentsSuccess.count	Number of documents successfully written to an OpenSearch cluster or collection.

Metric	Description
dynamodb-pipeline.opensearch.documentsSuccessFirstAttempt.count	Number of documents successfully indexed in OpenSearch on the first attempt.
dynamodb-pipeline.opensearch.documentsVersionConflictErrors.count	Count of errors due to version conflicts in documents during processing.
dynamodb-pipeline.opensearch.PipelineLatency.avg	Average latency of OpenSearch Ingestion pipeline to process the data by reading from the source to writint to the destination.
dynamodb-pipeline.opensearch.PipelineLatency.max	Maximum latency of OpenSearch Ingestion pipeline to process the data by reading from the source to writing the destination.
dynamodb-pipeline.opensearch.recordsIn.count	Count of records successfully ingested into OpenSearch. This metric is essential for tracking the volume of data being processed and stored.
dynamodb-pipeline.opensearch.s3.dlqS3RecordsFailed.count	Number of records that failed to write to DLQ.
dynamodb-pipeline.opensearch.s3.dlqS3RecordsSuccess.count	Number of records that are written to DLQ.
dynamodb-pipeline.opensearch.s3.dlqS3RequestLatency.count	Count of latency measurements for requests to the Amazon S3 dead-letter queue.
dynamodb-pipeline.opensearch.s3.dlqS3RequestLatency.sum	Total latency for all requests to the Amazon S3 dead-letter queue
dynamodb-pipeline.opensearch.s3.dlqS3RequestSizeBytes.sum	Total size in bytes of all requests made to the Amazon S3 dead-letter queue.

Metric	Description
dynamodb-pipeline.recordsProcessed.count	Total number of records processed in the pipeline, a key metric for overal throughput.
dynamodb.changeEventsProcessed.count	No records are being gathered from DynamoDB streams. This could be due to no activitiy on the table, an export being in progress, or an issue accessing the DynamoDB streams.
dynamodb.exportJobFailure.count	The attempt to trigger an export to S3 failed.
dynamodb-pipeline.opensearch.bulkRequestInvalidInputErrors.count	Count of bulk request errors in OpenSearch due to invalid input, crucial for monitoring data quality and operational issues.
opensearch.EndToEndLatency.avg	The end to end latnecy is higher than desired for reading from DynamoDB streams. This could be due to an underscaled OpenSearch cluster or a maximum pipeline OCU capacity that is too low for the WCU throughput on the DynamoDB table. This end to end latency will be high after an export and should decrease over time as it catches up to the latest DynamoDB streams.

Using an OpenSearch Ingestion pipeline with Amazon DocumentDB

You can use an OpenSearch Ingestion pipeline with Amazon DocumentDB to stream document changes (such as create, update, and delete) to Amazon OpenSearch Service domains and collections. The OpenSearch Ingestion pipeline can leverage change data capture (CDC) mechanisms, if available on your Amazon DocumentDB cluster, or API polling to provide a high-scale, low-latency way to continuously stream data from a Amazon DocumentDB cluster.

There are two ways that you can use Amazon DocumentDB as a source to process data—with and without a *full initial snapshot*.

A full initial snapshot is a bulk query of an entire Amazon DocumentDB collection. Amazon DocumentDB uploads this snapshot to Amazon S3. From there, an OpenSearch Ingestion pipeline sends it to one index in a domain, or partitions it to multiple indexes in a domain. To keep the data in Amazon DocumentDB and OpenSearch consistent, the pipeline syncs all of the create, update, and delete events in the Amazon DocumentDB collection with the documents saved in the OpenSearch index or indexes.

When you use a full initial snapshot, your OpenSearch Ingestion pipeline first ingests the snapshot and then starts reading data from Amazon DocumentDB change streams. It eventually catches up and maintains near real-time data consistency between Amazon DocumentDB and OpenSearch.

You can also use the OpenSearch Ingestion integration with Amazon DocumentDB to stream events without a snapshot. Choose this option if you already have a full snapshot from some other mechanism, or if you just want to stream current events from a Amazon DocumentDB collection with change streams.

With both of these options, you must enable a change stream on your Amazon DocumentDB collection if you enable a stream in your in pipeline configuration. If you only use full load or export, you don't need to enable a change stream.

Prerequisites

Before you create your OpenSearch Ingestion pipeline, perform the following steps:

1. Create a Amazon DocumentDB cluster with permission to read data by following the steps in Create an Amazon DocumentDB cluster in the *Amazon DocumentDB Developer Guide*. If you use CDC infrastructure, ensure that you configure your Amazon DocumentDB cluster to publish change streams.
2. Enable TLS on your Amazon DocumentDB cluster.
3. Set up a VPC CIDR of a private address space for use with OpenSearch Ingestion.
4. Set up authentication on your Amazon DocumentDB cluster with AWS Secrets Manager. Enable secrets rotation by following the steps in Automatically Rotating Passwords for Amazon DocumentDB. For more information, see Database Access Using Role-Based Access Control and Security in Amazon DocumentDB.
5. If you use a change stream to subscribe to data changes on your Amazon DocumentDB collection, avoid data loss by extending the retention period to up to 7 days using the `change_stream_log_retention_duration` parameter. Change streams events are stored

for 3 hours, by default, after the event has been recorded, which isn't enough time for large collections. To modify the change stream retention period, see Modifying the Change Stream Log Retention Duration.

6. Create an OpenSearch Service domain or OpenSearch Serverless collection. For more information, see Creating OpenSearch Service domains and Creating collections.
7. Attach a resource-based policy to your domain or a data access policy to your collection. These access policies allow OpenSearch Ingestion to write data from your Amazon DocumentDB cluster to your domain or collection.

   The following sample domain access policy allows the pipeline role, which you create in the next step, to write data to a domain. Make sure that you update the `resource` with your own ARN.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{pipeline-account-id}:role/pipeline-role"
      },
      "Action": [
        "es:DescribeDomain",
        "es:ESHttp*"
      ],
      "Resource": [
        "arn:aws:es:{region}:{account-id}:domain/domain-name"
      ]
    }
  ]
}
```

   To create an IAM role with the correct permissions to access write data to the collection or domain, see Required permissions for domains and Required permissions for collections.

Step 1: Configure the pipeline role

After you have your Amazon DocumentDB pipeline prerequisites set up, configure the pipeline role that you want to use in your pipeline configuration, and add the following Amazon DocumentDB permissions in the role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "allowS3ListObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::{s3_bucket}"
            ],
            "Condition": {
                "StringLike": {
                    "s3:prefix": "{s3_prefix}/*"
                }
            }
        },
        {
            "Sid": "allowReadAndWriteToS3ForExportStream",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::{s3_bucket}/{s3_prefix}/*"
            ]
        },
        {
            "Sid": "SecretsManagerReadAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": ["arn:aws:secretsmanager:{region}:{account-id}:secret:secret-
name"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AttachNetworkInterface",
```

```
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DetachNetworkInterface",
                "ec2:DescribeNetworkInterfaces"
            ],
            "Resource": [
                "arn:aws:ec2:*:{account-id}:network-interface/*",
                "arn:aws:ec2:*:{account-id}:subnet/*",
                "arn:aws:ec2:*:{account-id}:security-group/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:Describe*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": "arn:aws:ec2:*:*:network-interface/*",
            "Condition": {
               "StringEquals":
                    {
                        "aws:RequestTag/OSISManaged": "true"
                    }
            }
        }
    ]
}
```

You must provide the above Amazon EC2 permissions on the IAM role that you use to create the OpenSearch Ingestion pipeline because the pipeline uses these permissions to create and delete

a network interface in your VPC. The pipeline can only access the Amazon DocumentDB cluster through this network interface.

Step 2: Create the pipeline

You can then configure an OpenSearch Ingestion pipeline like the following, which specifies Amazon DocumentDB as the source. Note that to populate the index name, the `getMetadata` function uses *`documentdb_collection`* as a metadata key. If you want to use a different index name without the `getMetadata` method, you can use the configuration `index: "`*`my_index_name`*`"`.

```
version: "2"
documentdb-pipeline:
  source:
    documentdb:
      acknowledgments: true
      host: "https://docdb-cluster-id.us-east-1.docdb.amazonaws.com"
      port: 27017
      authentication:
        username: ${aws_secrets:secret:username}
        password: ${aws_secrets:secret:password}
      aws:
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
      s3_bucket: "bucket-name"
      s3_region: "bucket-region"
      s3_prefix: "path" #optional path for storing the temporary data
      collections:
        - collection: "dbname.collection"
          export: true
          stream: true
  sink:
  - opensearch:
      hosts: ["https://search-mydomain.us-east-1.es.amazonaws.com"]
      index: "${getMetadata(\"documentdb_collection\")}"
      index_type: custom
      document_id: "${getMetadata(\"primary_key\")}"
      action: "${getMetadata(\"opensearch_action\")}"
      document_version: "${getMetadata(\"document_version\")}"
      document_version_type: "external"
extension:
  aws:
    secrets:
      secret:
```

```
        secret_id: "my-docdb-secret"
        region: "us-east-1"
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
        refresh_interval: PT1H
```

You can use a preconfigured Amazon DocumentDB blueprint to create this pipeline. For more information, see the section called “Using blueprints to create a pipeline”.

If you're using the AWS Management Console to create your pipeline, you must also attach your pipeline to your VPC in order to use Amazon DocumentDB as a source. To do so, find the **Network configuration** section, select the **Attach to VPC** checkbox, and choose your CIDR from one of the provided default options, or select your own. You can use any CIDR from a private address space as defined in the RFC 1918 Best Current Practice.

To provide a custom CIDR, select **Other** from the dropdown menu. To avoid a collision in IP addresses between OpenSearch Ingestion and Amazon DocumentDB, ensure that the Amazon DocumentDB VPC CIDR is different from the CIDR for OpenSearch Ingestion.

For more information, see Configuring VPC access for a pipeline.

Data consistency

The pipeline ensures data consistency by continuously polling or receiving changes from the Amazon DocumentDB cluster and updating the corresponding documents in the OpenSearch index.

OpenSearch Ingestion supports end-to-end acknowledgement to ensure data durability. When a pipeline reads snapshots or streams, it dynamically creates partitions for parallel processing. The pipeline marks a partition as complete when it receives an acknowledgement after ingesting all records in the OpenSearch domain or collection.

If you want to ingest into an OpenSearch Serverless *search* collection, you can generate a document ID in the pipeline. If you want to ingest into an OpenSearch Serverless *time series* collection, note that the pipeline doesn't generate a document ID, so you must omit `document_id: "${getMetadata(\"primary_key\")}"` in your pipeline sink configuration.

An OpenSearch Ingestion pipeline also maps incoming event actions into corresponding bulk indexing actions to help ingest documents. This keeps data consistent, so that every data change in Amazon DocumentDB is reconciled with the corresponding document changes in OpenSearch.

Mapping data types

OpenSearch Service dynamically maps data types in each incoming document to the corresponding data type in Amazon DocumentDB. The following table shows how OpenSearch Service automatically maps various data types.

Data type	OpenSearch	Amazon DocumentDB
Integer	OpenSearch automatically maps Amazon DocumentDB integer values to OpenSearch integers. OpenSearch dynamically maps the field based on the first sent document. If you have a mix of data types for the same attribute in Amazon DocumentDB, automatic mapping might fail. For example, if your first document has an attribute that is a long, and a later document has that same attribute as an integer, OpenSearch fails to ingest the second document. In these cases, you should provide an explicit mapping template that chooses the most flexible number type, such as the following: `{ "template": { "mappings": { "properties": { "MixedNumberField": { "type": "float" } } } } }`	Amazon DocumentDB supports integers.

Data type	OpenSearch	Amazon DocumentDB
Long	OpenSearch automatically maps Amazon DocumentDB long values to OpenSearch longs. OpenSearch dynamically maps the field based on the first sent document. If you have a mix of data types for the same attribute in Amazon DocumentDB, automatic mapping might fail. For example, if your first document has an attribute that is a long, and a later document has that same attribute as an integer, OpenSearch fails to ingest the second document. In these cases, you should provide an explicit mapping template that chooses the most flexible number type, such as the following: `{ "template": { "mappings": { "properties": { "MixedNumberField": { "type": "float" } } } } }`	Amazon DocumentDB supports longs.

Data type	OpenSearch	Amazon DocumentDB
String	OpenSearch automatically maps string values as text. In some situations, such as enumerated values, you can map to the keyword type. The following example shows how to map a Amazon DocumentDB attribute named `PartType` to an OpenSearch keyword. `{ "template": { "mappings": { "properties": { "PartType": { "type": "keyword" } } } } }`	Amazon DocumentDB supports strings.

Data type	OpenSearch	Amazon DocumentDB
Double	OpenSearch automatically maps Amazon DocumentDB double values to OpenSearch doubles. OpenSearch dynamically maps the field based on the first sent document. If you have a mix of data types for the same attribute in Amazon DocumentDB, automatic mapping might fail. For example, if your first document has an attribute that is a long, and a later document has that same attribute as an integer, OpenSearch fails to ingest the second document. In these cases, you should provide an explicit mapping template that chooses the most flexible number type, such as the following: `{ "template": { "mappings": { "properties": { "MixedNumberField": { "type": "float" } } } } }`	Amazon DocumentDB supports doubles.

Data type	OpenSearch	Amazon DocumentDB
Date	By default, date maps to an integer in OpenSearch. You can define a custom mapping template to map a date to an OpenSearch date. `{ "template": { "mappings": { "properties": { "myDateField": { "type": "date", "format": "epoch_second" } } } } }`	Amazon DocumentDB supports dates.
Timestamp	By default, timestamp maps to an integer in OpenSearch. You can define a custom mapping template to map a date to an OpenSearch date. `{ "template": { "mappings": { "properties": { "myTimestampField": { "type": "date", "format": "epoch_second" } } } } }`	Amazon DocumentDB supports timestamps.

Data type	OpenSearch	Amazon DocumentDB
Boolean	OpenSearch maps a Amazon DocumentDB Boolean type into an OpenSearch Boolean type.	Amazon DocumentDB supports Boolean type attributes.
Decimal	OpenSearch maps Amazon DocumentDB map attributes to nested fields. The same mappings apply within a nested field. The following example maps a string in a nested field to a keyword type in OpenSearch: `{ "template": { "mappings": { "properties": { "myDecimalField": { "type": "double" } } } } }` With this custom mapping, you can query and aggregate the field with double-level precision. The original value retains the full precision in the `_source` property of the OpenSearch document. Without this mapping, OpenSearch uses text by default.	Amazon DocumentDB supports decimals.
Regular Expression	The regex type creates nested fields. These include *<myFieldName>* `.pattern` and *<myFieldName>* `.options`.	Amazon DocumentDB supports regular expressions.

Data type	OpenSearch	Amazon DocumentDB
Binary Data	OpenSearch automatically maps Amazon DocumentDB binary data to OpenSearch text. You can provide a mapping to write these as binary fields in OpenSearch. The following example shows how to map a Amazon DocumentDB field named `imageData` to an OpenSearch binary field. `{ "template": { "mappings": { "properties": { "imageData": { "type": "binary" } } } } }`	Amazon DocumentDB supports binary data fields.
ObjectId	Fields with a type of objectId map to OpenSearch text fields. The value will be the string representation of the objectId.	Amazon DocumentDB supports objectIds.

Data type	OpenSearch	Amazon DocumentDB
Null	OpenSearch can ingest documents with the Amazon DocumentDB null type. It saves the value as a null value in the document. There is no mapping for this type, and this field is not indexed or searchable. If the same attribute name is used for a null type and then later changes to different type such as string, OpenSearch creates a dynamic mapping for the first non-null value. Subsequent values can still be Amazon DocumentDB null values.	Amazon DocumentDB supports null type fields.
Undefined	OpenSearch can ingest documents with the Amazon DocumentDB undefined type. It saves the value as a null value in the document. There is no mapping for this type, and this field is not indexed or searchable. If the same field name is used for a undefined type and then later changes to different type such as string, OpenSearch creates a dynamic mapping for the first non-undefined value. Subsequent values can still be Amazon DocumentDB undefined values.	Amazon DocumentDB supports undefined type fields.

Data type	OpenSearch	Amazon DocumentDB
MinKey	OpenSearch can ingest documents with the Amazon DocumentDB minKey type. It saves the value as a null value in the document. There is no mapping for this type, and this field is not indexed or searchable. If the same field name is used for a minKey type and then later changes to different type such as string, OpenSearch creates a dynamic mapping for the first non-minKey value. Subsequent values can still be Amazon DocumentDB minKey values.	Amazon DocumentDB supports minKey type fields.
MaxKey	OpenSearch can ingest documents with the Amazon DocumentDB maxKey type. It saves the value as a null value in the document. There is no mapping for this type, and this field is not indexed or searchable. If the same field name is used for a maxKey type and then later changes to different type such as string, OpenSearch creates a dynamic mapping for the first non-maxKey value. Subsequent values can still be Amazon DocumentDB maxKey values.	Amazon DocumentDB supports maxKey type fields.

We recommend that you configure the dead-letter queue (DLQ) in your OpenSearch Ingestion pipeline. If you've configured the queue, OpenSearch Service sends all failed documents that can't be ingested due to dynamic mapping failures to the queue.

In case automatic mappings fail, you can use `template_type` and `template_content` in your pipeline configuration to define explicit mapping rules. Alternatively, you can create mapping templates directly in your search domain or collection before you start the pipeline.

Limitations

Consider the following limitations when you set up an OpenSearch Ingestion pipeline for Amazon DocumentDB:

- The OpenSearch Ingestion integration with Amazon DocumentDB currently doesn't support cross-Region ingestion. Your Amazon DocumentDB cluster and OpenSearch Ingestion pipeline must be in the same AWS Region.
- The OpenSearch Ingestion integration with Amazon DocumentDB currently doesn't support cross-account ingestion. Your Amazon DocumentDB cluster and OpenSearch Ingestion pipeline must be in the same AWS account.
- An OpenSearch Ingestion pipeline supports only one Amazon DocumentDB cluster as its source.
- The OpenSearch Ingestion integration with Amazon DocumentDB specifically supports Amazon DocumentDB instance-based clusters. It doesn't support Amazon DocumentDB elastic clusters.
- The OpenSearch Ingestion integration only supports AWS Secrets Manager as an authentication mechanism for your Amazon DocumentDB cluster.
- You can't update the existing pipeline configuration to ingest data from a different database or collection. Instead, you must create a new pipeline.

Recommended CloudWatch alarms

For the best performance, we recommend that you use the following CloudWatch alarms when you create an OpenSearch Ingestion pipeline to access an Amazon DocumentDB cluster as a source.

CloudWatch Alarm	Description
<pipeline-name> .doucmentdb.credentialsChanged	This metric indicates how often AWS secrets are rotated.
<pipeline-name> .doucmentdb.executorRefreshErrors	This metric indicates failures to refresh AWS secrets.

CloudWatch Alarm	Description
<pipeline-name> .doucmentdb.export RecordsTotal	This metric indicates the number of records exported from Amazon DocumentDB.
<pipeline-name> .doucmentdb.export RecordsProcessed	This metric indicates the number of records processed by OpenSearch Ingestion pipeline.
<pipeline-name> .doucmentdb.export RecordProcessingErrors	This metric indicates number of processing errors in an OpenSearch Ingestion pipeline while reading the data from an Amazon DocumentDB cluster.
<pipeline-name> .doucmentdb.export RecordsSuccessTotal	This metric indicates the total number of export records processed successfully.
<pipeline-name> .doucmentdb.export RecordsFailedTotal	This metric indicates the total number of export records that failed to process.
<pipeline-name> .doucmentdb.bytesR eceived	This metrics indicates the total number of bytes received by an OpenSearch Ingestion pipeline.
<pipeline-name> .doucmentdb.bytesP rocessed	This metrics indicates the total number of bytes processed by an OpenSearch Ingestion pipeline.
<pipeline-name> .doucmentdb.export PartitionQueryTotal	This metric indicates the export partition total.
<pipeline-name> .doucmentdb.stream RecordsSuccessTotal	This metric indicates the number of records successfully processed from the stream.
<pipeline-name> .doucmentdb.stream RecordsFailedTotal	This metrics indicates the total number of records failed to process from the stream.

Using an OpenSearch Ingestion pipeline with Confluent Cloud Kafka

You can use an OpenSearch Ingestion pipeline to stream data from Confluent Cloud Kafka clusters to Amazon OpenSearch Service domains and OpenSearch Serverless collections. OpenSearch Ingestion supports both public and private network configurations for the streaming of data from Confluent Cloud Kafka clusters to domains or collections managed by OpenSearch Service or OpenSearch Serverless.

Connectivity to Confluent Cloud public Kafka clusters

You can use OpenSearch Ingestion pipelines to migrate data from a Confluent Cloud Kafka cluster with a public configuration, which means that the domain DNS name can be publicly resolved. To do so, set up an OpenSearch Ingestion pipeline with Confluent Cloud public Kafka cluster as the source and OpenSearch Service or OpenSearch Serverless as the destination. This processes your streaming data from a self-managed source cluster to an AWS-managed destination domain or collection.

Prerequisites

Before you create your OpenSearch Ingestion pipeline, perform the following steps:

1. Create a Confluent Cloud Kafka clusters cluster acting as a source. The cluster should contain the data you want to ingest into OpenSearch Service.
2. Create an OpenSearch Service domain or OpenSearch Serverless collection where you want to migrate data to. For more information, see Creating OpenSearch Service domains and Creating collections.
3. Set up authentication on your Confluent Cloud Kafka cluster with AWS Secrets Manager. Enable secrets rotation by following the steps in Rotate AWS Secrets Manager secrets.
4. Attach a resource-based policy to your domain or a data access policy to your collection. These access policies allow OpenSearch Ingestion to write data from your self-managed cluster to your domain or collection.

   The following sample domain access policy allows the pipeline role, which you create in the next step, to write data to a domain. Make sure that you update the `resource` with your own ARN.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{pipeline-account-id}:role/pipeline-role"
      },
      "Action": [
        "es:DescribeDomain",
        "es:ESHttp*"
      ],
      "Resource": [
        "arn:aws:es:{region}:{account-id}:domain/domain-name"
      ]
    }
  ]
}
```

To create an IAM role with the correct permissions to access write data to the collection or domain, see Required permissions for domains and Required permissions for collections.

Step 1: Configure the pipeline role

After you have your Confluent Cloud Kafka cluster pipeline prerequisites set up, configure the pipeline role that you want to use in your pipeline configuration, and add permission to write to an OpenSearch Service domain or OpenSearch Serverless collection, as well as permission to read secrets from Secrets Manager.

The following permission is needed to manage the network interface:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AttachNetworkInterface",
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DetachNetworkInterface",
                "ec2:DescribeNetworkInterfaces"
            ],
```

```
            "Resource": [
                "arn:aws:ec2:*:{account-id}:network-interface/*",
                "arn:aws:ec2:*:{account-id}:subnet/*",
                "arn:aws:ec2:*:{account-id}:security-group/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:Describe*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [ "ec2:CreateTags" ],
            "Resource": "arn:aws:ec2:*:*:network-interface/*",
            "Condition": {
               "StringEquals": { "aws:RequestTag/OSISManaged": "true" }
            }
        }
    ]
}
```

The following is permission needed to read secrets from AWS Secrets Manager service:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerReadAccess",
            "Effect": "Allow",
            "Action": ["secretsmanager:GetSecretValue"],
            "Resource": ["arn:aws:secretsmanager:<region:<account-id>:secret:<,secret-
name>"]
        }
    ]
}
```

The following permissions are needed to write to an Amazon OpenSearch Service domain:

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{your-account-id}:role/{pipeline-role}"
      },
      "Action": ["es:DescribeDomain", "es:ESHttp*"],
      "Resource": "arn:aws:es:{region}:{your-account-id}:domain/{domain-name}/*"
    }
  ]
}
```

Step 2: Create the pipeline

You can then configure an OpenSearch Ingestion pipeline like the following, which specifies your Confluent Cloud Kafka as the source.

You can specify multiple OpenSearch Service domains as destinations for your data. This capability enables conditional routing or replication of incoming data into multiple OpenSearch Service domains.

You can also migrate data from a source Confluent Kafka cluster to an OpenSearch Serverless VPC collection. Ensure you provide a network access policy within the pipeline configuration. You can use a Confluent schema registry to define a Confluent schema.

```
version: "2"
kafka-pipeline:
  source:
    kafka:
      encryption:
        type: "ssl"
      topics:
        - name: "topic-name"
          group_id: "group-id"
      bootstrap_servers:
        - "bootstrap-server.us-west-2.aws.private.confluent.cloud:9092"
      authentication:
        sasl:
          plain:
            username: ${aws_secrets:confluent-kafka-secret:username}
```

```
            password: ${aws_secrets:confluent-kafka-secret:password}
      schema:
        type: confluent
        registry_url: https://my-registry.us-west-2.aws.confluent.cloud
        api_key: "${{aws_secrets:schema-secret:schema_registry_api_key}}"
        api_secret: "${{aws_secrets:schema-secret:schema_registry_api_secret}}"
        basic_auth_credentials_source: "USER_INFO"
  sink:
  - opensearch:
      hosts: ["https://search-mydomain.us-west-2.es.amazonaws.com"]
      aws:
          sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
          region: "us-west-2"
      index: "confluent-index"
extension:
  aws:
    secrets:
      confluent-kafka-secret:
        secret_id: "my-kafka-secret"
        region: "us-west-2"
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
      schema-secret:
        secret_id: "my-self-managed-kafka-schema"
        region: "us-west-2"
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
```

You can use a preconfigured blueprint to create this pipeline. For more information, see the section called “Using blueprints to create a pipeline”.

Connectivity to Confluent Cloud Kafka clusters in a VPC

You can also use OpenSearch Ingestion pipelines to migrate data from a Confluent Cloud Kafka cluster running in a VPC. To do so, set up an OpenSearch Ingestion pipeline with a Confluent Cloud Kafka cluster as a source and OpenSearch Service or OpenSearch Serverless as the destination. This processes your streaming data from a Confluent Cloud Kafka source cluster to an AWS-managed destination domain or collection.

OpenSearch Ingestion supports Confluent Cloud Kafka clusters configured in all supported network modes in Confluent. The following modes of network configuration are supported as a source in OpenSearch Ingestion:

- AWS VPC peering

- AWS PrivateLink for dedicated clusters
- AWS PrivateLink for Enterprise clusters
- AWS Transit Gateway

Prerequisites

Before you create your OpenSearch Ingestion pipeline, perform the following steps:

1. Create a Confluent Cloud Kafka cluster with a VPC network configuration that contains the data you want to ingest into OpenSearch Service.
2. Create an OpenSearch Service domain or OpenSearch Serverless collection where you want to migrate data to. For more information, see Creating OpenSearch Service domains and Creating collections.
3. Set up authentication on your Confluent Cloud Kafka cluster with AWS Secrets Manager. Enable secrets rotation by following the steps in Rotate AWS Secrets Manager secrets.
4. Obtain the ID of the VPC that that has access to self-managed Kafka. Choose the VPC CIDR to be used by OpenSearch Ingestion.

   > **Note**
   >
   > If you're using the AWS Management Console to create your pipeline, you must also attach your OpenSearch Ingestion pipeline to your VPC in order to use self-managed Kafka. To do so, find the **Network configuration** section, select the **Attach to VPC** checkbox, and choose your CIDR from one of the provided default options, or select your own. You can use any CIDR from a private address space as defined in the RFC 1918 Best Current Practice.
   >
   > To provide a custom CIDR, select **Other** from the dropdown menu. To avoid a collision in IP addresses between OpenSearch Ingestion and self-managed OpenSearch, ensure that the self-managed OpenSearch VPC CIDR is different from the CIDR for OpenSearch Ingestion.

5. Attach a resource-based policy to your domain or a data access policy to your collection. These access policies allow OpenSearch Ingestion to write data from your self-managed cluster to your domain or collection.

> **Note**
>
> If you are using AWS PrivateLink to connect your Confluent Cloud Kafka, you will need to configure VPC DHCP Options. *DNS hostnames* and *DNS resolutions*should be enabled.

The following sample domain access policy allows the pipeline role, which you create in the next step, to write data to a domain. Make sure that you update the `resource` with your own ARN.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{pipeline-account-id}:role/pipeline-role"
      },
      "Action": [
        "es:DescribeDomain",
        "es:ESHttp*"
      ],
      "Resource": [
        "arn:aws:es:{region}:{account-id}:domain/domain-name"
      ]
    }
  ]
}
```

To create an IAM role with the correct permissions to access write data to the collection or domain, see Required permissions for domains and Required permissions for collections.

Step 1: Configure the pipeline role

After you have your pipeline prerequisites set up, configure the pipeline role that you want to use in your pipeline configuration, and add the following permissions in the role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "SecretsManagerReadAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": ["arn:aws:secretsmanager:{region}:{account-id}:secret:secret-
name"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AttachNetworkInterface",
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DetachNetworkInterface",
                "ec2:DescribeNetworkInterfaces"
            ],
            "Resource": [
                "arn:aws:ec2:*:{account-id}:network-interface/*",
                "arn:aws:ec2:*:{account-id}:subnet/*",
                "arn:aws:ec2:*:{account-id}:security-group/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:Describe*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": "arn:aws:ec2:*:*:network-interface/*",
            "Condition": {
```

```
                "StringEquals":
                     {
                         "aws:RequestTag/OSISManaged": "true"
                     }
            }
        }
    ]
}
```

You must provide the above Amazon EC2 permissions on the IAM role that you use to create the OpenSearch Ingestion pipeline because the pipeline uses these permissions to create and delete a network interface in your VPC. The pipeline can only access the Kafka cluster through this network interface.

Step 2: Create the pipeline

You can then configure an OpenSearch Ingestion pipeline like the following, which specifies Kafka as the source.

You can specify multiple OpenSearch Service domains as destinations for your data. This capability enables conditional routing or replication of incoming data into multiple OpenSearch Service domains.

You can also migrate data from a source Confluent Kafka cluster to an OpenSearch Serverless VPC collection. Ensure you provide a network access policy within the pipeline configuration. You can use a Confluent schema registry to define a Confluent schema.

```
 version: "2"
kafka-pipeline:
  source:
    kafka:
      encryption:
        type: "ssl"
      topics:
        - name: "topic-name"
          group_id: "group-id"
      bootstrap_servers:
        - "bootstrap-server.us-west-2.aws.private.confluent.cloud:9092"
      authentication:
        sasl:
          plain:
            username: ${aws_secrets:confluent-kafka-secret:username}
```

```
            password: ${aws_secrets:confluent-kafka-secret:password}
      schema:
        type: confluent
        registry_url: https://my-registry.us-west-2.aws.confluent.cloud
        api_key: "${{aws_secrets:schema-secret:schema_registry_api_key}}"
        api_secret: "${{aws_secrets:schema-secret:schema_registry_api_secret}}"
        basic_auth_credentials_source: "USER_INFO"
  sink:
  - opensearch:
      hosts: ["https://search-mydomain.us-west-2.es.amazonaws.com"]
      aws:
          sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
          region: "us-west-2"
      index: "confluent-index"
extension:
  aws:
    secrets:
      confluent-kafka-secret:
        secret_id: "my-kafka-secret"
        region: "us-west-2"
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
      schema-secret:
        secret_id: "my-self-managed-kafka-schema"
        region: "us-west-2"
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
```

You can use a preconfigured blueprint to create this pipeline. For more information, see the section called “Using blueprints to create a pipeline”.

Using an OpenSearch Ingestion pipeline with Amazon Managed Streaming for Apache Kafka

You can use the Kafka plugin to ingest data from Amazon Managed Streaming for Apache Kafka (Amazon MSK) into your OpenSearch Ingestion pipeline. With Amazon MSK, you can build and run applications that use Apache Kafka to process streaming data. OpenSearch Ingestion uses AWS PrivateLink to connect to Amazon MSK. You can ingest data from both Amazon MSK and Amazon MSK Serverless clusters. The only difference between the two processes is the prerequisite steps you must take before you set up your pipeline.

Topics

- Amazon MSK prerequisites

- Amazon MSK Serverless prerequisites
- Configure a pipeline role
- Step 2: Create the pipeline
- Step 3: (Optional) Use the AWS Glue Schema Registry
- Step 4: (Optional) Configure recommended compute units (OCUs) for the Amazon MSK pipeline

Amazon MSK prerequisites

Before you create your OpenSearch Ingestion pipeline, perform the following steps:

1. Create an Amazon MSK provisioned cluster by following the steps in Creating a cluster in the *Amazon Managed Streaming for Apache Kafka Developer Guide*. For **Broker type**, choose any option except for `t3` types, as these aren't supported by OpenSearch Ingestion.
2. After the cluster has an **Active** status, follow the steps in Turn on multi-VPC connectivity.
3. Follow the steps in Attach a cluster policy to the MSK cluster to attach one of the following policies, depending on if your cluster and pipeline are in the same AWS account. This policy allows OpenSearch Ingestion to create a AWS PrivateLink connection to your Amazon MSK cluster and read data from Kafka topics. Make sure that you update the `resource` with your own ARN.

   The following policies applies when your cluster and pipeline are in the same AWS account:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "osis.amazonaws.com"
      },
      "Action": [
        "kafka:CreateVpcConnection",
        "kafka:DescribeClusterV2"
      ],
      "Resource": "arn:aws:kafka:us-east-1:{account-id}:cluster/cluster-name/cluster-
id"
    },
    {
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "osis-pipelines.amazonaws.com"
      },
      "Action": [
        "kafka:CreateVpcConnection",
        "kafka:GetBootstrapBrokers",
        "kafka:DescribeClusterV2"
      ],
      "Resource": "arn:aws:kafka:us-east-1:{account-id}:cluster/cluster-name/cluster-
id"
    }
  ]
}
```

If your Amazon MSK cluster is in a different AWS account than your pipeline, attach the following policy instead. Note that cross-account access is only possible with provisioned Amazon MSK clusters and not Amazon MSK Serverless clusters. The ARN for the AWS `principal` should be the ARN for the same pipeline role that you provide to your pipleine YAML configuration:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "osis.amazonaws.com"
      },
      "Action": [
        "kafka:CreateVpcConnection",
        "kafka:DescribeClusterV2"
      ],
      "Resource": "arn:aws:kafka:us-east-1:{msk-account-id}:cluster/cluster-
name/cluster-id"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "osis-pipelines.amazonaws.com"
      },
      "Action": [
        "kafka:CreateVpcConnection",
```

```
        "kafka:GetBootstrapBrokers",
        "kafka:DescribeClusterV2"
      ],
      "Resource": "arn:aws:kafka:us-east-1:{msk-account-id}:cluster/cluster-
name/cluster-id"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{pipeline-account-id}:role/pipeline-role"
      },
      "Action": [
        "kafka-cluster:*",
        "kafka:*"
      ],
      "Resource": [
        "arn:aws:kafka:us-east-1:{msk-account-id}:cluster/cluster-name/cluster-id",
        "arn:aws:kafka:us-east-1:{msk-account-id}:topic/cluster-name/cluster-id/*",
        "arn:aws:kafka:us-east-1:{msk-account-id}:group/cluster-name/*"
      ]
    }
  ]
}
```

4. Create a Kafka topic by following the steps in Create a topic. Make sure that *BootstrapServerString* is one of the private endpoint (single-VPC) bootstrap URLs. The value for `--replication-factor` should be 2 or 3, based on the number of zones your Amazon MSK cluster has. The value for `--partitions` should be at least 10.
5. Produce and consume data by following the steps in Produce and consume data. Again, make sure that *BootstrapServerString* is one of your private endpoint (single-VPC) bootstrap URLs.

Amazon MSK Serverless prerequisites

Before you create your OpenSearch Ingestion pipeline, perform the following steps:

1. Create an Amazon MSK Serverless cluster by following the steps in Create an MSK Serverless cluster in the *Amazon Managed Streaming for Apache Kafka Developer Guide*.

2. After the cluster has an **Active** status, follow the steps in Attach a cluster policy to the MSK cluster to attach the following policy. Make sure that you update the `resource` with your own ARN.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "osis.amazonaws.com"
      },
      "Action": [
        "kafka:CreateVpcConnection",
        "kafka:DescribeClusterV2"
      ],
      "Resource": "arn:aws:kafka:us-east-1:{account-id}:cluster/cluster-name/cluster-id"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "osis-pipelines.amazonaws.com"
      },
      "Action": [
        "kafka:CreateVpcConnection",
        "kafka:GetBootstrapBrokers",
        "kafka:DescribeClusterV2"
      ],
      "Resource": "arn:aws:kafka:us-east-1:{account-id}:cluster/cluster-name/cluster-id"
    }
  ]
}
```

   This policy allows OpenSearch Ingestion to create a AWS PrivateLink connection to your Amazon MSK Serverless cluster and read data from Kafka topics. This policy applies when your cluster and pipeline are in the same AWS account, which must be true as Amazon MSK Serverless doesn't support cross-account access.

3. Create a Kafka topic by following the steps in Create a topic. Make sure that *`BootstrapServerString`* is one of your Simple Authentication and Security Layer (SASL) IAM

bootstrap URLs. The value for `--replication-factor` should be 2 or 3, based on the number of zones your Amazon MSK Serverless cluster has. The value for `--partitions` should be at least 10.

4. Produce and consume data by following the steps in Produce and consume data. Again, make sure that *`BootstrapServerString`* is one of your Simple Authentication and Security Layer (SASL) IAM bootstrap URLs.

Configure a pipeline role

After you have your Amazon MSK provisoned or serverless cluster set up, add the following Kafka permissions in the pipeline role that you want to use in your pipeline configuration:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:Connect",
                "kafka-cluster:AlterCluster",
                "kafka-cluster:DescribeCluster",
                "kafka:DescribeClusterV2",
                "kafka:GetBootstrapBrokers"
            ],
            "Resource": [
                "arn:aws:kafka:us-east-1:{account-id}:cluster/cluster-name/cluster-id"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:*Topic*",
                "kafka-cluster:ReadData"
            ],
            "Resource": [
                "arn:aws:kafka:us-east-1:{account-id}:topic/cluster-name/cluster-
id/topic-name"
            ]
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "kafka-cluster:AlterGroup",
                "kafka-cluster:DescribeGroup"
            ],
            "Resource": [
                "arn:aws:kafka:us-east-1:{account-id}:group/cluster-name/*"
            ]
        }
    ]
}
```

Step 2: Create the pipeline

You can then configure an OpenSearch Ingestion pipeline like the following, which specifies Kafka as the source:

```
version: "2"
log-pipeline:
  source:
    kafka:
      acknowledgements: true
      topics:
      - name: "topic-name"
        group_id: "group-id"
      aws:
        msk:
          arn: "arn:aws:kafka:{region}:{account-id}:cluster/cluster-name/cluster-id"
        region: "us-west-2"
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
  processor:
  - grok:
      match:
        message:
        - "%{COMMONAPACHELOG}"
  - date:
      destination: "@timestamp"
      from_time_received: true
  sink:
  - opensearch:
      hosts: ["https://search-domain-endpoint.us-east-1.es.amazonaws.com"]
      index: "index_name"
      aws_sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
      aws_region: "us-east-1"
```

```
      aws_sigv4: true
```

You can use a preconfigured Amazon MSK blueprint to create this pipeline. For more information, see the section called "Using blueprints to create a pipeline".

Step 3: (Optional) Use the AWS Glue Schema Registry

When you use OpenSearch Ingestion with Amazon MSK, you can use the AVRO data format for schemas hosted in the AWS Glue Schema Registry. With the AWS Glue Schema Registry, you can centrally discover, control, and evolve data stream schemas.

To use this option, enable the schema `type` in your pipeline configuration:

```
schema:
  type: "aws_glue"
```

You must also provide AWS Glue with read access permissions in your pipeline role. You can use the AWS managed policy called AWSGlueSchemaRegistryReadonlyAccess. Additionally, your registry must be in the same AWS account and Region as your OpenSearch Ingestion pipeline.

Step 4: (Optional) Configure recommended compute units (OCUs) for the Amazon MSK pipeline

Each compute unit has one consumer per topic. Brokers balance partitions among these consumers for a given topic. However, when the number of partitions is greater than the number of consumers, Amazon MSK hosts multiple partitions on every consumer. OpenSearch Ingestion has built-in auto scaling to scale up or down based on CPU usage or number of pending records in the pipeline.

For optimal performance, distribute your partitions across many compute units for parallel processing. If topics have a large number of partitions (for example, more than 96, which is the maximum OCUs per pipeline), we recommend that you configure a pipeline with 1–96 OCUs. This is because it will automatically scale as needed. If a topic has a low number of partitions (for example, less than 96), keep the maximum compute unit the same as the number of partitions.

When a pipeline has more than one topic, choose the topic with the highest number of partitions as a reference to configure maximum computes units. By adding another pipeline with a new set of OCUs to the same topic and consumer group, you can scale the throughput almost linearly.

Using an OpenSearch Ingestion pipeline with Amazon S3

With OpenSearch Ingestion, you can use Amazon S3 as a source or as a destination. When you use Amazon S3 as a source, you send data to an OpenSearch Ingestion pipeline. When you use Amazon S3 as a destination, you write data from an OpenSearch Ingestion pipeline to one or more S3 buckets.

Topics

Amazon S3 as a source

There are two ways that you can use Amazon S3 as a source to process data—with *S3-SQS processing* and with *scheduled scans*.

Use S3-SQS processing when you require near real-time scanning of files after they are written to S3. You can configure Amazon S3 buckets to raise an event any time an object is stored or modified within the bucket. Use a one-time or recurring scheduled scan to batch process data in a S3 bucket.

Topics

Prerequisites

To use Amazon S3 as the source for an OpenSearch Ingestion pipeline for both a scheduled scan or S3-SQS processing, first create an S3 bucket.

> **Note**
>
> If the S3 bucket used as a source in the OpenSearch Ingestion pipeline is in a different AWS account, you also need to enable cross-account read permissions on the bucket. This allows the pipeline to read and process the data. To enable cross-account permissions, see Bucket owner granting cross-account bucket permissions in the *Amazon S3 User Guide*.

If your S3 buckets are in multiple accounts, use a `bucket_owners` map. For an example, see Cross-account S3 access in the OpenSearch documentation.

To set up S3-SQS processing, you also need to perform the following steps:

1. Create an Amazon SQS queue.
2. Enable event notifications on the S3 bucket with the SQS queue as a destination.

Step 1: Configure the pipeline role

Unlike other source plugins that *push* data to a pipeline, the S3 source plugin has a read-based architecture in which the pipeline *pulls* data from the source.

Therefore, in order for a pipeline to read from S3, you must specify a role within the pipeline's S3 source configuration that has access to both the S3 bucket and the Amazon SQS queue. The pipeline will assume this role in order to read data from the queue.

Note

The role that you specify within the S3 source configuration must be the pipeline role. Therefore, your pipeline role must contain two separate permissions policies—one to write to a sink, and one to pull from the S3 source. You must use the same `sts_role_arn` in all pipeline components.

The following sample policy shows the required permissions for using S3 as a source:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action":[
          "s3:ListBucket",
          "s3:GetBucketLocation",
          "s3:GetObject"
       ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-/*"
    },
```

```
    {
       "Effect":"Allow",
       "Action":"s3:ListAllMyBuckets",
       "Resource":"arn:aws:s3:::*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "sqs:DeleteMessage",
        "sqs:ReceiveMessage",
        "sqs:ChangeMessageVisibility"
      ],
      "Resource": "arn:aws:sqs:us-west-2:{account-id}:MyS3EventSqsQueue"
    }
  ]
}
```

You must attach these permissions to the IAM role that you specify in the `sts_role_arn` option within the S3 source plugin configuration:

```
version: "2"
source:
  s3:
    ...
    aws:
      ...
      sts_role_arn: arn:aws:iam::{account-id}:role/pipeline-role
processor:
  ...
sink:
  - opensearch:
      ...
```

Step 2: Create the pipeline

After you've set up your permissions, you can configure an OpenSearch Ingestion pipeline depending on your Amazon S3 use case.

S3-SQS processing

To set up S3-SQS processing, configure your pipeline to specify S3 as the source and set up Amazon SQS notifications:

```
version: "2"
s3-pipeline:
  source:
    s3:
      notification_type: "sqs"
      codec:
        newline: null
      sqs:
        queue_url: "https://sqs.us-east-1.amazonaws.com/{account-id}/ingestion-queue"
      compression: "none"
      aws:
        region: "us-east-1"

        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
  processor:
  - grok:
      match:
        message:
        - "%{COMMONAPACHELOG}"
  - date:
      destination: "@timestamp"
      from_time_received: true
  sink:
  - opensearch:
      hosts: ["https://search-domain-endpoint.us-east-1.es.amazonaws.com"]
      index: "index-name"
      aws:
        # IAM role that the pipeline assumes to access the domain sink
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
        region: "us-east-1"
```

If you observe low CPU utilization while processing small files on Amazon S3, consider increasing the throughput by modifying the value of the `workers` option. For more information, see the S3 plugin configuration options.

Scheduled scan

To set up a scheduled scan, configure your pipeline with a schedule at the scan level that applies to all your S3 buckets, or at the bucket level. A bucket-level schedule or a scan-interval configuration always overwrites a scan-level configuration.

You can configure scheduled scans with either a *one-time scan*, which is ideal for data migration, or a *recurring scan*, which is ideal for batch processing.

To configure your pipeline to read from Amazon S3, use the preconfigured Amazon S3 blueprints. You can edit the `scan` portion of your pipeline configuration to meet your scheduling needs. For more information, see the section called “Using blueprints to create a pipeline”.

One-time scan

A one-time scheduled scan runs once. In your YAML configuration, you can use a `start_time` and `end_time` to specify when you want the objects in the bucket to be scanned. Alternatively, you can use `range` to specify the interval of time relative to current time that you want the objects in the bucket to be scanned.

For example, a range set to PT4H scans all files created in the last four hours. To configure a one-time scan to run a second time, you must stop and restart the pipeline. If you don't have a range configured, you must also update the start and end times.

The following configuration sets up a one-time scan for all buckets and all objects in those buckets:

```
version: "2"
log-pipeline:
  source:
    s3:
      codec:
        csv:
      compression: "none"
      aws:
        region: "us-east-1"
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
      acknowledgments: true
      scan:
        buckets:
          - bucket:
              name: amzn-s3-demo-1
              filter:
                include_prefix:
                  - Objects1/
                exclude_suffix:
                  - .jpeg
                  - .png
          - bucket:
```

```
              name: my-bucket-2
              key_prefix:
                include:
                  - Objects2/
                exclude_suffix:
                  - .jpeg
                  - .png
      delete_s3_objects_on_read: false
  processor:
    - date:
        destination: "@timestamp"
        from_time_received: true
  sink:
    - opensearch:
        hosts: ["https://search-domain-endpoint.us-east-1.es.amazonaws.com"]
        index: "index-name"
        aws:
          sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
          region: "us-east-1"
        dlq:
          s3:
            bucket: "my-bucket-1"
            region: "us-east-1"
            sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
```

The following configuration sets up a one-time scan for all buckets during a specified time window. This means that S3 processes only those objects with creation times that fall within this window.

```
scan:
  start_time: 2023-01-21T18:00:00.000Z
  end_time: 2023-04-21T18:00:00.000Z
  buckets:
    - bucket:
        name: my-bucket-1
        filter:
          include:
            - Objects1/
          exclude_suffix:
            - .jpeg
            - .png
    - bucket:
        name: my-bucket-2
        filter:
```

```
          include:
            - Objects2/
          exclude_suffix:
            - .jpeg
            - .png
```

The following configuration sets up a one-time scan at both the scan level and the bucket level. Start and end times at the bucket level override start and end times at the scan level.

```
scan:
  start_time: 2023-01-21T18:00:00.000Z
  end_time: 2023-04-21T18:00:00.000Z
  buckets:
    - bucket:
        start_time: 2023-01-21T18:00:00.000Z
        end_time: 2023-04-21T18:00:00.000Z
        name: my-bucket-1
        filter:
          include:
            - Objects1/
          exclude_suffix:
            - .jpeg
            - .png
    - bucket:
        start_time: 2023-01-21T18:00:00.000Z
        end_time: 2023-04-21T18:00:00.000Z
        name: my-bucket-2
        filter:
          include:
            - Objects2/
          exclude_suffix:
            - .jpeg
            - .png
```

Stopping a pipeline removes any pre-existing reference of what objects have been scanned by the pipeline before the stop. If a single scan pipeline is stopped, it will rescan all objects again after its started, even if they were already scanned. If you need to stop a single scan pipeline, it is recommended you change your time window before starting the pipeline again.

If you need to filter objects by start time and end time, stopping and starting your pipeline is the only option. If you don't need to filter by start time and end time, you can filter objects

by name. Flitering by name doesn't require you to stop and start your pipeline. To do this, use `include_prefix` and `exclude_suffix`.

Recurring scan

A recurring scheduled scan runs a scan of your specified S3 buckets at regular, scheduled intervals. You can only configure these intervals at the scan level because individual bucket level configurations aren't supported.

In your YAML configuration, the `interval` specifies the frequency of the recurring scan, and can be between 30 seconds and 365 days. The first of these scans always occurs when you create the pipeline. The `count` defines the total number of scan instances.

The following configuration sets up a recurring scan, with a delay of 12 hours between the scans:

```
scan:
  scheduling:
    interval: PT12H
    count: 4
  buckets:
    - bucket:
        name: my-bucket-1
        filter:
          include:
            - Objects1/
          exclude_suffix:
            - .jpeg
            - .png
    - bucket:
        name: my-bucket-2
        filter:
          include:
            - Objects2/
          exclude_suffix:
            - .jpeg
            - .png
```

Amazon S3 as a destination

To write data from an OpenSearch Ingestion pipeline to an S3 bucket, use the preconfigured S3 blueprint to create a pipeline with an S3 sink. This pipeline routes selective data to an OpenSearch

sink and simultaneously sends all data for archival in S3. For more information, see the section called “Using blueprints to create a pipeline”.

When you create your S3 sink, you can specify your preferred formatting from a variety of sink codecs. For example, if you want to write data in columnar format, choose the Parquet or Avro codec. If you prefer a row-based format, choose JSON or ND-JSON. To write data to S3 in a specified schema, you can also define an inline schema within sink codecs using the Avro format.

The following example defines an inline schema in an S3 sink:

```
- s3:
  codec:
    parquet:
      schema: >
        {
           "type" : "record",
           "namespace" : "org.vpcFlowLog.examples",
           "name" : "VpcFlowLog",
           "fields" : [
             { "name" : "version", "type" : "string"},
             { "name" : "srcport", "type": "int"},
             { "name" : "dstport", "type": "int"},
             { "name" : "start", "type": "int"},
             { "name" : "end", "type": "int"},
             { "name" : "protocol", "type": "int"},
             { "name" : "packets", "type": "int"},
             { "name" : "bytes", "type": "int"},
             { "name" : "action", "type": "string"},
             { "name" : "logStatus", "type" : "string"}
           ]
         }
```

When you define this schema, specify a superset of all keys that might be present in the different types of events that your pipeline delivers to a sink.

For example, if an event has the possibility of a key missing, add that key in your schema with a `null` value. Null value declarations allow the schema to process non-uniform data (where some events have these keys and others don't). When incoming events do have these keys present, their values are written to sinks.

This schema definition acts as a filter that only allows defined keys to be sent to sinks, and drops undefined keys from incoming events.

You can also use `include_keys` and `exclude_keys` in your sink to filter data that's routed to other sinks. These two filters are mutually exclusive, so you can only use one at a time in your schema. Additionally, you can't use them within user-defined schemas.

To create pipelines with such filters, use the preconfigured sink filter blueprint. For more information, see the section called “Using blueprints to create a pipeline”.

Amazon S3 cross account as a source

You can grant access across accounts with Amazon S3 so that OpenSearch Ingestion pipelines can access S3 buckets in another account as a source. To enable cross-account access, see Bucket owner granting cross-account bucket permissions in the *Amazon S3 User Guide*. After you have granted access, ensure that your pipeline role has the required permissions.

Then, you can create a YAML configuration using `bucket_owners` to enable cross-account access to an Amazon S3 bucket as a source:

```
s3-pipeline:
 source:
  s3:
   notification_type: "sqs"
   codec:
    csv:
     delimiter: ","
     quote_character: "\""
     detect_header: True
   sqs:
    queue_url: "https://sqs.ap-northeast-1.amazonaws.com/401447383613/test-s3-queue"
   bucket_owners:
    my-bucket-01: 123456789012
    my-bucket-02: 999999999999
   compression: "gzip"
```

Using an OpenSearch Ingestion pipeline with Amazon Security Lake

You can use the S3 source plugin to ingest data from Amazon Security Lake into your OpenSearch Ingestion pipeline. Security Lake automatically centralizes security data from AWS environments, on-premises environments, and SaaS providers into a purpose-built data lake. You can create a subscription that replicates data from Security Lake to your OpenSearch Ingestion pipeline, which then writes it to your OpenSearch Service domain or OpenSearch Serverless collection.

To configure your pipeline to read from Security Lake, use the preconfigured Security Lake blueprint. The blueprint includes a default configuration for ingesting Open Cybersecurity Schema Framework (OCSF) parquet files from Security Lake. For more information, see the section called "Using blueprints to create a pipeline".

Topics

Using an OpenSearch Ingestion pipeline with Amazon Security Lake as a source

You can use an Amazon S3 source plugin to ingest data in to your OpenSearch Ingestion pipeline. to write data from any supported source in and ingest the data into an OpenSearch Ingestion. Security Lake automatically centralizes security data from AWS environments, on-premises enviroments, and SaaS providers into a purpose-built data lake.

Amazon Security Lake has the following metadata attributes:

- `bucket_name`: name of the bucket created by Security Lake.
- `path_prefix`: Security Lake custom source name added in the IAM role policy.
- `region`: region of the S3 bucket created by Security Lake.
- `accountID`: accountID where Security Lake is enabled.
- `sts_role_arn`: IAM role you plan use.

Prerequisites

Before you create your OpenSearch Ingestion pipeline, perform the following steps:

- Enable Security Lake.
- Create a subscriber in Security Lake.
  - Choose the sources that you want to ingest into your pipeline.
  - For **Subscriber credentials**, add the ID of the AWS account where you intend to create the pipeline. For the external ID, specify `OpenSearchIngestion-{accountid}`.
  - For **Data access method**, choose **S3**.
  - For **Notification details**, choose **SQS queue**.

When you create a subscriber, Security Lake automatically creates two inline permissions policies—one for S3 and one for SQS. The policies take the following format: `AmazonSecurityLake-{12345}-S3` and `AmazonSecurityLake-{12345}-SQS`. To allow your pipeline to access the subscriber sources, you must associate the required permissions with your pipeline role.

Configure the pipeline role

Create a new permissions policy in IAM that combines only the required permissions from the two policies that Security Lake automatically created. The following example policy shows the least privilege required for an OpenSearch Ingestion pipeline to read data from multiple Security Lake sources:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject"
         ],
         "Resource":[
            "arn:aws:s3:::aws-security-data-lake-{region}-abcde/aws/
LAMBDA_EXECUTION/1.0/*",
            "arn:aws:s3:::aws-security-data-lake-{region}-abcde/aws/S3_DATA/1.0/*",
            "arn:aws:s3:::aws-security-data-lake-{region}-abcde/aws/VPC_FLOW/1.0/*",
            "arn:aws:s3:::aws-security-data-lake-{region}-abcde/aws/ROUTE53/1.0/*",
            "arn:aws:s3:::aws-security-data-lake-{region}-abcde/aws/SH_FINDINGS/1.0/*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "sqs:ReceiveMessage",
            "sqs:DeleteMessage"
         ],
         "Resource":[
            "arn:aws:sqs:{region}:{account-id}:AmazonSecurityLake-abcde-Main-Queue"
         ]
      }
   ]
}
```

> **Important**
>
> Security Lake doesn't manage the pipeline role policy for you. If you add or remove sources from your Security Lake subscription, you must manually update the policy. Security Lake creates partitions for each log source, so you need to manually add or remove permissions in the pipeline role.

You must attach these permissions to the IAM role that you specify in the `sts_role_arn` option within the S3 source plugin configuration, under `sqs`.

```
version: "2"
source:
  s3:
    ...
    sqs:
      queue_url: "https://sqs.{region}.amazonaws.com/{account-id}/
AmazonSecurityLake-abcde-Main-Queue"
    aws:
      ...
      sts_role_arn: arn:aws:iam::{account-id}:role/pipeline-role
processor:
  ...
sink:
  - opensearch:
      ...
```

Create the pipeline

After you add the permissions to the pipeline role, use the preconfigured Security Lake blueprint to create the pipeline. For more information, see the section called "Using blueprints to create a pipeline".

You must specify the `queue_url` option within the `s3` source configuration, which is the Amazon SQS queue URL to read from. To format the URL, locate the **Subscription endpoint** in the subscriber configuration and change `arn:aws:` to `https://`. For example, `https://sqs.{region}.amazonaws.com/{account-id}/AmazonSecurityLake-abdcef-Main-Queue`.

The `sts_role_arn` that you specify within the S3 source configuration must be the ARN of the pipeline role.

Using an OpenSearch Ingestion pipeline with Amazon Security Lake as a sink

You can use an Amazon S3 sink plugin in OpenSearch Ingestion to write data from any supported source in OpenSearch Ingestion and ingest the data into Amazon Security Lake. Security Lake automatically centralizes security data from AWS environments, on-premises enviroments, and SaaS providers into a purpose-built data lake.

To configure your pipeline to write log data to Security Lake, use the preconfigured Security Lake blueprint. The blueprint includes a default configuration for ingesting Open Cybersecurity Schema Framework (OCSF) parqet files from Security Lake.

Amazon Security Lake has the following metadata attributes:

- `bucket_name`: name of the bucket created by Security Lake.
- `path_prefix`: Security Lake custom source name added in the IAM role policy.
- `region`: region of the S3 bucket created by Security Lake.
- `accountID`: accountID where Security Lake is enabled.
- `sts_role_arn`: IAM role you plan use.

Prerequisites

Before you create your OpenSearch Ingestion pipeline, perform the following steps:

1. Configure an Amazon Security Lake. To do this, see Amazon Security Lake Documentation data stream acting as a source. The stream should contain the data you want to ingest into OpenSearch Service.
2. Create a custom source in Security Lake. A custom source is required to be configured for Security Lake in order to use OpenSearch Ingestion pipelines for writing data from any log source to a Security Lake S3 bucket.
3. Set up the necessary IAM and OpenSearch Ingestionpermissions for the pipeline role so that your OpenSearch Ingestion pipeline has the ability to read, process, and write data to your Security Lake S3 bucket. For information on setting up an ingestion role, see Ingestion role. To read more on setting up a pipeline role, see Pipeline role. You can also use CloudWatch metrics to monitor pipeline performance. To see how to enable CloudWatch metrics, see CloudWatch permissions.

Create the pipeline

After you add permissions to the pipeline role, use the preconfigured Security Lake blueprint to create the pipeline. For more infromation, see Using blueprints to create a pipeline.

> **Note**
>
> OpenSearch Ingestion supports only one IAM role in pipeline configuration so the IAM role used in custom input source configuration should be used in the OpenSearch Ingestion pipeline configuration.

The following sample pipeline configuration is for generating OCSF events from a sample firewall logs stored in S3 bucket in csv format. You can use any source with a valid OCSF schema mapping to read the logs from and ingest the logs to Security Lake S3 bucket.

```
version: "2"
securitylake-firewall-traffic-pipeline:
  source:
    s3:

      compression: "none"
      codec:
        csv:
      sqs:

      aws:
        region: "<<us-east-1>>"
        sts_role_arn: "<<arn:aws:iam::123456789012:role/Example-Role>>"

  processor:
    - date:
        match:
          - key: Start_Time
            patterns:
              - 'yyyy-MM-dd''T''HH:mm:ss'
              - 'yyyy-MM-dd''T''HH:mm:ss''Z'''
        destination: time_dt
        output_format: 'yyyy-MM-dd''T''HH:mm:ss'
    - date:
        match:
          - key: Start_Time
```

```
          patterns:
            - 'yyyy-MM-dd''T''HH:mm:ss'
            - 'yyyy-MM-dd''T''HH:mm:ss''Z'''
        destination: time
        output_format: epoch_second
    - date:
        match:
          - key: Generated_Time
            patterns:
              - 'yyyy-MM-dd''T''HH:mm:ss'
              - 'yyyy-MM-dd''T''HH:mm:ss''Z'''
        destination: metadata/processed_time
        output_format: epoch_second
    - date:
        match:
          - key: Generated_Time
            patterns:
              - 'yyyy-MM-dd''T''HH:mm:ss'
              - 'yyyy-MM-dd''T''HH:mm:ss''Z'''
        destination: metadata/processed_time_dt
        output_format: 'yyyy-MM-dd''T''HH:mm:ss'
    - date:
        match:
          - key: Receive_Time
            patterns:
              - 'yyyy-MM-dd''T''HH:mm:ss'
              - 'yyyy-MM-dd''T''HH:mm:ss''Z'''
        destination: metadata/logged_time
        output_format: epoch_second
    - date:
        match:
          - key: Receive_Time
            patterns:
              - 'yyyy-MM-dd''T''HH:mm:ss'
              - 'yyyy-MM-dd''T''HH:mm:ss''Z'''
        destination: metadata/logged_time_dt
        output_format: 'yyyy-MM-dd''T''HH:mm:ss'
    - convert_type:
        keys: [ time, metadata/processed_time ]
        type: integer
    - add_entries:
        entries:
          - key: category_uid
            value: 4
```

```
          - key: category_name
            value: Network Activity
          - key: class_uid
            value: 4001
          - key: class_name
            value: Network Activity
          - key: metadata/product/name
            value: Palo Alto Networks Next-Generation Firewall
          - key: metadata/product/vendor_name
            value: Palo Alto Networks
          - key: metadata/profiles
            value:
              - security_control
              - network_proxy
              - host
              - datetime
          - key: metadata/version
            value: 1.4.0
          - key: severity_id
            value: 1
          - key: severity
            value: Informational
          - key: device/type_id
            value: 9
          - key: connection_info/direction_id
            value: 1
          - key: observables_0/name
            value: src_endpoint.ip
          - key: observables_0/type
            value: IP Address
          - key: observables_0/type_id
            value: '2'
          - key: observables_0/value
            format: '${Source_Address}'
          - key: observables_1/name
            value: dst_endpoint.ip
          - key: observables_1/type
            value: IP Address
          - key: observables_1/type_id
            value: '2'
          - key: observables_1/value
            format: '${Destination_Address}'
          - key: observables_2/name
            value: firewall_rule.uid
```

```
        - key: observables_2/type
          value: Resource UID
        - key: observables_2/type_id
          value: '10'
        - key: observables_2/value
          format: '${Rule_UUID}'
  - convert_type:
      keys:
        - observables_0/type_id
        - observables_1/type_id
        - observables_2/type_id
      type: integer
  - add_entries:
      entries:
        - key: observables
          value: []
        - key: observables
          value_expression: /observables_0
          append_if_key_exists: true
        - key: observables
          value_expression: /observables_1
          append_if_key_exists: true
        - key: observables
          value_expression: /observables_2
          append_if_key_exists: true
  - rename_keys:
      entries:
        - from_key: Source_Address
          to_key: src_endpoint/ip
          overwrite_if_to_key_exists: true
        - from_key: Source_Port
          to_key: src_endpoint/port
          overwrite_if_to_key_exists: true
        - from_key: Virtual_System
          to_key: src_endpoint/instance_uid
          overwrite_if_to_key_exists: true
        - from_key: Virtual_System_Name
          to_key: src_endpoint/name
          overwrite_if_to_key_exists: true
        - from_key: NAT_Source_IP
          to_key: src_endpoint/proxy_endpoint/ip
          overwrite_if_to_key_exists: true
        - from_key: NAT_Source_Port
          to_key: src_endpoint/proxy_endpoint/port
```

```
            overwrite_if_to_key_exists: true
          - from_key: Source_Zone
            to_key: src_endpoint/zone
            overwrite_if_to_key_exists: true
          - from_key: Inbound_Interface
            to_key: src_endpoint/interface_uid
            overwrite_if_to_key_exists: true
          - from_key: Source_Location
            to_key: src_endpoint/location/country
            overwrite_if_to_key_exists: true
          - from_key: Source_Device_Category
            to_key: src_endpoint/type
            overwrite_if_to_key_exists: true
          - from_key: Source_MAC_Address
            to_key: src_endpoint/mac
            overwrite_if_to_key_exists: true
          - from_key: Source_Hostname
            to_key: src_endpoint/hostname
            overwrite_if_to_key_exists: true
          - from_key: Source_Device_OS_Version
            to_key: src_endpoint/os/version
            overwrite_if_to_key_exists: true
          - from_key: Source_Device_OS_Family
            to_key: src_endpoint/os/type
            overwrite_if_to_key_exists: true
          - from_key: Source_Device_Model
            to_key: src_endpoint/device_hw_info/cpu_type
            overwrite_if_to_key_exists: true
          - from_key: Source_Device_Profile
            to_key: unmapped/Source_Device_Profile
            overwrite_if_to_key_exists: true
          - from_key: Source_Device_Vendor
            to_key: src_endpoint/device_hw_info/bios_manufacturer
            overwrite_if_to_key_exists: true
          - from_key: Destination_Device_Category
            to_key: dst_endpoint/type
            overwrite_if_to_key_exists: true
          - from_key: Destination_MAC_Address
            to_key: dst_endpoint/mac
            overwrite_if_to_key_exists: true
          - from_key: Destination_Hostname
            to_key: dst_endpoint/hostname
            overwrite_if_to_key_exists: true
          - from_key: Destination_Device_OS_Version
```

```
            to_key: dst_endpoint/os/version
            overwrite_if_to_key_exists: true
          - from_key: Destination_Device_OS_Family
            to_key: dst_endpoint/os/type
            overwrite_if_to_key_exists: true
          - from_key: Destination_Device_Model
            to_key: dst_endpoint/device_hw_info/cpu_type
            overwrite_if_to_key_exists: true
          - from_key: Destination_Device_Vendor
            to_key: dst_endpoint/device_hw_info/bios_manufacturer
            overwrite_if_to_key_exists: true
          - from_key: Destination_Device_Profile
            to_key: unmapped/Destination_Device_Profile
            overwrite_if_to_key_exists: true
          - from_key: Destination_Location
            to_key: dst_endpoint/location/country
            overwrite_if_to_key_exists: true
          - from_key: Destination_Zone
            to_key: dst_endpoint/zone
            overwrite_if_to_key_exists: true
          - from_key: Destination_Address
            to_key: dst_endpoint/ip
            overwrite_if_to_key_exists: true
          - from_key: Destination_Port
            to_key: dst_endpoint/port
            overwrite_if_to_key_exists: true
          - from_key: NAT_Destination_IP
            to_key: dst_endpoint/proxy_endpoint/ip
            overwrite_if_to_key_exists: true
          - from_key: NAT_Destination_Port
            to_key: dst_endpoint/proxy_endpoint/port
            overwrite_if_to_key_exists: true
          - from_key: Outbound_Interface
            to_key: dst_endpoint/interface_uid
            overwrite_if_to_key_exists: true
          - from_key: XFF_Address
            to_key: proxy_endpoint/ip
            overwrite_if_to_key_exists: true
          - from_key: Application_Subcategory
            to_key: unmapped/Application_Risk
            overwrite_if_to_key_exists: true
          - from_key: Application_Category
            to_key: unmapped/Application_Category
            overwrite_if_to_key_exists: true
```

```
          - from_key: Application_Technology
            to_key: unmapped/Application_Technology
            overwrite_if_to_key_exists: true
          - from_key: Application_Risk
            to_key: unmapped/Application_Risk
            overwrite_if_to_key_exists: true
          - from_key: XFF_Address
            to_key: proxy_endpoint/ip
            overwrite_if_to_key_exists: true
          - from_key: Session_ID
            to_key: connection_info/session/uid
            overwrite_if_to_key_exists: true
          - from_key: Repeat_Count
            to_key: connection_info/session/count
            overwrite_if_to_key_exists: true
          - from_key: Flags
            to_key: connection_info/tcp_flags
            overwrite_if_to_key_exists: true
          - from_key: Protocol
            to_key: connection_info/protocol_name
            overwrite_if_to_key_exists: true
          - from_key: Serial_Number
            to_key: device/hw_info/serial_number
            overwrite_if_to_key_exists: true
          - from_key: Device_Name
            to_key: device/hostname
            overwrite_if_to_key_exists: true
          - from_key: Host_ID
            to_key: device/uid
            overwrite_if_to_key_exists: true
          - from_key: Container_ID
            to_key: device/container/uid
            overwrite_if_to_key_exists: true
          - from_key: POD_Namespace
            to_key: unmapped/POD_Namespace
            overwrite_if_to_key_exists: true
          - from_key: POD_Name
            to_key: device/container/pod_uuid
            overwrite_if_to_key_exists: true
          - from_key: HTTP2_Connection
            to_key: unmapped/HTTP2_Connection
            overwrite_if_to_key_exists: true
          - from_key: Parent_Session_ID
            to_key: unmapped/Parent_Session_ID
```

```
            overwrite_if_to_key_exists: true
          - from_key: Source_VM_UUID
            to_key: unmapped/Source_VM_UUID
            overwrite_if_to_key_exists: true
          - from_key: Destination_VM_UUID
            to_key: unmapped/Source_VM_UUID
            overwrite_if_to_key_exists: true
          - from_key: Device_Group_Hierarchy_Level_1
            to_key: unmapped/Device_Group_Hierarchy_Level_1
            overwrite_if_to_key_exists: true
          - from_key: Device_Group_Hierarchy_Level_2
            to_key: unmapped/Device_Group_Hierarchy_Level_2
            overwrite_if_to_key_exists: true
          - from_key: Device_Group_Hierarchy_Level_3
            to_key: unmapped/Device_Group_Hierarchy_Level_3
            overwrite_if_to_key_exists: true
          - from_key: Device_Group_Hierarchy_Level_4
            to_key: unmapped/Device_Group_Hierarchy_Level_4
            overwrite_if_to_key_exists: true
          - from_key: High_Resolution_Timestamp
            to_key: unmapped/High_Resolution_Timestamp
            overwrite_if_to_key_exists: true
          - from_key: Log_Action
            to_key: unmapped/Log_Action
            overwrite_if_to_key_exists: true
          - from_key: Action_Flags
            to_key: unmapped/Action_Flags
            overwrite_if_to_key_exists: true
          - from_key: Tunnel_ID_IMSI
            to_key: unmapped/Tunnel_ID_IMSI
            overwrite_if_to_key_exists: true
          - from_key: Monitor_Tag_IMEI
            to_key: unmapped/Monitor_Tag_IMEI
            overwrite_if_to_key_exists: true
          - from_key: Tunnel_Type
            to_key: unmapped/Tunnel_Type
            overwrite_if_to_key_exists: true
          - from_key: SCTP_Association_ID
            to_key: unmapped/SCTP_Association_ID
            overwrite_if_to_key_exists: true
          - from_key: App_Flap_Count
            to_key: unmapped/App_Flap_Count
            overwrite_if_to_key_exists: true
          - from_key: Policy_ID
```

```
            to_key: unmapped/Policy_ID
            overwrite_if_to_key_exists: true
          - from_key: SD_WAN_Cluster
            to_key: unmapped/SD_WAN_Cluster
            overwrite_if_to_key_exists: true
          - from_key: SD_WAN_Device_Type
            to_key: unmapped/SD_WAN_Device_Type
            overwrite_if_to_key_exists: true
          - from_key: SD_WAN_Cluster_Type
            to_key: unmapped/SD_WAN_Cluster_Type
            overwrite_if_to_key_exists: true
          - from_key: SD_WAN_Site
            to_key: unmapped/SD_WAN_Site
            overwrite_if_to_key_exists: true
          - from_key: Link_Switches
            to_key: unmapped/Link_Switches
            overwrite_if_to_key_exists: true
          - from_key: A_Slice_Service_Type
            to_key: unmapped/A_Slice_Service_Type
            overwrite_if_to_key_exists: true
          - from_key: A_Slice_Differentiator
            to_key: unmapped/A_Slice_Differentiator
            overwrite_if_to_key_exists: true
          - from_key: Source_External_Dynamic_List
            to_key: unmapped/Source_External_Dynamic_List
            overwrite_if_to_key_exists: true
          - from_key: Destination_External_Dynamic_List
            to_key: unmapped/Destination_External_Dynamic_List
            overwrite_if_to_key_exists: true
          - from_key: Source_Dynamic_Address_Group
            to_key: unmapped/Source_Dynamic_Address_Group
            overwrite_if_to_key_exists: true
          - from_key: Destination_Dynamic_Address_Group
            to_key: unmapped/Destination_Dynamic_Address_Group
            overwrite_if_to_key_exists: true
          - from_key: Application_Characteristic
            to_key: unmapped/Application_Characteristic
            overwrite_if_to_key_exists: true
          - from_key: Application_Container
            to_key: unmapped/Application_Container
            overwrite_if_to_key_exists: true
          - from_key: Tunneled_Application
            to_key: unmapped/Tunneled_Application
            overwrite_if_to_key_exists: true
```

```
          - from_key: Application_SaaS
            to_key: unmapped/Application_SaaS
            overwrite_if_to_key_exists: true
          - from_key: Application_Sanctioned_State
            to_key: unmapped/Application_Sanctioned_State
            overwrite_if_to_key_exists: true
          - from_key: Offloaded
            to_key: unmapped/Offloaded
            overwrite_if_to_key_exists: true
          - from_key: Session_Owner
            to_key: unmapped/Session_Owner
            overwrite_if_to_key_exists: true
          - from_key: Packets_Sent
            to_key: traffic/packets_out
            overwrite_if_to_key_exists: true
          - from_key: Packets_Received
            to_key: traffic/packets_in
            overwrite_if_to_key_exists: true
          - from_key: Packets
            to_key: traffic/packets
            overwrite_if_to_key_exists: true
          - from_key: Bytes_Sent
            to_key: traffic/bytes_out
            overwrite_if_to_key_exists: true
          - from_key: Bytes_Received
            to_key: traffic/bytes_in
            overwrite_if_to_key_exists: true
          - from_key: Bytes
            to_key: traffic/bytes
            overwrite_if_to_key_exists: true
          - from_key: SCTP_Chunks_Sent
            to_key: traffic/chunks_out
            overwrite_if_to_key_exists: true
          - from_key: SCTP_Chunks_Received
            to_key: traffic/chunks_in
            overwrite_if_to_key_exists: true
          - from_key: SCTP_Chunks
            to_key: traffic/chunks
            overwrite_if_to_key_exists: true
          - from_key: Application
            to_key: unmapped/Application
            overwrite_if_to_key_exists: true
          - from_key: Source_User
            to_key: actor/user/name
```

```
            overwrite_if_to_key_exists: true
          - from_key: Destination_User
            to_key: actor/invoked_by
            overwrite_if_to_key_exists: true
          - from_key: Dynamic_User_Group_Name
            to_key: unmapped/Dynamic_User_Group_Name
            overwrite_if_to_key_exists: true
          - from_key: Rule_Name
            to_key: firewall_rule/name
            overwrite_if_to_key_exists: true
          - from_key: Rule_UUID
            to_key: firewall_rule/uid
            overwrite_if_to_key_exists: true
          - from_key: Session_End_Reason
            to_key: firewall_rule/condition
            overwrite_if_to_key_exists: true
          - from_key: Action_Source
            to_key: firewall_rule/match_location
            overwrite_if_to_key_exists: true
          - from_key: Category
            to_key: unmapped/Category
            overwrite_if_to_key_exists: true
          - from_key: Type
            to_key: metadata/product/feature/name
            overwrite_if_to_key_exists: true
          - from_key: Threat_Content_Type
            to_key: metadata/log_name
            overwrite_if_to_key_exists: true
          - from_key: Sequence_Number
            to_key: metadata/log_version
            overwrite_if_to_key_exists: true
          - from_key: Elapsed_Time
            to_key: unmapped/Elapsed_Time
            overwrite_if_to_key_exists: true
          - from_key: Parent_Start_Time
            to_key: unmapped/Parent_Start_Time
            overwrite_if_to_key_exists: true
    - translate:
        mappings:
          - source: Action
            targets:
              - target: activity_id
                default: 99
                map:
```

```
                allow: 1
                deny: 5
                drop: 2
                drop ICMP: 2
                reset both: 3
                reset client: 3
                reset server: 3
            - target: activity_name
              default: Other
              map:
                allow: Open
                deny: Refuse
                drop: Close
                drop ICMP: Close
                reset both: Reset
                reset client: Reset
                reset server: Reset
            - target: action_id
              default: 0
              map:
                allow: 1
                deny: 2
                drop: 99
                drop ICMP: 99
                reset both: 99
                reset client: 99
                reset server: 99
            - target: action
              default: Other
              map:
                allow: Allowed
                deny: Denied
                drop: Other
                drop ICMP: Other
                reset both: Other
                reset client: Other
                reset server: Other
            - target: type_uid
              default: 400199
              map:
                allow: 400101
                deny: 400105
                drop: 400102
                drop ICMP: 400102
```

```
                reset both: 400103
                reset client: 400103
                reset server: 400103
            - target: type_name
              default: 'Network Activity: Unknown'
              map:
                allow: 'Network Activity: Open'
                deny: 'Network Activity: Refuse'
                drop: 'Network Activity: Close'
                drop ICMP: 'Network Activity: Close'
                reset both: 'Network Activity: Reset'
                reset client: 'Network Activity: Reset'
                reset server: 'Network Activity: Reset'
            - target: status_id
              default: 0
              map:
                allow: 1
                deny: 2
                drop: 99
                drop ICMP: 99
                reset both: 99
                reset client: 99
                reset server: 99
            - target: status
              default: 0
              map:
                allow: Success
                deny: Failure
                drop: Other
                drop ICMP: Other
                reset both: Other
                reset client: Other
                reset server: Other
    - rename_keys:
        entries:
          - from_key: Action
            to_key: unmapped/Action
            overwrite_if_to_key_exists: true
    - convert_type:
        keys:
          - status_id
          - action_id
          - type_uid
          - activity_id
```

```
          - metadata/logged_time
          - connection_info/direction_id
          - connection_info/session/count
          - connection_info/tcp_flags
          - dst_endpoint/port
          - dst_endpoint/proxy_endpoint/port
          - src_endpoint/port
          - src_endpoint/proxy_endpoint/port
          - traffic/bytes
          - traffic/bytes_in
          - traffic/bytes_out
          - traffic/packets
          - traffic/packets_in
          - traffic/packets_out
          - traffic/chunks
          - traffic/chunks_in
          - traffic/chunks_out
        type: integer
    - convert_type:
        keys:
          - metadata/log_version
          - metadata/logged_time
          - connection_info/uid
          - connection_info/session/uid
          - connection_info/uid
          - connection_info/session/uid
        type: string
    - delete_entries:
        with_keys:
          - s3
          - message
          - Generated_Time
          - Start_Time
          - Receive_Time
          - FUTURE_USE_1
          - FUTURE_USE_2
          - FUTURE_USE_3
          - FUTURE_USE_4
          - FUTURE_USE_5
          - observables
          - observables_0
          - observables_1
          - observables_2
```

```
  sink:
    - s3:
        aws:

          sts_role_arn: "<<arn:aws:iam::123456789012:role/Example-Role>>"

          region: "<<us-east-1>>"

          bucket: "<<your-security-lake-bucket-name>>"
          object_key:
            path_prefix: "ext/<<CustomSourceName>>/1.0/region=<<region>>/
accountId=<<acount-id>>/eventDay=%{yyyyMMdd}/"
          object_metadata:
            number_of_events_key: asl_rows

          threshold:

            event_collect_timeout: 60s

            event_count: 10
          codec:
            parquet:
              auto_schema: true
```

Using an OpenSearch Ingestion pipeline with Fluent Bit

This sample Fluent Bit configuration file sends log data from Fluent Bit to an OpenSearch Ingestion pipeline. For more information about ingesting log data, see Log Analytics in the Data Prepper documentation.

Note the following:

- The `host` value must be your pipeline endpoint. For example, *`pipeline-endpoint`*`.us-east-1.osis.amazonaws.com`.
- The `aws_service` value must be `osis`.
- The `aws_role_arn` value is the ARN of the AWS IAM role for the client to assume and use for Signature Version 4 authentication.

```
[INPUT]
  name                  tail
```

```
  refresh_interval      5
  path                  /var/log/test.log
  read_from_head        true

[OUTPUT]
  Name http
  Match *
  Host pipeline-endpoint.us-east-1.osis.amazonaws.com
  Port 443
  URI /log/ingest
  Format json
  aws_auth true
  aws_region us-east-1
  aws_service osis
  aws_role_arn arn:aws:iam::{account-id}:role/ingestion-role
  Log_Level trace
  tls On
```

You can then configure an OpenSearch Ingestion pipeline like the following, which has HTTP as the source:

```
version: "2"
unaggregated-log-pipeline:
  source:
    http:
      path: "/log/ingest"
  processor:
    - grok:
        match:
          log:
            - "%{TIMESTAMP_ISO8601:timestamp} %{NOTSPACE:network_node}
 %{NOTSPACE:network_host} %{IPORHOST:source_ip}:%{NUMBER:source_port:int} ->
 %{IPORHOST:destination_ip}:%{NUMBER:destination_port:int} %{GREEDYDATA:details}"
    - grok:
        match:
          details:
            - "'%{NOTSPACE:http_method} %{NOTSPACE:http_uri}' %{NOTSPACE:protocol}"
            - "TLS%{NOTSPACE:tls_version} %{GREEDYDATA:encryption}"
            - "%{NUMBER:status_code:int} %{NUMBER:response_size:int}"
    - delete_entries:
        with_keys: ["details", "log"]

  sink:
```

```
    - opensearch:
        hosts: ["https://search-domain-endpoint.us-east-1.es.amazonaws.com"]
        index: "index_name"
        index_type: custom
        bulk_size: 20
        aws:
          # IAM role that the pipeline assumes to access the domain sink
          sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
          region: "us-east-1"
```

Using an OpenSearch Ingestion pipeline with Fluentd

Fluentd is an open-source data collection ecosystem that provides SDKs for different languages and sub-projects like Fluent Bit. This sample Fluentd configuration file sends log data from Fluentd to an OpenSearch Ingestion pipeline. For more information about ingesting log data, see Log Analytics in the Data Prepper documentation.

Note the following:

- The `endpoint` value must be your pipeline endpoint. For example, *`pipeline-endpoint`*`.us-east-1.osis.amazonaws.com/apache-log-pipeline/logs`.
- The `aws_service` value must be `osis`.
- The `aws_role_arn` value is the ARN of the AWS IAM role for the client to assume and use for Signature Version 4 authentication.

```
<source>
  @type tail
  path logs/sample.log
  path_key log
  tag apache
  <parse>
    @type none
  </parse>
</source>

<filter apache>
  @type record_transformer
  <record>
    log ${record["message"]}
  </record>
```

```
</filter>

<filter apache>
  @type record_transformer
  remove_keys message
</filter>

<match apache>
  @type http
  endpoint pipeline-endpoint.us-east-1.osis.amazonaws.com/apache-log-pipeline/logs
  json_array true

  <auth>
    method aws_sigv4
    aws_service osis
    aws_region us-east-1
    aws_role_arn arn:aws:iam::{account-id}:role/ingestion-role
  </auth>

  <format>
    @type json
  </format>

  <buffer>
    flush_interval 1s
  </buffer>
</match>
```

You can then configure an OpenSearch Ingestion pipeline like the following, which has HTTP as the source:

```
version: "2"
apache-log-pipeline:
  source:
    http:
      path: "/${pipelineName}/logs"
  processor:
    - grok:
        match:
          log:
            - "%{TIMESTAMP_ISO8601:timestamp} %{NOTSPACE:network_node}
 %{NOTSPACE:network_host} %{IPORHOST:source_ip}:%{NUMBER:source_port:int} ->
 %{IPORHOST:destination_ip}:%{NUMBER:destination_port:int} %{GREEDYDATA:details}"
```

```
  sink:
    - opensearch:
        hosts: ["https://search-domain-endpoint.us-east-1.es.amazonaws.com"]
        index: "index_name"
        aws_sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
        aws_region: "us-east-1"
        aws_sigv4: true
```

Using an OpenSearch Ingestion pipeline with OpenTelemetry Collector

This sample OpenTelemetry configuration file exports trace data from the OpenTelemetry Collector and sends it to an OpenSearch Ingestion pipeline. For more information about ingesting trace data, see Trace Analytics in the Data Prepper documentation.

Note the following:

- The `endpoint` value must include your pipeline endpoint. For example, `https://pipeline-endpoint.us-east-1.osis.amazonaws.com`.
- The `service` value must be `osis`.
- The `compression` option for the OTLP/HTTP Exporter must match the `compression` option on the pipeline's OpenTelemetry source.

```
extensions:
  sigv4auth:
    region: "us-east-1"
    service: "osis"

receivers:
  jaeger:
    protocols:
      grpc:

exporters:
  otlphttp:
    traces_endpoint: "https://pipeline-endpoint.us-east-1.osis.amazonaws.com/v1/traces"
    auth:
      authenticator: sigv4auth
    compression: none

service:
```

```
  extensions: [sigv4auth]
  pipelines:
    traces:
      receivers: [jaeger]
      exporters: [otlphttp]
```

You can then configure an OpenSearch Ingestion pipeline like the following, which specifies the OTel trace plugin as the source:

```
version: "2"
otel-trace-pipeline:
  source:
    otel_trace_source:
      path: "/v1/traces"
  processor:
    - trace_peer_forwarder:
  sink:
    - pipeline:
        name: "trace-pipeline"
    - pipeline:
        name: "service-map-pipeline"
trace-pipeline:
  source:
    pipeline:
      name: "otel-trace-pipeline"
  processor:
    - otel_traces:
  sink:
    - opensearch:
        hosts: ["https://search-domain-endpoint.us-east-1.es.amazonaws.com"]
        index_type: trace-analytics-raw
        aws:
          # IAM role that OpenSearch Ingestion assumes to access the domain sink
          sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
          region: "us-east-1"

service-map-pipeline:
  source:
    pipeline:
      name: "otel-trace-pipeline"
  processor:
    - service_map:
  sink:
```

```
    - opensearch:
        hosts: ["https://search-domain-endpoint.us-east-1.es.amazonaws.com"]
        index_type: trace-analytics-service-map
        aws:
          # IAM role that the pipeline assumes to access the domain sink
          sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
          region: "us-east-1"
```

For another example pipeline, see the preconfigured trace analytics blueprint. For more information, see the section called “Using blueprints to create a pipeline”.

Using an OpenSearch Ingestion pipeline with Kafka

You can use an OpenSearch Ingestion pipeline with self-managed Kafka to stream data to Amazon OpenSearch Service domains and OpenSearch Serverless collections. OpenSearch Ingestion supports both public and private network configurations for the streaming of data from self-managed Kafka to domains or collections managed by OpenSearch Service or OpenSearch Serverless.

Connectivity to public Kafka clusters

You can use OpenSearch Ingestion pipelines to migrate data from a self-managed Kafka cluster with a public configuration, which means that the domain DNS name can be publicly resolved. To do so, set up an OpenSearch Ingestion pipeline with self-managed Kafka as the source and OpenSearch Service or OpenSearch Serverless as the destination. This processes your streaming data from a self-managed source cluster to an AWS-managed destination domain or collection.

Prerequisites

Before you create your OpenSearch Ingestion pipeline, perform the following steps:

1. Create a self-managed Kafka cluster with a public network configuration. The cluster should contain the data you want to ingest into OpenSearch Service.
2. Create an OpenSearch Service domain or OpenSearch Serverless collection where you want to migrate data to. For more information, see Creating OpenSearch Service domains and Creating collections.
3. Set up authentication on your self-managed cluster with AWS Secrets Manager. Enable secrets rotation by following the steps in Rotate AWS Secrets Manager secrets.

4. Attach a resource-based policy to your domain or a data access policy to your collection. These access policies allow OpenSearch Ingestion to write data from your self-managed cluster to your domain or collection.

   The following sample domain access policy allows the pipeline role, which you create in the next step, to write data to a domain. Make sure that you update the `resource` with your own ARN.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{pipeline-account-id}:role/pipeline-role"
      },
      "Action": [
        "es:DescribeDomain",
        "es:ESHttp*"
      ],
      "Resource": [
        "arn:aws:es:{region}:{account-id}:domain/domain-name"
      ]
    }
  ]
}
```

   To create an IAM role with the correct permissions to access write data to the collection or domain, see Required permissions for domains and Required permissions for collections.

Step 1: Configure the pipeline role

After you have your Kafka pipeline prerequisites set up, configure the pipeline role that you want to use in your pipeline configuration, and add permission to write to an OpenSearch Service domain or OpenSearch Serverless collection, as well as permission to read secrets from Secrets Manager.

Step 2: Create the pipeline

You can then configure an OpenSearch Ingestion pipeline like the following, which specifies Kafka as the source.

You can specify multiple OpenSearch Service domains as destinations for your data. This capability enables conditional routing or replication of incoming data into multiple OpenSearch Service domains.

You can also migrate data from a source Confluent Kafka cluster to an OpenSearch Serverless VPC collection. Ensure you provide a network access policy within the pipeline configuration. You can use a Confluent schema registry to define a Confluent schema.

```
version: "2"
kafka-pipeline:
  source:
    kafka:
      encryption:
        type: "ssl"
      topics:
        - name: "topic-name"
          group_id: "group-id"
      bootstrap_servers:
        - "bootstrap-server.us-west-2.aws.private.confluent.cloud:9092"
      authentication:
        sasl:
          plain:
            username: ${aws_secrets:confluent-kafka-secret:username}
            password: ${aws_secrets:confluent-kafka-secret:password}
      schema:
        type: confluent
        registry_url: https://my-registry.us-west-2.aws.confluent.cloud
        api_key: "${{aws_secrets:schema-secret:schema_registry_api_key}}"
        api_secret: "${{aws_secrets:schema-secret:schema_registry_api_secret}}"
        basic_auth_credentials_source: "USER_INFO"
  sink:
  - opensearch:
      hosts: ["https://search-mydomain.us-west-2.es.amazonaws.com"]
      aws:
          sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
          region: "us-west-2"
      index: "confluent-index"
extension:
  aws:
    secrets:
      confluent-kafka-secret:
        secret_id: "my-kafka-secret"
        region: "us-west-2"
```

```
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
      schema-secret:
        secret_id: "my-self-managed-kafka-schema"
        region: "us-west-2"
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
```

You can use a preconfigured blueprint to create this pipeline. For more information, see the section called “Using blueprints to create a pipeline”.

Connectivity to Kafka clusters in a VPC

You can also use OpenSearch Ingestion pipelines to migrate data from a self-managed Kafka cluster running in a VPC. To do so, set up an OpenSearch Ingestion pipeline with self-managed Kafka as the source and OpenSearch Service or OpenSearch Serverless as the destination. This processes your streaming data from a self-managed source cluster to an AWS-managed destination domain or collection.

Prerequisites

Before you create your OpenSearch Ingestion pipeline, perform the following steps:

1. Create a self-managed Kafka cluster with a VPC network configuration that contains the data you want to ingest into OpenSearch Service.
2. Create an OpenSearch Service domain or OpenSearch Serverless collection where you want to migrate data to. For more information, see Creating OpenSearch Service domains and Creating collections.
3. Set up authentication on your self-managed cluster with AWS Secrets Manager. Enable secrets rotation by following the steps in Rotate AWS Secrets Manager secrets.
4. Obtain the ID of the VPC that that has access to self-managed Kafka. Choose the VPC CIDR to be used by OpenSearch Ingestion.

> **Note**
>
> If you're using the AWS Management Console to create your pipeline, you must also attach your OpenSearch Ingestion pipeline to your VPC in order to use self-managed Kafka. To do so, find the **Network configuration** section, select the **Attach to VPC** checkbox, and choose your CIDR from one of the provided default options, or select your own. You can use any CIDR from a private address space as defined in the RFC 1918 Best Current Practice.

To provide a custom CIDR, select **Other** from the dropdown menu. To avoid a collision in IP addresses between OpenSearch Ingestion and self-managed OpenSearch, ensure that the self-managed OpenSearch VPC CIDR is different from the CIDR for OpenSearch Ingestion.

5. Attach a resource-based policy to your domain or a data access policy to your collection. These access policies allow OpenSearch Ingestion to write data from your self-managed cluster to your domain or collection.

   The following sample domain access policy allows the pipeline role, which you create in the next step, to write data to a domain. Make sure that you update the `resource` with your own ARN.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{pipeline-account-id}:role/pipeline-role"
      },
      "Action": [
        "es:DescribeDomain",
        "es:ESHttp*"
      ],
      "Resource": [
        "arn:aws:es:{region}:{account-id}:domain/domain-name"
      ]
    }
  ]
}
```

   To create an IAM role with the correct permissions to access write data to the collection or domain, see Required permissions for domains and Required permissions for collections.

Step 1: Configure the pipeline role

After you have your pipeline prerequisites set up, configure the pipeline role that you want to use in your pipeline configuration, and add the following permissions in the role:

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerReadAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": ["arn:aws:secretsmanager:{region}:{account-id}:secret:secret-
name"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AttachNetworkInterface",
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DetachNetworkInterface",
                "ec2:DescribeNetworkInterfaces"
            ],
            "Resource": [
                "arn:aws:ec2:*:{account-id}:network-interface/*",
                "arn:aws:ec2:*:{account-id}:subnet/*",
                "arn:aws:ec2:*:{account-id}:security-group/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:Describe*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
```

```
            ],
            "Resource": "arn:aws:ec2:*:*:network-interface/*",
            "Condition": {
               "StringEquals":
                    {
                        "aws:RequestTag/OSISManaged": "true"
                    }
            }
        }
    ]
}
```

You must provide the above Amazon EC2 permissions on the IAM role that you use to create the OpenSearch Ingestion pipeline because the pipeline uses these permissions to create and delete a network interface in your VPC. The pipeline can only access the Kafka cluster through this network interface.

Step 2: Create the pipeline

You can then configure an OpenSearch Ingestion pipeline like the following, which specifies Kafka as the source.

You can specify multiple OpenSearch Service domains as destinations for your data. This capability enables conditional routing or replication of incoming data into multiple OpenSearch Service domains.

You can also migrate data from a source Confluent Kafka cluster to an OpenSearch Serverless VPC collection. Ensure you provide a network access policy within the pipeline configuration. You can use a Confluent schema registry to define a Confluent schema.

```
 version: "2"
kafka-pipeline:
  source:
    kafka:
      encryption:
        type: "ssl"
      topics:
        - name: "topic-name"
          group_id: "group-id"
      bootstrap_servers:
        - "bootstrap-server.us-west-2.aws.private.confluent.cloud:9092"
      authentication:
```

```
        sasl:
          plain:
            username: ${aws_secrets:confluent-kafka-secret:username}
            password: ${aws_secrets:confluent-kafka-secret:password}
      schema:
        type: confluent
        registry_url: https://my-registry.us-west-2.aws.confluent.cloud
        api_key: "${{aws_secrets:schema-secret:schema_registry_api_key}}"
        api_secret: "${{aws_secrets:schema-secret:schema_registry_api_secret}}"
        basic_auth_credentials_source: "USER_INFO"
  sink:
  - opensearch:
      hosts: ["https://search-mydomain.us-west-2.es.amazonaws.com"]
      aws:
          sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
          region: "us-west-2"
      index: "confluent-index"
extension:
  aws:
    secrets:
      confluent-kafka-secret:
        secret_id: "my-kafka-secret"
        region: "us-west-2"
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
      schema-secret:
        secret_id: "my-self-managed-kafka-schema"
        region: "us-west-2"
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
```

You can use a preconfigured blueprint to create this pipeline. For more information, see the section called “Using blueprints to create a pipeline”.

Using an OpenSearch Ingestion pipeline with OpenSearch

You can use an OpenSearch Ingestion pipeline with self-managed OpenSearch or Elasticsearch to migrate data to Amazon OpenSearch Service domains and OpenSearch Serverless collections. OpenSearch Ingestion supports both public and private network configurations for the migration of data from self-managed OpenSearch and Elasticsearch.

Connectivity to public OpenSearch clusters

You can use OpenSearch Ingestion pipelines to migrate data from a self-managed OpenSearch or Elasticsearch cluster with a public configuration, which means that the domain DNS name

can be publicly resolved. To do so, set up an OpenSearch Ingestion pipeline with self-managed OpenSearch or Elasticsearch as the source and OpenSearch Service or OpenSearch Serverless as the destination. This effectively migrates your data from a self-managed source cluster to an AWS-managed destination domain or collection.

Prerequisites

Before you create your OpenSearch Ingestion pipeline, perform the following steps:

1. Create a self-managed OpenSearch or Elastisearch cluster that contains the data you want to migrate and configure a public DNS name.
2. Create an OpenSearch Service domain or OpenSearch Serverless collection where you want to migrate data to. For more information, see Creating OpenSearch Service domains and Creating collections.
3. Set up authentication on your self-managed cluster with AWS Secrets Manager. Enable secrets rotation by following the steps in Rotate AWS Secrets Manager secrets.
4. Attach a resource-based policy to your domain or a data access policy to your collection. These access policies allow OpenSearch Ingestion to write data from your self-managed cluster to your domain or collection.

   The following sample domain access policy allows the pipeline role, which you create in the next step, to write data to a domain. Make sure that you update the `resource` with your own ARN.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{pipeline-account-id}:role/pipeline-role"
      },
      "Action": [
        "es:DescribeDomain",
        "es:ESHttp*"
      ],
      "Resource": [
        "arn:aws:es:{region}:{account-id}:domain/domain-name"
      ]
    }
  ]
```

```
}
```

To create an IAM role with the correct permissions to access write data to the collection or domain, see Required permissions for domains and Required permissions for collections.

Step 1: Configure the pipeline role

After you have your OpenSearch pipeline prerequisites set up, configure the pipeline role that you want to use in your pipeline configuration, and add permission to write to an OpenSearch Service domain or OpenSearch Serverless collection, as well as permission to read secrets from Secrets Manager.

Step 2: Create the pipeline

You can then configure an OpenSearch Ingestion pipeline like the following, which specifies OpenSearch as the source.

You can specify multiple OpenSearch Service domains as destinations for your data. This capability enables conditional routing or replication of incoming data into multiple OpenSearch Service domains.

You can also migrate data from a source OpenSearch or Elasticsearch cluster to an OpenSearch Serverless VPC collection. Ensure you provide a network access policy within the pipeline configuration.

```
version: "2"
opensearch-migration-pipeline:
  source:
    opensearch:
      acknowledgments: true
      host: [ "https://my-self-managed-cluster-name:9200" ]
      indices:
        include:
          - index_name_regex: "include-.*"
        exclude:
          - index_name_regex: '\..*'
      authentication:
        username: ${aws_secrets:secret:username}
        password: ${aws_secrets:secret:password}
        scheduling:
           interval: "PT2H"
```

```
            index_read_count: 3
            start_time: "2023-06-02T22:01:30.00Z"
  sink:
  - opensearch:
      hosts: ["https://search-mydomain.us-east-1.es.amazonaws.com"]
      aws:
          sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
          region: "us-east-1"
          #Uncomment the following lines if your destination is an OpenSearch
 Serverless collection
          #serverless: true
          # serverless_options:
          #     network_policy_name: "network-policy-name"
      index: "${getMetadata(\"opensearch-index\")}"
      document_id: "${getMetadata(\"opensearch-document_id\")}"
      enable_request_compression: true
      dlq:
        s3:
          bucket: "bucket-name"
          key_path_prefix: "apache-log-pipeline/logs/dlq"
          region: "us-east-1"
          sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
extension:
  aws:
    secrets:
      secret:
        secret_id: "my-opensearch-secret"
        region: "us-east-1"
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
        refresh_interval: PT1H
```

You can use a preconfigured blueprint to create this pipeline. For more information, see the section called “Using blueprints to create a pipeline”.

Connectivity to OpenSearch clusters in a VPC

You can also use OpenSearch Ingestion pipelines to migrate data from a self-managed OpenSearch or Elasticsearch cluster running in a VPC. To do so, set up an OpenSearch Ingestion pipeline with self-managed OpenSearch or Elasticsearch as the source and OpenSearch Service or OpenSearch Serverless as the destination. This effectively migrates your data from a self-managed source cluster to an AWS-managed destination domain or collection.

Prerequisites

Before you create your OpenSearch Ingestion pipeline, perform the following steps:

1. Create a self-managed OpenSearch or Elastisearch cluster with a VPC network configuration that contains the data you want to migrate.
2. Create an OpenSearch Service domain or OpenSearch Serverless collection where you want to migrate data to. For more information, see Creating OpenSearch Service domains and Creating collections.
3. Set up authentication on your self-managed cluster with AWS Secrets Manager. Enable secrets rotation by following the steps in Rotate AWS Secrets Manager secrets.
4. Obtain the ID of the VPC that that has access to self-managed OpenSearch or Elasticsearch. Choose the VPC CIDR to be used by OpenSearch Ingestion.

   > **Note**
   >
   > If you're using the AWS Management Console to create your pipeline, you must also attach your OpenSearch Ingestion pipeline to your VPC in order to use self-managed OpenSearch or Elasticsearch. To do so, find the **Network configuration** section, select the **Attach to VPC** checkbox, and choose your CIDR from one of the provided default options, or select your own. You can use any CIDR from a private address space as defined in the RFC 1918 Best Current Practice.
   > To provide a custom CIDR, select **Other** from the dropdown menu. To avoid a collision in IP addresses between OpenSearch Ingestion and self-managed OpenSearch, ensure that the self-managed OpenSearch VPC CIDR is different from the CIDR for OpenSearch Ingestion.

5. Attach a resource-based policy to your domain or a data access policy to your collection. These access policies allow OpenSearch Ingestion to write data from your self-managed cluster to your domain or collection.

   The following sample domain access policy allows the pipeline role, which you create in the next step, to write data to a domain. Make sure that you update the `resource` with your own ARN.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{pipeline-account-id}:role/pipeline-role"
      },
      "Action": [
        "es:DescribeDomain",
        "es:ESHttp*"
      ],
      "Resource": [
        "arn:aws:es:{region}:{account-id}:domain/domain-name"
      ]
    }
  ]
}
```

To create an IAM role with the correct permissions to access write data to the collection or domain, see Required permissions for domains and Required permissions for collections.

Step 1: Configure the pipeline role

After you have your pipeline prerequisites set up, configure the pipeline role that you want to use in your pipeline configuration, and add the following permissions in the role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerReadAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": ["arn:aws:secretsmanager:{region}:{account-id}:secret:secret-
name"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AttachNetworkInterface",
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
```

```
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DetachNetworkInterface",
                "ec2:DescribeNetworkInterfaces"
            ],
            "Resource": [
                "arn:aws:ec2:*:{account-id}:network-interface/*",
                "arn:aws:ec2:*:{account-id}:subnet/*",
                "arn:aws:ec2:*:{account-id}:security-group/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:Describe*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": "arn:aws:ec2:*:*:network-interface/*",
            "Condition": {
               "StringEquals":
                    {
                        "aws:RequestTag/OSISManaged": "true"
                    }
            }
        }
    ]
}
```

You must provide the above Amazon EC2 permissions on the IAM role that you use to create the OpenSearch Ingestion pipeline because the pipeline uses these permissions to create and delete a network interface in your VPC. The pipeline can only access the OpenSearch cluster through this network interface.

Step 2: Create the pipeline

You can then configure an OpenSearch Ingestion pipeline like the following, which specifies OpenSearch as the source.

You can specify multiple OpenSearch Service domains as destinations for your data. This capability enables conditional routing or replication of incoming data into multiple OpenSearch Service domains.

You can also migrate data from a source OpenSearch or Elasticsearch cluster to an OpenSearch Serverless VPC collection. Ensure you provide a network access policy within the pipeline configuration.

```
version: "2"
opensearch-migration-pipeline:
  source:
    opensearch:
      acknowledgments: true
      host: [ "https://my-self-managed-cluster-name:9200" ]
      indices:
        include:
          - index_name_regex: "include-.*"
        exclude:
          - index_name_regex: '\..*'
      authentication:
        username: ${aws_secrets:secret:username}
        password: ${aws_secrets:secret:password}
        scheduling:
           interval: "PT2H"
           index_read_count: 3
           start_time: "2023-06-02T22:01:30.00Z"
  sink:
  - opensearch:
      hosts: ["https://search-mydomain.us-east-1.es.amazonaws.com"]
      aws:
          sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
          region: "us-east-1"
          #Uncomment the following lines if your destination is an OpenSearch
 Serverless collection
          #serverless: true
          # serverless_options:
          #     network_policy_name: "network-policy-name"
      index: "${getMetadata(\"opensearch-index\")}"
```

```
      document_id: "${getMetadata(\"opensearch-document_id\")}"
      enable_request_compression: true
      dlq:
        s3:
          bucket: "bucket-name"
          key_path_prefix: "apache-log-pipeline/logs/dlq"
          region: "us-east-1"
          sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
extension:
  aws:
    secrets:
      secret:
        secret_id: "my-opensearch-secret"
        region: "us-east-1"
        sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
        refresh_interval: PT1H
```

You can use a preconfigured blueprint to create this pipeline. For more information, see the section called “Using blueprints to create a pipeline”.

Using an OpenSearch Ingestion pipeline with Amazon Kinesis Data Streams

You can use an OpenSearch Ingestion pipeline to stream data from Amazon Kinesis Data Streams to Amazon OpenSearch Service domains and OpenSearch Serverless collections. OpenSearch Ingestion supports both public and private network configurations for the streaming of data from Amazon Kinesis Data Streams to domains or collections managed by OpenSearch Service or OpenSearch Serverless.

Connectivity to Amazon Kinesis Data Streams

You can use OpenSearch Ingestion pipelines to migrate data from Amazon Kinesis Data Streams with public configuration, which means that the domain DNS name can be publicly resolved. To do so, set up an OpenSearch Ingestion pipeline with Amazon Kinesis Data Streams as the source and OpenSearch Service or OpenSearch Serverless as the destination. This processes your streaming data from a self-managed source cluster to an AWS-managed destination domain or collection.

Prerequisites

Before you create your OpenSearch Ingestion pipeline, perform the following steps:

1. Create an Amazon Kinesis data stream acting as a source. The stream should contain the data you want to ingest into OpenSearch Service.
2. Create an OpenSearch Service domain or OpenSearch Serverless collection where you want to migrate data to. For more information, see Creating OpenSearch Service domains and Creating collections.
3. Set up authentication on your Amazon Kinesis data stream with AWS Secrets Manager. Enable secrets rotation by following the steps in Rotate AWS Secrets Manager secrets.
4. Attach a resource-based policy to your domain or a data access policy to your collection. These access policies allow OpenSearch Ingestion to write data from your self-managed cluster to your domain or collection.

   The following sample domain access policy allows the pipeline role, which you create in the next step, to write data to a domain. Make sure that you update the `resource` with your own ARN.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{pipeline-account-id}:role/pipeline-role"
      },
      "Action": [
        "es:DescribeDomain",
        "es:ESHttp*"
      ],
      "Resource": [
        "arn:aws:es:{region}:{account-id}:domain/domain-name"
      ]
    }
  ]
}
```

   To create an IAM role with the correct permissions to access write data to the collection or domain, see Required permissions for domains and Required permissions for collections.

Step 1: Configure the pipeline role

After you have your Amazon Kinesis Data Streams pipeline prerequisites set up, configure the pipeline role that you want to use in your pipeline configuration, and add permission to write to an OpenSearch Service domain or OpenSearch Serverless collection, as well as permission to read secrets from Secrets Manager.

The following permission is needed to write to an Amazon S3 bucket, domain, and collection:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "allowReadFromStream",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:DescribeStreamConsumer",
                "kinesis:DescribeStreamSummary",
                "kinesis:GetRecords",
                "kinesis:GetShardIterator",
                "kinesis:ListShards",
                "kinesis:ListStreams",
                "kinesis:ListStreamConsumers",
                "kinesis:RegisterStreamConsumer",
                "kinesis:SubscribeToShard"
            ],
            "Resource": [
                "arn:aws:kinesis:{{region}}:{{account-id}}:stream/{{stream-name}}"
            ]
        }
    ]
}
```

If server-side encryption is enabled for streams, the following KMS policy will decrypt the stream records:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "allowDecryptionOfCustomManagedKey",
```

```
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:{{region}}:{{account-id}}:key/{{key-id}}"
        }
    ]
}
```

In order for a pipeline to write data to a domain, the domain must have a domain-level access policy that allows the **sts_role_arn** pipeline role to access it. The following sample domain access policy allows the pipeline role named `pipeline-role` , which you created in the previous step, to write data to the domain named `ingestion-domain` :

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{your-account-id}:role/{pipeline-role}"
      },
``      "Action": ["es:DescribeDomain", "es:ESHttp*"],
      "Resource": "arn:aws:es:{region}:{your-account-id}:domain/{domain-name}/*"
    }
  ]
}
```

Step 2: Create the pipeline

You can then configure an OpenSearch Ingestion pipeline which specifying Amazon Kinesis as the source. The metadata attributes available are:

- `stream_name`: name of the Kinesis data stream that the record is being ingested from.
- `partition_key`: partition key of the Kinesis data stream record that is being ingested.
- `sequence_number`: sequence number of the Kinesis data stream record that is being ingested.
- `sub_sequence_number`: sub sequence number of the Kinesis data stream record that is being ingested.

You can specify multiple OpenSearch Service domains as destinations for your data. This capability enables conditional routing or replication of incoming data into multiple OpenSearch Service domains.

You can also migrate data from Amazon Kinesis to an OpenSearch Serverless VPC collection. There is a blueprint available on the OpenSearch Ingestion console for creating a pipeline. To create a pipeline, you can use the following `AWS-KinesisDataStreamsPipeline` blueprint.

```
version: "2"
     kinesis_data_streams_pipeline:
       source:
         kinesis_data_streams:
           acknowledgments: true
           codec:
             newline:
           streams:
             - stream_name: "<stream name>"
             - stream_name: "<stream name>"

           aws:
             sts_role_arn: "<<arn:aws:iam::123456789012:role/Example-Role>>"
             region: "<<us-east-1>>"

       sink:
         - opensearch:
             hosts: [ "<<https://search-mydomain-1a2a3a4a5a6a7a8a9a0a9a8a7a.us-
east-1.es.amazonaws.com>>" ]
             index: "index_${getMetadata(\"stream_name\")}"
             document_id: "${getMetadata(\"partition_key\")}"
             aws:
               sts_role_arn: "<<arn:aws:iam::123456789012:role/Example-Role>>"
               region: "<<us-east-1>>"

               s3:
                 bucket: "<<your-dlq-bucket-name>>"
                 region: "<<us-east-1>>"
                 sts_role_arn: "<<arn:aws:iam::123456789012:role/Example-Role>>"
```

You can use a preconfigured blueprint to create this pipeline. For more information, see the section called “Using blueprints to create a pipeline”. You can also review the opensource Opensearch documentation for additional confiugration options. To learn more, see configuration options.

Data consistency

OpenSearch Ingestion supports end-to-end acknowledgement to ensure data durability. When the pipeline reads stream records from Kinesis, it dynamically distributes the work of reading stream records based on the shards associated with the stream. Pipeline will automatically checkpoint streams when it receives an acknowledgement after ingesting all records in the OpenSearch domain or collection. This will avoid duplicate processing of stream records.

> **Note**
>
> If you want to create the index based on the stream name, you can define the index in the opensearch sink section as **"index_${getMetadata(\"*stream_name*\")}".**

(Optional) Configure recommended compute units (OCUs) for the Kinesis Data Streams pipeline

A minimum of 2 compute units (OCU) are recommended when creating a Kinesis source pipeline. This will allow for the Kinesis data stream records per shard processing to be distributed evenly among the compute units thereby ensuring a low-latency mechansim for stream records ingestion.

An OpenSearchKinesis data streams source pipeline can also be configured to ingest stream records from more than one stream. It is recommended to add an additional compute unit per new stream.

> **Note**
>
> If your pipeline has more compute units (OCU) than there are shards in the set of streams configured in the pipeline, some compute units could sit idle without processing any stream records per shard.

Next steps

After you export your data to a pipeline, you can query it from the OpenSearch Service domain that is configured as a sink for the pipeline. The following resources can help you get started:

- *Observability*
- the section called "Trace Analytics"
- the section called "Piped Processing Language"

Using an OpenSearch Ingestion pipeline with AWS Lambda

You can use an OpenSearch Ingestion pipeline with AWS Lambda for custom enrichment of your data from all sources and destinations supported by OpenSearch Ingestion. Lambda as a processor can enrich data using custom code and return the processed events back to your integration pipeline for further processing.

Prerequisites

Before you create your OpenSearch Ingestion pipeline, perform the following steps:

1. Create an AWS Lambda function. To do this, see AWS Lambda Documentation data stream acting as a source. The stream should contain the data you want to ingest into OpenSearch Service.
2. Create an OpenSearch Service domain or OpenSearch Serverless collection where you want to migrate data to. For more information, see Creating OpenSearch Service domains and Creating collections.
3. Set up the pipeline role with permissions to write to a destination domain or collection. To do this, see pipeline role.
4. In order for a pipeline to write data to a domain, the domain must have a domain-level access policy that allows the **sts_role_arn** pipeline role to access it. You can refer to the Granting Amazon OpenSearch Ingestion pipelines access to domains for granting domain permission. You can refer to the Granting Amazon OpenSearch Ingestion access to collections for granting collection permission.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "allowinvokeFunction",
            "Effect": "Allow",
            "Action": [
                "lambda:invokeFunction",
                "lambda:InvokeAsync",
                "lambda:ListFunctions"
            ],
            "Resource": "arn:aws:lambda:{{region}}:{{account-id}}:function:
{{function-name}}"
```

```
        }
    ]
}
```

To create an IAM role with the correct permissions to access write data to the collection or domain, see Required permissions for domains and Required permissions for collections.

> **Note**
>
> The payload size limit for a Lambda processor or sink for a single event is 5MB. Additionally, a Lambda processor supports responses only in a JSON array format.

Create a pipeline

To use AWS Lambda as a processor, you must first configure an OpenSearch Ingestion pipeline and specify AWS Lambda as a processor for your pipeline. You can also use a preconfigured Lambda processor blueprint to create this pipeline. For more information, see Using blueprints to create a pipeline. The following pipeline example receives data from a http source, enriches the data using a date processor and the AWS Lambda processor, and ingests the processed data to an OpenSearch Domain.

To set up AWS Lambda as processor, see the following example pipeline:

```
version: "2"
lambda-processor-pipeline:
  source:
    http:
      path: "/${pipelineName}/logs"

  processor:
      - date:
        destination: "@timestamp"
        from_time_received: true

    - aws_lambda:
        function_name: "my-lambda-function"

        tags_on_failure: ["lambda_failure"]
        batch:
            key_name: "events"
```

```
        aws:
          region: us-east-1
          sts_role_arn: "<<arn:aws:iam::123456789012:role/Example-Role>>"
  sink:
    - opensearch:
        hosts: [ "<<https://search-mydomain-1a2a3a4a5a6a7a8a9a0a9a8a7a.us-
east-1.es.amazonaws.com>>" ]
        index: "table-index"
        aws:
          sts_role_arn: "<<arn:aws:iam::123456789012:role/Example-Role>>"
          region: "<<us-east-1>>"
          serverless: false
```

The following example is an AWS Lambda function:

```
import json

def lambda_handler(event, context):
    input_array = event.get('events', [])
    output = []
    for input in input_arr:
        input["transformed"] = "true";
        output.append(input)

    return output
```

Batching in an AWS Lambda processor

OpenSearch Ingestion pipelines will send batched events to your Lambda processor. OpenSearch Ingestion pipelines will dynamically determine batch size and limit the size of the batch to below 5MB.

The following is an example of a pipeline batch:

```
batch:
    key_name: "events"

input_arrary = event.get('events', [])
```

> **Note**
>
> When creating an OpenSearch Ingestion pipeline, ensure that the **key_name** in the lambda configuration of the pipeline matches the event key in the Lambda handler.

Conditional Filtering in AWS Lambda processor

Conditional filtering allows you to control when your AWS Lambda processor invokes the Lambda function based on specific conditions in event data. This is particularly useful when you want to selectively process certain types of events while ignoring others.

To perform conditional fitlering, see the following example:

```
processors:
  - aws_lambda:
      function_name: "my-lambda-function"
      aws:
        region: "us-east-1"
        sts_role_arn: "arn:aws:iam::123456789012:role/my-lambda-role"
      lambda_when: "/sourceIp == 10.10.10.10"
```

Metrics and Monitoring

A Lambda processor offers support for several CloudWatch metrics which can be used for monitoring the performance of your OpenSearch Ingestion pipelines and Lambda processor. These metrics can help you to identify the amount of data sent to and processed by your Lambda processor. You can configure a CloudWatch alarm to perform actions when one of these metrics exceeds a specified value. The following are processor metrics that can be used to monitor the performance and health of your Lambda integrations:

- `recordsSuccessfullySentToLambda`: Counter for successfully processed events
- `recordsFailedToSendToLambda`: Counter for failed events
- `lambdafunctionlatency`: Timer for Lambda invocation latency
- `requestPayloadSize`: Gauge for request payload size
- `numberOfRequestsSucceeded`: Number of requests to Lambda that succeeded
- `numberOfRequestsFailed`: Number of requests to Lambda that failed
- `requestPayloadSize`: Lambda request payload size

- `responsePayloadSize`: Lambda response payload size

Migrating data between domains and collections using Amazon OpenSearch Ingestion

You can use OpenSearch Ingestion pipelines to migrate data between Amazon OpenSearch Service domains or OpenSearch Serverless VPC collections. To do so, you set up a pipeline in which you configure one domain or collection as the source, and another domain or collection as the sink. This effectively migrates your data from one domain or collection to the other.

To migrate data, you must have the following resources:

- A source OpenSearch Service domain or OpenSearch Serverless VPC collection. This domain or collection contains the data that you want to migrate. If you're using a domain, it must be running OpenSearch version 1.0 or later, or Elasticsearch version 7.4 or later. The domain must also have an access policy that grants the appropriate permissions to your pipeline role.
- A separate domain or VPC collection that you want to migrate your data to. This domain or collection will act as the pipeline *sink*.
- An pipeline role that OpenSearch Ingestion will use to read and write to your collection or domain. You include the Amazon Resource Name (ARN) of this role in your pipeline configuration. For more information, see the following resources:
  - the section called “Granting pipelines access to domains”
  - the section called “Granting pipelines access to collections”

Topics

Limitations

The following limitations apply when you designate OpenSearch Service domains or OpenSearch Serverless collections as sinks:

- A pipeline can't write to more than one VPC domain.
- You can only migrate data to or from OpenSearch Serverless collections that use VPC access. Public collections aren't supported.
- You can't specify a combination of VPC and public domains in a single pipeline configuration.
- You can have a maximum of 20 non-pipeline sinks within a single pipeline configuration.
- You can specify sinks from a maximum of three different AWS Regions in a single pipeline configuration.
- A pipeline with multiple sinks might experience a reduction in processing speed over time if any of the sinks are down for too long, or are not provisioned with enough capacity to receive incoming data.

OpenSearch Service as a source

The domain or collection that you specify as the source is where the data is migrated *from*.

Creating a pipeline role in IAM

To create your OpenSearch Ingestion pipeline, you must first create a pipeline role to grant read and write access between domains or collections. To do this, perform the following steps:

1. Create a new permissions policy in IAM to attach to the pipeline role. Make sure you allow permissions to read from the source and write to the sink. For more information on setting IAM pipeline permissions for OpenSearch Service domains, see the section called “Granting pipelines access to domains” and the section called “Granting pipelines access to collections”.
2. Specify the following permissions within the pipeline role to read from the source:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"es:ESHttpGet",
         "Resource":[
            "arn:aws:es:us-east-1:{account-id}:domain/{domain-name}/",
            "arn:aws:es:us-east-1:{account-id}:domain/{domain-name}/_cat/indices",
            "arn:aws:es:us-east-1:{account-id}:domain/{domain-name}/_search",
            "arn:aws:es:us-east-1:{account-id}:domain/{domain-name}/_search/scroll",
            "arn:aws:es:us-east-1:{account-id}:domain/{domain-name}/*/_search"
```

```
            ]
         },
         {
            "Effect":"Allow",
            "Action":"es:ESHttpPost",
            "Resource":[
               "arn:aws:es:us-east-1:{account-id}:domain/{domain-name}/*/_search/
point_in_time",
               "arn:aws:es:us-east-1:{account-id}:domain/{domain-name}/*/_search/scroll"
            ]
         },
         {
            "Effect":"Allow",
            "Action":"es:ESHttpDelete",
            "Resource":[
               "arn:aws:es:us-east-1:{account-id}:domain/{domain-name}/_search/
point_in_time",
               "arn:aws:es:us-east-1:{account-id}:domain/{domain-name}/_search/scroll"
            ]
         }
      ]
}
```

Creating a pipeline

After you attach the policy to the pipeline role, use the **AWSOpenSearchDataMigrationPipeline** migration blueprint to create the pipeline. This blueprint includes a default configuration for migrating data between OpenSearch Service domains or collections. For more information, see the section called “Using blueprints to create a pipeline”.

> **Note**
>
> OpenSearch Ingestion uses your source domain version and distribution to determine what mechanism to use for migration. Some versions support the `point_in_time` option. OpenSearch Serverless uses the `search_after` option because it doesn't support `point_in_time` or `scroll`.

New indexes might be in the process of being created during the migration process, or documents might be updating while migration is in progress. Because of this, you might need to perform either a single scan or multiple scans of your domain index data to pick up new or updated data.

Specify the number of scans to run by configuring the `index_read_count` and `interval` in the pipeline configuration. The following example shows how to perform multiple scans:

```
scheduling:
    interval: "PT2H"
    index_read_count: 3
    start_time: "2023-06-02T22:01:30.00Z"
```

OpenSearch Ingestion uses the following configuration to ensure that your data is written to the same index and maintains the same document ID:

```
index: "${getMetadata(\"opensearch-index\")}"
document_id: "${getMetadata(\"opensearch-document_id\")}"
```

Specifying multiple OpenSearch Service domain sinks

You can specify multiple public OpenSearch Service domains as destinations for your data. You can use this capability to perform conditional routing or replicate incoming data into multiple OpenSearch Service domains. You can specify up to 10 different public OpenSearch Service domains as sinks.

In the following example, incoming data is conditionally routed to different OpenSearch Service domains:

```
...
  route:
    - 2xx_status: "/response >= 200 and /response < 300"
    - 5xx_status: "/response >= 500 and /response < 600"
  sink:
    - opensearch:
        hosts: [ "https://search-response-2xx.us-east-1.es.amazonaws.com" ]
        aws:
          sts_role_arn: "arn:aws:iam::123456789012:role/Example-Role"
          region: "us-east-1"
        index: "response-2xx"
        routes:
```

```
        - 2xx_status
    - opensearch:
        hosts: [ "https://search-response-5xx.us-east-1.es.amazonaws.com" ]
        aws:
          sts_role_arn: "arn:aws:iam::123456789012:role/Example-Role"
          region: "us-east-1"
        index: "response-5xx"
        routes:
          - 5xx_status
```

Migrating data to an OpenSearch Serverless VPC collection

You can use OpenSearch Ingestion to migrate data from a source OpenSearch Service domain or OpenSearch Serverless collection to a VPC collection sink. You must provide a network access policy within the pipeline configuration. For more information about data ingestion into OpenSearch Serverless VPC collections, see the section called "Tutorial: Ingest data into a collection".

To migrate data to a VPC collection

1. Create an OpenSearch Serverless collection. For instructions, see the section called "Tutorial: Ingest data into a collection".
2. Create a network policy for the collection that specifies VPC access to both the collection endpoint and the Dashboards endpoint. For instructions, see the section called "Network access".
3. Create the pipeline role if you don't already have one. For instructions, see the section called "Pipeline role".
4. Create the pipeline. For instructions, see the section called "Using blueprints to create a pipeline".

Using the AWS SDKs to interact with Amazon OpenSearch Ingestion

This section includes an example of how to use the AWS SDKs to interact with Amazon OpenSearch Ingestion. The code example demonstrates how to create a domain and a pipeline, and then ingest data into the pipeline.

Topics

Python

The following sample script uses the AWS SDK for Python (Boto3) to create an IAM pipeline role, a domain to write data to, and a pipeline to ingest data through. It then ingests a sample log file into the pipeline using the `requests` HTTP library.

To install the required dependencies, run the following commands:

```
pip install boto3
pip install botocore
pip install requests
pip install requests-auth-aws-sigv4
```

Within the script, replace the account IDs in the access policies with your AWS account ID. You can also optionally modify the `region`.

```
import boto3
import botocore
from botocore.config import Config
import requests
from requests_auth_aws_sigv4 import AWSSigV4
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default region.

my_config = Config(
    # Optionally lets you specify a Region other than your default.
    region_name='us-east-1'
)

opensearch = boto3.client('opensearch', config=my_config)
iam = boto3.client('iam', config=my_config)
osis = boto3.client('osis', config=my_config)

domainName = 'test-domain'  # The name of the domain
pipelineName = 'test-pipeline' # The name of the pipeline

def createPipelineRole(iam, domainName):
```

```
    """Creates the pipeline role"""
    response = iam.create_policy(
        PolicyName='pipeline-policy',
        PolicyDocument=f'{{\"Version\":\"2012-10-17\",\"Statement\":[{{\"Effect
\":\"Allow\",\"Action\":\"es:DescribeDomain\",\"Resource\":\"arn:aws:es:us-
east-1:123456789012:domain\/{domainName}\"}},{{\"Effect\":\"Allow\",\"Action\":
\"es:ESHttp*\",\"Resource\":\"arn:aws:es:us-east-1:123456789012:domain\/{domainName}\/*
\"}}]}}'
    )
    policyarn = response['Policy']['Arn']

    response = iam.create_role(
        RoleName='PipelineRole',
        AssumeRolePolicyDocument='{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect
\":\"Allow\",\"Principal\":{\"Service\":\"osis-pipelines.amazonaws.com\"},\"Action\":
\"sts:AssumeRole\"}]}'
    )
    rolename=response['Role']['RoleName']

    response = iam.attach_role_policy(
        RoleName=rolename,
        PolicyArn=policyarn
    )

    print('Creating pipeline role...')
    time.sleep(10)
    print('Role created: ' + rolename)

def createDomain(opensearch, domainName):
    """Creates a domain to ingest data into"""
    response = opensearch.create_domain(
        DomainName=domainName,
        EngineVersion='OpenSearch_2.3',
        ClusterConfig={
            'InstanceType': 't2.small.search',
            'InstanceCount': 5,
            'DedicatedMasterEnabled': True,
            'DedicatedMasterType': 't2.small.search',
            'DedicatedMasterCount': 3
        },
        # Many instance types require EBS storage.
        EBSOptions={
            'EBSEnabled': True,
            'VolumeType': 'gp2',
```

```
            'VolumeSize': 10
        },
        AccessPolicies=f'{{\"Version\":\"2012-10-17\",\"Statement\":[{{\"Effect\":
\"Allow\",\"Principal\":{{\"AWS\":\"arn:aws:iam::123456789012:role\/PipelineRole
\"}},\"Action\":\"es:*\",\"Resource\":\"arn:aws:es:us-east-1:123456789012:domain\/
{domainName}\/*\"}}]}}',
        NodeToNodeEncryptionOptions={
            'Enabled': True
        }
    )
    return(response)

def waitForDomainProcessing(opensearch, domainName):
    """Waits for the domain to be active"""
    try:
        response = opensearch.describe_domain(
            DomainName=domainName
        )
        # Every 30 seconds, check whether the domain is processing.
        while 'Endpoint' not in response['DomainStatus']:
            print('Creating domain...')
            time.sleep(60)
            response = opensearch.describe_domain(
                DomainName=domainName)

        # Once we exit the loop, the domain is ready for ingestion.
        endpoint = response['DomainStatus']['Endpoint']
        print('Domain endpoint ready to receive data: ' + endpoint)
        createPipeline(osis, endpoint)

    except botocore.exceptions.ClientError as error:
        if error.response['Error']['Code'] == 'ResourceNotFoundException':
            print('Domain not found.')
        else:
            raise error

def createPipeline(osis, endpoint):
    """Creates a pipeline using the domain and pipeline role"""
    try:
        definition = f'version: \"2\"\nlog-pipeline:\n  source:\n    http:\n      path:
 \"/${{pipelineName}}/logs\"\n  processor:\n    - date:\n        from_time_received:
 true\n        destination: \"@timestamp\"\n  sink:\n    - opensearch:\n        hosts:
 [ \"https://{endpoint}\" ]\n        index: \"application_logs\"\n        aws:\n
```

```
     sts_role_arn: \"arn:aws:iam::123456789012:role/PipelineRole\"\n          region:
 \"us-east-1\"'
        response = osis.create_pipeline(
            PipelineName=pipelineName,
            MinUnits=4,
            MaxUnits=9,
            PipelineConfigurationBody=definition
        )

        response = osis.get_pipeline(
                PipelineName=pipelineName
        )

        # Every 30 seconds, check whether the pipeline is active.
        while response['Pipeline']['Status'] == 'CREATING':
            print('Creating pipeline...')
            time.sleep(30)
            response = osis.get_pipeline(
                PipelineName=pipelineName)

        # Once we exit the loop, the pipeline is ready for ingestion.
        ingestionEndpoint = response['Pipeline']['IngestEndpointUrls'][0]
        print('Pipeline ready to ingest data at endpoint: ' + ingestionEndpoint)
        ingestData(ingestionEndpoint)

    except botocore.exceptions.ClientError as error:
        if error.response['Error']['Code'] == 'ResourceAlreadyExistsException':
            print('Pipeline already exists.')
            response = osis.get_pipeline(
                PipelineName=pipelineName
            )
            ingestionEndpoint = response['Pipeline']['IngestEndpointUrls'][0]
            ingestData(ingestionEndpoint)
        else:
            raise error


def ingestData(ingestionEndpoint):
    """Ingests a sample log file into the pipeline"""
    endpoint = 'https://' + ingestionEndpoint
    r = requests.request('POST', f'{endpoint}/log-pipeline/logs',

 data='[{"time":"2014-08-11T11:40:13+00:00","remote_addr":"122.226.223.69","status":"404","requ
```

```
 http://www.k2proxy.com//hello.html HTTP/1.1","http_user_agent":"Mozilla/4.0
 (compatible; WOW64; SLCC2;)"}]',
    auth=AWSSigV4('osis'))
    print('Ingesting sample log file into pipeline')
    print('Response: ' + r.text)

def main():
    createPipelineRole(iam, domainName)
    createDomain(opensearch, domainName)
    waitForDomainProcessing(opensearch, domainName)

if __name__ == "__main__":
    main()
```

Security in Amazon OpenSearch Ingestion

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using OpenSearch Ingestion. The following topics show you how to configure OpenSearch Ingestion to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your OpenSearch Ingestion resources.

Topics

- Configuring VPC access for Amazon OpenSearch Ingestion pipelines
- Identity and Access Management for Amazon OpenSearch Ingestion

- Logging Amazon OpenSearch Ingestion API calls using AWS CloudTrail

Configuring VPC access for Amazon OpenSearch Ingestion pipelines

You can access your Amazon OpenSearch Ingestion pipelines using an interface VPC endpoint. A VPC is a virtual network that's dedicated to your AWS account. It's logically isolated from other virtual networks in the AWS Cloud. Accessing a pipeline through a VPC endpoint enables secure communication between OpenSearch Ingestion 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.

OpenSearch Ingestion establishes this private connection by creating an *interface endpoint*, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you specify during pipeline creation. These are requester-managed network interfaces that serve as the entry point for traffic destined for the OpenSearch Ingestion pipeline. You can also choose to create and manage the interface endpoints yourself.

Using a VPC allows you to enforce data flow through your OpenSearch Ingestion pipelines within the boundaries of the VPC, rather than over the public internet. Pipelines that aren't within a VPC send and receive data over public-facing endpoints and the internet.

A pipeline with VPC access can write to public or VPC OpenSearch Service domains, and to public or VPC OpenSearch Serverless collections.

Topics

Considerations

Consider the following when you configure VPC access for a pipeline.

- A pipeline doesn't need to be in the same VPC as its sink. You also don't need to establish a connection between the two VPCs. OpenSearch Ingestion takes care of connecting them for you.
- You can only specify one VPC for your pipeline.
- Unlike with public pipelines, a VPC pipeline must be in the same AWS Region as the domain or collection sink that it's writing to.
- You can choose to deploy a pipeline into one, two, or three subnets of your VPC. The subnets are distributed across the same Availability Zones that your Ingestion OpenSearch Compute Units (OCUs) are deployed in.
- If you only deploy a pipeline in one subnet and the Availability Zone goes down, you won't be able to ingest data. To ensure high availability, we recommend that you configure pipelines with two or three subnets.
- Specifying a security group is optional. If you don't provide a security group, OpenSearch Ingestion uses the default security group that is specified in the VPC.

Limitations

Pipelines with VPC access have the following limitations.

- You can't change a pipeline's network configuration after you create it. If you launch a pipeline within a VPC, you can't later change it to a public endpoint, and vice versa.
- You can either launch your pipeline with an interface VPC endpoint or a public endpoint, but you can't do both. You must choose one or the other when you create a pipeline.
- After you provision a pipeline with VPC access, you can't move it to a different VPC, and you can't change its subnets or security group settings.
- If your pipeline writes to a domain or collection sink that uses VPC access, you can't go back later and change the sink (VPC or public) after the pipeline is created. You must delete and recreate the pipeline with a new sink. You can still switch from a public sink to a sink with VPC access.
- You can't provide cross-account ingestion access to VPC pipelines.

Prerequisites

Before you can provision a pipeline with VPC access, you must do the following:

- **Create a VPC**

To create your VPC, you can use the Amazon VPC console, the AWS CLI, or one of the AWS SDKs. For more information, see Working with VPCs in the *Amazon VPC User Guide*. If you already have a VPC, you can skip this step.

- **Reserve IP addresses**

  OpenSearch Ingestion places an *elastic network interface* in each subnet that you specify during pipeline creation. Each network interface is associated with an IP address. You must reserve one IP address per subnet for the network interfaces.

Configuring VPC access for a pipeline

You can enable VPC access for a pipeline within the OpenSearch Service console or using the AWS CLI.

Console

You configure VPC access during pipeline creation. Under **Network**, choose **VPC access** and configure the following settings:

Setting	Description
Endpoint management	Choose whether you want to create your VPC endpoints yourself, or have OpenSearch Ingestion create them for you.
VPC	Choose the ID of the virtual private cloud (VPC) that you want to use. The VPC and pipeline must be in the same AWS Region.
Subnets	Choose one or more subnets. OpenSearch Service will place a VPC endpoint and *elastic network interfaces* in the subnets.
Security groups	Choose one or more VPC security groups that allow your required applicati on to reach the OpenSearch Ingestion pipeline on the ports (80 or 443) and protocols (HTTP or HTTPs) exposed by the pipeline.
VPC attachment options	If your source is a self-managed endpoint, attach your pipeline to a VPC. Choose one of the default CIDR options provided, or use a custom CIDR.

CLI

To configure VPC access using the AWS CLI, specify the `--vpc-options` parameter:

```
aws osis create-pipeline \
  --pipeline-name vpc-pipeline \
  --min-units 4 \
  --max-units 10 \
  --vpc-options
 SecurityGroupIds={sg-12345678,sg-9012345},SubnetIds=subnet-1212234567834asdf \
  --pipeline-configuration-body "file://pipeline-config.yaml"
```

Self-managed VPC endpoints

When you create a pipeline, you can use endpoint management to create a pipeline with self-managed endpoints or service-managed endpoints. Endpoint management is optional, and defaults to endpoints managed by OpenSearch Ingestion.

To create a pipeline with a self-managed VPC endpoint in the AWS Management Console, see Creating pipelines with the OpenSearch Service console. To create a pipeline with a self-managed VPC endpoint in the AWS CLI, you can use the `--vpc-options` parameter in the create-pipeline command:

```
--vpc-options SubnetIds=subnet-abcdef01234567890,VpcEndpointManagement=CUSTOMER
```

You can create an endpoint to your pipeline yourself when you specify your endpoint service. To find your endpoint service, use the get-pipeline command, which returns a response similar to the following:

```
"vpcEndpointService" : "com.amazonaws.osis.us-east-1.pipeline-
id-1234567890abcdef1234567890",
"vpcEndpoints" : [
  {
    "vpcId" : "vpc-1234567890abcdef0",
    "vpcOptions" : {
      "subnetIds" : [ "subnet-abcdef01234567890", "subnet-021345abcdef6789" ],
      "vpcEndpointManagement" : "CUSTOMER"
    }
  }
```

Use the `vpcEndpointService` from the response to create a VPC endpoint with the AWS Management Console or AWS CLI.

If you use self-managed VPC endpoints, you must enable the DNS attributes `enableDnsSupport` and `enableDnsHostnames` in your VPC. Note that if you have a pipeline with a self-managed endpoint that you stop and restart, you must recreate the VPC endpoint in your account.

Service-linked role for VPC access

A service-linked role is a unique type of IAM role that delegates permissions to a service so that it can create and manage resources on your behalf. If you choose a service-managed VPC endpoint, OpenSearch Ingestion requires a service-linked role called **AWSServiceRoleForAmazonOpenSearchIngestionService** to access your VPC, create the pipeline endpoint, and place network interfaces in a subnet of your VPC.

If you choose a self-managed VPC endpoint, OpenSearch Ingestion requires a service-linked role called **AWSServiceRoleForOpensearchIngestionSelfManagedVpce**. For more information on these roles, their permissions, and how to delete them, see the section called “Pipeline creation role”.

OpenSearch Ingestion automatically creates the role when you create an ingestion pipeline. For this automatic creation to succeed, the user creating the first pipeline in an account must have permissions for the `iam:CreateServiceLinkedRole` action. To learn more, see Service-linked role permissions in the *IAM User Guide*. You can view the role in the AWS Identity and Access Management (IAM) console after it's created.

Identity and Access Management for Amazon OpenSearch Ingestion

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use OpenSearch Ingestion resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Identity-based policies for OpenSearch Ingestion
- Policy actions for OpenSearch Ingestion
- Policy resources for OpenSearch Ingestion
- Policy condition keys for Amazon OpenSearch Ingestion
- ABAC with OpenSearch Ingestion

- Using temporary credentials with OpenSearch Ingestion
- Service-linked roles for OpenSearch Ingestion
- Identity-based policy examples for OpenSearch Ingestion

Identity-based policies for OpenSearch Ingestion

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for OpenSearch Ingestion

To view examples of OpenSearch Ingestion identity-based policies, see the section called "Identity-based policy examples".

Policy actions for OpenSearch Ingestion

Supports policy actions: Yes

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in OpenSearch Ingestion use the following prefix before the action:

```
osis
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "osis:action1",
      "osis:action2"
         ]
```

You can specify multiple actions using wildcard characters (*). For example, to specify all actions that begin with the word `List`, include the following action:

```
"Action": "osis:List*"
```

To view examples of OpenSearch Ingestion identity-based policies, see Identity-based policy examples for OpenSearch Serverless.

Policy resources for OpenSearch Ingestion

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

Policy condition keys for Amazon OpenSearch Ingestion

Supports service-specific policy condition keys: No

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use

condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of OpenSearch Ingestion condition keys, see Condition keys for Amazon OpenSearch Ingestion in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Amazon OpenSearch Ingestion.

ABAC with OpenSearch Ingestion

Supports ABAC (tags in policies): Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

For more information about tagging OpenSearch Ingestion resources, see the section called "Tagging pipelines".

Using temporary credentials with OpenSearch Ingestion

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Service-linked roles for OpenSearch Ingestion

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

OpenSearch Ingestion uses a service-linked role called `AWSServiceRoleForAmazonOpenSearchIngestionService`. The service-linked role named `AWSServiceRoleForOpensearchIngestionSelfManagedVpce` is also available for pipelines

with self-managed VPC endpoints. For details about creating and managing OpenSearch Ingestion service-linked roles, see the section called “Pipeline creation role”.

Identity-based policy examples for OpenSearch Ingestion

By default, users and roles don't have permission to create or modify OpenSearch Ingestion resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by Amazon OpenSearch Ingestion, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for Amazon OpenSearch Ingestion in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using OpenSearch Ingestion in the console
- Administering OpenSearch Ingestion pipelines
- Ingesting data into an OpenSearch Ingestion pipeline

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete OpenSearch Ingestion resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

Identity-based policies determine whether someone can create, access, or delete OpenSearch Ingestion resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies

that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.

- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using OpenSearch Ingestion in the console

To access OpenSearch Ingestion within the OpenSearch Service console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the OpenSearch Ingestion resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (such as IAM roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

The following policy allows a user to access OpenSearch Ingestion within the OpenSearch Service console:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Resource": "*",
            "Effect": "Allow",
            "Action": [
                "osis:ListPipelines",
                "osis:GetPipeline",
                "osis:ListPipelineBlueprints",
                "osis:GetPipelineBlueprint",
                "osis:GetPipelineChangeProgress"
            ]
        }
    ]
}
```

Alternately, you can use the the section called “AmazonOpenSearchIngestionReadOnlyAccess” AWS managed policy, which grants read-only access to all OpenSearch Ingestion resources for an AWS account.

Administering OpenSearch Ingestion pipelines

This policy is an example of a "pipeline admin" policy that allows a user to manage and administer Amazon OpenSearch Ingestion pipelines. The user can create, view, and delete pipelines.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Resource": "arn:aws:osis:region:123456789012:pipeline/*",
            "Action": [
                "osis:CreatePipeline",
                "osis:DeletePipeline",
                "osis:UpdatePipeline",
                "osis:ValidatePipeline",
                "osis:StartPipeline",
                "osis:StopPipeline"
            ],
```

```
            "Effect": "Allow"
        },
        {
            "Resource": "*",
            "Action": [
                "osis:ListPipelines",
                "osis:GetPipeline",
                "osis:ListPipelineBlueprints",
                "osis:GetPipelineBlueprint",
                "osis:GetPipelineChangeProgress"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Ingesting data into an OpenSearch Ingestion pipeline

This example policy allows a user or other entity to ingest data into an Amazon OpenSearch Ingestion pipeline in their account. The user can't modify the pipelines.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Resource": "arn:aws:osis:region:123456789012:pipeline/*",
            "Action": [
                "osis:Ingest"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Logging Amazon OpenSearch Ingestion API calls using AWS CloudTrail

Amazon OpenSearch Ingestion is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in OpenSearch Ingestion.

CloudTrail captures all API calls for OpenSearch Ingestion as events. The calls captured include calls from the OpenSearch Ingestion section of the OpenSearch Service console and code calls to the OpenSearch Ingestion API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for OpenSearch Ingestion. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**.

Using the information collected by CloudTrail, you can determine the request that was made to OpenSearch Ingestion, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

OpenSearch Ingestion information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in OpenSearch Ingestion, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history.

For an ongoing record of events in your AWS account, including events for OpenSearch Ingestion, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions.

The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

All OpenSearch Ingestion actions are logged by CloudTrail and are documented in the OpenSearch Ingestion API reference. For example, calls to the `CreateCollection`, `ListCollections`, and `DeleteCollection` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding OpenSearch Ingestion log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries.

An event represents a single request from any source. It includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `DeletePipeline` action.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn":"arn:aws:iam::123456789012:user/test-user",
        "accountId": "123456789012",
        "accessKeyId": "access-key",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-04-21T16:48:33Z",
                "mfaAuthenticated": "false"
            }
        }
```

```
    },
    "eventTime": "2023-04-21T16:49:22Z",
    "eventSource": "osis.amazonaws.com",
    "eventName": "UpdatePipeline",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "123.456.789.012",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36",
    "requestParameters": {
        "pipelineName": "my-pipeline",
        "pipelineConfigurationBody": "version: \"2\"\nlog-pipeline:\n  source:\n
 http:\n        path: \"/test/logs\"\n  processor:\n    - grok:\n        match:\n
        log: [ '%{COMMONAPACHELOG}' ]\n    - date:\n        from_time_received: true
\n        destination: \"@timestamp\"\n  sink:\n    - opensearch:\n        hosts:
 [ \"https://search-b5zd22mwxhggheqpj5ftslgyle.us-west-2.es.amazonaws.com\" ]\n
  index: \"apache_logs2\"\n        aws_sts_role_arn: \"arn:aws:iam::709387180454:role/
canary-bootstrap-OsisRole-J1BARLD26QKN\"\n        aws_region: \"us-west-2\"\n
 aws_sigv4: true\n"
    },
    "responseElements": {
        "pipeline": {
            "pipelineName": "my-pipeline",sourceIPAddress
            "pipelineArn": "arn:aws:osis:us-west-2:123456789012:pipeline/my-pipeline",
            "minUnits": 1,
            "maxUnits": 1,
            "status": "UPDATING",
            "statusReason": {
                "description": "An update was triggered for the pipeline. It is still
 available to ingest data."
            },
            "pipelineConfigurationBody": "version: \"2\"\nlog-pipeline:\n  source:\n
    http:\n        path: \"/test/logs\"\n  processor:\n    - grok:\n        match:
\n          log: [ '%{COMMONAPACHELOG}' ]\n    - date:\n        from_time_received:
 true\n        destination: \"@timestamp\"\n  sink:\n    - opensearch:\n        hosts:
 [ \"https://search-b5zd22mwxhggheqpj5ftslgyle.us-west-2.es.amazonaws.com\" ]\n
  index: \"apache_logs2\"\n        aws_sts_role_arn: \"arn:aws:iam::709387180454:role/
canary-bootstrap-OsisRole-J1BARLD26QKN\"\n        aws_region: \"us-west-2\"\n
 aws_sigv4: true\n",
            "createdAt": "Mar 29, 2023 1:03:44 PM",
            "lastUpdatedAt": "Apr 21, 2023 9:49:21 AM",
            "ingestEndpointUrls": [
                "my-pipeline-tu33ldsgdltgv7x7tjqiudvf7m.us-west-2.osis.amazonaws.com"
            ]
        }
```

```
    },
    "requestID": "12345678-1234-1234-1234-987654321098",
    "eventID": "12345678-1234-1234-1234-987654321098",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "709387180454",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "osis.us-west-2.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```

Tagging Amazon OpenSearch Ingestion pipelines

Tags let you assign arbitrary information to an Amazon OpenSearch Ingestion pipeline so you can categorize and filter on that information. A *tag* is a metadata label that you assign or that AWS assigns to an AWS resource. Each tag consists of a *key* and a *value*. For tags that you assign, you define the key and value. For example, you might define the key as `stage` and the value for one resource as `test`.

Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you could assign the same tag to an OpenSearch Ingestion pipeline that you assign to an Amazon OpenSearch Service domain.
- Track your AWS costs. You activate these tags on the AWS Billing and Cost Management dashboard. AWS uses the tags to categorize your costs and deliver a monthly cost allocation report to you. For more information, see Use Cost Allocation Tags in the AWS Billing User Guide.
- Restrict access to pipelines using attribute based access control. For more information, see Controlling access based on tag keys in the IAM User Guide.

In OpenSearch Ingestion, the primary resource is a pipeline. You can use the OpenSearch Service console, the AWS CLI, OpenSearch Ingestion APIs, or the AWS SDKs to add, manage, and remove tags from a pipeline.

Topics

Permissions required

OpenSearch Ingestion uses the following AWS Identity and Access Management Access Analyzer (IAM) permissions for tagging pipelines:

- `osis:TagResource`
- `osis:ListTagsForResource`
- `osis:UntagResource`

For more information about each permission, see Actions, resources, and condition keys for OpenSearch Ingestion in the *Service Authorization Reference*.

Working with tags (console)

The console is the simplest way to tag a pipeline.

To create a tag

1. Sign in to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Choose **Ingestion** on the left navigation pane.
3. Select the pipeline you want to add tags to and go to the **Tags** tab.
4. Choose **Manage** and **Add new tag**.
5. Enter a tag key and an optional value.
6. Choose **Save**.

To delete a tag, follow the same steps and choose **Remove** on the **Manage tags** page.

For more information about using the console to work with tags, see Tag Editor in the *AWS Management Console Getting Started Guide*.

Working with tags (AWS CLI)

To tag a pipeline using the AWS CLI, send a `TagResource` request:

```
aws osis tag-resource
  --arn arn:aws:osis:us-east-1:123456789012:pipeline/my-pipeline
  --tags Key=service,Value=osis Key=source,Value=otel
```

Remove tags from a pipeline using the `UntagResource` command:

```
aws osis untag-resource
  --arn arn:aws:osis:us-east-1:123456789012:pipeline/my-pipeline
  --tag-keys service
```

View the existing tags for a pipeline with the `ListTagsForResource` command:

```
aws osis list-tags-for-resource
  --arn arn:aws:osis:us-east-1:123456789012:pipeline/my-pipeline
```

Logging and monitoring Amazon OpenSearch Ingestion with Amazon CloudWatch

Amazon OpenSearch Ingestion publishes metrics and logs to Amazon CloudWatch.

Topics

Monitoring pipeline logs

You can enable logging for Amazon OpenSearch Ingestion pipelines to expose error and warning messages raised during pipeline operations and ingestion activity. OpenSearch Ingestion publishes all logs to *Amazon CloudWatch Logs*. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.

Logs from OpenSearch Ingestion might indicate failed processing of requests, authentication errors from the source to the sink, and other warnings that can be helpful for troubleshooting. For its

logs, OpenSearch Ingestion uses the log levels of `INFO`, `WARN`, `ERROR`, and `FATAL`. We recommend enabling log publishing for all pipelines.

Permissions required

In order to enable OpenSearch Ingestion to send logs to CloudWatch Logs, you must be signed in as a user that has certain IAM permissions.

You need the following CloudWatch Logs permissions in order to create and update log delivery resources:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:PutResourcePolicy",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:DescribeResourcePolicies",
                "logs:GetLogDelivery",
                "logs:ListLogDeliveries"
            ]
        }
    ]
}
```

Enabling log publishing

You can enable log publishing on existing pipelines, or while creating a pipeline. For steps to enable log publishing during pipeline creation, see the section called “Creating pipelines”.

Console

To enable log publishing on an existing pipeline

1. Sign in to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.

2. Choose **Ingestion** in the left navigation pane and select the pipeline that you want to enable logs for.
3. Choose **Edit log publishing options**.
4. Select **Publish to CloudWatch Logs**.
5. Either create a new log group or select an existing one. We recommend that you format the name as a path, such as `/aws/vendedlogs/OpenSearchIngestion/pipeline-name/audit-logs`. This format makes it easier to apply a CloudWatch access policy that grants permissions to all log groups under a specific path such as `/aws/vendedlogs/OpenSearchService/OpenSearchIngestion`.

   > **Important**
   >
   > You must include the prefix `vendedlogs` in the log group name, otherwise creation fails.

6. Choose **Save**.

CLI

To enable log publishing using the AWS CLI, send the following request:

```
aws osis update-pipeline \
  --pipeline-name my-pipeline \
  --log-publishing-options  IsLoggingEnabled=true,CloudWatchLogDestination={LogGroup="/
aws/vendedlogs/OpenSearchIngestion/pipeline-name"}
```

Monitoring pipeline metrics

You can monitor Amazon OpenSearch Ingestion pipelines using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the Amazon CloudWatch User Guide.

The OpenSearch Ingestion console displays a series of charts based on the raw data from CloudWatch on the **Performance** tab for each pipeline.

OpenSearch Ingestion reports metrics from most supported plugins. If certain plugins don't have their own table below, it means that they don't report any plugin-specific metrics. Pipeline metrics are published in the `AWS/OSIS` namespace.

Topics

Common metrics

The following metrics are common to all processors and sinks.

Each metric is prefixed by the sub-pipeline name and plugin name, in the format <*sub_pipeline_name*><*plugin*><*metric_name*>. For example, the full name of the `recordsIn.count` metric for a sub-pipeline named `my-pipeline` and the date processor would be `my-pipeline.date.recordsIn.count`.

Metric suffix	Description
`recordsIn.count`	The ingress of records to a pipeline component. This metric applies to processors and sinks. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`recordsOut.count`	The egress of records from a pipeline component. This metric applies to processors and sources. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`timeElapsed.count`	A count of data points recorded during execution of a pipeline component. This metric applies to processors and sinks. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`timeElapsed.sum`	The total time elapsed during execution of a pipeline component. This metric applies to processors and sinks, in milliseconds. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`timeElapsed.max`	The maximum time elapsed during execution of a pipeline component. This metric applies to processors and sinks, in milliseconds. **Relevant statistics**: Max **Dimension**: `PipelineName`

Buffer metrics

The following metrics apply to the default Bounded blocking buffer that OpenSearch Ingestion automatically configures for all pipelines.

Each metric is prefixed by the sub-pipeline name and buffer name, in the format <*sub_pipeline_name*><*buffer_name*><*metric_name*>. For example, the full name of the `recordsWritten.count` metric for a sub-pipeline named `my-pipeline` would be `my-pipeline.BlockingBuffer.recordsWritten.count`.

Metric suffix	Description
`recordsWritten.count`	The number of records written to a buffer. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`recordsRead.count`	The number of records read from a buffer. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`recordsInFlight.value`	The number of unchecked records read from a buffer. **Relevant statistics**: Average **Dimension**: `PipelineName`
`recordsInBuffer.value`	The number of records currently in a buffer. **Relevant statistics**: Average **Dimension**: `PipelineName`
`recordsProcessed.count`	The number of records read from a buffer and processed by a pipeline. **Relevant statistics**: Sum **Dimension**: `PipelineName`

Metric suffix	Description
`recordsWriteFailed.count`	The number of records that the pipeline failed to write to the sink. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`writeTimeElapsed.count`	A count of data points recorded while writing to a buffer. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`writeTimeElapsed.sum`	The total time elapsed while writing to a buffer, in milliseconds. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`writeTimeElapsed.max`	The maximum time elapsed while writing to a buffer, in milliseconds. **Relevant statistics**: Max **Dimension**: `PipelineName`
`writeTimeouts.count`	The count of write timeouts to a buffer. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`readTimeElapsed.count`	A count of data points recorded while reading from a buffer. **Relevant statistics**: Sum **Dimension**: `PipelineName`

Metric suffix	Description
`readTimeElapsed.sum`	The total time elapsed while reading from a buffer, in milliseconds. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`readTimeElapsed.max`	The maximum time elapsed while reading from a buffer, in milliseconds. **Relevant statistics**: Max **Dimension**: `PipelineName`
`checkpointTimeElapsed.count`	A count of data points recorded while checkpointing. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`checkpointTimeElapsed.sum`	The total time elapsed while checkpointing, in milliseconds. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`checkpointTimeElapsed.max`	The maximum time elapsed while checkpointing, in milliseconds. **Relevant statistics**: Max **Dimension**: `PipelineName`

Signature V4 metrics

The following metrics apply to the ingestion endpoint for a pipeline and are associate with the source plugins (`http`, `otel_trace`, and `otel_metrics`). All requests to the ingestion endpoint

must be signed using Signature Version 4. These metrics can help you identify authorization issues when connecting to your pipeline, or confirm that you're successfully authenticating.

Each metric is prefixed by the sub-pipeline name and `osis_sigv4_auth`. For example, *`sub_pipeline_name`*`.osis_sigv4_auth.httpAuthSuccess.count`.

Metric suffix	Description
`httpAuthSuccess.count`	The number of successful Signature V4 requests to the pipeline. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`httpAuthFailure.count`	The number of failed Signature V4 requests to the pipeline. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`httpAuthServerError.count`	The number of Signature V4 requests to the pipeline that returned server errors. **Relevant statistics**: Sum **Dimension**: `PipelineName`

Bounded blocking buffer metrics

The following metrics apply to the bounded blocking buffer. Each metric is prefixed by the sub-pipeline name and `BlockingBuffer`. For example, *`sub_pipeline_name`*`.BlockingBuffer.bufferUsage.value`.

Metric suffix	Description
`bufferUsage.value`	Percent usage of the `buffer_size` based on the number of records in the buffer. `buffer_size`

Metric suffix	Description
	represents the maximum number of records written into the buffer as well as in-flight records that have not been checked. **Relevant statistics**: Average **Dimension**: `PipelineName`

Otel trace source metrics

The following metrics apply to the OTel trace source. Each metric is prefixed by the sub-pipeline name and `otel_trace_source`. For example, *`sub_pipeline_name`*`.otel_trace_source.requestTimeouts.count`.

Metric suffix	Description
`requestTimeouts.count`	The number of requests that timed out. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`requestsReceived.count`	The number of requests received by the plugin. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`successRequests.count`	The number of requests that were successfully processed by the plugin. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`badRequests.count`	The number of requests with an invalid format that were processed by the plugin.

Metric suffix	Description
	Relevant statistics: Sum **Dimension**: `PipelineName`
`requestsTooLarge.count`	The number of requests of which the number of spans in the content is larger than the buffer capacity. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`internalServerError.count`	The number of requests processed by the plugin with a custom exception type. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`requestProcessDuration.count`	A count of data points recorded while processing requests by the plugin. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`requestProcessDuration.sum`	The total latency of requests processed by the plugin, in milliseconds. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`requestProcessDuration.max`	The maximum latency of requests processed by the plugin, in milliseconds. **Relevant statistics**: Max **Dimension**: `PipelineName`

Metric suffix	Description
`payloadSize.count`	A count of the distribution of payload sizes of incoming requests, in bytes. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`payloadSize.sum`	The total distribution of the payload sizes of incoming requests, in bytes. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`payloadSize.max`	The maximum distribution of payload sizes of incoming requests, in bytes. **Relevant statistics**: Max **Dimension**: `PipelineName`

Otel metrics source metrics

The following metrics apply to the OTel metrics source. Each metric is prefixed by the sub-pipeline name and `otel_metrics_source`. For example, *`sub_pipeline_name`*`.otel_metrics_source.requestTimeouts.count`.

Metric suffix	Description
`requestTimeouts.count`	The total number of requests to the plugin that time out. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`requestsReceived.count`	The total number of requests received by the plugin. **Relevant statistics**: Sum

Metric suffix	Description
	Dimension: `PipelineName`
`successRequests.count`	The number of requests successfully processed (200 response status code) by the plugin. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`requestProcessDuration.count`	A count of the latency of requests processed by the plugin, in seconds. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`requestProcessDuration.sum`	The total latency of requests processed by the plugin, in milliseconds. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`requestProcessDuration.max`	The maximum latency of requests processed by the plugin, in milliseconds. **Relevant statistics**: Max **Dimension**: `PipelineName`
`payloadSize.count`	A count of the distribution of payload sizes of incoming requests, in bytes. **Relevant statistics**: Sum **Dimension**: `PipelineName`

Metric suffix	Description
`payloadSize.sum`	The total distribution of the payload sizes of incoming requests, in bytes. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`payloadSize.max`	The maximum distribution of payload sizes of incoming requests, in bytes. **Relevant statistics**: Max **Dimension**: `PipelineName`

Http metrics

The following metrics apply to the HTTP source. Each metric is prefixed by the sub-pipeline name and `http`. For example, *`sub_pipeline_name`*`.http.requestsReceived.count`.

Metric suffix	Description
`requestsReceived.count`	The number of requests received by the `/log/ingest` endpoint. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`requestsRejected.count`	The number of requests rejected (429 response status code) by the plugin. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`successRequests.count`	The number of requests successfully processed (200 response status code) by the plugin.

Metric suffix	Description
	Relevant statistics: Sum **Dimension**: `PipelineName`
`badRequests.count`	The number of requests with invalid content type or format (400 response status code) processed by the plugin. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`requestTimeouts.count`	The number of requests that time out in the HTTP source server (415 response status code). **Relevant statistics**: Sum **Dimension**: `PipelineName`
`requestsTooLarge.count`	The number of requests of which the events size in the content is larger than the buffer capacity (413 response status code). **Relevant statistics**: Sum **Dimension**: `PipelineName`
`internalServerError.count`	The number of requests processed by the plugin with a custom exception type (500 response status code). **Relevant statistics**: Sum **Dimension**: `PipelineName`
`requestProcessDuration.count`	A count of the latency of requests processed by the plugin, in seconds. **Relevant statistics**: Sum **Dimension**: `PipelineName`

Metric suffix	Description
`requestProcessDuration.sum`	The total latency of requests processed by the plugin, in milliseconds. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`requestProcessDuration.max`	The maximum latency of requests processed by the plugin, in milliseconds. **Relevant statistics**: Max **Dimension**: `PipelineName`
`payloadSize.count`	A count of the distribution of payload sizes of incoming requests, in bytes. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`payloadSize.sum`	The total distribution of the payload sizes of incoming requests, in bytes. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`payloadSize.max`	The maximum distribution of payload sizes of incoming requests, in bytes. **Relevant statistics**: Max **Dimension**: `PipelineName`

S3 metrics

The following metrics apply to the S3 source. Each metric is prefixed by the sub-pipeline name and `s3`. For example, *`sub_pipeline_name`*`.s3.s3ObjectsFailed.count`.

Metric suffix	Description
`s3ObjectsFailed.count`	The total number of S3 objects that the plugin failed to read. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3ObjectsNotFound.count`	The number of S3 objects that the plugin failed to read due to a `Not Found` error from S3. These metrics also count toward the `s3ObjectsFailed` metric. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3ObjectsAccessDenied.count`	The number of S3 objects that the plugin failed to read due to an `Access Denied` or `Forbidden` error from S3. These metrics also count toward the `s3ObjectsFailed` metric. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3ObjectReadTimeElapsed.count`	The amount of time the plugin takes to perform a GET request for an S3 object, parse it, and write events to the buffer. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3ObjectReadTimeElapsed.sum`	The total amount of time that the plugin takes to perform a GET request for an S3 object, parse it, and write events to the buffer, in milliseconds. **Relevant statistics**: Sum

Metric suffix	Description
	Dimension: `PipelineName`
`s3ObjectReadTimeElapsed.max`	The maximum amount of time that the plugin takes to perform a GET request for an S3 object, parse it, and write events to the buffer, in milliseconds. **Relevant statistics**: Max **Dimension**: `PipelineName`
`s3ObjectSizeBytes.count`	The count of the distribution of S3 object sizes, in bytes. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3ObjectSizeBytes.sum`	The total distribution of S3 object sizes, in bytes. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3ObjectSizeBytes.max`	The maximum distribution of S3 object sizes, in bytes. **Relevant statistics**: Max **Dimension**: `PipelineName`
`s3ObjectProcessedBytes.count`	The count of the distribution of S3 objects processed by the plugin, in bytes. **Relevant statistics**: Sum **Dimension**: `PipelineName`

Metric suffix	Description
`s3ObjectProcessedBytes.sum`	The total distribution of S3 objects processed by the plugin, in bytes. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3ObjectProcessedBytes.max`	The maximum distribution of S3 objects processed by the plugin, in bytes. **Relevant statistics**: Max **Dimension**: `PipelineName`
`s3ObjectsEvents.count`	The count of the distribution of S3 events received by the plugin. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3ObjectsEvents.sum`	The total distribution of S3 events received by the plugin. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3ObjectsEvents.max`	The maximum distribution of S3 events received by the plugin. **Relevant statistics**: Max **Dimension**: `PipelineName`

Metric suffix	Description
`sqsMessageDelay.count`	A count of data points recorded while S3 records an event time for the creation of an object to when it's fully parsed. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`sqsMessageDelay.sum`	The total amount of time between when S3 records an event time for the creation of an object to when it's fully parsed, in milliseconds. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`sqsMessageDelay.max`	The maximum amount of time between when S3 records an event time for the creation of an object to when it's fully parsed, in milliseconds. **Relevant statistics**: Max **Dimension**: `PipelineName`
`s3ObjectsSucceeded.count`	The number of S3 objects that the plugin successfully read. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`sqsMessagesReceived.count`	The number of Amazon SQS messages received from the queue by the plugin. **Relevant statistics**: Sum **Dimension**: `PipelineName`

Metric suffix	Description
`sqsMessagesDeleted.count`	The number of Amazon SQS messages deleted from the queue by the plugin. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`sqsMessagesFailed.count`	The number of Amazon SQS messages that the plugin failed to parse. **Relevant statistics**: Sum **Dimension**: `PipelineName`

Aggregate metrics

The following metrics apply to the Aggregate processor. Each metric is prefixed by the sub-pipeline name and `aggregate`. For example, *`sub_pipeline_name`*`.aggregate.actionHandleEventsOut.count`.

Metric suffix	Description
`actionHandleEventsOut.count`	The number of events that have been returned from the `handleEvent` call to the configured action. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`actionHandleEventsDropped.count`	The number of events that have been returned from the `handleEvent` call to the configured action. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`actionHandleEventsProcessingErrors.count`	The number of calls made to `handleEvent` for the configured action that resulted in an error.

Metric suffix	Description
	Relevant statistics: Sum **Dimension**: `PipelineName`
`actionConcludeGroupEventsOut.count`	The number of events that have been returned from the `concludeGroup` call to the configured action. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`actionConcludeGroupEventsDropped.count`	The number of events that have not been returned from the `condludeGroup` call to the configured action. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`actionConcludeGroupEventsProcessingErrors.count`	The number of calls made to `concludeGroup` for the configured action that resulted in an error. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`currentAggregateGroups.value`	The current number of groups. This gauge decreases when groups are concluded, and increases when an event initiates the creation of a new group. **Relevant statistics**: Average **Dimension**: `PipelineName`

Date metrics

The following metrics apply to the Date processor. Each metric is prefixed by the sub-pipeline name and `date`. For example, *`sub_pipeline_name`*`.date.dateProcessingMatchSuccess.count`.

Metric suffix	Description
`dateProcessingMatchSuccess.count`	The number of records that match at least one of the patterns specified in the `match` configuration option. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`dateProcessingMatchFailure.count`	The number of records that didn't match any of the patterns specified in the `match` configuration option. **Relevant statistics**: Sum **Dimension**: `PipelineName`

Grok metrics

The following metrics apply to the Grok processor. Each metric is prefixed by the sub-pipeline name and `grok`. For example, *`sub_pipeline_name`*`.grok.grokProcessingMatch.count`.

Metric suffix	Description
`grokProcessingMatch.count`	The number of records that found at least one pattern match from the `match` configuration option. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`grokProcessingMismatch.count`	The number of records that didn't match any of the patterns specified in the `match` configuration option. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`grokProcessingErrors.count`	The number of record processing errors. **Relevant statistics**: Sum

Metric suffix	Description
	Dimension: `PipelineName`
`grokProcessingTimeouts.count`	The number of records that timed out while matching. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`grokProcessingTime.count`	A count of data points recorded while an individual record matched against patterns from the `match` configuration option. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`grokProcessingTime.sum`	The total amount of time that each individual record takes to match against patterns from the `match` configuration option, in milliseconds. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`grokProcessingTime.max`	The maximum amount of time that each individual record takes to match against patterns from the `match` configuration option, in milliseconds. **Relevant statistics**: Max **Dimension**: `PipelineName`

Otel trace raw metrics

The following metrics apply to the OTel trace raw processor. Each metric is prefixed by the sub-pipeline name and `otel_trace_raw`. For example, *`sub_pipeline_name`*`.otel_trace_raw.traceGroupCacheCount.value`.

Metric suffix	Description
`traceGroupCacheCount.value`	The number of trace groups in the trace group cache. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`spanSetCount.value`	The number of span sets in the span set collection. **Relevant statistics**: Sum **Dimension**: `PipelineName`

Otel trace group metrics

The following metrics apply to the OTel trace group processor. Each metric is prefixed by the sub-pipeline name and `otel_trace_group`. For example, *`sub_pipeline_name`*`.otel_trace_group.recordsInMissingTraceGroup.count`.

Metric suffix	Description
`recordsInMissingTraceGroup.count`	The number of ingress records missing trace group fields. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`recordsOutFixedTraceGroup.count`	The number of egress records with trace group fields that were filled successfully. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`recordsOutMissingTraceGroup.count`	The number of egress records missing trace group fields. **Relevant statistics**: Sum **Dimension**: `PipelineName`

Service map stateful metrics

The following metrics apply to the Service-map stateful processor. Each metric is prefixed by the sub-pipeline name and `service-map-stateful`. For example, *`sub_pipeline_name`*`.service-map-stateful.spansDbSize.count`.

Metric suffix	Description
`spansDbSize.value`	The in-memory byte sizes of spans in MapDB across the current and previous window durations. **Relevant statistics**: Average **Dimension**: `PipelineName`
`traceGroupDbSize.value`	The in-memory byte sizes of trace groups in MapDB across the current and previous window durations. **Relevant statistics**: Average **Dimension**: `PipelineName`
`spansDbCount.value`	The count of spans in MapDB across the current and previous window durations. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`traceGroupDbCount.value`	The count of trace groups in MapDB across the current and previous window durations. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`relationshipCount.value`	The count of relationships stored across the current and previous window durations. **Relevant statistics**: Sum

Metric suffix	Description
	Dimension: `PipelineName`

OpenSearch metrics

The following metrics apply to the OpenSearch sink. Each metric is prefixed by the sub-pipeline name and `opensearch`. For example, *`sub_pipeline_name`*`.opensearch.bulkRequestErrors.count`.

Metric suffix	Description
`bulkRequestErrors.count`	The total number of errors encountered while sending bulk requests. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`documentsSuccess.count`	The number of documents successfully sent to the OpenSearch Service by bulk request, including retries. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`documentsSuccessFirstAttempt.count`	The number of documents successfully sent to OpenSearch Service by bulk request on the first attempt. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`documentErrors.count`	The number of documents that failed to be sent by bulk requests. **Relevant statistics**: Sum **Dimension**: `PipelineName`

Metric suffix	Description
`bulkRequestFailed.count`	The number of bulk requests that failed. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`bulkRequestNumberOfRetries.count`	The number of retries of failed bulk requests. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`bulkBadRequestErrors.count`	The number of `Bad Request` errors encountered while sending bulk requests. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`bulkRequestNotAllowedErrors.count`	The number of `Request Not Allowed` errors encountered while sending bulk requests. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`bulkRequestInvalidInputErrors.count`	The number of `Invalid Input` errors encountered while sending bulk requests. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`bulkRequestNotFoundErrors.count`	The number of `Request Not Found` errors encountered while sending bulk requests. **Relevant statistics**: Sum **Dimension**: `PipelineName`

Metric suffix	Description
`bulkRequestTimeoutErrors.count`	The number of `Request Timeout` errors encountered while sending bulk requests. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`bulkRequestServerErrors.count`	The number of `Server Error` errors encountered while sending bulk requests. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`bulkRequestSizeBytes.count`	A count of the distribution of payload sizes of bulk requests, in bytes. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`bulkRequestSizeBytes.sum`	The total distribution of payload sizes of bulk requests, in bytes. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`bulkRequestSizeBytes.max`	The maximum distribution of payload sizes of bulk requests, in bytes. **Relevant statistics**: Max **Dimension**: `PipelineName`

Metric suffix	Description
`bulkRequestLatency.count`	A count of data points recorded while requests are sent to the plugin, including retries. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`bulkRequestLatency.sum`	The total latency of requests sent to the plugin, including retries, in milliseconds. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`bulkRequestLatency.max`	The maximum latency of requests sent to the plugin, including retries, in milliseconds. **Relevant statistics**: Max **Dimension**: `PipelineName`
`s3.dlqS3RecordsSuccess.count`	The number of records successfully sent to the S3 dead letter queue. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3.dlqS3RecordsFailed.count`	The number of recourds that failed to be sent to the S3 dead letter queue. **Relevant statistics**: Sum **Dimension**: `PipelineName`

Metric suffix	Description
`s3.dlqS3RequestSuccess.count`	The number of successful requests to the S3 dead letter queue. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3.dlqS3RequestFailed.count`	The number of failed requests to the S3 dead letter queue. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3.dlqS3RequestLatency.count`	A count of data points recorded while requests are sent to the S3 dead letter queue, including retries. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3.dlqS3RequestLatency.sum`	The total latency of requests sent to the S3 dead letter queue, including retries, in milliseconds. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3.dlqS3RequestLatency.max`	The maximum latency of requests sent to the S3 dead letter queue, including retries, in milliseconds. **Relevant statistics**: Max **Dimension**: `PipelineName`

Metric suffix	Description
`s3.dlqS3RequestSizeBytes.count`	A count of the distribution of payload sizes of requests to the S3 dead letter queue, in bytes. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3.dlqS3RequestSizeBytes.sum`	The total distribution of payload sizes of requests to the S3 dead letter queue, in bytes. **Relevant statistics**: Sum **Dimension**: `PipelineName`
`s3.dlqS3RequestSizeBytes.max`	The maximum distribution of payload sizes of requests to the S3 dead letter queue, in bytes. **Relevant statistics**: Max **Dimension**: `PipelineName`

System and metering metrics

The following metrics apply to the overall OpenSearch Ingestion system. These metrics aren't prefixed by anything.

Metric	Description
`system.cpu.usage.value`	The percentage of available CPU usage for all data nodes. **Relevant statistics**: Average **Dimension**: `PipelineName` , `area`, `id`
`system.cpu.count.value`	The total amount of CPU usage for all data nodes. **Relevant statistics**: Average

Metric	Description
	Dimension: `PipelineName , area, id`
`jvm.memory.max.value`	The maximum amount of memory that can be used for memory management, in bytes. **Relevant statistics**: Average **Dimension**: `PipelineName , area, id`
`jvm.memory.used.value`	The total amount of memory used, in bytes. **Relevant statistics**: Average **Dimension**: `PipelineName , area, idsigna`
`jvm.memory.committ ed.value`	The amount of memory that is committed for use by the Java virtual machine (JVM), in bytes. **Relevant statistics**: Average **Dimension**: `PipelineName , area, id`
`computeUnits`	The number of Ingestion OpenSearch Compute Units (Ingestion OCUs) in use by a pipeline. **Relevant statistics**: Max, Sum, Average **Dimension**: `PipelineName`

Best practices for Amazon OpenSearch Ingestion

This topic provides best practices for creating and managing Amazon OpenSearch Ingestion pipelines 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.

Topics

General best practices

The following general best practices apply to creating and managing pipelines.

- To ensure high availability, configure VPC pipelines with two or three subnets. If you only deploy a pipeline in one subnet and the Availability Zone goes down, you won't be able to ingest data.
- Within each pipeline, we recommend limiting the number of sub-pipelines to 5 or fewer.
- If you're using the S3 source plugin, use evenly-sized S3 files for optimal performance.
- If you're using the S3 source plugin, add 30 seconds of additional visibility timeout for every 0.25 GB of file size in the S3 bucket for optimal performance.
- Include a dead-letter queue (DLQ) in your pipeline configuration so that you can offload failed events and make them accessible for analysis. If your sinks reject data due to incorrect mappings or other issues, you can route the data to the DLQ in order to troubleshoot and fix the issue.

Recommended CloudWatch alarms

CloudWatch alarms perform an action when a CloudWatch metric exceeds a specified value for some amount of time. For example, you might want AWS to email you if your cluster health status is `red` for longer than one minute. This section includes some recommended alarms for Amazon OpenSearch Ingestion and how to respond to them.

For more information about configuring alarms, see Creating Amazon CloudWatch Alarms in the *Amazon CloudWatch User Guide*.

Alarm	Issue
`computeUnits` maximum is = the configured `maxUnits` for 15 minute, 3 consecutive times	The pipeline has reached the maximum capacity and might require a `maxUnits` update. Increase the maximum capacity of your pipeline
`opensearch.documentErrors.count` sum is = *`{sub_pipe`*	The pipeline is unable to write to the OpenSearch sink. Check the pipeline permissions and confirm that the domain or collection is healthy. You can also check the dead letter queue (DLQ) for failed events, if it's configured.

Alarm	Issue
line_name } .opensearch.recordsIn.count sum for 1 minute, 1 consecutive time	
bulkRequestLatency.max max is >= *x* for 1 minute, 1 consecutive time	The pipeline is experiencing high latency sending data to the OpenSearch sink. This is likely due to the sink being undersized, or a poor sharding strategy, which is causing the sink to fall behind. Sustained high latency can impact pipeline performance and will likely lead to backpressure on the clients.
httpAuthFailure.count sum >= 1 for 1 minute, 1 consecutive time	Ingestion requests are not being authenticated. Confirm that all clients have Signature Version 4 authentication enabled correctly.
system.cpu.usage.value average >= 80% for 15 minutes, 3 consecutive times	Sustained high CPU usage can be problematic. Consider increasing the maximum capacity for the pipeline.
bufferUsage.value average >= 80% for 15 minutes, 3 consecutive times	Sustained high buffer usage can be problematic. Consider increasing the maximum capacity for the pipeline.

Other alarms you might consider

Consider configuring the following alarms depending on which Amazon OpenSearch Ingestion features you regularly use.

Alarm	Issue
dynamodb.exportJobFailure.count sum 1	The attempt to trigger an export to Amazon S3 failed.
opensearch.EndtoEndLatency.avg average > X for 15 minutes, 4 consecutive times	The EndtoEndLatency is higher than desired for reading from DynamoDB streams. This could be caused by an underscaled OpenSearch cluster or a maximum pipeline OCU capacity that is too low for the WCU throughput on the DynamoDB table. EndtoEndLatency will be higher after an export but should decrease over time as it catches up to the latest DynamoDB streams.
dynamodb.changeEventsProcessed.count sum == 0 for X minutes	No records are being gathered from DynamoDB streams. This could be caused by to no activity on the table, or an issue accessing DynamoDB streams.
opensearch.s3.dlqS3RecordsSuccess.count sum >= opensearch.documentSuccess.count sum for 1 minute, 1 consecutive time	A larger number of records are being sent to the DLQ than the OpenSearch sink. Review the OpenSearch sink plugin metrics to investigate and determine the root cause.
grok.grokProcessingTimeouts.count sum = recordsIn.count sum for 1 minute, 5 consecutive times	All data is timing out while the Grok processor is trying to pattern match. This is likely impacting performance and slowing your pipeline down. Consider adjusting your patterns to reduce timeouts.

Alarm	Issue
`grok.grokProcessingErrors.count` sum is >= 1 for 1 minute, 1 consecutive time	The Grok processor is failing to match patterns to the data in the pipeline, resulting in errors. Review your data and Grok plugin configurations to ensure the pattern matching is expected.
`grok.grokProcessingMismatch.count` sum = recordsIn.count sum for 1 minute, 5 consecutive times	The Grok processor is unable to match patterns to the data in the pipeline. Review your data and Grok plugin configurations to ensure the pattern matching is expected.
`date.dateProcessingMatchFailure.count` sum = recordsIn.count sum for 1 minut, 5 consecutive times	The Date processor is unable to match any patterns to the data in the pipeline. Review your data and Date plugin configurations to ensure the pattern is expected.
`s3.s3ObjectsFailed.count` sum >= 1 for 1 minute, 1 consecutive time	This issue is either occurring because the S3 object doesn't exist, or the pipeline has insufficient privileges. Reivew the `s3Objects NotFound.count` and `s3ObjectsAccessDenied.count` metrics to determine the root cause. Confirm that the S3 object exists and/or update the permissions.
`s3.sqsMessagesFailed.count` sum >= 1 for 1 minute, 1 consecutive time	The S3 plugin failed to process an Amazon SQS message. If you have a DLQ enabled on your SQS queue, review the failed message. The queue might be receiving invalid data that the pipeline is attempting to process.

Alarm	Issue
`http.badRequests.count` sum >= 1 for 1 minute, 1 consecutive times	The client is sending a bad request. Confirm that all clients are sending the proper payload.
`http.requestsTooLarge.count` sum >= 1 for 1 minute, 1 consecutive time	Requests from the HTTP source plugin contain too much data, which is exceeding the buffer capacity. Adjust the batch size for your clients.
`http.internalServerError.count` sum >= 0 for 1 minute, 1 consecutive time	The HTTP source plugin is having trouble receiving events.
`http.requestTimeouts.count` sum >= 0 for 1 minute, 1 consecutive time	Source timeouts are likely the result of the pipeline being underprovisioned. Consider increasing the pipeline `maxUnits` to handle additional workload.
`otel_trace.badRequests.count` sum >= 1 for 1 minute, 1 consecutive time	The client is sending a bad request. Confirm that all clients are sending the proper payload.

Alarm	Issue
`otel_trace.requestsTooLarge.count` sum >= 1 for 1 minute, 1 consecutive time	Requests from the Otel Trace source plugin contain too much data, which is exceeding the buffer capacity. Adjust the batch size for your clients.
`otel_trace.internalServerError.count` sum >= 0 for 1 minute, 1 consecutive time	The Otel Trace source plugin is having trouble receiving events.
`otel_trace.requestTimeouts.count` sum >= 0 for 1 minute, 1 consecutive time	Source timeouts are likely the result of the pipeline being underprovisioned. Consider increasing the pipeline `maxUnits` to handle additional workload.
`otel_metrics.requestTimeouts.count` sum >= 0 for 1 minute, 1 consecutive time	Source timeouts are likely the result of the pipeline being underprovisioned. Consider increasing the pipeline `maxUnits` to handle additional workload.

Amazon OpenSearch Serverless

Amazon OpenSearch Serverless is an on-demand, auto-scaling configuration for Amazon OpenSearch Service. Unlike provisioned OpenSearch domains, which require manual capacity management, an OpenSearch Serverless collection automatically scales compute resources based on your application's needs.

OpenSearch Serverless offers a cost-effective solution for workloads that are infrequent, intermittent, or unpredictable. It optimizes costs by automatically scaling compute capacity based on your application's usage. Serverless collections use the same high-capacity, distributed, and highly available storage volume as provisioned OpenSearch Service domains.

OpenSearch Serverless collections are always encrypted. You can choose the encryption key, but you can't disable encryption. For more information, see the section called “Encryption”

Benefits

OpenSearch Serverless has the following benefits:

- **Simpler than provisioned** – OpenSearch Serverless removes much of the complexity of managing OpenSearch clusters and capacity. It automatically sizes and tunes your clusters, and takes care of shard and index lifecycle management. It also manages service software updates and OpenSearch version upgrades. All updates and upgrades are non-disruptive.
- **Cost-effective** – When you use OpenSearch Serverless, you only pay for the resources that you consume. This removes the need for upfront provisioning and overprovisioning for peak workloads.
- **Highly available** – OpenSearch Serverless supports production workloads with redundancy to protect against Availability Zone outages and infrastructure failures.
- **Scalable** – OpenSearch Serverless automatically scales resources to maintain consistently fast data ingestion rates and query response times.

What is Amazon OpenSearch Serverless?

Amazon OpenSearch Serverless is an on-demand, serverless option for Amazon OpenSearch Service that eliminates the operational complexity of provisioning, configuring, and tuning OpenSearch clusters. It’s ideal for organizations that prefer not to self-manage their clusters or

lack the dedicated resources and expertise to operate large-scale deployments. With OpenSearch Serverless, you can search and analyze large volumes of data without managing the underlying infrastructure.

An OpenSearch Serverless *collection* is a group of OpenSearch indexes that work together to support a specific workload or use case. Collections simplify operations compared to self-managed OpenSearch clusters, which require manual provisioning.

Collections use the same high-capacity, distributed, and highly available storage as provisioned OpenSearch Service domains, but further reduce complexity by eliminating manual configuration and tuning. Data within a collection is encrypted in transit. OpenSearch Serverless also supports OpenSearch Dashboards, providing an interface for data analysis.

Currently, serverless collections run OpenSearch version 2.0.x. As new versions are released, OpenSearch Serverless automatically upgrades collections to incorporate new features, bug fixes, and performance improvements.

OpenSearch Serverless supports the same ingest and query API operations as the OpenSearch open source suite, so you can continue to use your existing clients and applications. Your clients must be compatible with OpenSearch 2.x in order to work with OpenSearch Serverless. For more information, see the section called “Ingesting data into collections”.

Topics

Use cases for OpenSearch Serverless

OpenSearch Serverless supports two primary use cases:

- **Log analytics** - The log analytics segment focuses on analyzing large volumes of semi-structured, machine-generated time series data for operational and user behavior insights.

- **Full-text search** - The full-text search segment powers applications in your internal networks (content management systems, legal documents) and internet-facing applications, such as ecommerce website content search.

When you create a collection, you choose one of these use cases. For more information, see the section called “Choosing a collection type”.

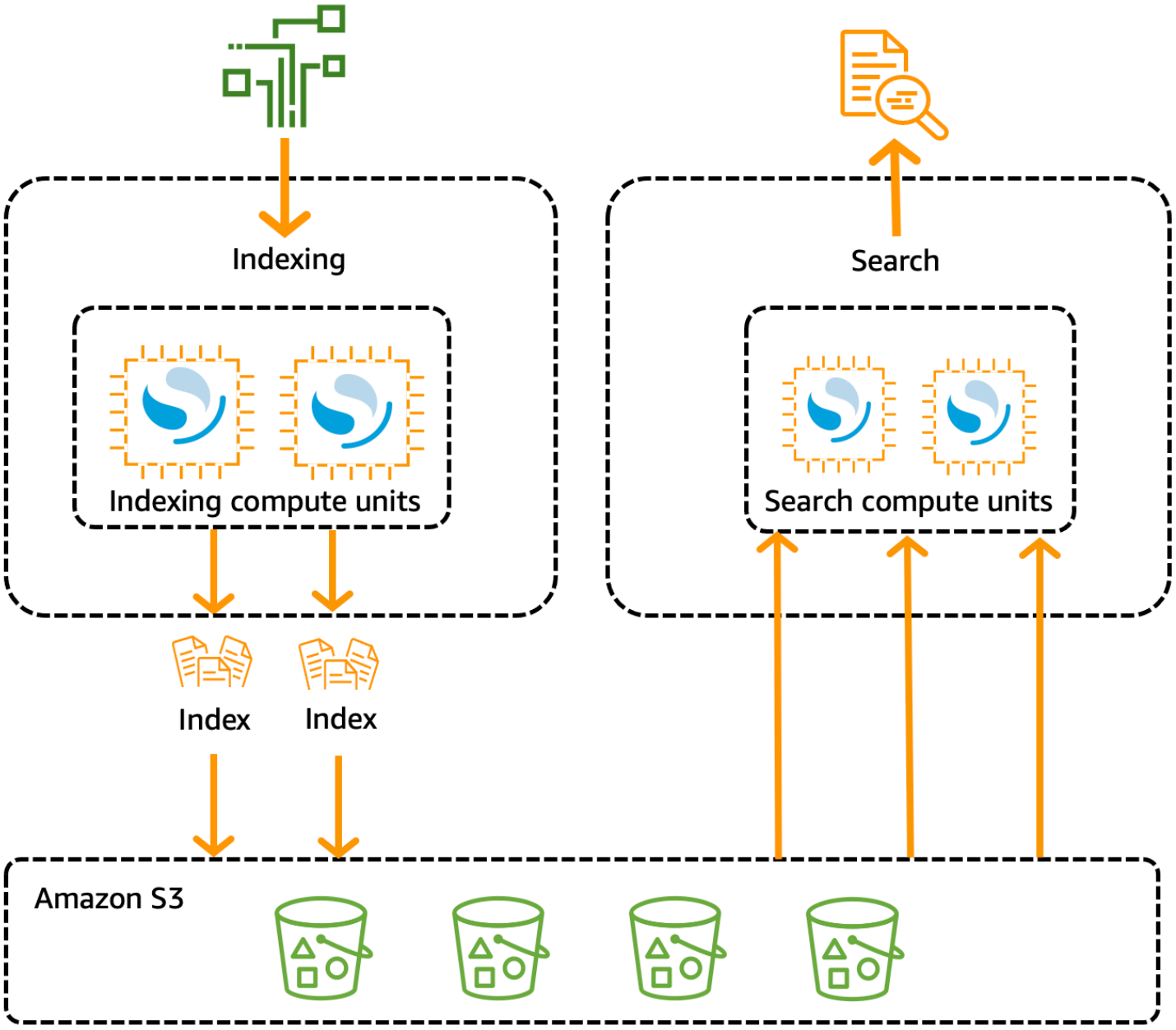
How it works

Traditional OpenSearch clusters have a single set of instances that perform both indexing and search operations, and index storage is tightly coupled with compute capacity. By contrast, OpenSearch Serverless uses a cloud-native architecture that separates the indexing (ingest) components from the search (query) components, with Amazon S3 as the primary data storage for indexes.

This decoupled architecture lets you scale search and indexing functions independently of each other, and independently of the indexed data in S3. The architecture also provides isolation for ingest and query operations so that they can run concurrently without resource contention.

When you write data to a collection, OpenSearch Serverless distributes it to the *indexing* compute units. The indexing compute units ingest the incoming data and move the indexes to S3. When you perform a search on the collection data, OpenSearch Serverless routes requests to the *search* compute units that hold the data being queried. The search compute units download the indexed data directly from S3 (if it's not already cached locally), run search operations, and perform aggregations.

The following image illustrates this decoupled architecture:



OpenSearch Serverless compute capacity for data ingestion, searching, and querying are measured in OpenSearch Compute Units (OCUs). Each OCU is a combination of 6 GiB of memory and corresponding virtual CPU (vCPU), as well as data transfer to Amazon S3. Each OCU includes enough hot ephemeral storage for 120 GiB of index data.

When you create your first collection, OpenSearch Serverless instantiates two OCUs—one for indexing and one for search. To ensure high availability, it also launches a standby set of nodes in another Availability Zone. For development and testing purposes, you can disable the **Enable redundancy** setting for a collection, which eliminates the two standby replicas and only

instantiates two OCUs. By default, the redundant active replicas are enabled, which means that a total of four OCUs are instantiated for the first collection in an account.

These OCUs exist even when there's no activity on any collection endpoints. All subsequent collections share these OCUs. When you create additional collections in the same account, OpenSearch Serverless only adds additional OCUs for search and ingest as needed to support the collections, according to the capacity limits that you specify. Capacity scales back down as your compute usage decreases.

For information about how you're billed for these OCUs, see the section called “Pricing”.

Choosing a collection type

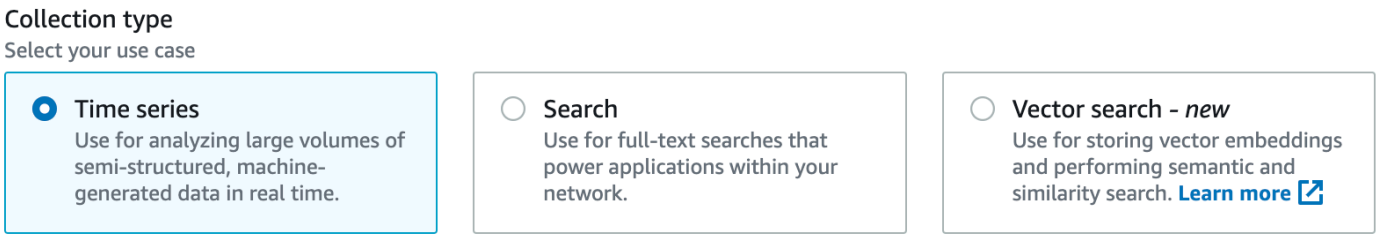
OpenSearch Serverless supports three primary collection types:

Time series – The log analytics segment that analyzes large volumes of semi-structured, machine-generated data in real-time, providing insights into operations, security, user behavior, and business performance.

Search – Full-text search that enables applications within internal networks, such as content management systems and legal document repositories, as well as internet-facing applications like e-commerce site search and content discovery.

Vector search – Semantic search on vector embeddings simplifies vector data management and enables machine learning (ML)-augmented search experiences. It supports generative AI applications such as chatbots, personal assistants, and fraud detection.

You choose a collection type when you first create a collection:



The collection type that you choose depends on the kind of data that you plan to ingest into the collection, and how you plan to query it. You can't change the collection type after you create it.

The collection types have the following notable **differences**:

- For *search* and *vector search* collections, all data is stored in hot storage to ensure fast query response times. *Time series* collections use a combination of hot and warm storage, where the most recent data is kept in hot storage to optimize query response times for more frequently accessed data.
- For *time series* and *vector search* collections, you can't index by custom document ID or update by upsert requests. This operation is reserved for search use cases. You can update by document ID instead. For more information, see the section called “Supported OpenSearch API operations and permissions”.
- For *search* and *time series* collections, you can't use k-NN type indexes.

Pricing

AWS charges you for the following OpenSearch Serverless components:

- Data ingestion compute
- Search and query compute
- Storage retained in Amazon S3

It bills OCU on an hourly basis, with per-second granularity. In your account statement, you see an entry for compute in OCU-hours with a label for data ingestion and a label for search. AWS also bills you on a monthly basis for data stored in Amazon S3. It doesn't charge you for using OpenSearch Dashboards.

You're billed for a minimum of 2 OCUs (0.5 OCU x 2) for ingestion and 1 OCU (0.5 OCU x 2) for search when you create a collection and enable redundant active replicas. You're billed for a minimum of 1 OCU (0.5 OCU x 2) for the first collection in your account if you disable redundant active replicas. All subsequent collections can share those OCUs.

OpenSearch Serverless adds additional OCUs in increments of 1 OCU based on the compute power and storage needed to support your collections. You can configure a maximum number of OCUs for your account in order to control costs.

> **Note**
>
> Collections with unique AWS KMS keys can't share OCUs with other collections.

OpenSearch Serverless attempts to use the minimum required resources to account for changing workloads. The number of OCUs provisioned at any time can vary and isn't exact. Over time, the algorithm that OpenSearch Serverless uses will continue to improve in order to better minimize system usage.

For full pricing details, see Amazon OpenSearch Service pricing.

Supported AWS Regions

OpenSearch Serverless is available in a subset of AWS Regions that OpenSearch Service is available in. For a list of supported Regions, see Amazon OpenSearch Service endpoints and quotas in the *AWS General Reference*.

Limitations

OpenSearch Serverless has the following limitations:

- Some OpenSearch API operations aren't supported. See the section called "Supported OpenSearch API operations and permissions".
- Some OpenSearch plugins aren't supported. See the section called "Supported OpenSearch plugins".
- There's currently no way to automatically migrate your data from a managed OpenSearch Service domain to a serverless collection. You must reindex your data from a domain to a collection.
- Cross-account access to collections isn't supported. You can't include collections from other accounts in your encryption or data access policies.
- Custom OpenSearch plugins aren't supported.
- You can't take or restore snapshots of OpenSearch Serverless collections.
- Cross-Region search and replication aren't supported.
- There are limits on the number of serverless resources that you can have in a single account and Region. See OpenSearch Serverless quotas.
- The refresh interval for indexes in vector search collections is approximately 60 seconds. The refresh interval for indexes in search and time series collections is approximately 10 seconds.
- The number of shards, number of intervals, and refresh interval are not modifiable and are handled by OpenSearch Serverless. The sharding strategy is based off the collection type and traffic. For example, a time series collection scales primary shards based on write traffic bottlenecks.

- Geospatial features available on OpenSearch versions up to 2.1 are supported.

Comparing OpenSearch Service and OpenSearch Serverless

In OpenSearch Serverless, some concepts and features are different than their corresponding feature for a provisioned OpenSearch Service domain. For example, one important difference is that OpenSearch Serverless doesn't have the concept of a cluster or node.

The following table describes how important features and concepts in OpenSearch Serverless differ from the equivalent feature in a provisioned OpenSearch Service domain.

Feature	OpenSearch Service	OpenSearch Serverless
Domains versus collections	Indexes are held in *domains*, which are pre-provisioned OpenSearch clusters. For more information, see *Creating and managing domains*.	Indexes are held in *collections*, which are logical groupings of indexes that represent a specific workload or use case. For more information, see the section called "Creating, listing, and deleting collections".
Node types and capacity management	You build a cluster with node types that meet your cost and performance specifications. You must calculate your own storage requirements and choose an instance type for your domain. For more information, see the section called "Sizing domains".	OpenSearch Serverless automatically scales and provisions additional compute units for your account based on your capacity usage. For more information, see the section called "Managing capacity limits".
Billing	You pay for each hour of use of an EC2 instance and for the cumulative size of any EBS storage volumes attached to your instances.	You're charged in OCU-hours for compute for data ingestion, compute for search and query, and storage retained in S3.

Feature	OpenSearch Service	OpenSearch Serverless
	For more information, see the section called “Pricing”.	For more information, see the section called “Pricing”.
Encryption	Encryption at rest is *optional* for domains. For more information, see the section called “Encryption at rest”.	Encryption at rest is *required* for collections. For more information, see the section called “Encryption”.
Data access control	Access to the data within domains is determined by IAM policies and fine-grained access control.	Access to data within collections is determine d by data access policies.
Supported OpenSearch operations	OpenSearch Service supports a subset of all of the OpenSearch API operations. For more information, see the section called “Supported operations”.	OpenSearch Serverless supports a different subset of OpenSearch API operations. For more information, see the section called “Supported operations and plugins”.
Dashboards sign-in	Sign in with a username and password. For more information, see the section called “Accessin g OpenSearch Dashboards as the master user”.	If you're logged into the AWS console and navigate to your Dashboard URL, you'll automatically log in. For more information, see the section called “Accessing OpenSearch Dashboards”.
APIs	Interact programmatically with OpenSearch Service using the OpenSearch Service API operations.	Interact programmatically with OpenSearch Serverless using the OpenSearch Serverless API operations.

Feature	OpenSearch Service	OpenSearch Serverless
Network access	Network settings for a domain apply to the domain endpoint as well as the OpenSearch Dashboards endpoint. Network access for both is tightly coupled.	Network settings for the domain endpoint and the OpenSearch Dashboards endpoint are decoupled. You can choose to not configure network access for OpenSearch Dashboards. For more information, see the section called "Network access".
Signing requests	Use the OpenSearch high and low-level REST clients to sign requests. Specify the service name as `es`.	At this time, OpenSearch Serverless supports a subset of clients that OpenSearch Service supports. When you sign requests, specify the service name as `aoss`. The `x-amz-content-sha256` header is required. For more information, see the section called "Other clients".
OpenSearch version upgrades	You manually upgrade your domains as new versions of OpenSearch become available. You're responsible for ensuring that your domain meets the upgrade requirements, and that you've addressed any breaking changes.	OpenSearch Serverless automatically upgrades your collections to new OpenSearch versions. Upgrades don't necessarily happen as soon as a new version is available.
Service software updates	You manually apply service software updates to your domain as they become available.	OpenSearch Serverless automatically updates your collections to consume the latest bug fixes, features, and performance improvements.

Feature	OpenSearch Service	OpenSearch Serverless
VPC access	You can provision your domain within a VPC. You can also create additional OpenSearch Service-managed VPC endpoints to access the domain.	You create one or more OpenSearch Serverless-managed VPC endpoints for your account. Then, you include these endpoints within network policies.
SAML authentication	You enable SAML authentication on a per-domain basis. For more information, see the section called "SAML authentication for OpenSearch Dashboards".	You configure one or more SAML providers at the account level, then you include the associated user and group IDs within data access policies. For more information, see the section called "SAML authentication".
Transport Layer Security (TLS)	OpenSearch Service supports TLS 1.2 but it is recommend you use TLS 1.3.	OpenSearch Serverless supports TLS 1.2 but it is recommended you use TLS 1.3.

Tutorial: Getting started with Amazon OpenSearch Serverless

This tutorial walks you through the basic steps to get an Amazon OpenSearch Serverless *search* collection up and running quickly. A search collection allows you to power applications in your internal networks and internet-facing applications, such as ecommerce website search and content search.

To learn how to use a *vector search* collection, see the section called "Working with vector search collections". For more detailed information about using collections, see the section called "Creating, listing, and deleting collections" and the other topics within this guide.

You'll complete the following steps in this tutorial:

1. Configure permissions
2. Create a collection

3. Upload and search data
4. Delete the collection

> **Note**
>
> It is recommened that you use only ASCII characters for your `IndexName`. If you do not use ASCII characters for your `IndexName`, the `IndexName` in CloudWatch metrics will be converted to a URL encoded format for Non-ASCII characters.

Step 1: Configure permissions

In order to complete this tutorial, and to use OpenSearch Serverless in general, you must have the correct IAM permissions. In this tutorial, you will create a collection, upload and search data, and then delete the collection.

Your user or role must have an attached identity-based policy with the following minimum permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "aoss:CreateCollection",
        "aoss:ListCollections",
        "aoss:BatchGetCollection",
        "aoss:DeleteCollection",
        "aoss:CreateAccessPolicy",
        "aoss:ListAccessPolicies",
        "aoss:UpdateAccessPolicy",
        "aoss:CreateSecurityPolicy",
        "aoss:GetSecurityPolicy",
        "aoss:UpdateSecurityPolicy",
        "iam:ListUsers",
        "iam:ListRoles"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
```

```
}
```

For more information about OpenSearch Serverless IAM permissions, see the section called "Identity and Access Management".

Step 2: Create a collection

A collection is a group of OpenSearch indexes that work together to support a specific workload or use case.

To create an OpenSearch Serverless collection

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Choose **Collections** in the left navigation pane and choose **Create collection**.
3. Name the collection **movies**.
4. For collection type, choose **Search**. For more information, see Choosing a collection type.
5. For **Security**, choose **Standard create**.
6. Under **Encryption**, select **Use AWS owned key**. This is the AWS KMS key that OpenSearch Serverless will use to encrypt your data.
7. Under **Network**, configure network settings for the collection.
   - For the access type, select **Public**.
   - For the resource type, choose both **Enable access to OpenSearch endpoints** and **Enable access to OpenSearch Dashboards**. Since you'll upload and search data using OpenSearch Dashboards, you need to enable both.
8. Choose **Next**.
9. For **Configure data access**, set up access settings for the collection. Data access policies allow users and roles to access the data within a collection. In this tutorial, we'll provide a single user the permissions required to index and search data in the *movies* collection.

   Create a single rule that provides access to the *movies* collection. Name the rule **Movies collection access**.
10. Choose **Add principals**, **IAM users and roles** and select the user or role that you'll use to sign in to OpenSearch Dashboards and index data. Choose **Save**.
11. Under **Index permissions**, select all of the permissions.

12. Choose **Next**.
13. For the access policy settings, choose **Create a new data access policy** and name the policy **movies**.
14. Choose **Next**.
15. Review your collection settings and choose **Submit**. Wait several minutes for the collection status to become `Active`.

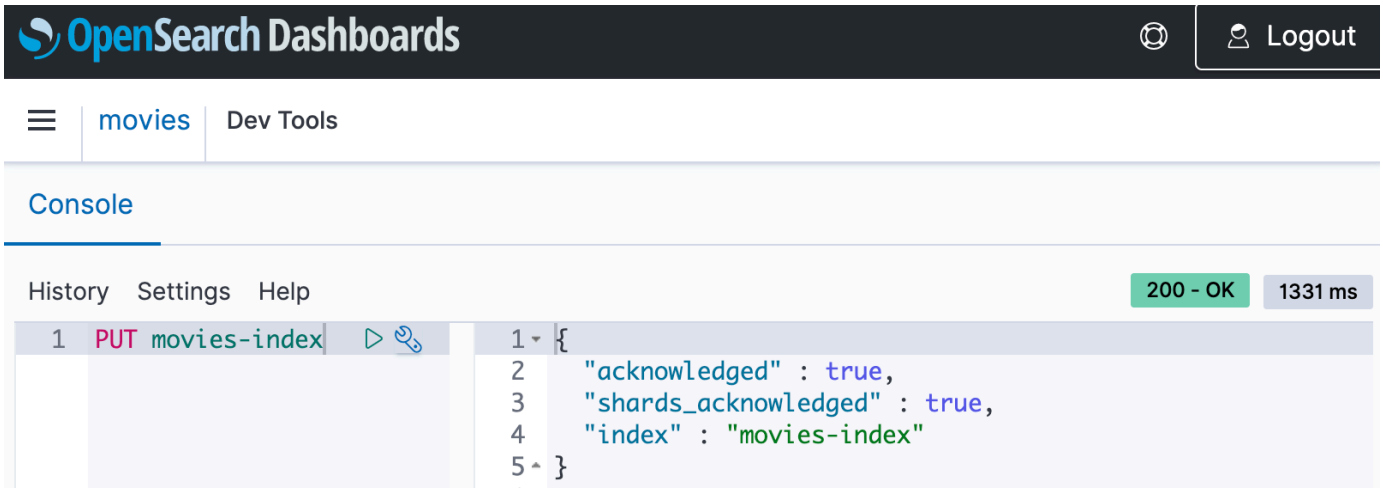
Step 3: Upload and search data

You can upload data to an OpenSearch Serverless collection using Postman or cURL. For brevity, these examples use **Dev Tools** within the OpenSearch Dashboards console.

To index and search data in the movies collection

1. Choose **Collections** in the left navigation pane and choose the **movies** collection to open its details page.
2. Choose the OpenSearch Dashboards URL for the collection. The URL takes the format `https://dashboards.{region}.aoss.amazonaws.com/_login/?collectionId={collection-id}`.
3. Within OpenSearch Dashboards, open the left navigation pane and choose **Dev Tools**.
4. To create a single index called *movies-index*, send the following request:

```
PUT movies-index
```



5. To index a single document into *movies-index*, send the following request:

```
PUT movies-index/_doc/1
{
  "title": "Shawshank Redemption",
  "genre": "Drama",
  "year": 1994
}
```

6. To search data in OpenSearch Dashboards, you need to configure at least one index pattern. OpenSearch uses these patterns to identify which indexes you want to analyze. Open the left navigation pane, choose **Stack Management**, choose **Index Patterns**, and then choose **Create index pattern**. For this tutorial, enter *movies*.
7. Choose **Next step** and then choose **Create index pattern**. After the pattern is created, you can view the various document fields such as `title` and `genre`.
8. To begin searching your data, open the left navigation pane again and choose **Discover**, or use the search API within Dev Tools.

Step 4: Delete the collection

Because the *movies* collection is for test purposes, make sure to delete it when you're done experimenting.

To delete an OpenSearch Serverless collection

1. Go back to the **Amazon OpenSearch Service** console.
2. Choose **Collections** in the left navigation pane and select the **movies** collection.
3. Choose **Delete** and confirm deletion.

Next steps

Now that you know how to create a collection and index data, you might want to try some of the following exercises:

- See more advanced options for creating a collection. For more information, see the section called “Creating, listing, and deleting collections”.
- Learn how to configure security policies to manage collection security at scale. For more information, see the section called “Security in OpenSearch Serverless”.

- Discover other ways to index data into collections. For more information, see the section called "Ingesting data into collections".

Using OpenSearch SQL with Amazon OpenSearch Serverless

You can now use OpenSearch SQL plugin as an alternative way to query in Amazon OpenSearch Serverless. OpenSearch SQL plugin provides SQL query and Piped Processing Language (PPL) capabilites that enable you to execute read-only SQL and PPL queries as well as execute SQL joins and subqueries.

Additionally, you can perform deep pagination using sql and ppl queries, for more information on this, see Paginating results.

The following features are supported for OpenSearch SQL plugin:

- SQL pagination
- SQL join queries
- `WHERE` Clause/Filtering
- SQL queries

The following features are not supported for OpenSearch SQL plugin:

- Delete queries
- SQL stats API

You do not need to install anything to enable use of OpenSearch SQL plugin. For more information on use of the plugin, see Introduction to OpenSearch Plugins.

Amazon OpenSearch Serverless support use of the following sql and ppl APIs:

- `POST /_plugins/_sql`
- `POST /_plugins/_ppl`
- `POST /_plugins/_sql/_explain`
- `POST /_plugins/_ppl/_explain`
- `POST /_plugins/_sql/close`

All APIs are managed via `aoss:ReadDocument` data access control. For more information on supported APIs, see Supported operations in plugins in Amazon OpenSearch Serverless.

Querying using SQL plugin

OpenSearch SQL plugin converts a query string via SQL query engine and converts it to OpenSearch Service DSL to make a `SearchRequest` The following is an example of a SQL query. Here is an example query:

```
POST _plugins/_sql
{
  "query": "SELECT * FROM my-index LIMIT 50"
}
```

Querying using PPL plugin

OpenSearch SQL plugin converts a query string via SQL query engine and converts it to OpenSearch Service DSL to make a `SearchRequest` The following is an example of a SQL query. Here is an example query:

```
POST _plugins/_ppl
{
  "query": "source=my-index | fields firstname, age | head 50"
}
```

Pagination using OpenSearch SQL plugin

OpenSearch SQL plugin converts a query string via SQL query engine and converts it to OpenSearch Service DSL to make a `SearchRequest` The following is an example of a SQL query. Here is an example query:

```
POST _plugins/_sql
{
  "query": "SELECT * FROM my-index LIMIT 50"
}
```

Using Pit with Amazon OpenSearch Serverless

You can now use the PIT (Point in Time) plugin to run different queries against a dataset that is fixed in time. Typically, if you run a query on an index multiple times, the same query may return a different result due to data being continually indexed, updated, and deleted. If you need to run a query against the same data, you can preserve that data's state by creating a PIT. To learn more about Point in Time, see Point in Time.

Amazon OpenSearch Serverless support use of the following PIT APIs:

- `POST /<target_indexes>/_search/point_in_time`
- `GET /_search/point_in_time/_all`
- `DELETE /_search/point_in_time/_all`
- `DELETE /_search/point_in_time`

All APIs are managed via `aoss:ReadDocument` data access control. For more information on supported APIs, see Supported operations in plugins in Amazon OpenSearch Serverless.

OpenSearch offers various methods of pagination that can be can be coupled with Point in Time to perform deep pagination of search results. These features include the ability to specify the `from` and `size` paramaters of your search and the `search_after` parameter for deep pagination of search results.To learn more about pagination, see Paginate results.

Point in Time (PIT) with `search_after` is recommended pagination method in Amazon OpenSearch Serverless, especially for deep pagination. It bypasses the limitations of all other methods because it operates on a dataset that is frozen in time, it is not bound to a query, and it supports consistent pagination going forward.

You can also slice a PIT search into multiple slices if you want to jump from a page to a non-consecutive page by using the `slice.id` and `slice.max` parameters. To learn more on search slicing, see Search slicing.

Slicing performs better when you specify a field parameter to the slice. The field specified must be a numeric doc value type such as short, integer, or long. This field is used to create slice buckets, so there should be at least as many unique values as slices, and those values should uniformly distributed.

Create a PIT

Create a PIT using the following example:

```
POST /my-index-1/_search/point_in_time?keep_alive=100m

{
    "pit_id":
 "o123QQEEeEeeeE5eEEeeEEEeEEEeEeEEEE45eee3E3EeaEEeeEE1EEEeEeeEEeEeeEEeEEEmEEE2EEE6eEe1eEEeEeeAA
    "_shards": {
        "total": 1,
        "successful": 1,
        "skipped": 0,
        "failed": 0
    },
    "creation_time": 1658146050064
}
```

Selecting parameter for pagination with PIT

For effective pagination, a unique `sort` parameter must be selected. Without a unique `sort` parameter, documents may be skipped during pagination. You must select a tiebreaker field, or a set of tiebreaker fields, which are random enough to ensure no two documents have the same `sort` ordering. OpenSearch discourages the use of `_id` for tiebreakers because it requires `fielddata` which is very cost-intensive. If no tiebreaker is available, it is recommended you index documents with additional random integer field which can later be used as a tiebreaker during search.

Search using the PIT context id and the `search_after` parameter to retrieve the next page of results as shown in the following example:

Pagination with PIT

Search using the PIT context id and the `search_after` parameter to retrieve the next page of results as shown in the following example:

```
GET /_search
{
    "size": 10000,
    "query": {
```

```
        "match" : {
            "user.id" : "elkbee"
        }
    },
    "pit": {
        "id":
 "46ToAwMDaWR5BXV1aWQyKwZub2RlXzMAAAAAAAAAACoBYwADaWR4BXV1aWQxAgZub2RlXzEAAAAAAAAAAAEBYQADaWR5B
        "keep_alive": "100m"
    },
    "sort": [
        {"@timestamp": {"order": "asc"}}
    ],
    "search_after": [
        "2021-05-20T05:30:04.832Z"
    ]
}
```

Extend PIT using search request

To extend a point in time search, you will need to provide a `keep_alive` parameter in the `"pit"` object during search. See the following example:

```
GET /_search
{
    "size": 10000,
    "query": {
        "match" : {
            "user.id" : "elkbee"
        }
    },
    "pit": {
        "id":
 "46ToAwMDaWR5BXV1aWQyKwZub2RlXzMAAAAAAAAAACoBYwADaWR4BXV1aWQxAgZub2RlXzEAAAAAAAAAAAEBYQADaWR5B
        "keep_alive": "100m"
    },
    "sort": [
        {"@timestamp": {"order": "asc"}}
    ],
    "search_after": [
        "2021-05-20T05:30:04.832Z"
    ]
}
```

List all PITs

To list all PITs, see the following example:

```
GET /_search/point_in_time/_all

{
    "pits": [{
        "pit_id":
 "o463QQEPbXktaW5kZXgtMDAwMDAxFnNOWU43ckt3U3IyaFVpbGE1UWEtMncAFjFyeXBsRGJmVFM2RTB6eVg1aVVqQncAA
        "creation_time": 1658146048666,
        "keep_alive": 6000000
    },
    {
        "pit_id":
 "o463QQEPbXktaW5kZXgtMDAwMDAxFnNOWU43ckt3U3IyaFVpbGE1UWEtMncAFjFyeXBsRGJmVFM2RTB6eVg1aVVqQncAA
        "creation_time": 1658146050064,
        "keep_alive": 6000000
    }
    ]
}
```

Delete a PIT

To delete a PIT, see the following example

```
DELETE /_search/point_in_time
{
    "pit_id": [

 "o463QQEPbXktaW5kZXgtMDAwMDAxFkhGN09fMVlPUkVPLXh6MUExZ1hpaEEAFjBGbmVEZHdGU1EtaFhhUFc4ZkR5cWcAA

 "o463QQEPbXktaW5kZXgtMDAwMDAxFkhGN09fMVlPUkVPLXh6MUExZ1hpaEEAFjBGbmVEZHdGU1EtaFhhUFc4ZkR5cWcAA
    ]
}

{
    "pits": [
        {
            "successful": true,
            "pit_id":
 "o463QQEPbXktaW5kZXgtMDAwMDAxFkhGN09fMVlPUkVPLXh6MUExZ1hpaEEAFjBGbmVEZHdGU1EtaFhhUFc4ZkR5cWcAA
```

```
        },
        {
            "successful": false,
            "pit_id":
 "o463QQEPbXktaW5kZXgtMDAwMDAxFkhGN09fMVlPUkVPLXh6MUExZ1hpaEEAFjBGbmVEZHdGU1EtaFhhUFc4ZkR5cWcAA
        }
    ]
}
```

Creating and managing Amazon OpenSearch Serverless collections

You can create Amazon OpenSearch Serverless collections using the console, the AWS CLI and API, the AWS SDKs, and AWS CloudFormation.

Topics

- Creating, listing, and deleting Amazon OpenSearch Serverless collections
- Working with vector search collections
- Using data lifecycle policies with Amazon OpenSearch Serverless
- Using the AWS SDKs to interact with Amazon OpenSearch Serverless
- Using AWS CloudFormation to create Amazon OpenSearch Serverless collections

Creating, listing, and deleting Amazon OpenSearch Serverless collections

A *collection* in Amazon OpenSearch Serverless is a logical grouping of one or more indexes that represent an analytics workload. OpenSearch Service automatically manages and tunes the collection, requiring minimal manual input.

Topics

- Permissions required
- Creating collections
- Accessing OpenSearch Dashboards
- Viewing collections
- Deleting collections

Permissions required

OpenSearch Serverless uses the following AWS Identity and Access Management (IAM) permissions for creating and managing collections. You can specify IAM conditions to restrict users to specific collections.

- `aoss:CreateCollection` – Create a collection.
- `aoss:ListCollections` – List collections in the current account.
- `aoss:BatchGetCollection` – Get details about one or more collections.
- `aoss:UpdateCollection` – Modify a collection.
- `aoss:DeleteCollection` – Delete a collection.

The following sample identity-based access policy provides the minimum permissions necessary for a user to manage a single collection named `Logs`:

```
[
   {
      "Sid":"Allows managing logs collections",
      "Effect":"Allow",
      "Action":[
         "aoss:CreateCollection",
         "aoss:ListCollections",
         "aoss:BatchGetCollection",
         "aoss:UpdateCollection",
         "aoss:DeleteCollection",
         "aoss:CreateAccessPolicy",
         "aoss:CreateSecurityPolicy"
      ],
      "Resource":"*",
      "Condition":{
         "StringEquals":{
            "aoss:collection":"Logs"
         }
      }
   }
]
```

`aoss:CreateAccessPolicy` and `aoss:CreateSecurityPolicy` are included because encryption, network, and data access policies are required in order for a collection to function properly. For more information, see the section called “Identity and Access Management”.

> **Note**
>
> If you're creating the first collection in your account, you also need the `iam:CreateServiceLinkedRole` permission. For more information, see the section called “Collection creation role”.

Creating collections

You can use the console or the AWS CLI to create a serverless collection. These steps cover how to create a *search* or *time series* collection. To create a *vector search* collection, see the section called “Working with vector search collections”.

Create a collection (console)

To create a collection using the console

1. Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home/.
2. Expand **Serverless** in the left navigation pane and choose **Collections**.
3. Choose **Create collection**.
4. Provide a name and description for the collection. The name must meet the following criteria:
   - Is unique to your account and AWS Region
   - Starts with a lowercase letter
   - Contains between 3 and 32 characters
   - Contains only lowercase letters a-z, the numbers 0–9, and the hyphen (-)
5. Choose a collection type:
   - **Search** – Full-text search that powers applications in your internal networks and internet-facing applications. All search data is stored in hot storage to ensure fast query response times.

- **Time series** – Log analytics segment that focuses on analyzing large volumes of semi-structured, machine-generated data. At least 24 hours of data is stored on hot indexes, and the rest remains in warm storage.
- **Vector search** – Semantic search on vector embeddings that simplifies vector data management. Powers machine learning (ML) augmented search experiences and generative AI applications such as chatbots, personal assistants, and fraud detection.

For more information, see the section called “Choosing a collection type”.

6. Under **Deployment type**, choose the redundancy setting for your collection. By default, each collection is created with redundancy, meaning that the indexing and search OpenSearch Compute Units (OCUs) each have their own standby replicas in a different Availability Zone. For development and testing purposes, you can choose to disable redundancy, which reduces the number of OCUs in your collection to two. For more information, see the section called “How it works”.
7. Under **Encryption**, choose an AWS KMS key to encrypt your data with. OpenSearch Serverless notifies you if the collection name that you entered matches a pattern defined in an encryption policy. You can choose to keep this match or override it with unique encryption settings. For more information, see the section called “Encryption”.
8. Under **Network access settings**, configure network access for the collection.

   - For **Access type**, select public or private. Then, specify which VPC endpoints and AWS services can access the collection.
     - **VPC endpoints for access** – Specify one or more VPC endpoints to allow access through. To create a VPC endpoint, see the section called “VPC endpoints”.
     - **AWS service private access** – Select one or more supported services to allow access to.
   - For **Resource type**, select whether the collection will be accessible through its *OpenSearch* endpoint (to make API calls through curl, Postman, and so on), through the *OpenSearch Dashboards* endpoint (to work with visualizations and make API calls through the console), or through both.

   > **Note**
   >
   > AWS service private access applies only to the OpenSearch endpoint, not to the OpenSearch Dashboards endpoint.

OpenSearch Serverless notifies you if the collection name that you entered matches a pattern defined in a network policy. You can choose to keep this match or override it with custom network settings. For more information, see the section called "Network access".

9. (Optional) Add one or more tags to the collection. For more information, see the section called "Tagging collections".
10. Choose **Next**.
11. Configure data access rules for the collection, which define who can access the data within the collection. For each rule that you create, perform the following steps:

    - Choose **Add principals** and select one or more IAM roles or SAML users and groups to provide data access to.
    - Under **Grant permissions**, select the alias, template, and index permissions to grant the associated principals. For a full list of permissions and the access they allow, see the section called "Supported OpenSearch API operations and permissions".

    OpenSearch Serverless notifies you if the collection name that you entered matches a pattern defined in a data access policy. You can choose to keep this match or override it with unique data access settings. For more information, see the section called "Data access control".
12. Choose **Next**.
13. Under **Data access policy settings**, choose what to do with the rules you just created. You can either use them to create a new data access policy, or add them to an existing policy.
14. Review your collection configuration and choose **Submit**.

The collection status changes to `Creating` as OpenSearch Serverless creates the collection.

Create a collection (CLI)

Before you create a collection using the AWS CLI, you must have an encryption policy with a resource pattern that matches the intended name of the collection. For example, if you plan to name your collection *logs-application*, you might create an encryption policy like this:

```
aws opensearchserverless create-security-policy \
  --name logs-policy \
  --type encryption --policy "{\"Rules\":[{\"ResourceType\":\"collection\",\"Resource
\":[\"collection\/logs-application\"]}],\"AWSOwnedKey\":true}"
```

If you plan to use the policy for additional collections, you can make the rule more broad, such as `collection/logs*` or `collection/*`.

You also need to configure network settings for the collection in the form of a network policy. Using the previous *logs-application* example, you might create the following network policy:

```
aws opensearchserverless create-security-policy \
  --name logs-policy \
  --type network --policy "[{\"Description\":\"Public access for logs collection
\",\"Rules\":[{\"ResourceType\":\"dashboard\",\"Resource\":[\"collection\/logs-
application\"]},{\"ResourceType\":\"collection\",\"Resource\":[\"collection\/logs-
application\"]}],\"AllowFromPublic\":true}]"
```

> **Note**
>
> You can create network policies after you create a collection, but we recommend doing it beforehand.

To create a collection, send a CreateCollection request:

```
aws opensearchserverless create-collection --name "logs-application" --type SEARCH --
description "A collection for storing log data"
```

For `type`, specify either `SEARCH` or `TIMESERIES`. For more information, see the section called "Choosing a collection type".

Sample response

```
{
    "createCollectionDetail": {
        "id": "07tjusf2h91cunochc",
        "name": "books",
        "description":"A collection for storing log data",
        "status": "CREATING",
        "type": "SEARCH",
        "kmsKeyArn": "auto",
        "arn": "arn:aws:aoss:us-east-1:123456789012:collection/07tjusf2h91cunochc",
        "createdDate": 1665952577473
    }
```

```
}
```

If you don't specify a collection type in the request, it defaults to `TIMESERIES`. If your collection is encrypted with an AWS owned key, the `kmsKeyArn` is `auto` rather than an ARN.

> **Important**
>
> After you create a collection, you won't be able to access it unless it matches a data access policy. For instructions to create data access policies, see the section called "Data access control".

Accessing OpenSearch Dashboards

After you create a collection with the AWS Management Console, you can navigate to the collection's OpenSearch Dashboards URL. You can find the Dashboards URL by choosing **Collections** in the left navigation pane and selecting the collection to open its details page. The URL takes the format `https://dashboards.us-east-1.aoss.amazonaws.com/_login/?collectionId=07tjusf2h91cunochc`. Once you navigate to the URL, you'll automatically log into Dashboards.

If you already have the OpenSearch Dashboards URL available but aren't on the AWS Management Console, calling the Dashboards URL from the browser will redirect to the console. Once you enter your AWS credentials, you'll automatically log in to Dashboards. For information about accessing collections for SAML, see Accessing OpenSearch Dashboards with SAML.

The OpenSearch Dashboards console timeout is one hour and isn't configurable.

> **Note**
>
> On May 10, 2023, OpenSearch introduced a common global endpoint for OpenSearch Dashboards. You can now navigate to OpenSearch Dashboards in the browser with a URL that takes the format `https://dashboards.us-east-1.aoss.amazonaws.com/_login/?collectionId=07tjusf2h91cunochc`. To ensure backward compatibility, we'll continue to support the existing collection specific OpenSearch Dashboards endpoints with the format `https://07tjusf2h91cunochc.us-east-1.aoss.amazonaws.com/_dashboards`.

Viewing collections

You can view the existing collections in your AWS account on the **Collections** tab of the Amazon OpenSearch Service console.

To list collections along with their IDs, send a ListCollections request.

```
aws opensearchserverless list-collections
```

Sample response

```
{
   "collectionSummaries":[
      {
         "arn":"arn:aws:aoss:us-east-1:123456789012:collection/07tjusf2h91cunochc",
         "id":"07tjusf2h91cunochc",
         "name":"my-collection",
         "status":"CREATING"
      }
   ]
}
```

To limit the search results, use collection filters. This request filters the response to collections in the `ACTIVE` state:

```
aws opensearchserverless list-collections --collection-filters '{ "status": "ACTIVE" }'
```

To get more detailed information about one or more collections, including the OpenSearch endpoint and the OpenSearch Dashboards endpoint, send a BatchGetCollection request:

```
aws opensearchserverless batch-get-collection --ids ["07tjusf2h91cunochc",
 "1iu5usc4rame"]
```

> **Note**
>
> You can include `--names` or `--ids` in the request, but not both.

Sample response

```
{
   "collectionDetails":[
      {
         "id": "07tjusf2h91cunochc",
         "name": "my-collection",
         "status": "ACTIVE",
         "type": "SEARCH",
         "description": "",
         "arn": "arn:aws:aoss:us-east-1:123456789012:collection/07tjusf2h91cunochc",
         "kmsKeyArn": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab",
         "createdDate": 1667446262828,
         "lastModifiedDate": 1667446300769,
         "collectionEndpoint": "https://07tjusf2h91cunochc.us-
east-1.aoss.amazonaws.com",
         "dashboardEndpoint": "https://07tjusf2h91cunochc.us-east-1.aoss.amazonaws.com/
_dashboards"
      },
      {
         "id": "178ukvtg3i82dvopdid",
         "name": "another-collection",
         "status": "ACTIVE",
         "type": "TIMESERIES",
         "description": "",
         "arn": "arn:aws:aoss:us-east-1:123456789012:collection/178ukvtg3i82dvopdid",
         "kmsKeyArn": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab",
         "createdDate": 1667446262828,
         "lastModifiedDate": 1667446300769,
         "collectionEndpoint": "https://178ukvtg3i82dvopdid.us-
east-1.aoss.amazonaws.com",
         "dashboardEndpoint": "https://178ukvtg3i82dvopdid.us-
east-1.aoss.amazonaws.com/_dashboards"
      }
   ],
   "collectionErrorDetails":[]
}
```


Deleting collections

Deleting a collection deletes all data and indexes in the collection. You can't recover collections after you delete them.

To delete a collection using the console

1. From the **Collections** panel of the Amazon OpenSearch Service console, select the collection you want to delete.
2. Choose **Delete** and confirm deletion.

To delete a collection using the AWS CLI, send a DeleteCollection request:

```
aws opensearchserverless delete-collection --id 07tjusf2h91cunochc
```

Sample response

```
{
   "deleteCollectionDetail":{
      "id":"07tjusf2h91cunochc",
      "name":"my-collection",
      "status":"DELETING"
   }
}
```

Working with vector search collections

The *vector search* collection type in OpenSearch Serverless provides a similarity search capability that is scalable and high performing. It makes it easy for you to build modern machine learning (ML) augmented search experiences and generative artificial intelligence (AI) applications without having to manage the underlying vector database infrastructure.

Use cases for vector search collections include image searches, document searches, music retrieval, product recommendations, video searches, location-based searches, fraud detection, and anomaly detection.

Because the vector engine for OpenSearch Serverless is powered by the k-nearest neighbor (k-NN) search feature in OpenSearch, you get the same functionality with the simplicity of a serverless environment. The engine supports k-NN plugin API. With these operations, you can take advantage of full-text search, advanced filtering, aggregations, geospatial queries, nested queries for faster retrieval of data, and enhanced search results.

The vector engine provides distance metrics such as Euclidean distance, cosine similarity, and dot product similarity, and can accommodate 16,000 dimensions. You can store fields with various

data types for metadata, such as numbers, Booleans, dates, keywords, and geopoints. You can also store fields with text for descriptive information to add more context to stored vectors. Colocating the data types reduces complexity, increases maintainability, and avoids data duplication, version compatibility challenges, and licensing issues.

> **Note**
>
> Amazon OpenSearch Serverless supports Faiss 16-bit scalar quantization which can be used to perform conversions between 32-bit floating and 16-bit vectors. To learn more, see Faiss 16-bit scalar quantization. You can also use binary vectors to reduce memory costs. For more information, see Binary vectors.

Getting started with vector search collections

In this tutorial, you complete the following steps to store, search, and retrieve vector embeddings in real time:

1. Configure permissions
2. Create a collection
3. Upload and search data
4. Delete the collection

Step 1: Configure permissions

To complete this tutorial (and to use OpenSearch Serverless in general), you must have the correct AWS Identity and Access Management (IAM) permissions. In this tutorial, you create a collection, upload and search data, and then delete the collection.

Your user or role must have an attached identity-based policy with the following minimum permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "aoss:CreateCollection",
        "aoss:ListCollections",
```

```
        "aoss:BatchGetCollection",
        "aoss:DeleteCollection",
        "aoss:CreateAccessPolicy",
        "aoss:ListAccessPolicies",
        "aoss:UpdateAccessPolicy",
        "aoss:CreateSecurityPolicy",
        "iam:ListUsers",
        "iam:ListRoles"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

For more information about OpenSearch Serverless IAM permissions, see the section called "Identity and Access Management".

Step 2: Create a collection

A *collection* is a group of OpenSearch indexes that work together to support a specific workload or use case.

To create an OpenSearch Serverless collection

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Choose **Collections** in the left navigation pane and choose **Create collection**.
3. Name the collection **housing**.
4. For collection type, choose **Vector search**. For more information, see the section called "Choosing a collection type".
5. Under **Deployment type**, clear **Enable redundancy (active replicas)**. This creates a collection in development or testing mode, and reduces the number of OpenSearch Compute Units (OCUs) in your collection to two. If you want to create a production environment in this tutorial, leave the check box selected.
6. Under **Security**, select **Easy create** to streamline your security configuration. All the data in the vector engine is encrypted in transit and at rest by default. The vector engine supports fine-grained IAM permissions so that you can define who can create, update, and delete encryptions, networks, collections, and indexes.
7. Choose **Next**.

8. Review your collection settings and choose **Submit**. Wait several minutes for the collection status to become `Active`.

Step 3: Upload and search data

An *index* is a collection of documents with a common data schema that provides a way for you to store, search, and retrieve your vector embeddings and other fields. You can create and upload data to indexes in an OpenSearch Serverless collection by using the Dev Tools console in OpenSearch Dashboards, or an HTTP tool such as Postman or awscurl. This tutorial uses Dev Tools.

To index and search data in the movies collection

1. To create a single index for your new collection, send the following request in the Dev Tools console. By default, this creates an index with an `nmslib` engine and Euclidean distance.

```
PUT housing-index
{
   "settings": {
      "index.knn": true
   },
   "mappings": {
      "properties": {
         "housing-vector": {
            "type": "knn_vector",
            "dimension": 3
         },
         "title": {
            "type": "text"
         },
         "price": {
            "type": "long"
         },
         "location": {
            "type": "geo_point"
         }
      }
   }
}
```

2. To index a single document into *housing-index*, send the following request:

```
POST housing-index/_doc
```

```
{
  "housing-vector": [
    10,
    20,
    30
  ],
  "title": "2 bedroom in downtown Seattle",
  "price": "2800",
  "location": "47.71, 122.00"
}
```

3. To search for properties that are similar to the ones in your index, send the following query:

```
GET housing-index/_search
{
    "size": 5,
    "query": {
        "knn": {
            "housing-vector": {
                "vector": [
                    10,
                    20,
                    30
                ],
                "k": 5
            }
        }
    }
}
```

Step 4: Delete the collection

Because the *housing* collection is for test purposes, make sure to delete it when you're done experimenting.

To delete an OpenSearch Serverless collection

1. Go back to the **Amazon OpenSearch Service** console.
2. Choose **Collections** in the left navigation pane and select the **properties** collection.
3. Choose **Delete** and confirm the deletion.

Filtered search

You can use filters to refine your semantic search results. To create an index and perform a filtered search on your documents, substitute Upload and search data in the previous tutorial with the following instructions. The other steps remain the same. For more information about filters, see k-NN search with filters.

To index and search data in the movies collection

1. To create a single index for your collection, send the following request in the Dev Tools console:

```
PUT housing-index-filtered
{
  "settings": {
    "index.knn": true
  },
  "mappings": {
    "properties": {
      "housing-vector": {
        "type": "knn_vector",
        "dimension": 3,
        "method": {
          "engine": "faiss",
          "name": "hnsw"
        }
      },
      "title": {
        "type": "text"
      },
      "price": {
        "type": "long"
      },
      "location": {
        "type": "geo_point"
      }
    }
  }
}
```

2. To index a single document into *housing-index-filtered*, send the following request:

```
POST housing-index-filtered/_doc
{
  "housing-vector": [
    10,
    20,
    30
  ],
  "title": "2 bedroom in downtown Seattle",
  "price": "2800",
  "location": "47.71, 122.00"
}
```

3. To search your data for an apartment in Seattle under a given price and within a given distance of a geographical point, send the following request:

```
GET housing-index-filtered/_search
{
  "size": 5,
  "query": {
    "knn": {
      "housing-vector": {
        "vector": [
          0.1,
          0.2,
          0.3
        ],
        "k": 5,
        "filter": {
          "bool": {
            "must": [
              {
                "query_string": {
                  "query": "Find me 2 bedroom apartment in Seattle under $3000 ",
                  "fields": [
                    "title"
                  ]
                }
              },
              {
                "range": {
                  "price": {
                    "lte": 3000
```

```
                }
              }
            },
            {
              "geo_distance": {
                "distance": "100miles",
                "location": {
                  "lat": 48,
                  "lon": 121
                }
              }
            }
          ]
        }
      }
    }
  }
}
```

Billion scale workloads

Vector search collections support workloads with billions of vectors. You don't need to reindex for scaling purposes because auto scaling does this for you. If you have millions of vectors (or more) with a high number of dimensions and need more than 200 OCUs, contact AWS Support to raise your the maximum OpenSearch Compute Units (OCUs) for your account.

Limitations

Vector search collections have the following limitations:

- Vector search collections don't support the Apache Lucene ANN engine.
- Vector search collections only support the HNSW algorithm with Faiss and do not support IVF and IVFQ.
- Vector search collections don't support the warmup, stats, and model training API operations.
- Vector search collections don't support inline or stored scripts.
- Index count information isn't available in the AWS Management Console for vector search collections.
- The refresh interval for indexes on vector search collections is 60 seconds.

Next steps

Now that you know how to create a vector search collection and index data, you might want to try some of the following exercises:

- Use the OpenSearch Python client to work with vector search collections. See this tutorial on GitHub.
- Use the OpenSearch Java client to work with vector search collections. See this tutorial on GitHub.
- Set up LangChain to use OpenSearch as a vector store. LangChain is an open source framework for developing applications powered by language models. For more information, see the LangChain documentation.

Using data lifecycle policies with Amazon OpenSearch Serverless

A data lifecycle policy for an Amazon OpenSearch Serverless *time series* collection determines the lifespan of the data in that collection. OpenSearch Serverless retains the data for the period of time that you configure.

You can configure a separate data lifecycle policy for each index of each *time series* collection in your AWS account. OpenSearch Serverless retains documents in indexes for, at minimum, the retention period you configure in the policy. It then automatically deletes them on a best-effort basis, typically within 48 hours or 10% of the retention period, whichever is longer.

Only *time series* collections support data lifecycle policies. They are not supported by *search* or *vector search* collections.

Topics

Data lifecycle policies

In a data lifecycle policy, you specify a series of *rules*. The data lifecycle policy lets you manage the retention period of data associated to indexes or collections that match these rules. These rules define the retention period for data in an index or group of indexes. Each rule consists of a resource type (`index`), a retention period, and a list of resources (indexes) that the retention period applies to.

You define the retention period with one of the following formats:

- `"MinIndexRetention": "24h"` – OpenSearch Serverless retains index data for the specified period in hours or days. You can set this period to be from `24h` to `3650d`.
- `"NoMinIndexRetention": true` – OpenSearch Serverless retains index data indefinitely.

In the following sample policy, the first rule specifies a retention period of 15 days for all indexes within the collection `marketing`. The second rule specifies that all index names that begin with `log` in the `finance` collection have no retention period set and will be retained indefinitely.

```
{
   "lifeCyclePolicyDetail": {
      "type": "retention",
      "name": "my-policy",
      "policyVersion": "MTY4ODI0NTM2OTk1N18x",
      "policy": {
         "Rules": [
            {
            "ResourceType":"index",
            "Resource":[
               "index/marketing/*"
            ],
            "MinIndexRetention": "15d"
         },
         {
            "ResourceType":"index",
            "Resource":[
               "index/finance/log*"
            ],
            "NoMinIndexRetention": true
         }
         ]
      },
```

```
      "createdDate": 1688245369957,
      "lastModifiedDate": 1688245369957
   }
}
```

In the following sample policy rule, OpenSearch Serverless indefinitely retains the data in all indexes for all collections within the account.

```
{
   "Rules": [
      {
         "ResourceType": "index",
         "Resource": [
            "index/*/*"
         ]
      }
   ],
   "NoMinIndexRetention": true
}
```

Permissions required

Lifecycle policies for OpenSearch Serverless use the following AWS Identity and Access Management (IAM) permissions. You can specify IAM conditions to restrict users to data lifecycle policies associated with specific collections and indexes.

- `aoss:CreateLifecyclePolicy` – Create a data lifecycle policy.
- `aoss:ListLifecyclePolicies` – List all data lifecycle policies in the current account.
- `aoss:BatchGetLifecyclePolicy` – View a data lifecycle policy associated with an account or policy name.
- `aoss:BatchGetEffectiveLifecyclePolicy` – View a data lifecycle policy for a given resource (`index` is the only supported resource).
- `aoss:UpdateLifecyclePolicy` – Modify a given data lifecycle policy, and change its retention setting or resource.
- `aoss:DeleteLifecyclePolicy` – Delete a data lifecycle policy.

The following identity-based access policy allows a user to view all data lifecycle policies, and update policies with the resource pattern `collection/application-logs`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "aoss:UpdateLifecyclePolicy"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aoss:collection": "application-logs"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "aoss:ListLifecyclePolicies",
                "aoss:BatchGetLifecyclePolicy"
            ],
            "Resource": "*"
        }
    ]
}
```

Policy precedence

There can be situations where data lifecycle policy rules overlap, within or across policies. When this happens, a rule with a more specific resource name or pattern for an index overrides a rule with a more general resource name or pattern for any indexes that are common to *both* rules.

For example, in the following policy, two rules apply to an index `index/sales/logstash`. In this situation, the second rule takes precedence because `index/sales/log*` is the longest match to `index/sales/logstash`. Therefore, OpenSearch Serverless sets no retention period for the index.

```
{
      "Rules":[
         {
            "ResourceType":"index",
            "Resource":[
```

```
            "index/sales/*",
         ],
         "MinIndexRetention": "15d"
      },
      {
         "ResourceType":"index",
         "Resource":[
            "index/sales/log*",
         ],
         "NoMinIndexRetention": true
      }
   ]
}
```

Policy syntax

Provide one or more *rules*. These rules define data lifecycle settings for your OpenSearch Serverless indexes.

Each rule contains the following elements. You can either provide `MinIndexRetention` or `NoMinIndexRetention` in each rule, but not both.

Element	Description
Resource type	The type of resource that the rule applies to. The only supported option for data lifecycle policies is `index`.
Resource	A list of resource names and/or patterns. Patterns consist of a prefix and a wildcard (*), which allow the associated permissio ns to apply to multiple resources. For example, `index/`*`<collection-name|p attern>`* `/`*`<index-name|pattern>`* .
MinIndexRetention	The minimum period, in days (d) or hours (h), to retain the document in the index. The lower bound is 24h and the upper bound is 3650d.

Element	Description
NoMinIndexRetention	If `true`, OpenSearch Serverless retains documents indefinitely.

The following are some examples:

```
{
  "Rules": [
    {
      "ResourceType": "index",
      "Resource": [
        "index/autoparts-inventory/*"
      ],
      "MinIndexRetention": "20d"
    },
    {
      "ResourceType": "index",
      "Resource": [
        "index/auto*/gear"
      ],
      "MinIndexRetention": "24h"
    },
    {
      "ResourceType": "index",
      "Resource": [
        "index/autoparts-inventory/tires"
      ],
      "NoMinIndexRetention": true
    }
  ]
}
```

Creating data lifecycle policies (AWS CLI)

To create a data lifecycle policy using the OpenSearch Serverless API operations, use the CreateLifecyclePolicy command. This command accepts both inline policies and .json files. Inline policies must be encoded as a JSON escaped string.

The following request creates a data lifecycle policy:

```
aws opensearchserverless create-lifecycle-policy \
    --name my-policy \
    --type retention \
    --policy "{\"Rules\":[{\"ResourceType\":\"index\",\"Resource\":[\"index/autoparts-
inventory/*\"],\"MinIndexRetention\": \"81d\"},{\"ResourceType\":\"index\",\"Resource
\":[\"index/sales/orders*\"],\"NoMinIndexRetention\":true}]}"
```

To provide the policy in a JSON file, use the format `--policy file://my-policy.json`

Viewing data lifecycle policies

Before you create a collection, you might want to preview the existing data lifecycle policies in your account to see which one has a resource pattern that matches your collection's name. The following ListLifecyclePolicies request lists all data lifecycle policies in your account:

```
aws opensearchserverless list-lifecycle-policies --type retention
```

The request returns information about all configured data lifecycle policies. To view the pattern rules defined in the one specific policy, find the policy information in the contents of the `lifecyclePolicySummaries` element in the response. Note the `name` and `type` of this policy and use these properties in a BatchGetLifecyclePolicy request to receive a response with the following policy details:

```
{
    "lifecyclePolicySummaries": [
        {
            "type": "retention",
            "name": "my-policy",
            "policyVersion": "MTY2MzY5MTY1MDA3Ml8x",
            "createdDate": 1663691650072,
            "lastModifiedDate": 1663691650072
        }
    ]
}
```

To limit the results to policies that contain specific collections or indexes, you can include resource filters:

```
aws opensearchserverless list-lifecycle-policies --type retention --resources
 "index/autoparts-inventory/*"
```

To view detailed information about a specific policy, use the BatchGetLifecyclePolicy command.

Updating data lifecycle policies

When you modify a data lifecycle policy, all associated collections are impacted. To update a data lifecycle policy in the OpenSearch Serverless console, expand **Data lifecycle policies**, select the policy to modify, and choose **Edit**. Make your changes and choose **Save**.

To update a data lifecycle policy using the OpenSearch Serverless API, use the UpdateLifecyclePolicy command. You must include a policy version in the request. You can retrieve the policy version by using the `ListLifecyclePolicies` or `BatchGetLifecyclePolicy` commands. Including the most recent policy version ensures that you don't inadvertently override a change made by someone else.

The following request updates a data lifecycle policy with a new policy JSON document:

```
aws opensearchserverless update-lifecycle-policy \
    --name my-policy \
    --type retention \
    --policy-version MTY2MzY5MTY1MDA3Ml8x \
    --policy file://my-new-policy.json
```

There might be a few minutes of lag time between when you update the policy and when the new retention periods are enforced.

Deleting data lifecycle policies

When you delete a data lifecycle policy, it no longer applies to any matching indexes. To delete a policy in the OpenSearch Serverless console, select the policy and choose **Delete**.

You can also use the DeleteLifecyclePolicy command:

```
aws opensearchserverless delete-lifecycle-policy --name my-policy --type retention
```

Using the AWS SDKs to interact with Amazon OpenSearch Serverless

This section includes examples of how to use the AWS SDKs to interact with Amazon OpenSearch Serverless. These code samples show how to create security policies and collections, and how to query collections.

> **Note**
>
> We're currently building out these code samples. If you want to contribute a code sample (Java, Go, etc.), please open a pull request directly within the GitHub repository.

Topics

Python

The following sample script uses the AWS SDK for Python (Boto3), as well as the opensearch-py client for Python, to create encryption, network, and data access policies, create a matching collection, and index some sample data.

To install the required dependencies, run the following commands:

```
pip install opensearch-py
pip install boto3
pip install botocore
pip install requests-aws4auth
```

Within the script, replace the `Principal` element with the Amazon Resource Name (ARN) of the user or role that's signing the request. You can also optionally modify the `region`.

```
from opensearchpy import OpenSearch, RequestsHttpConnection
from requests_aws4auth import AWS4Auth
import boto3
import botocore
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default region.

client = boto3.client('opensearchserverless')
service = 'aoss'
region = 'us-east-1'
credentials = boto3.Session().get_credentials()
```

```
awsauth = AWS4Auth(credentials.access_key, credentials.secret_key,
                   region, service, session_token=credentials.token)


def createEncryptionPolicy(client):
    """Creates an encryption policy that matches all collections beginning with tv-"""
    try:
        response = client.create_security_policy(
            description='Encryption policy for TV collections',
            name='tv-policy',
            policy="""
                {
                    \"Rules\":[
                        {
                            \"ResourceType\":\"collection\",
                            \"Resource\":[
                                \"collection\/tv-*\"
                            ]
                        }
                    ],
                    \"AWSOwnedKey\":true
                }
                """,
            type='encryption'
        )
        print('\nEncryption policy created:')
        print(response)
    except botocore.exceptions.ClientError as error:
        if error.response['Error']['Code'] == 'ConflictException':
            print(
                '[ConflictException] The policy name or rules conflict with an existing
 policy.')
        else:
            raise error


def createNetworkPolicy(client):
    """Creates a network policy that matches all collections beginning with tv-"""
    try:
        response = client.create_security_policy(
            description='Network policy for TV collections',
            name='tv-policy',
            policy="""
                [{
```

```
                    \"Description\":\"Public access for TV collection\",
                    \"Rules\":[
                        {
                            \"ResourceType\":\"dashboard\",
                            \"Resource\":[\"collection\/tv-*\"]
                        },
                        {
                            \"ResourceType\":\"collection\",
                            \"Resource\":[\"collection\/tv-*\"]
                        }
                    ],
                    \"AllowFromPublic\":true
                }]
                """,
            type='network'
        )
        print('\nNetwork policy created:')
        print(response)
    except botocore.exceptions.ClientError as error:
        if error.response['Error']['Code'] == 'ConflictException':
            print(
                '[ConflictException] A network policy with this name already exists.')
        else:
            raise error


def createAccessPolicy(client):
    """Creates a data access policy that matches all collections beginning with tv-"""
    try:
        response = client.create_access_policy(
            description='Data access policy for TV collections',
            name='tv-policy',
            policy="""
                [{
                    \"Rules\":[
                        {
                            \"Resource\":[
                                \"index\/tv-*\/*\"
                            ],
                            \"Permission\":[
                                \"aoss:CreateIndex\",
                                \"aoss:DeleteIndex\",
                                \"aoss:UpdateIndex\",
                                \"aoss:DescribeIndex\",
```

```
                            \"aoss:ReadDocument\",
                            \"aoss:WriteDocument\"
                        ],
                        \"ResourceType\": \"index\"
                    },
                    {
                        \"Resource\":[
                            \"collection\/tv-*\"
                        ],
                        \"Permission\":[
                            \"aoss:CreateCollectionItems\"
                        ],
                        \"ResourceType\": \"collection\"
                    }
                ],
                \"Principal\":[
                    \"arn:aws:iam::123456789012:role\/Admin\"
                ]
            }]
            """,
        type='data'
    )
    print('\nAccess policy created:')
    print(response)
except botocore.exceptions.ClientError as error:
    if error.response['Error']['Code'] == 'ConflictException':
        print(
            '[ConflictException] An access policy with this name already exists.')
    else:
        raise error


def createCollection(client):
    """Creates a collection"""
    try:
        response = client.create_collection(
            name='tv-sitcoms',
            type='SEARCH'
        )
        return(response)
    except botocore.exceptions.ClientError as error:
        if error.response['Error']['Code'] == 'ConflictException':
            print(
```

```
                '[ConflictException] A collection with this name already exists. Try
 another name.')
        else:
            raise error


def waitForCollectionCreation(client):
    """Waits for the collection to become active"""
    response = client.batch_get_collection(
        names=['tv-sitcoms'])
    # Periodically check collection status
    while (response['collectionDetails'][0]['status']) == 'CREATING':
        print('Creating collection...')
        time.sleep(30)
        response = client.batch_get_collection(
            names=['tv-sitcoms'])
    print('\nCollection successfully created:')
    print(response["collectionDetails"])
    # Extract the collection endpoint from the response
    host = (response['collectionDetails'][0]['collectionEndpoint'])
    final_host = host.replace("https://", "")
    indexData(final_host)


def indexData(host):
    """Create an index and add some sample data"""
    # Build the OpenSearch client
    client = OpenSearch(
        hosts=[{'host': host, 'port': 443}],
        http_auth=awsauth,
        use_ssl=True,
        verify_certs=True,
        connection_class=RequestsHttpConnection,
        timeout=300
    )
    # It can take up to a minute for data access rules to be enforced
    time.sleep(45)

    # Create index
    response = client.indices.create('sitcoms-eighties')
    print('\nCreating index:')
    print(response)

    # Add a document to the index.
```

```
    response = client.index(
        index='sitcoms-eighties',
        body={
            'title': 'Seinfeld',
            'creator': 'Larry David',
            'year': 1989
        },
        id='1',
    )
    print('\nDocument added:')
    print(response)


def main():
    createEncryptionPolicy(client)
    createNetworkPolicy(client)
    createAccessPolicy(client)
    createCollection(client)
    waitForCollectionCreation(client)


if __name__ == "__main__":
    main()
```

JavaScript

The following sample script uses the SDK for JavaScript in Node.js, as well as the opensearch-js client for JavaScript, to create encryption, network, and data access policies, create a matching collection, create an index, and index some sample data.

To install the required dependencies, run the following commands:

```
npm i aws-sdk
npm i aws4
npm i @opensearch-project/opensearch
```

Within the script, replace the `Principal` element with the Amazon Resource Name (ARN) of the user or role that's signing the request. You can also optionally modify the `region`.

```
var AWS = require('aws-sdk');
var aws4 = require('aws4');
var {
```

```
    Client,
    Connection
} = require("@opensearch-project/opensearch");
var {
    OpenSearchServerlessClient,
    CreateSecurityPolicyCommand,
    CreateAccessPolicyCommand,
    CreateCollectionCommand,
    BatchGetCollectionCommand
} = require("@aws-sdk/client-opensearchserverless");
var client = new OpenSearchServerlessClient();

async function execute() {
    await createEncryptionPolicy(client)
    await createNetworkPolicy(client)
    await createAccessPolicy(client)
    await createCollection(client)
    await waitForCollectionCreation(client)
}

async function createEncryptionPolicy(client) {
    // Creates an encryption policy that matches all collections beginning with 'tv-'
    try {
        var command = new CreateSecurityPolicyCommand({
            description: 'Encryption policy for TV collections',
            name: 'tv-policy',
            type: 'encryption',
            policy: " \
        { \
            \"Rules\":[ \
                { \
                    \"ResourceType\":\"collection\", \
                    \"Resource\":[ \
                        \"collection\/tv-*\" \
                    ] \
                } \
            ], \
            \"AWSOwnedKey\":true \
        }"
        });
        const response = await client.send(command);
        console.log("Encryption policy created:");
        console.log(response['securityPolicyDetail']);
    } catch (error) {
```

```
        if (error.name === 'ConflictException') {
            console.log('[ConflictException] The policy name or rules conflict with an
 existing policy.');
        } else
            console.error(error);
    };
}

async function createNetworkPolicy(client) {
    // Creates a network policy that matches all collections beginning with 'tv-'
    try {
        var command = new CreateSecurityPolicyCommand({
            description: 'Network policy for TV collections',
            name: 'tv-policy',
            type: 'network',
            policy: " \
            [{ \
                \"Description\":\"Public access for television collection\", \
                \"Rules\":[ \
                    { \
                        \"ResourceType\":\"dashboard\", \
                        \"Resource\":[\"collection\/tv-*\"] \
                    }, \
                    { \
                        \"ResourceType\":\"collection\", \
                        \"Resource\":[\"collection\/tv-*\"] \
                    } \
                ], \
                \"AllowFromPublic\":true \
            }]"
        });
        const response = await client.send(command);
        console.log("Network policy created:");
        console.log(response['securityPolicyDetail']);
    } catch (error) {
        if (error.name === 'ConflictException') {
            console.log('[ConflictException] A network policy with that name already
 exists.');
        } else
            console.error(error);
    };
}

async function createAccessPolicy(client) {
```

```
    // Creates a data access policy that matches all collections beginning with 'tv-'
    try {
        var command = new CreateAccessPolicyCommand({
            description: 'Data access policy for TV collections',
            name: 'tv-policy',
            type: 'data',
            policy: " \
            [{ \
                \"Rules\":[ \
                    { \
                        \"Resource\":[ \
                            \"index\/tv-*\/*\" \
                        ], \
                        \"Permission\":[ \
                            \"aoss:CreateIndex\", \
                            \"aoss:DeleteIndex\", \
                            \"aoss:UpdateIndex\", \
                            \"aoss:DescribeIndex\", \
                            \"aoss:ReadDocument\", \
                            \"aoss:WriteDocument\" \
                        ], \
                        \"ResourceType\": \"index\" \
                    }, \
                    { \
                        \"Resource\":[ \
                            \"collection\/tv-*\" \
                        ], \
                        \"Permission\":[ \
                            \"aoss:CreateCollectionItems\" \
                        ], \
                        \"ResourceType\": \"collection\" \
                    } \
                ], \
                \"Principal\":[ \
                    \"arn:aws:iam::123456789012:role\/Admin\" \
                ] \
            }]"
        });
        const response = await client.send(command);
        console.log("Access policy created:");
        console.log(response['accessPolicyDetail']);
    } catch (error) {
        if (error.name === 'ConflictException') {
```

```
            console.log('[ConflictException] An access policy with that name already
 exists.');
        } else
            console.error(error);
    };
}

async function createCollection(client) {
    // Creates a collection to hold TV sitcoms indexes
    try {
        var command = new CreateCollectionCommand({
            name: 'tv-sitcoms',
            type: 'SEARCH'
        });
        const response = await client.send(command);
        return (response)
    } catch (error) {
        if (error.name === 'ConflictException') {
            console.log('[ConflictException] A collection with this name already
 exists. Try another name.');
        } else
            console.error(error);
    };
}

async function waitForCollectionCreation(client) {
    // Waits for the collection to become active
    try {
        var command = new BatchGetCollectionCommand({
            names: ['tv-sitcoms']
        });
        var response = await client.send(command);
        while (response.collectionDetails[0]['status'] == 'CREATING') {
            console.log('Creating collection...')
            await sleep(30000) // Wait for 30 seconds, then check the status again
            function sleep(ms) {
                return new Promise((resolve) => {
                    setTimeout(resolve, ms);
                });
            }
            var response = await client.send(command);
        }
        console.log('Collection successfully created:');
        console.log(response['collectionDetails']);
```

```
        // Extract the collection endpoint from the response
        var host = (response.collectionDetails[0]['collectionEndpoint'])
        // Pass collection endpoint to index document request
        indexDocument(host)
    } catch (error) {
        console.error(error);
    };
}

async function indexDocument(host) {

    var client = new Client({
        node: host,
        Connection: class extends Connection {
            buildRequestObject(params) {
                var request = super.buildRequestObject(params)
                request.service = 'aoss';
                request.region = 'us-east-1'; // e.g. us-east-1
                var body = request.body;
                request.body = undefined;
                delete request.headers['content-length'];
                request.headers['x-amz-content-sha256'] = 'UNSIGNED-PAYLOAD';
                request = aws4.sign(request, AWS.config.credentials);
                request.body = body;

                return request
            }
        }
    });

    // Create an index
    try {
        var index_name = "sitcoms-eighties";

        var response = await client.indices.create({
            index: index_name
        });

        console.log("Creating index:");
        console.log(response.body);

        // Add a document to the index
        var document = "{ \"title\": \"Seinfeld\", \"creator\": \"Larry David\", \"year
\": \"1989\" }\n";
```

```
        var response = await client.index({
            index: index_name,
            body: document
        });

        console.log("Adding document:");
        console.log(response.body);
    } catch (error) {
        console.error(error);
    };
}

execute()
```

Using AWS CloudFormation to create Amazon OpenSearch Serverless collections

You can use AWS CloudFormation to create Amazon OpenSearch Serverless resources such as collections, security policies, and VPC endpoints. For the comprehensive OpenSearch Serverless CloudFormation reference, see Amazon OpenSearch Serverless in the *AWS CloudFormation User Guide*.

The following sample CloudFormation template creates a simple data access policy, network policy, and security policy, as well as a matching collection. It's a good way to get up and running quickly with Amazon OpenSearch Serverless and provision the necessary elements to create and use a collection.

> **Important**
>
> This example uses public network access, which isn't recommended for production workloads. We recommend using VPC access to protect your collections. For more information, see AWS::OpenSearchServerless::VpcEndpoint and the section called “VPC endpoints”.

```
AWSTemplateFormatVersion: 2010-09-09
Description: 'Amazon OpenSearch Serverless template to create an IAM user, encryption
 policy, data access policy and collection'
Resources:
```

```
  IAMUSer:
    Type: 'AWS::IAM::User'
    Properties:
      UserName:  aossadmin
  DataAccessPolicy:
    Type: 'AWS::OpenSearchServerless::AccessPolicy'
    Properties:
      Name: quickstart-access-policy
      Type: data
      Description: Access policy for quickstart collection
      Policy: !Sub >-
        [{"Description":"Access for cfn user","Rules":
[{"ResourceType":"index","Resource":["index/*/*"],"Permission":["aoss:*"]},
        {"ResourceType":"collection","Resource":["collection/quickstart"],"Permission":
["aoss:*"]}],
        "Principal":["arn:aws:iam::${AWS::AccountId}:user/aossadmin"]}]
  NetworkPolicy:
    Type: 'AWS::OpenSearchServerless::SecurityPolicy'
    Properties:
      Name: quickstart-network-policy
      Type: network
      Description: Network policy for quickstart collection
      Policy: >-
        [{"Rules":[{"ResourceType":"collection","Resource":["collection/
quickstart"]}, {"ResourceType":"dashboard","Resource":["collection/
quickstart"]}],"AllowFromPublic":true}]
  EncryptionPolicy:
    Type: 'AWS::OpenSearchServerless::SecurityPolicy'
    Properties:
      Name: quickstart-security-policy
      Type: encryption
      Description: Encryption policy for quickstart collection
      Policy: >-
        {"Rules":[{"ResourceType":"collection","Resource":["collection/
quickstart"]}],"AWSOwnedKey":true}
  Collection:
    Type: 'AWS::OpenSearchServerless::Collection'
    Properties:
      Name: quickstart
      Type: TIMESERIES
      Description: Collection to holds timeseries data
    DependsOn: EncryptionPolicy
Outputs:
  IAMUser:
```

```
    Value: !Ref IAMUSer
  DashboardURL:
    Value: !GetAtt Collection.DashboardEndpoint
  CollectionARN:
    Value: !GetAtt Collection.Arn
```

Managing capacity limits for Amazon OpenSearch Serverless

With Amazon OpenSearch Serverless, you don't have to manage capacity yourself. OpenSearch Serverless automatically scales the compute capacity for your account based on the current workload. Serverless compute capacity is measured in *OpenSearch Compute Units* (OCUs). Each OCU is a combination of 6 GiB of memory and corresponding virtual CPU (vCPU), as well as data transfer to Amazon S3. For more information about the decoupled architecture in OpenSearch Serverless, see the section called “How it works”.

When you create your first collection, OpenSearch Serverless instantiates a total of four OCUs (two for indexing and two for search). These OCUs always exist, even when there's no indexing or search activity. All subsequent collections can share these OCUs (except for collections with unique AWS KMS keys, which instantiate their own set of four OCUs). If needed, OpenSearch Serverless automatically scales out and adds additional OCUs as your indexing and search usage grows. When traffic on your collection endpoint decreases, capacity scales back down to the minimum number of OCUs required for your data size. For the search and time series collection, the number of OCUs required when idle is proportional to the data size and index count. For vectors, it depends on both the memory (RAM) to store vector graphs and disk space to store indices. If not in an idle state, OCU requirements take both of these into account.

Vector collections keep index data in OCU local storage. OCU RAM limits are reached faster than OCU disk limits, causing vector collections to be restricted by RAM space. At most, it will scale down to 1 OCU [0.5 OCU x 2] for indexing and 1 OCU [0.5 OCU x 2] for search. Scaling also factors in the number of shards needed for your collection or index. Each OCU can support a specified number of shards. The number of indexes should be proportional to the shard count. The total number of base OCUs required is the maximum amount of your data, memory, and shards required. To learn more, see Amazon OpenSearch Serverless cost-effective search capabilities, at any scale.

For *search* and *vector search* collections, all data is stored on hot indexes to ensure fast query response times. *Time series* collections use a combination of hot and warm storage, keeping the

most recent data in hot storage to optimize query response times for more frequently accessed data. For more information, see the section called “Choosing a collection type”.

> **Note**
>
> A vector search collection can't share OCUs with *search* and *time series* collections, even if the vector search collection uses the same KMS key as the *search* or *time series* collections. A new set of OCUs will be created for your first vector collection. The OCUs of vector collections are shared among the same KMS key collections.

To manage capacity for your collections and to control costs, you can specify the overall maximum indexing and search capacity for the current account and Region, and OpenSearch Serverless scales out your collection resources automatically based on these specifications.

Because indexing and search capacity scale separately, you specify account-level limits for each:

- **Maximum indexing capacity** – OpenSearch Serverless can increase indexing capacity up to this number of OCUs.
- **Maximum search capacity** – OpenSearch Serverless can increase search capacity up to this number of OCUs.

> **Note**
>
> At this time, capacity settings only apply at the account level. You can't configure per-collection capacity limits.

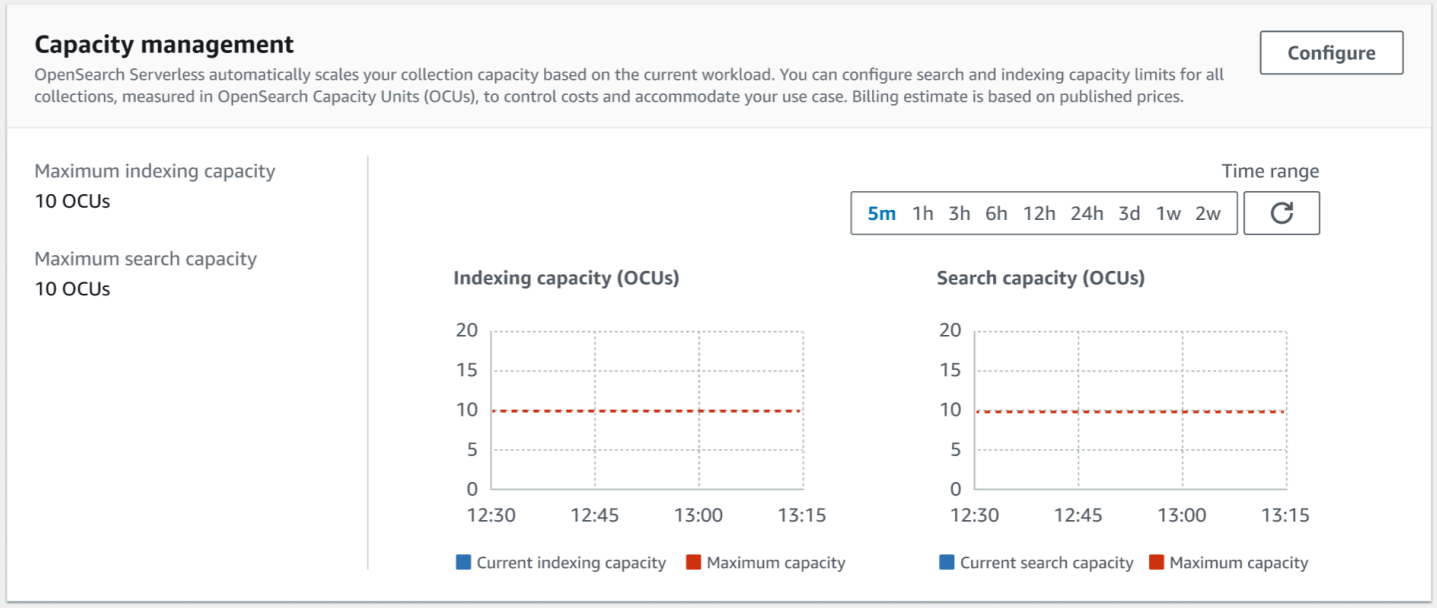
Your goal should be to ensure that the maximum capacity is high enough to handle spikes in workload. Based on your settings, OpenSearch Serverless automatically scales out the number of OCUs for your collections to process the indexing and search workload.

Topics

- Configuring capacity settings
- Maximum capacity limits
- Monitoring capacity usage

Configuring capacity settings

To configure capacity settings in the OpenSearch Serverless console, expand **Serverless** in the left navigation pane and select **Dashboard**. Specify the maximum indexing and search capacity under **Capacity management**:



To configure capacity using the AWS CLI, send an [UpdateAccountSettings] request:

```
aws opensearchserverless update-account-settings \
    --capacity-limits '{ "maxIndexingCapacityInOCU": 8,"maxSearchCapacityInOCU": 9 }'
```

Maximum capacity limits

The maximum total of indexes a collection can contain is 1000. For all three types of collections, the default maximum OCU capacity is 10 OCUs for indexing and 10 OCUs for search. The minimum OCU capacity allowed for an account is 1 OCU [0.5 OCU x 2] for indexing and 1 OCU [0.5 OCU x 2] for search. For all collections, the maximum allowed capacity is 500 OCUs for indexing and 500 OCUs for search. You can configure the OCU count to be any number from 1 to the maximum allowed capacity, in multiples of 2.

Each OCU includes enough hot ephemeral storage for 120 GiB of index data. OpenSearch Serverless supports up to 1 TiB of data per index in *search* and *vector search* collections, and 30 TiB of hot data per index in a *time series* collection. For time series collections, you can still ingest more data, which can be stored as warm data in S3.

For a list of all quotas, see OpenSearch Serverless quotas.

Monitoring capacity usage

You can monitor the `Search0CU` and `Indexing0CU` account-level CloudWatch metrics to understand how your collections are scaling. We recommend that you configure alarms to notify you if your account is approaching a threshold for metrics related to capacity, so you can adjust your capacity settings accordingly.

You can also use these metrics to determine if your maximum capacity settings are appropriate, or if you need to adjust them. Analyze these metrics to focus your efforts for optimizing the efficiency of your collections. For more information about the metrics that OpenSearch Serverless sends to CloudWatch, see the section called "Monitoring OpenSearch Serverless".

Ingesting data into Amazon OpenSearch Serverless collections

These sections provide details about the supported ingest pipelines for data ingestion into Amazon OpenSearch Serverless collections. They also cover some of the clients that you can use to interact with the OpenSearch API operations. Your clients should be compatible with OpenSearch 2.x in order to integrate with OpenSearch Serverless.

Topics

- Minimum required permissions
- OpenSearch Ingestion
- Fluent Bit
- Amazon Data Firehose
- Go
- Java
- JavaScript
- Logstash
- Python
- Ruby
- Signing HTTP requests with other clients

Minimum required permissions

In order to ingest data into an OpenSearch Serverless collection, the principal that is writing the data must have the following minimum permissions assigned in a data access policy:

```
[
   {
      "Rules":[
         {
            "ResourceType":"index",
            "Resource":[
               "index/target-collection/logs"
            ],
            "Permission":[
               "aoss:CreateIndex",
               "aoss:WriteDocument",
               "aoss:UpdateIndex"
            ]
         }
      ],
      "Principal":[
         "arn:aws:iam::123456789012:user/my-user"
      ]
   }
]
```

The permissions can be more broad if you plan to write to additional indexes. For example, rather than specifying a single target index, you can allow permission to all indexes (index/*target-collection*/*), or a subset of indexes (index/*target-collection*/*logs**).

For a reference of all available OpenSearch API operations and their associated permissions, see the section called "Supported operations and plugins".

OpenSearch Ingestion

Rather than using a third-party client to send data directly to an OpenSearch Serverless collection, you can use Amazon OpenSearch Ingestion. You configure your data producers to send data to OpenSearch Ingestion, and it automatically delivers the data to the collection that you specify. You can also configure OpenSearch Ingestion to transform your data before delivering it. For more information, see *Amazon OpenSearch Ingestion*.

An OpenSearch Ingestion pipeline needs permission to write to an OpenSearch Serverless collection that is configured as its sink. These permissions include the ability to describe the collection and send HTTP requests to it. For instructions to use OpenSearch Ingestion to add data to a collection, see the section called "Granting pipelines access to collections".

To get started with OpenSearch Ingestion, see the section called "Tutorial: Ingest data into a collection".

Fluent Bit

You can use AWS for Fluent Bit image and the OpenSearch output plugin to ingest data into OpenSearch Serverless collections.

> **Note**
>
> You must have version 2.30.0 or later of the AWS for Fluent Bit image in order to integrate with OpenSearch Serverless.

Example configuration:

This sample output section of the configuration file shows how to use an OpenSearch Serverless collection as a destination. The important addition is the `AWS_Service_Name` parameter, which is `aoss`. `Host` is the collection endpoint.

```
[OUTPUT]
    Name  opensearch
    Match *
    Host  collection-endpoint.us-west-2.aoss.amazonaws.com
    Port  443
    Index  my_index
    Trace_Error On
    Trace_Output On
    AWS_Auth On
    AWS_Region <region>
    AWS_Service_Name aoss
    tls     On
    Suppress_Type_Name On
```

Amazon Data Firehose

Firehose supports OpenSearch Serverless as a delivery destination. For instructions to send data into OpenSearch Serverless, see Creating a Kinesis Data Firehose Delivery Stream and Choose OpenSearch Serverless for Your Destination in the *Amazon Data Firehose Developer Guide*.

The IAM role that you provide to Firehose for delivery must be specified within a data access policy with the `aoss:WriteDocument` minimum permission for the target collection, and you must have a preexisting index to send data to. For more information, see the section called "Minimum required permissions".

Before you send data to OpenSearch Serverless, you might need to perform transforms on the data. To learn more about using Lambda functions to perform this task, see Amazon Kinesis Data Firehose Data Transformation in the same guide.

Go

The following sample code uses the opensearch-go client for Go to establish a secure connection to the specified OpenSearch Serverless collection and create a single index. You must provide values for `region` and `host`.

```
package main

import (
  "context"
  "log"
  "strings"
  "github.com/aws/aws-sdk-go-v2/aws"
  "github.com/aws/aws-sdk-go-v2/config"
  opensearch "github.com/opensearch-project/opensearch-go/v2"
  opensearchapi "github.com/opensearch-project/opensearch-go/v2/opensearchapi"
  requestsigner "github.com/opensearch-project/opensearch-go/v2/signer/awsv2"
)

const endpoint = "" // serverless collection endpoint

func main() {
 ctx := context.Background()

 awsCfg, err := config.LoadDefaultConfig(ctx,
  config.WithRegion("<AWS_REGION>"),
  config.WithCredentialsProvider(
```

```
  getCredentialProvider("<AWS_ACCESS_KEY>", "<AWS_SECRET_ACCESS_KEY>",
"<AWS_SESSION_TOKEN>"),
 ),
)
if err != nil {
 log.Fatal(err) // don't log.fatal in a production-ready app
}

// create an AWS request Signer and load AWS configuration using default config folder
or env vars.
signer, err := requestsigner.NewSignerWithService(awsCfg, "aoss") // "aoss" for Amazon
OpenSearch Serverless
if err != nil {
 log.Fatal(err) // don't log.fatal in a production-ready app
}

// create an opensearch client and use the request-signer
client, err := opensearch.NewClient(opensearch.Config{
 Addresses: []string{endpoint},
 Signer:    signer,
})
if err != nil {
 log.Fatal("client creation err", err)
}

indexName := "go-test-index"

 // define index mapping
mapping := strings.NewReader(`{
 "settings": {
   "index": {
        "number_of_shards": 4
        }
      }
 }`)

// create an index
createIndex := opensearchapi.IndicesCreateRequest{
 Index: indexName,
   Body: mapping,
}
createIndexResponse, err := createIndex.Do(context.Background(), client)
if err != nil {
 log.Println("Error ", err.Error())
```

```
  log.Println("failed to create index ", err)
  log.Fatal("create response body read err", err)
 }
 log.Println(createIndexResponse)

 // delete the index
 deleteIndex := opensearchapi.IndicesDeleteRequest{
  Index: []string{indexName},
 }

 deleteIndexResponse, err := deleteIndex.Do(context.Background(), client)
 if err != nil {
  log.Println("failed to delete index ", err)
  log.Fatal("delete index response body read err", err)
 }
 log.Println("deleting index", deleteIndexResponse)
}

func getCredentialProvider(accessKey, secretAccessKey, token string)
 aws.CredentialsProviderFunc {
 return func(ctx context.Context) (aws.Credentials, error) {
  c := &aws.Credentials{
   AccessKeyID:     accessKey,
   SecretAccessKey: secretAccessKey,
   SessionToken:    token,
  }
  return *c, nil
 }
}
```

Java

The following sample code uses the opensearch-java client for Java to establish a secure connection to the specified OpenSearch Serverless collection and create a single index. You must provide values for `region` and `host`.

The important difference compared to OpenSearch Service *domains* is the service name (`aoss` instead of `es`).

```
// import OpenSearchClient to establish connection to OpenSearch Serverless collection
import org.opensearch.client.opensearch.OpenSearchClient;

SdkHttpClient httpClient = ApacheHttpClient.builder().build();
```

```
// create an opensearch client and use the request-signer
OpenSearchClient client = new OpenSearchClient(
    new AwsSdk2Transport(
        httpClient,
        "...us-west-2.aoss.amazonaws.com", // serverless collection endpoint
        "aoss" // signing service name
        Region.US_WEST_2, // signing service region
        AwsSdk2TransportOptions.builder().build()
    )
);

String index = "sample-index";

// create an index
CreateIndexRequest createIndexRequest = new
 CreateIndexRequest.Builder().index(index).build();
CreateIndexResponse createIndexResponse = client.indices().create(createIndexRequest);
System.out.println("Create index reponse: " + createIndexResponse);

// delete the index
DeleteIndexRequest deleteIndexRequest = new
 DeleteIndexRequest.Builder().index(index).build();
DeleteIndexResponse deleteIndexResponse = client.indices().delete(deleteIndexRequest);
System.out.println("Delete index reponse: " + deleteIndexResponse);

httpClient.close();
```

The following sample code again establishes a secure connection, and then searches an index.

```
import org.opensearch.client.opensearch.OpenSearchClient;
>>>>>>> aoss-slr-update

SdkHttpClient httpClient = ApacheHttpClient.builder().build();

OpenSearchClient client = new OpenSearchClient(
    new AwsSdk2Transport(
        httpClient,
        "...us-west-2.aoss.amazonaws.com", // serverless collection endpoint
        "aoss" // signing service name
        Region.US_WEST_2, // signing service region
        AwsSdk2TransportOptions.builder().build()
    )
);
```

```
Response response = client.generic()
    .execute(
        Requests.builder()
            .endpoint("/" + "users" + "/_search?typed_keys=true")
            .method("GET")
            .json("{"
                + "    \"query\": {"
                + "        \"match_all\": {}"
                + "    }"
                + "}")
            .build());

httpClient.close();
```

JavaScript

The following sample code uses the opensearch-js client for JavaScript to establish a secure connection to the specified OpenSearch Serverless collection, create a single index, add a document, and delete the index. You must provide values for `node` and `region`.

The important difference compared to OpenSearch Service *domains* is the service name (`aoss` instead of `es`).

Version 3

This example uses version 3 of the SDK for JavaScript in Node.js.

```
const { defaultProvider } = require('@aws-sdk/credential-provider-node');
const { Client } = require('@opensearch-project/opensearch');
const { AwsSigv4Signer } = require('@opensearch-project/opensearch/aws');

async function main() {
    // create an opensearch client and use the request-signer
    const client = new Client({
        ...AwsSigv4Signer({
            region: 'us-west-2',
            service: 'aoss',
            getCredentials: () => {
                const credentialsProvider = defaultProvider();
                return credentialsProvider();
            },
        }),
```

```
        node: '' # // serverless collection endpoint
    });

    const index = 'movies';

    // create index if it doesn't already exist
    if (!(await client.indices.exists({ index })).body) {
        console.log((await client.indices.create({ index })).body);
    }

    // add a document to the index
    const document = { foo: 'bar' };
    const response = await client.index({
        id: '1',
        index: index,
        body: document,
    });
    console.log(response.body);

    // delete the index
    console.log((await client.indices.delete({ index })).body);
}

main();
```

Version 2

This example uses version 2 of the SDK for JavaScript in Node.js.

```
const AWS = require('aws-sdk');
const { Client } = require('@opensearch-project/opensearch');
const { AwsSigv4Signer } = require('@opensearch-project/opensearch/aws');

async function main() {
    // create an opensearch client and use the request-signer
    const client = new Client({
        ...AwsSigv4Signer({
            region: 'us-west-2',
            service: 'aoss',
            getCredentials: () =>
                new Promise((resolve, reject) => {
                    AWS.config.getCredentials((err, credentials) => {
                        if (err) {
                            reject(err);
```

```
                        } else {
                            resolve(credentials);
                        }
                    });
                }),
        }),
        node: '' # // serverless collection endpoint
    });

    const index = 'movies';

    // create index if it doesn't already exist
    if (!(await client.indices.exists({ index })).body) {
        console.log((await client.indices.create({
            index
        })).body);
    }

    // add a document to the index
    const document = {
        foo: 'bar'
    };
    const response = await client.index({
        id: '1',
        index: index,
        body: document,
    });
    console.log(response.body);

    // delete the index
    console.log((await client.indices.delete({ index })).body);
}

main();
```

Logstash

You can use the Logstash OpenSearch plugin to publish logs to OpenSearch Serverless collections.

To use Logstash to send data to OpenSearch Serverless

1. Install version *2.0.0 or later* of the logstash-output-opensearch plugin using Docker or Linux.

Docker

Docker hosts the Logstash OSS software with the OpenSearch output plugin preinstalled: opensearchproject/logstash-oss-with-opensearch-output-plugin. You can pull the image just like any other image:

```
docker pull opensearchproject/logstash-oss-with-opensearch-output-plugin:latest
```

Linux

First, install the latest version of Logstash if you haven't already. Then, install version 2.0.0 of the output plugin:

```
cd logstash-8.5.0/
bin/logstash-plugin install --version 2.0.0 logstash-output-opensearch
```

If the plugin is already installed, update it to the latest version:

```
bin/logstash-plugin update logstash-output-opensearch
```

Starting with version 2.0.0 of the plugin, the AWS SDK uses version 3. If you're using a Logstash version earlier than 8.4.0, you must remove any pre-installed AWS plugins and install the `logstash-integration-aws` plugin:

```
/usr/share/logstash/bin/logstash-plugin remove logstash-input-s3
/usr/share/logstash/bin/logstash-plugin remove logstash-input-sqs
/usr/share/logstash/bin/logstash-plugin remove logstash-output-s3
/usr/share/logstash/bin/logstash-plugin remove logstash-output-sns
/usr/share/logstash/bin/logstash-plugin remove logstash-output-sqs
/usr/share/logstash/bin/logstash-plugin remove logstash-output-cloudwatch

/usr/share/logstash/bin/logstash-plugin install --version 0.1.0.pre logstash-
integration-aws
```

2. In order for the OpenSearch output plugin to work with OpenSearch Serverless, you must make the following modifications to the `opensearch` output section of logstash.conf:

   - Specify `aoss` as the `service_name` under `auth_type`.
   - Specify your collection endpoint for `hosts`.

- Add the parameters `default_server_major_version` and `legacy_template`. These parameters are required for the plugin to work with OpenSearch Serverless.

```
output {
  opensearch {
    hosts => "collection-endpoint:443"
    auth_type => {
      ...
      service_name => 'aoss'
    }
    default_server_major_version => 2
    legacy_template => false
  }
}
```

This example configuration file takes its input from files in an S3 bucket and sends them to an OpenSearch Serverless collection:

```
input {
  s3  {
    bucket => "my-s3-bucket"
    region => "us-east-1"
  }
}

output {
  opensearch {
    ecs_compatibility => disabled
    hosts => "https://my-collection-endpoint.us-east-1.aoss.amazonaws.com:443"
    index => my-index
    auth_type => {
      type => 'aws_iam'
      aws_access_key_id => 'your-access-key'
      aws_secret_access_key => 'your-secret-key'
      region => 'us-east-1'
      service_name => 'aoss'
    }
    default_server_major_version => 2
    legacy_template => false
  }
}
```

3. Then, run Logstash with the new configuration to test the plugin:

```
bin/logstash -f config/test-plugin.conf
```

Python

The following sample code uses the opensearch-py client for Python to establish a secure connection to the specified OpenSearch Serverless collection, create a single index, and search that index. You must provide values for `region` and `host`.

The important difference compared to OpenSearch Service *domains* is the service name (`aoss` instead of `es`).

```
from opensearchpy import OpenSearch, RequestsHttpConnection, AWSV4SignerAuth
import boto3

host = ''  # serverless collection endpoint, without https://
region = ''  # e.g. us-east-1

service = 'aoss'
credentials = boto3.Session().get_credentials()
auth = AWSV4SignerAuth(credentials, region, service)

# create an opensearch client and use the request-signer
client = OpenSearch(
    hosts=[{'host': host, 'port': 443}],
    http_auth=auth,
    use_ssl=True,
    verify_certs=True,
    connection_class=RequestsHttpConnection,
    pool_maxsize=20,
)

# create an index
index_name = 'books-index'
create_response = client.indices.create(
    index_name
)

print('\nCreating index:')
print(create_response)
```

```
# index a document
document = {
  'title': 'The Green Mile',
  'director': 'Stephen King',
  'year': '1996'
}

response = client.index(
    index = 'books-index',
    body = document,
    id = '1'
)


# delete the index
delete_response = client.indices.delete(
    index_name
)

print('\nDeleting index:')
print(delete_response)
```

Ruby

The `opensearch-aws-sigv4` gem provides access to OpenSearch Serverless, along with OpenSearch Service, out of the box. It has all features of the opensearch-ruby client because it's a dependency of this gem.

When instantiating the Sigv4 signer, specify `aoss` as the service name:

```
require 'opensearch-aws-sigv4'
require 'aws-sigv4'

signer = Aws::Sigv4::Signer.new(service: 'aoss',
                                region: 'us-west-2',
                                access_key_id: 'key_id',
                                secret_access_key: 'secret')

# create an opensearch client and use the request-signer
client = OpenSearch::Aws::Sigv4Client.new(
  { host: 'https://your.amz-opensearch-serverless.endpoint',
    log: true },
```

```
  signer)

# create an index
index = 'prime'
client.indices.create(index: index)

# insert data
client.index(index: index, id: '1', body: { name: 'Amazon Echo',
                                            msrp: '5999',
                                            year: 2011 })

# query the index
client.search(body: { query: { match: { name: 'Echo' } } })

# delete index entry
client.delete(index: index, id: '1')

# delete the index
client.indices.delete(index: index)
```

Signing HTTP requests with other clients

The following requirements apply when signing requests to OpenSearch Serverless collections when you construct HTTP requests with another clients.

- You must specify the service name as `aoss`.
- The `x-amz-content-sha256` header is required for all AWS Signature Version 4 requests. It provides a hash of the request payload. If there's a request payload, set the value to its Secure Hash Algorithm (SHA) cryptographic hash (SHA256). If there's no request payload, set the value to `e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855`, which is the hash of an empty string.

Topics

- Indexing with cURL
- Indexing with Postman

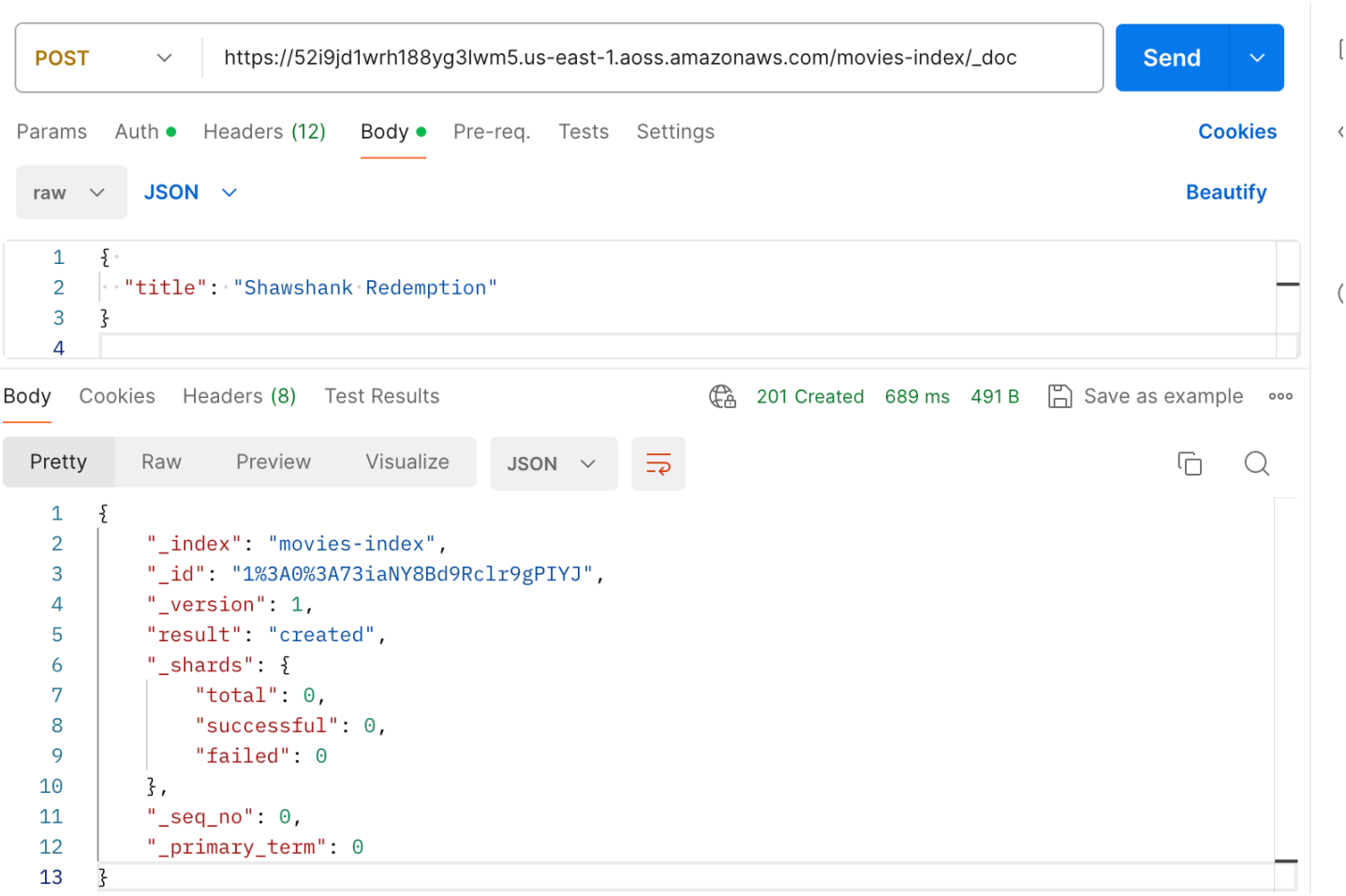
Indexing with cURL

The following example request uses the Client URL Request Library (cURL) to send a single document to an index named `movies-index` within a collection:

```
curl -XPOST \
    --user "$AWS_ACCESS_KEY_ID":"$AWS_SECRET_ACCESS_KEY" \
    --aws-sigv4 "aws:amz:us-east-1:aoss" \
    --header "x-amz-content-sha256: $REQUEST_PAYLOAD_SHA_HASH" \
    --header "x-amz-security-token: $AWS_SESSION_TOKEN" \
    "https://my-collection-endpoint.us-east-1.aoss.amazonaws.com/movies-index/_doc" \
    -H "Content-Type: application/json" -d '{"title": "Shawshank Redemption"}'
```

Indexing with Postman

The following image shows how to send a requests to a collection using Postman. For instructions to authenticate, see Authenticate with AWS Signature authentication workflow in Postman.

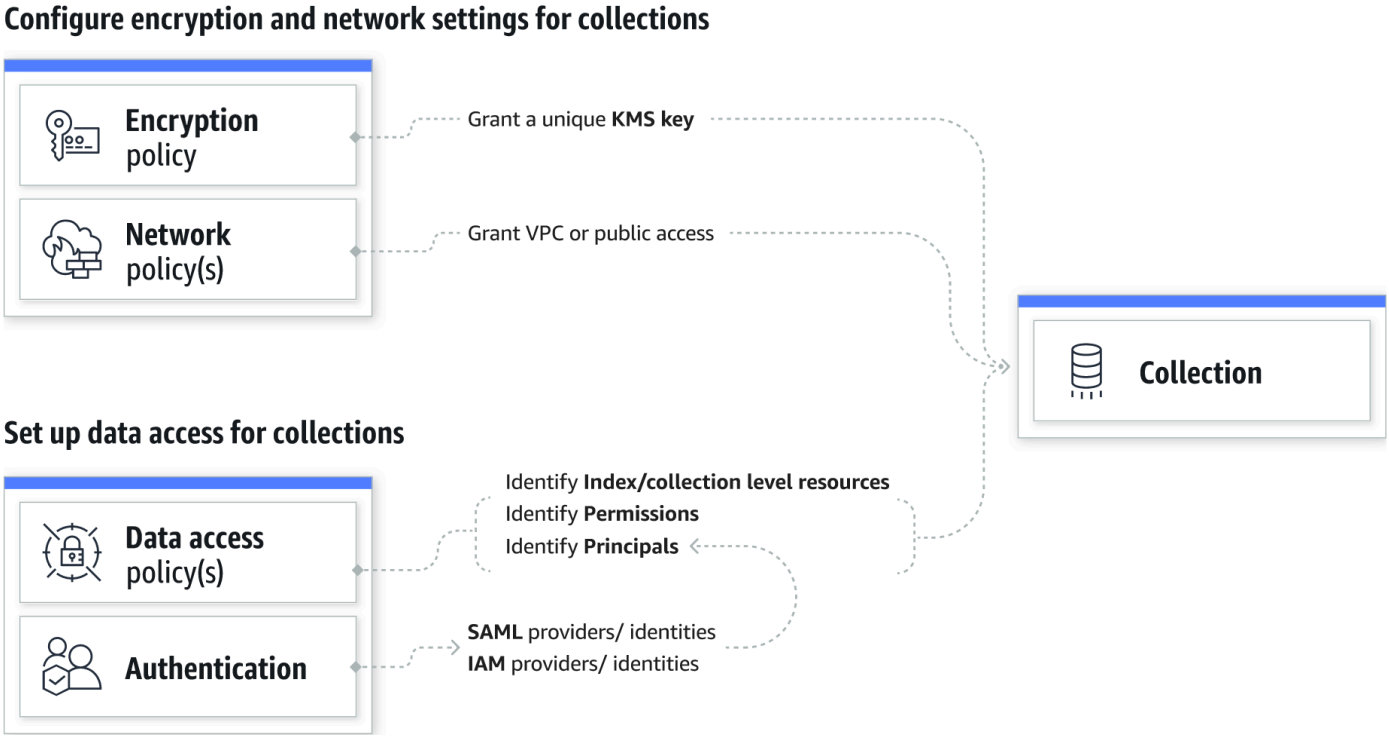


Overview of security in Amazon OpenSearch Serverless

Security in Amazon OpenSearch Serverless differs fundamentally from security in Amazon OpenSearch Service in the following ways:

Feature	OpenSearch Service	OpenSearch Serverless
Data access control	Data access is determined by IAM policies and fine-grained access control.	Data access is determined by data access policies.
Encryption at rest	Encryption at rest is *optional* for domains.	Encryption at rest is *required* for collections.
Security setup and administration	You must configure network, encryption, and data access individually for each domain.	You can use security policies to manage security settings for multiple collections at scale.

The following diagram illustrates the security components that make up a functional collection. A collection must have an assigned encryption key, network access settings, and a matching data access policy that grants permission to its resources.

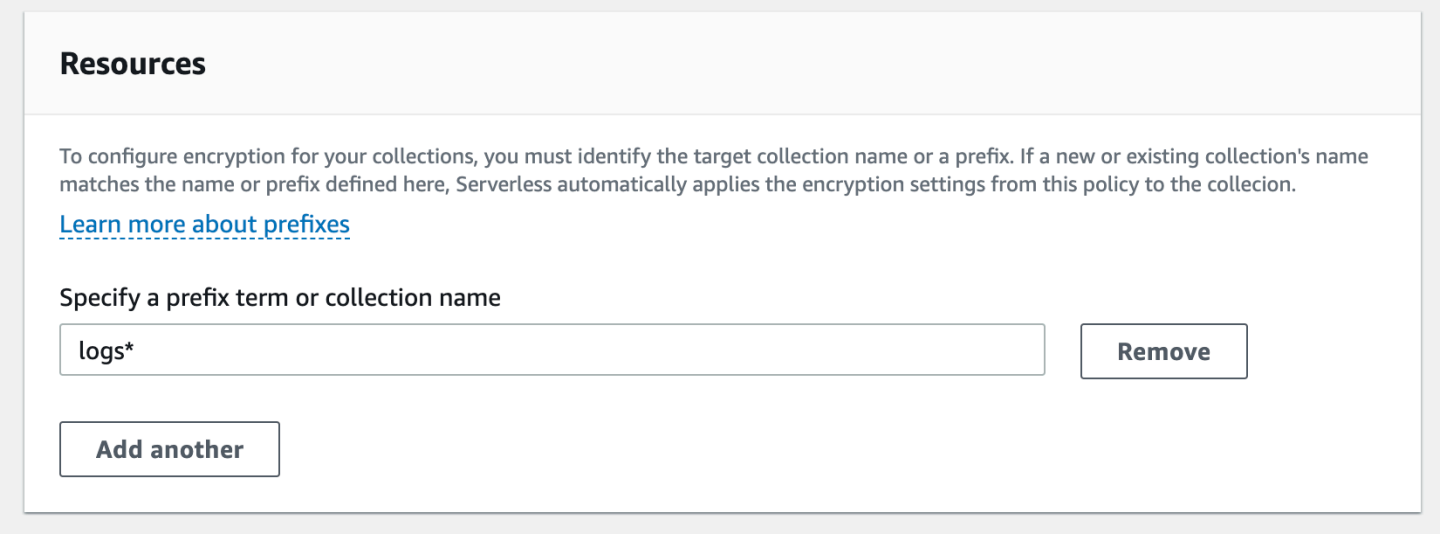


Topics

Encryption policies

Encryption policies define whether your collections are encrypted with an AWS owned key or a customer managed key. Encryption policies consist of two components: a **resource pattern** and an **encryption key**. The resource pattern defines which collection or collections the policy applies to. The encryption key determines how the associated collections will be secured.

To apply a policy to multiple collections, you include a wildcard (*) in the policy rule. For example, the following policy applies to all collections with names that begin with "logs".

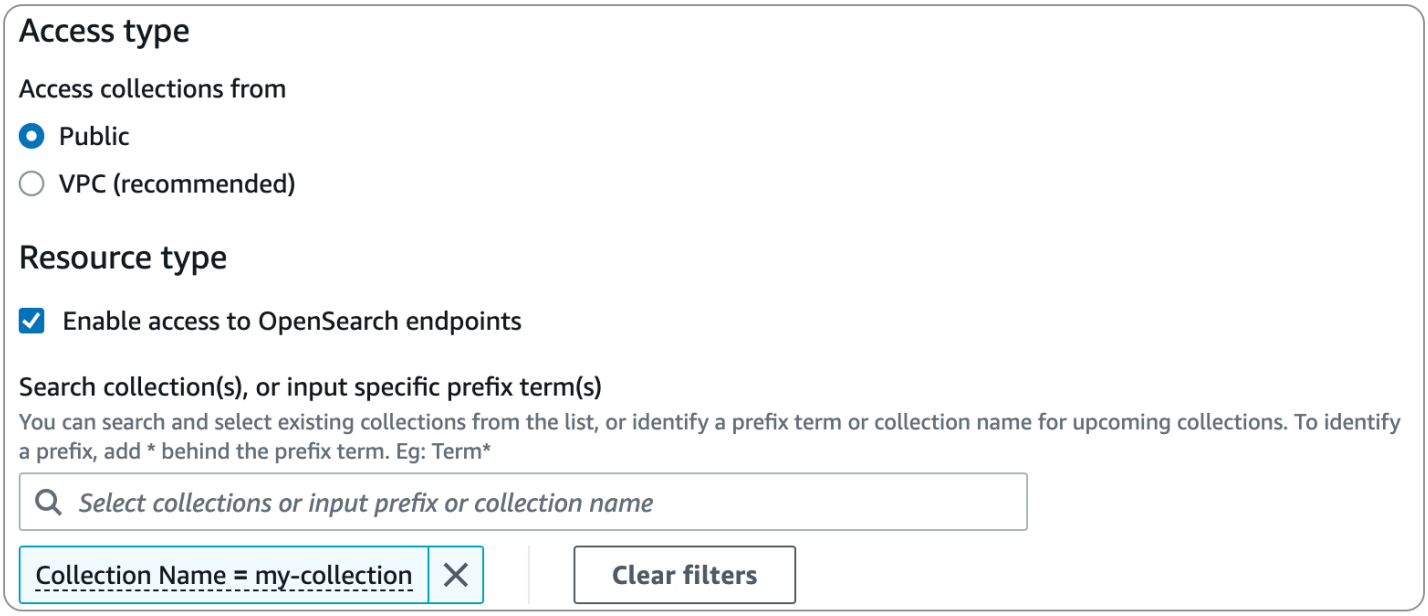


Encryption policies streamline the process of creating and managing collections, especially when you do so programmatically. You can create a collection by simply specifying a name, and an encryption key is automatically assigned to it upon creation.

Network policies

Network policies define whether your collections are accessible privately, or over the internet from public networks. Private collections can be accessed through OpenSearch Serverless–managed VPC endpoints, or by specific AWS services such as Amazon Bedrock using *AWS service private access*. Just like encryption policies, network policies can apply to multiple collections, which allows you to manage network access for many collections at scale.

Network policies consist of two components: an **access type** and a **resource type**. The access type can either be public or private. The resource type determines whether the access you choose applies to the collection endpoint, the OpenSearch Dashboards endpoint, or both.



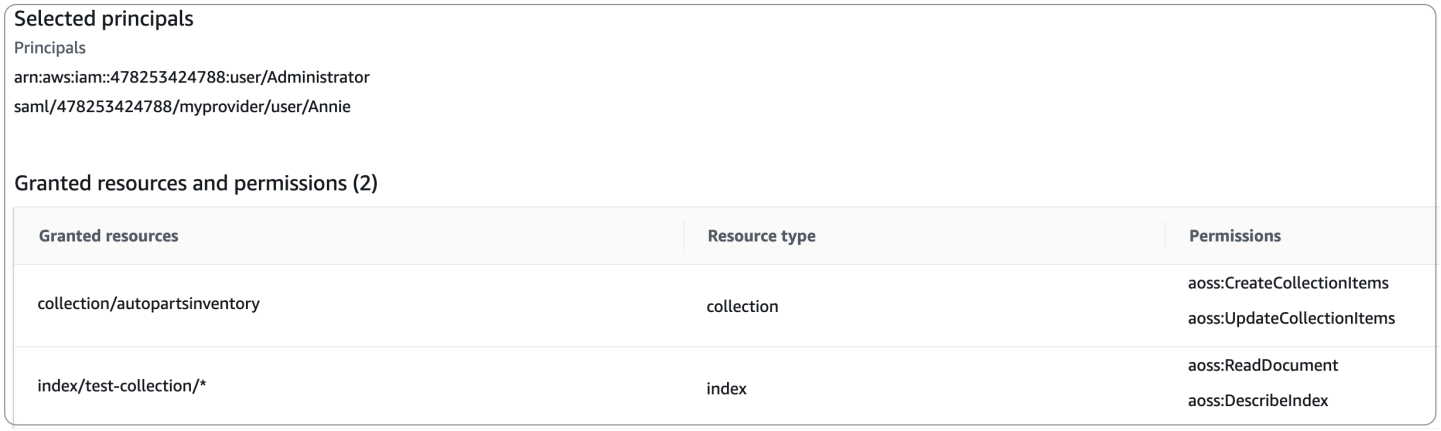
If you plan to configure VPC access within a network policy, you must first create one or more OpenSearch Serverless-managed VPC endpoints. These endpoints let you access OpenSearch Serverless as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection.

Private access to AWS services can only apply to the collection's OpenSearch endpoint, not to the OpenSearch Dashboards endpoint. AWS services cannot be granted access to OpenSearch Dashboards.

Data access policies

Data access policies define how your users access the data within your collections. Data access policies help you manage collections at scale by automatically assigning access permissions to collections and indexes that match a specific pattern. Multiple policies can apply to a single resource.

Data access policies consist of a set of rules, each with three components: a **resource type**, **granted resources**, and a set of **permissions**. The resource type can be a collection or index. The granted resources can be collection/index names or patterns with a wildcard (*). The list of permissions specifies which OpenSearch API operations the policy grants access to. In addition, the policy contains a list of **principals**, which specify the IAM roles, users, and SAML identities to grant access to.



For more information about the format of a data access policy, see the policy syntax.

Before you create a data access policy, you must have one or more IAM roles or users, or SAML identities, to provide access to in the policy. For details, see the next section.

> **Note**
>
> Switching from Public to Private Access for your collection, will remove the Indexes Tab in the OpenSearch Serverless Collection Console.

IAM and SAML authentication

IAM principals and SAML identities are one of the building blocks of a data access policy. Within the `principal` statement of an access policy, you can include IAM roles, users, and SAML identities. These principals are then granted the permissions that you specify in the associated policy rules.

```
[
   {
      "Rules":[
         {
            "ResourceType":"index",
            "Resource":[
               "index/marketing/orders*"
            ],
            "Permission":[
               "aoss:*"
            ]
         }
```

```
      ],
      "Principal":[
         "arn:aws:iam::123456789012:user/Dale",
         "arn:aws:iam::123456789012:role/RegulatoryCompliance",
         "saml/123456789012/myprovider/user/Annie"
      ]
   }
]
```

You configure SAML authentication directly within OpenSearch Serverless. For more information, see the section called “SAML authentication”.

Infrastructure security

Amazon OpenSearch Serverless is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon OpenSearch Serverless through the network. Clients must support Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3. For a list of supported ciphers for TLS 1.3, see TLS protocols and ciphers in the Elastic Load Balancing documentation.

Additionally, you must sign requests using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Getting started with security in Amazon OpenSearch Serverless

The following tutorials help you get started using Amazon OpenSearch Serverless. Both tutorials accomplish the same basic steps, but one uses the console while the other uses the AWS CLI.

Note that the use cases in these tutorials are simplified. The network and security policies are fairly open. In production workloads, we recommend that you configure more robust security features such as SAML authentication, VPC access, and restrictive data access policies.

Topics

- Tutorial: Getting started with security in Amazon OpenSearch Serverless (console)

- Tutorial: Getting started with security in Amazon OpenSearch Serverless (CLI)

Tutorial: Getting started with security in Amazon OpenSearch Serverless (console)

This tutorial walks you through the basic steps to create and manage security policies using the Amazon OpenSearch Serverless console.

You will complete the following steps in this tutorial:

1. Configure permissions
2. Create an encryption policy
3. Create a network policy
4. Configure a data access policy
5. Create a collection
6. Upload and search data

This tutorial walks you through setting up a collection using the AWS Management Console. For the same steps using the AWS CLI, see the section called "Tutorial: Getting started with security (CLI)".

Step 1: Configure permissions

> **Note**
>
> You can skip this step if you're already using a more broad identity-based policy, such as `Action":"aoss:*"` or `Action":"*"`. In production environments, however, we recommend that you follow the principal of least privilege and only assign the minimum permissions necessary to complete a task.

In order to complete this tutorial, you must have the correct IAM permissions. Your user or role must have an attached identity-based policy with the following minimum permissions:

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Action": [
        "aoss:ListCollections",
        "aoss:BatchGetCollection",
        "aoss:CreateCollection",
        "aoss:CreateSecurityPolicy",
        "aoss:GetSecurityPolicy",
        "aoss:ListSecurityPolicies",
        "aoss:CreateAccessPolicy",
        "aoss:GetAccessPolicy",
        "aoss:ListAccessPolicies"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

For a full list of OpenSearch Serverless permissions, see the section called "Identity and Access Management".

Step 2: Create an encryption policy

Encryption policies specify the AWS KMS key that OpenSearch Serverless will use to encrypt the collection. You can encrypt collections with an AWS managed key or a different key. For simplicity in this tutorial, we'll encrypt our collection with an AWS managed key.

To create an encryption policy

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Expand **Serverless** in the left navigation pane and choose **Encryption policies**.
3. Choose **Create encryption policy**.
4. Name the policy **books-policy**. For the description, enter **Encryption policy for books collection**.
5. Under **Resources**, enter **books**, which is what you'll name your collection. If you wanted to be more broad, you could include an asterisk (`books*`) to make the policy apply to all collections beginning with the word "books".
6. For **Encryption**, keep **Use AWS owned key** selected.
7. Choose **Create**.

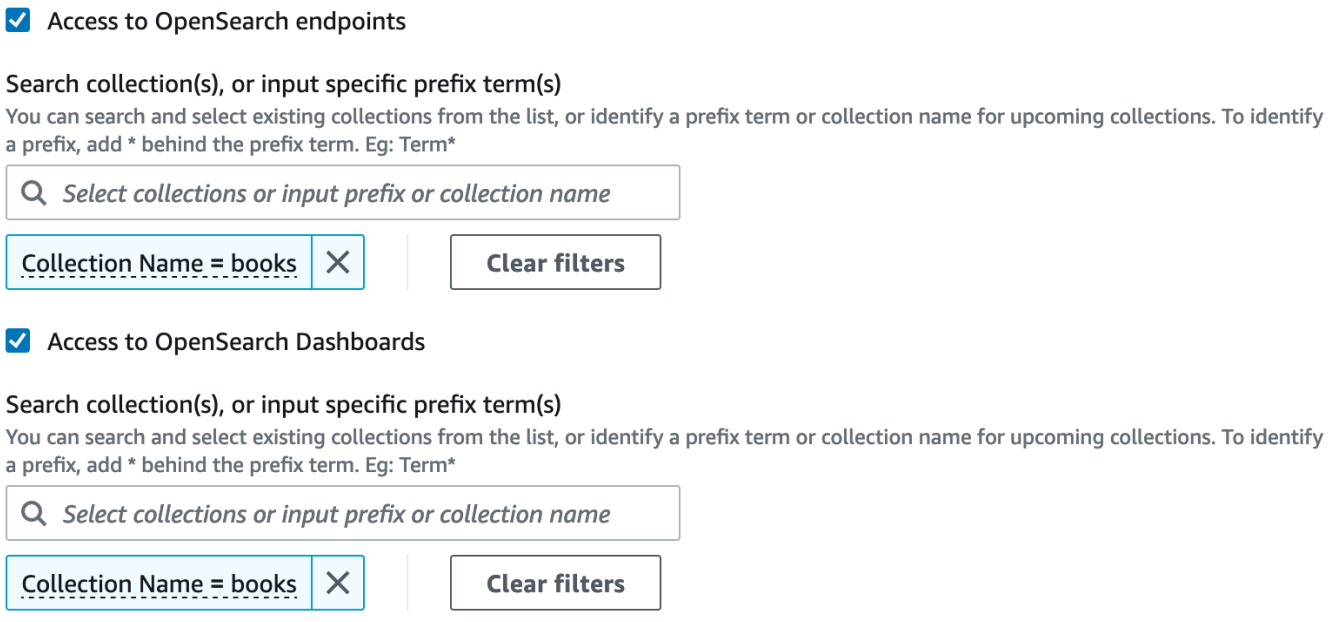
Step 3: Create a network policy

Network policies determine whether your collection is accessible over the internet from public networks, or whether it must be accessed through OpenSearch Serverless–managed VPC endpoints. In this tutorial, we'll configure public access.

To create a network policy

1. Choose **Network policies** in the left navigation pane and choose **Create network policy**.
2. Name the policy **books-policy**. For the description, enter **Network policy for books collection**.
3. Under **Rule 1**, name the rule **Public access for books collection**.
4. For simplicity in this tutorial, we'll configure public access for the *books* collection. For the access type, select **Public**.
5. We're going to access the collection from OpenSearch Dashboards. In order to do this, you need to configure network access for Dashboards *and* the OpenSearch endpoint, otherwise Dashboards won't function.

   For the resource type, enable both **Access to OpenSearch endpoints** and **Access to OpenSearch Dashboards**.
6. In both input boxes, enter **Collection Name = books**. This setting scopes the policy down so that it only applies to a single collection (`books`). Your rule should look like this:



7. Choose **Create**.

Step 4: Create a data access policy

Your collection data won't be accessible until you configure data access. Data access policies are separate from the IAM identity-based policy that you configured in step 1. They allow users to access the actual data within a collection.

In this tutorial, we'll provide a single user the permissions required to index data into the *books* collection.

To create a data access policy

1. Choose **Data access policies** in the left navigation pane and choose **Create access policy**.
2. Name the policy **books-policy**. For the description, enter **Data access policy for books collection**.
3. Select **JSON** for the policy definition method and paste the following policy in the JSON editor.

   Replace the principal ARN with the ARN of the account that you'll use to log in to OpenSearch Dashboards and index data.

```
[
   {
      "Rules":[
         {
            "ResourceType":"index",
            "Resource":[
               "index/books/*"
            ],
            "Permission":[
               "aoss:CreateIndex",
               "aoss:DescribeIndex",
               "aoss:ReadDocument",
               "aoss:WriteDocument",
               "aoss:UpdateIndex",
               "aoss:DeleteIndex"
            ]
         }
      ],
      "Principal":[
         "arn:aws:iam::123456789012:user/my-user"
      ]
   }
```

```
]
```

This policy provides a single user the minimum permissions required to create an index in the *books* collection, index some data, and search for it.

4. Choose **Create**.

Step 5: Create a collection

Now that you configured encryption and network policies, you can create a matching collection and the security settings will be automatically applied to it.

To create an OpenSearch Serverless collection

1. Choose **Collections** in the left navigation pane and choose **Create collection**.
2. Name the collection **books**.
3. For collection type, choose **Search**.
4. Under **Encryption**, OpenSearch Serverless informs you that the collection name matches the `books-policy` encryption policy.
5. Under **Network access settings**, OpenSearch Serverless informs you that the collection name matches the `books-policy` network policy.
6. Choose **Next**.
7. Under **Data access policy options**, OpenSearch Serverless informs you that the collection name matches the `books-policy` data access policy.
8. Choose **Next**.
9. Review the collection configuration and choose **Submit**. Collections typically take less than a minute to initialize.

Step 6: Upload and search data

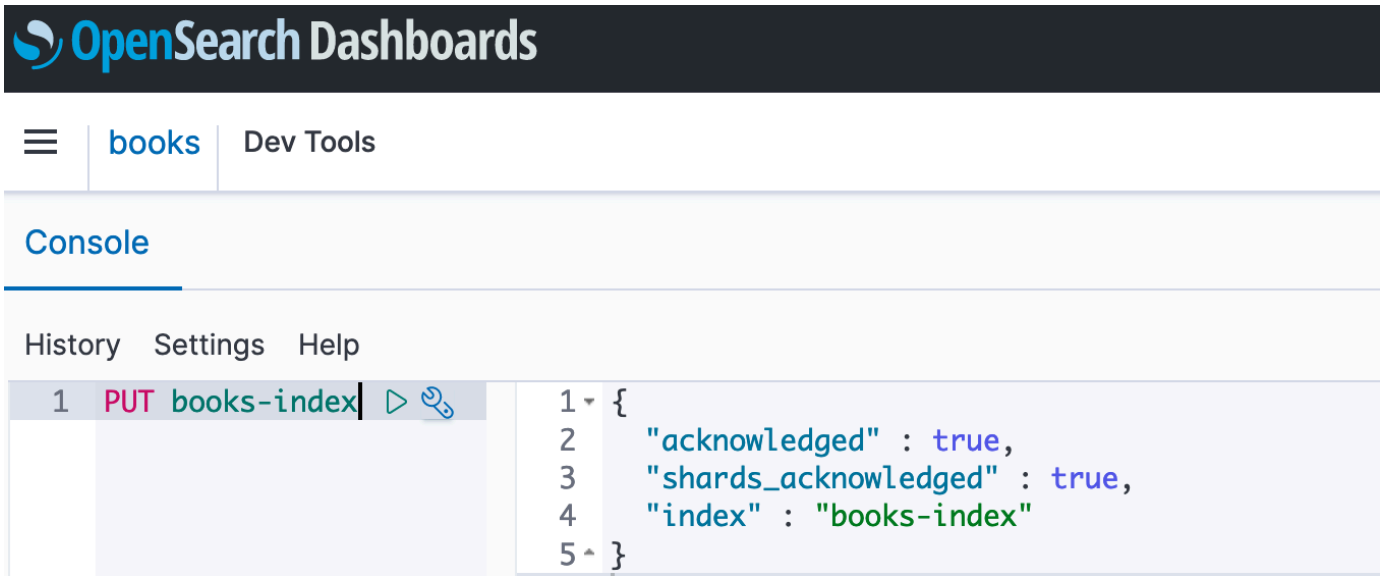
You can upload data to an OpenSearch Serverless collection using Postman or curl. For brevity, these examples use **Dev Tools** within the OpenSearch Dashboards console.

To index and search data in a collection

1. Choose **Collections** in the left navigation pane and choose the **books** collection to open its details page.

2. Choose the OpenSearch Dashboards URL for the collection. The URL takes the format `https://collection-id.us-east-1.aoss.amazonaws.com/_dashboards`.
3. Sign in to OpenSearch Dashboards using the AWS access and secret keys for the principal that you specified in your data access policy.
4. Within OpenSearch Dashboards, open the left navigation menu and choose **Dev Tools**.
5. To create a single index called *books-index*, run the following command:

```
PUT books-index
```



6. To index a single document into *books-index*, run the following command:

```
PUT books-index/_doc/1
{
  "title": "The Shining",
  "author": "Stephen King",
  "year": 1977
}
```

7. To search data in OpenSearch Dashboards, you need to configure at least one index pattern. OpenSearch uses these patterns to identify which indexes you want to analyze. Open the Dashboards main menu, choose **Stack Management**, choose **Index Patterns**, and then choose **Create index pattern**. For this tutorial, enter *books-index*.
8. Choose **Next step** and then choose **Create index pattern**. After the pattern is created, you can view the various document fields such as `author` and `title`.

9. To begin searching your data, open the main menu again and choose **Discover**, or use the search API.

Tutorial: Getting started with security in Amazon OpenSearch Serverless (CLI)

This tutorial walks you through the steps described in the console getting started tutorial for security, but uses the AWS CLI rather than the OpenSearch Service console.

You'll complete the following steps in this tutorial:

1. Create an IAM permissions policy
2. Attatch the IAM policy to an IAM role
3. Create an encryption policy
4. Create a network policy
5. Create a collection
6. Configure a data access policy
7. Retrieve the collection endpoint
8. Upload data to your connection
9. Search data in your collection

The goal of this tutorial is to set up a single OpenSearch Serverless collection with fairly simple encryption, network, and data access settings. For example, we'll configure public network access, an AWS managed key for encryption, and a simplified data access policy that grants minimal permissions to a single user.

In a production scenario, consider implementing a more robust configuration, including SAML authentication, a custom encryption key, and VPC access.

To get started with security policies in OpenSearch Serverless

1.
> **Note**
>
> You can skip this step if you're already using a more broad identity-based policy, such as `Action":"aoss:*"` or `Action":"*"`. In production environments, however, we recommend that you follow the principal of least privilege and only assign the minimum permissions necessary to complete a task.

To start, create an AWS Identity and Access Management policy with the minimum required permissions to perform the steps in this tutorial. We'll name the policy `TutorialPolicy`:

```
aws iam create-policy \
  --policy-name TutorialPolicy \
  --policy-document "{\"Version\": \"2012-10-17\",\"Statement\":
 [{\"Action\": [\"aoss:ListCollections\",\"aoss:BatchGetCollection\",
\"aoss:CreateCollection\",\"aoss:CreateSecurityPolicy\",\"aoss:GetSecurityPolicy\",
\"aoss:ListSecurityPolicies\",\"aoss:CreateAccessPolicy\",\"aoss:GetAccessPolicy\",
\"aoss:ListAccessPolicies\"],\"Effect\": \"Allow\",\"Resource\": \"*\"}]}"
```

Sample response

```
{
    "Policy": {
        "PolicyName": "TutorialPolicy",
        "PolicyId": "ANPAW6WRAECKG6QJWUV7U",
        "Arn": "arn:aws:iam::123456789012:policy/TutorialPolicy",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2022-10-16T20:57:18+00:00",
        "UpdateDate": "2022-10-16T20:57:18+00:00"
    }
}
```

2. Attach `TutorialPolicy` to the IAM role who will index and search data in the collection. We'll name the user `TutorialRole`:

```
aws iam attach-role-policy \
  --role-name TutorialRole \
  --policy-arn arn:aws:iam::123456789012:policy/TutorialPolicy
```

3. Before you create a collection, you need to create an encryption policy that assigns an AWS owned key to the *books* collection that you'll create in a later step.

   Send the following request to create an encryption policy for the *books* collection:

```
aws opensearchserverless create-security-policy \
  --name books-policy \
  --type encryption --policy "{\"Rules\":[{\"ResourceType\":\"collection\",
\"Resource\":[\"collection\/books\"]}],\"AWSOwnedKey\":true}"
```

Sample response

```
{
    "securityPolicyDetail": {
        "type": "encryption",
        "name": "books-policy",
        "policyVersion": "MTY2OTI0MDAwNTk5MF8x",
        "policy": {
            "Rules": [
                {
                    "Resource": [
                        "collection/books"
                    ],
                    "ResourceType": "collection"
                }
            ],
            "AWSOwnedKey": true
        },
        "createdDate": 1669240005990,
        "lastModifiedDate": 1669240005990
    }
}
```

4. Create a network policy that provides public access to the *books* collection:

```
aws opensearchserverless create-security-policy --name books-policy --type network
 \
  --policy "[{\"Description\":\"Public access for books collection\",\"Rules
\":[{\"ResourceType\":\"dashboard\",\"Resource\":[\"collection\/books\"]},
{\"ResourceType\":\"collection\",\"Resource\":[\"collection\/books\"]}],
\"AllowFromPublic\":true}]"
```

Sample response

```
{
    "securityPolicyDetail": {
```

```
        "type": "network",
        "name": "books-policy",
        "policyVersion": "MTY2OTI0MDI1Njk1NV8x",
        "policy": [
            {
                "Rules": [
                    {
                        "Resource": [
                            "collection/books"
                        ],
                        "ResourceType": "dashboard"
                    },
                    {
                        "Resource": [
                            "collection/books"
                        ],
                        "ResourceType": "collection"
                    }
                ],
                "AllowFromPublic": true,
                "Description": "Public access for books collection"
            }
        ],
        "createdDate": 1669240256955,
        "lastModifiedDate": 1669240256955
    }
}
```

5. Create the *books* collection:

```
aws opensearchserverless create-collection --name books --type SEARCH
```

Sample response

```
{
    "createCollectionDetail": {
        "id": "8kw362bpwg4gx9b2f6e0",
        "name": "books",
        "status": "CREATING",
        "type": "SEARCH",
        "arn": "arn:aws:aoss:us-
east-1:123456789012:collection/8kw362bpwg4gx9b2f6e0",
        "kmsKeyArn": "auto",
```

```
        "createdDate": 1669240325037,
        "lastModifiedDate": 1669240325037
    }
}
```

6. Create a data access policy that provides the minimum permissions to index and search data in the *books* collection. Replace the principal ARN with the ARN of `TutorialRole` from step 1:

```
aws opensearchserverless create-access-policy \
  --name books-policy \
  --type data \
  --policy "[{\"Rules\":[{\"ResourceType\":\"index\",\"Resource\":
[\"index\/books\/books-index\"],\"Permission\":[\"aoss:CreateIndex
\",\"aoss:DescribeIndex\",\"aoss:ReadDocument\",\"aoss:WriteDocument
\",\"aoss:UpdateIndex\",\"aoss:DeleteIndex\"]}],\"Principal\":
[\"arn:aws:iam::123456789012:role\/TutorialRole\"]}]"
```

Sample response

```
{
    "accessPolicyDetail": {
        "type": "data",
        "name": "books-policy",
        "policyVersion": "MTY2OTI0MDM5NDY1M18x",
        "policy": [
            {
                "Rules": [
                    {
                        "Resource": [
                            "index/books/books-index"
                        ],
                        "Permission": [
                            "aoss:CreateIndex",
                            "aoss:DescribeIndex",
                            "aoss:ReadDocument",
                            "aoss:WriteDocument",
                            "aoss:UpdateDocument",
                            "aoss:DeleteDocument"
                        ],
                        "ResourceType": "index"
                    }
                ],
                "Principal": [
```

```
                    "arn:aws:iam::123456789012:role/TutorialRole"
                ]
            }
        ],
        "createdDate": 1669240394653,
        "lastModifiedDate": 1669240394653
    }
}
```

`TutorialRole` should now be able to index and search documents in the *books* collection.

7. To make calls to the OpenSearch API, you need the collection endpoint. Send the following request to retrieve the `collectionEndpoint` parameter:

```
aws opensearchserverless batch-get-collection --names books
```

Sample response

```
{
    "collectionDetails": [
        {
            "id": "8kw362bpwg4gx9b2f6e0",
            "name": "books",
            "status": "ACTIVE",
            "type": "SEARCH",
            "description": "",
            "arn": "arn:aws:aoss:us-
east-1:123456789012:collection/8kw362bpwg4gx9b2f6e0",
            "createdDate": 1665765327107,
            "collectionEndpoint": "https://8kw362bpwg4gx9b2f6e0.us-
east-1.aoss.amazonaws.com",
            "dashboardEndpoint": "https://8kw362bpwg4gx9b2f6e0.us-
east-1.aoss.amazonaws.com/_dashboards"
        }
    ],
    "collectionErrorDetails": []
}
```

> **Note**
>
> You won't be able to see the collection endpoint until the collection status changes to `ACTIVE`. You might have to make multiple calls to check the status until the collection is successfully created.

8. Use an HTTP tool such as Postman or curl to index data into the *books* collection. We'll create an index called *books-index* and add a single document.

   Send the following request to the collection endpoint that you retrieved in the previous step, using the credentials for `TutorialRole`.

```
PUT https://8kw362bpwg4gx9b2f6e0.us-east-1.aoss.amazonaws.com/books-index/_doc/1
{
  "title": "The Shining",
  "author": "Stephen King",
  "year": 1977
}
```

   Sample response

```
{
  "_index" : "books-index",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 0,
    "successful" : 0,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 0
}
```

9. To begin searching data in your collection, use the search API. The following query performs a basic search:

```
GET https://8kw362bpwg4gx9b2f6e0.us-east-1.aoss.amazonaws.com/books-index/_search
```

Sample response


```
{
    "took": 405,
    "timed_out": false,
    "_shards": {
        "total": 6,
        "successful": 6,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": {
            "value": 2,
            "relation": "eq"
        },
        "max_score": 1.0,
        "hits": [
            {
                "_index": "books-index:0::3xJq14MBUaOS0wL26UU9:0",
                "_id": "F_bt4oMBLle5pYmm5q4T",
                "_score": 1.0,
                "_source": {
                    "title": "The Shining",
                    "author": "Stephen King",
                    "year": 1977
                }
            }
        ]
    }
}
```


Identity and Access Management for Amazon OpenSearch Serverless

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use OpenSearch Serverless resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Identity-based policies for OpenSearch Serverless
- Policy actions for OpenSearch Serverless
- Policy resources for OpenSearch Serverless
- Policy condition keys for Amazon OpenSearch Serverless
- ABAC with OpenSearch Serverless
- Using temporary credentials with OpenSearch Serverless
- Service-linked roles for OpenSearch Serverless
- Identity-based policy examples for OpenSearch Serverless
- IAM Identity Center support for Amazon OpenSearch Serverless

Identity-based policies for OpenSearch Serverless

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for OpenSearch Serverless

To view examples of OpenSearch Serverless identity-based policies, see the section called "Identity-based policy examples".

Policy actions for OpenSearch Serverless

Supports policy actions: Yes

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation.

There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in OpenSearch Serverless use the following prefix before the action:

```
aoss
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "aoss:action1",
      "aoss:action2"
         ]
```

You can specify multiple actions using wildcard characters (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "aoss:List*"
```

To view examples of OpenSearch Serverless identity-based policies, see Identity-based policy examples for OpenSearch Serverless.

Policy resources for OpenSearch Serverless

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

Policy condition keys for Amazon OpenSearch Serverless

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

In addition to attribute-based access control (ABAC), OpenSearch Serverless supports the following condition keys:

- `aoss:collection`
- `aoss:CollectionId`
- `aoss:index`

You can use these condition keys even when providing permissions for access policies and security policies. For example:

```
[
   {
      "Effect":"Allow",
```

```
      "Action":[
         "aoss:CreateAccessPolicy",
         "aoss:CreateSecurityPolicy"
      ],
      "Resource":"*",
      "Condition":{
         "StringLike":{
            "aoss:collection":"log"
         }
      }
   }
]
```

In this example, the condition applies to policies that contain *rules* that match a collection name or pattern. The conditions have the following behavior:

- `StringEquals` - Applies to policies with rules that contain the *exact* resource string "log" (i.e. `collection/log`).
- `StringLike` - Applies to policies with rules that contain a resource string that *includes* the string "log" (i.e. `collection/log` but also `collection/logs-application` or `collection/applogs123`).

> **Note**
>
> *Collection* condition keys don't apply at the index level. For example, in the policy above, the condition wouldn't apply to an access or security policy containing the resource string `index/logs-application/*`.

To see a list of OpenSearch Serverless condition keys, see Condition keys for Amazon OpenSearch Serverless in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Amazon OpenSearch Serverless.

ABAC with OpenSearch Serverless

Supports ABAC (tags in policies): Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or

roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

For more information about tagging OpenSearch Serverless resources, see the section called "Tagging collections".

Using temporary credentials with OpenSearch Serverless

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Service-linked roles for OpenSearch Serverless

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating and managing OpenSearch Serverless service-linked roles, see the section called “Collection creation role”.

Identity-based policy examples for OpenSearch Serverless

By default, users and roles don't have permission to create or modify OpenSearch Serverless resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by Amazon OpenSearch Serverless, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for Amazon OpenSearch Serverless in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using OpenSearch Serverless in the console
- Administering OpenSearch Serverless collections
- Viewing OpenSearch Serverless collections
- Using OpenSearch API operations

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete OpenSearch Serverless resources in your account. These actions can incur costs for

your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

Identity-based policies determine whether someone can create, access, or delete OpenSearch Serverless resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using OpenSearch Serverless in the console

To access OpenSearch Serverless within the OpenSearch Service console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the OpenSearch Serverless resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (such as IAM roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

The following policy allows a user to access OpenSearch Serverless within the OpenSearch Service console:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Resource": "*",
            "Effect": "Allow",
            "Action": [
                "aoss:ListCollections",
                "aoss:BatchGetCollection",
                "aoss:ListAccessPolicies",
                "aoss:ListSecurityConfigs",
                "aoss:ListSecurityPolicies",
                "aoss:ListTagsForResource",
                "aoss:ListVpcEndpoints",
                "aoss:GetAccessPolicy",
                "aoss:GetAccountSettings",
                "aoss:GetSecurityConfig",
                "aoss:GetSecurityPolicy"
            ]
        }
    ]
}
```

Administering OpenSearch Serverless collections

This policy is an example of a "collection admin" policy that allows a user to manage and administer Amazon OpenSearch Serverless collections. The user can create, view, and delete collections.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Resource": "arn:aws:aoss:region:123456789012:collection/*",
            "Action": [
                "aoss:CreateCollection",
                "aoss:DeleteCollection",
                "aoss:UpdateCollection"
            ],
            "Effect": "Allow"
        },
        {
            "Resource": "*",
            "Action": [
                "aoss:BatchGetCollection",
                "aoss:ListCollections",
                "aoss:CreateAccessPolicy",
                "aoss:CreateSecurityPolicy"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Viewing OpenSearch Serverless collections

This example policy allows a user to view details for all Amazon OpenSearch Serverless collections in their account. The user can't modify the collections or any associated security policies.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Resource": "*",
            "Action": [
                "aoss:ListAccessPolicies",
```

```
                "aoss:ListCollections",
                "aoss:ListSecurityPolicies",
                "aoss:ListTagsForResource",
                "aoss:BatchGetCollection"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Using OpenSearch API operations

Data plane API operations consist of the functions you use in OpenSearch Serverless to derive realtime value from the service. Control plane API operations consist of the functions you use to set up the environment.

To access Amazon OpenSearch Serverless data plane APIs and OpenSearch Dashboards from the browser, you need to add two IAM permissions for collection resources. These permissions are `aoss:APIAccessAll` and `aoss:DashboardsAccessAll`.

> **Note**
>
> Starting May 10, 2023, OpenSearch Serverless requires these two new IAM permissions for collection resources. The `aoss:APIAccessAll` permission allows data plane access, and the `aoss:DashboardsAccessAll` permission allows OpenSearch Dashboards from the browser. Failure to add the two new IAM permissions results in a 403 error.

This example policy allows a user to access data plane APIs for a specified collection in their account, and to access OpenSearch Dashboards for all collections in their account.

```
{
    "Version": "2012-10-17",
    "Statement": [
         {
            "Effect": "Allow",
            "Action": "aoss:APIAccessAll",
            "Resource": "arn:aws:aoss:region:account-id:collection/collection-id"
        },
        {
            "Effect": "Allow",
```

```
            "Action": "aoss:DashboardsAccessAll",
            "Resource": "arn:aws:aoss:region:account-id:dashboards/default"
        }
    ]
}
```

Both `aoss:APIAccessAll` and `aoss:DashboardsAccessAll` give full IAM permission to the collection resources, while the Dashboards permission also provides OpenSearch Dashboards access. Each permission works independently, so an explicit deny on `aoss:APIAccessAll` doesn't block `aoss:DashboardsAccessAll` access to the resources, including Dev Tools. The same is true for a deny on `aoss:DashboardsAccessAll`. OpenSearch Serverless supports the following global condition keys:

- `aws:CalledVia`
- `aws:CalledViaAWSService`
- `aws:CalledViaFirst`
- `aws:CalledViaLast`
- `aws:CurrentTime`
- `aws:EpochTime`
- `aws:PrincipalAccount`
- `aws:PrincipalArn`
- `aws:PrincipallsAWSService`
- `aws:PrincipalOrgID`
- `aws:PrincipalOrgPaths`
- `aws:PrincipalType`
- `aws:PrincipalServiceName`
- `aws:PrincipalServiceNamesList`
- `aws:ResourceAccount`
- `aws:ResourceOrgID`
- `aws:ResourceOrgPaths`
- `aws:RequestedRegion`
- `aws:SourceIp`
- `aws:userid`

- `aws:username`

The following is an example of using `aws:SourceIp` in the condition block in your principal's IAM policy for data plane calls:

```
"Condition": {
    "IpAddress": {
         "aws:SourceIp": "52.95.4.14"
    }
}
```

Additonally, support is offered for the following OpenSearch Serverless specific keys:

- `aoss:CollectionId`
- `aoss:collection`

The following is an example of using `aoss:collection` in the condition block in your principal's IAM policy for data plane calls:

```
"Condition": {
    "StringLike": {
         "aoss:collection": "log-*"
    }
}
```

IAM Identity Center support for Amazon OpenSearch Serverless

IAM Identity Center support for Amazon OpenSearch Serverless

You can use IAM Identity Center principals (users and groups) to access Amazon OpenSearch Serverless data through Amazon OpenSearch Applications. In order to enable IAM Identity Center support for Amazon OpenSearch Serverless, you will need to enable use of IAM Identity Center. To learn more on how to do this, see What is IAM Identity Center?

After the IAM Identity Center instance is created, the customer account administrator needs to create an IAM Identity Center application for the Amazon OpenSearch Serverless service. This can be done by calling the CreateSecurityConfig:. The customer account administrator can specify what attributes will be used for authorizing the request. The default attributes used are `UserId` and `GroupId`.

The IAM Identity Center integration for Amazon OpenSearch Serverless uses the following AWS IAM Identity Center (IAM) permsions:

- `aoss:CreateSecurityConfig` – Create an IAM Identity Center provider
- `aoss:ListSecurityConfig` – List all IAM Identity Center providers in the current account.
- `aoss:GetSecurityConfig` – View IAM Identity Center provider information.
- `aoss:UpdateSecurityConfig` – Modify a given IAM Identity Center configuration
- `aoss:DeleteSecurityConfig` – Delete an IAM Identity Centerprovider.

The following idenity-based access policy can be used to manage all IAM Identity Center configurations:

```
{
"Version": "2012-10-17",
    "Statement": [
        {
"Action": [
                "aoss:CreateSecurityConfig",
                "aoss:DeleteSecurityConfig",
                "aoss:GetSecurityConfig",
                "aoss:UpdateSecurityConfig",
                "aoss:ListSecurityConfigs"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Note

The `Resource` element must be a wildcard.

Creating an IAM Identity Center provider (console)

You can create an IAM Identity Center provider to enable authentication with OpenSearch Application. To enable IAM Identity Center authentication for OpenSearch Dashboards, perform the following steps:

1. Sign in to the Amazon OpenSearch Service console.
2. On the left navigation panel, expand **Serverless** and choose **Authentication**.
3. Choose **IAM Identity Center authentication**.
4. Select **Edit**
5. Check the box next to Authenticate with IAM Identity Center.
6. Select the **user and group** attribute key from the dropdown menu. User attributes will be used to authorize users based on `UserName`, `UserId`, and `Email`. Group attributes will be used to authenticate users based on `GroupName` and `GroupId`.
7. Select the **IAM Identity Center** instance.
8. Select **Save**

Creating IAM Identity Center provider (AWS CLI)

To create an IAM Identity Center provider using the AWS Command Line Interface (AWS CLI) use the following command:

```
aws opensearchserverless create-security-config \
--region us-east-2 \
--name "iamidentitycenter-config" \
--description "description" \
--type "iamidentitycenter" \
--iam-identity-center-options '{
    "instanceArn": "arn:aws:sso:::instance/ssoins-99199c99e99ee999",
    "userAttribute": "UserName",
    "groupAttribute": "GroupId"
}'
```

After an IAM Identity Center is enabled, customers can only modify **user and group** attributes.

```
aws opensearchserverless update-security-config \
--region us-east-1 \
--id <id_from_list_security_configs> \
--config-version <config_version_from_get_security_config> \
--iam-identity-center-options-updates '{
    "userAttribute": "UserId",
    "groupAttribute": "GroupId"
}'
```

In order to view the IAM Identity Center provider using the AWS Command Line Interface, use the following command:

```
aws opensearchserverless list-security-configs --type iamidentitycenter
```

Deleting an IAM Identity Center provider

IAM Identity Center offers two instances of providers, one for your organization account and one for your member account. If you need to change your IAM Identity Center instance, you need to delete your security configuration through the `DeleteSecurityConfig` API and create a new security configuration using the new IAM Identity Center instance. The following command can be used to delete an IAM Identity Center provider:

```
aws opensearchserverless delete-security-config \
--region us-east-1 \
--id <id_from_list_security_configs>
```

Granting IAM Identity Center access to collection data

After your IAM Identity Center provider is enabled, you can update the collection data access policy to include IAM Identity Center principals. IAM Identity Center principals need to be updated in the following format:

```
[
   {
"Rules":[
       ...
      ],
      "Principal":[
         "iamidentitycenter/<iamidentitycenter-instance-id>/user/<UserName>",
         "iamidentitycenter/<iamidentitycenter-instance-id>/group/<GroupId>"
      ]
   }
]
```

Note

Amazon OpenSearch Serverless supports only one IAM Identity Center instance for all customer collections and can support up to 100 groups for a single user. If you try to use

more than the number of allowed instances, you will experience inconsistency with your data access policy authorization processing and receive a `403`error message.

You can grant access to collections, indexes, or both. If you want different users to have different permssions, you will need to create multiple rules. For a list of available permissions, see Identity and Access Management in Amazon OpenSearch Service. For information about how to format an access policy, see Granting SAML identities access to collection data .

IAM Identity Center offers two instances of providers, one for your organization account and one for your member accout. If you need to change your IAM Identity Center instance, you need to delete your security configuration through the `DeleteSecurityConfig` API and create a new security configuration using the new IAM Identity Center instance. The following command can be used to delete an IAM Identity Center provider:

```
aws opensearchserverless delete-security-config \
--region us-east-1 \
--id <id_from_list_security_configs>
```

Encryption in Amazon OpenSearch Serverless

Encryption at rest

Each Amazon OpenSearch Serverless collection that you create is protected with encryption of data at rest, a security feature that helps prevent unauthorized access to your data. Encryption at rest uses AWS Key Management Service (AWS KMS) to store and manage your encryption keys. It uses the Advanced Encryption Standard algorithm with 256-bit keys (AES-256) to perform the encryption.

Topics

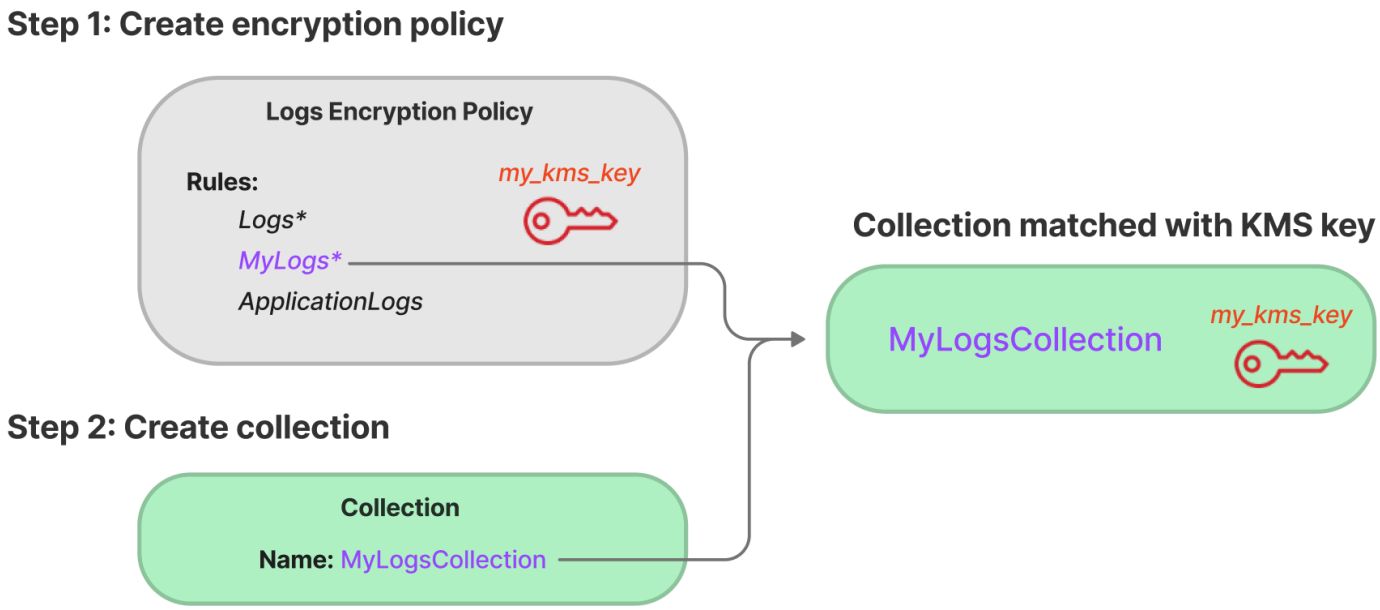
- Encryption policies
- Considerations
- Permissions required
- Key policy for a customer managed key
- How OpenSearch Serverless uses grants in AWS KMS
- Creating encryption policies (console)

- Creating encryption policies (AWS CLI)
- Viewing encryption policies
- Updating encryption policies
- Deleting encryption policies

Encryption policies

With encryption policies, you can manage many collections at scale by automatically assigning an encryption key to newly created collections that match a specific name or pattern.

When you create an encryption policy, you can either specify a *prefix*, which is a wildcard-based matching rule such as `MyCollection*`, or enter a single collection name. Then, when you create a collection that matches that name or prefix pattern, the policy and corresponding KMS key are automatically assigned to it.



Encryption policies contain the following elements:

- `Rules` – one or more collection matching rules, each with the following sub-elements:
  - `ResourceType` – Currently the only option is "collection". Encryption policies apply to collection resources only.
  - `Resource` – One or more collection names or patterns that the policy will apply to, in the format `collection/<collection name|pattern>`.
- `AWSOwnedKey` – Whether to use an AWS owned key.

- KmsARN – If you set AWSOwnedKey to false, specify the Amazon Resource Name (ARN) of the KMS key to encrypt the associated collections with. If you include this parameter, OpenSearch Serverless ignores the AWSOwnedKey parameter.

The following sample policy will assign a customer managed key to any future collection named autopartsinventory, as well as collections that begin with the term "sales":

```
{
   "Rules":[
      {
         "ResourceType":"collection",
         "Resource":[
            "collection/autopartsinventory",
            "collection/sales*"
         ]
      }
   ],
   "AWSOwnedKey":false,
   "KmsARN":"arn:aws:encryption:us-east-1:123456789012:key/93fd6da4-a317-4c17-
bfe9-382b5d988b36"
}
```

Even if a policy matches a collection name, you can choose to override this automatic assignment during collection creation if the resource pattern contains a wildcard (*). If you choose to override automatic key assignment, OpenSearch Serverless creates an encryption policy for you named **auto-<*collection-name*>** and attaches it to the collection. The policy initially only applies to a single collection, but you can modify it to include additional collections.

If you modify policy rules to no longer match a collection, the associated KMS key won't be unassigned from that collection. The collection always remains encrypted with its initial encryption key. If you want to change the encryption key for a collection, you must recreate the collection.

If rules from multiple policies match a collection, the more specific rule is used. For example, if one policy contains a rule for collection/log*, and another for collection/logSpecial, the encryption key for the second policy is used because it's more specific.

You can't use a name or a prefix in a policy if it already exists in another policy. OpenSearch Serverless displays an error if you try to configure identical resource patterns in different encryption policies.

Considerations

Consider the following when you configure encryption for your collections:

- Encryption at rest is *required* for all serverless collections.
- You have the option to use a customer managed key or an AWS owned key. If you choose a customer managed key, we recommend that you enable automatic key rotation.
- You can't change the encryption key for a collection after the collection is created. Carefully choose which AWS KMS to use the first time you set up a collection.
- A collection can only match a single encryption policy.
- Collections with unique KMS keys can't share OpenSearch Compute Units (OCUs) with other collections. Each collection with a unique key requires its own 4 OCUs.
- If you update the KMS key in an encryption policy, the change doesn't affect existing matching collections with KMS keys already assigned.
- OpenSearch Serverless doesn't explicitly check user permissions on customer managed keys. If a user has permissions to access a collection through a data access policy, they will be able to ingest and query the data that is encrypted with the associated key.

Permissions required

Encryption at rest for OpenSearch Serverless uses the following AWS Identity and Access Management (IAM) permissions. You can specify IAM conditions to restrict users to specific collections.

- `aoss:CreateSecurityPolicy` – Create an encryption policy.
- `aoss:ListSecurityPolicies` – List all encryption policies and collections that they are attached to.
- `aoss:GetSecurityPolicy` – See details of a specific encryption policy.
- `aoss:UpdateSecurityPolicy` – Modify an encryption policy.
- `aoss:DeleteSecurityPolicy` – Delete an encryption policy.

The following sample identity-based access policy provides the minimum permissions necessary for a user to manage encryption policies with the resource pattern `collection/application-logs`.

```
{
```

```
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "aoss:CreateSecurityPolicy",
            "aoss:UpdateSecurityPolicy",
            "aoss:DeleteSecurityPolicy",
            "aoss:GetSecurityPolicy"
         ],
         "Resource":"*",
         "Condition":{
            "StringEquals":{
               "aoss:collection":"application-logs"
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "aoss:ListSecurityPolicies"
         ],
         "Resource":"*"
      }
   ]
}
```

Key policy for a customer managed key

If you select a customer managed key to protect a collection, OpenSearch Serverless gets permission to use the KMS key on behalf of the principal who makes the selection. That principal, a user or role, must have the permissions on the KMS key that OpenSearch Serverless requires. You can provide these permissions in a key policy or an IAM policy.

At a minimum, OpenSearch Serverless requires the following permissions on a customer managed key:

- kms:DescribeKey
- kms:CreateGrant

For example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:DescribeKey",
        "kms:CreateGrant"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "aoss.us-east-1.amazonaws.com"
        },
        "Bool": {
          "kms:GrantIsForAWSResource": "true"
        }
      }
    }
  ]
}
```

OpenSearch Serverless create a grant with the kms:GenerateDataKey and kms:Decrypt permissions.

For more information, see Using key policies in AWS KMS in the *AWS Key Management Service Developer Guide*.

How OpenSearch Serverless uses grants in AWS KMS

OpenSearch Serverless requires a grant in order to use a customer managed key.

When you create an encryption policy in your account with a new key, OpenSearch Serverless creates a grant on your behalf by sending a CreateGrant request to AWS KMS. Grants in AWS KMS are used to give OpenSearch Serverless access to a KMS key in a customer account.

OpenSearch Serverless requires the grant to use your customer managed key for the following internal operations:

- Send DescribeKey requests to AWS KMS to verify that the symmetric customer managed key ID provided is valid.
- Send GenerateDataKey requests to KMS key to create data keys with which to encrypt objects.

- Send Decrypt requests to AWS KMS to decrypt the encrypted data keys so that they can be used to encrypt your data.

You can revoke access to the grant, or remove the service's access to the customer managed key at any time. If you do, OpenSearch Serverless won't be able to access any of the data encrypted by the customer managed key, which affects all the operations that are dependent on that data, leading to `AccessDeniedException` errors and failures in the asynchronous workflows.

OpenSearch Serverless retires grants in an asynchronous workflow when a given customer managed key isn't associated with any security policies or collections.

Creating encryption policies (console)

In an encryption policy, you specify an KMS key and a series of collection patterns that the policy will apply to. Any new collections that match one of the patterns defined in the policy will be assigned the corresponding KMS key when you create the collection. We recommend that you create encryption policies *before* you start creating collections.

To create an OpenSearch Serverless encryption policy

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. On the left navigation panel, expand **Serverless** and choose **Encryption policies**.
3. Choose **Create encryption policy**.
4. Provide a name and description for the policy.
5. Under **Resources**, enter one or more resource patterns for this encryption policy. Any newly created collections in the current AWS account and Region that match one of the patterns are automatically assigned to this policy. For example, if you enter `ApplicationLogs` (with no wildcard), and later create a collection with that name, the policy and corresponding KMS key are assigned to that collection.

   You can also provide a prefix such as `Logs*`, which assigns the policy to any new collections with names beginning with `Logs`. By using wildcards, you can manage encryption settings for multiple collections at scale.
6. Under **Encryption**, choose an KMS key to use.
7. Choose **Create**.

Next step: Create collections

After you configure one or more encryption policies, you can start creating collections that match the rules defined in those policies. For instructions, see the section called "Creating collections".

In the **Encryptions** step of collection creation, OpenSearch Serverless informs you that the name that you entered matches the pattern defined in an encryption policy, and automatically assigns the corresponding KMS key to the collection. If the resource pattern contains a wildcard (*), you can choose to override the match and select your own key.

Creating encryption policies (AWS CLI)

To create an encryption policy using the OpenSearch Serverless API operations, you specify resource patterns and an encryption key in JSON format. The CreateSecurityPolicy request accepts both inline policies and .json files.

Encryption policies take the following format. This sample `my-policy.json` file matches any future collection named `autopartsinventory`, as well as any collections with names beginning with `sales`.

```
{
   "Rules":[
      {
         "ResourceType":"collection",
         "Resource":[
            "collection/autopartsinventory",
            "collection/sales*"
         ]
      }
   ],
   "AWSOwnedKey":false,
   "KmsARN":"arn:aws:encryption:us-east-1:123456789012:key/93fd6da4-a317-4c17-
bfe9-382b5d988b36"
}
```

To use a service-owned key, set `AWSOwnedKey` to `true`:

```
{
   "Rules":[
      {
         "ResourceType":"collection",
         "Resource":[
```

```
            "collection/autopartsinventory",
            "collection/sales*"
         ]
      }
   ],
   "AWSOwnedKey":true
}
```

The following request creates the encryption policy:

```
aws opensearchserverless create-security-policy \
    --name sales-inventory \
    --type encryption \
    --policy file://my-policy.json
```

Then, use the CreateCollection API operation to create one or more collections that match one of the resource patterns.

Viewing encryption policies

Before you create a collection, you might want to preview the existing encryption policies in your account to see which one has a resource pattern that matches your collection's name. The following ListSecurityPolicies request lists all encryption policies in your account:

```
aws opensearchserverless list-security-policies --type encryption
```

The request returns information about all configured encryption policies. Use the contents of the `policy` element to view the pattern rules that are defined in the policy:

```
{
   "securityPolicyDetails": [
      {
         "createdDate": 1663693217826,
         "description": "Sample encryption policy",
         "lastModifiedDate": 1663693217826,
         "name": "my-policy",
         "policy": "{\"Rules\":[{\"ResourceType\":\"collection\",\"Resource\":
[\"collection/autopartsinventory\",\"collection/sales*\"]}],\"AWSOwnedKey\":true}",
         "policyVersion": "MTY2MzY5MzIxNzgyNl8x",
         "type": "encryption"
      }
   ]
```

```
}
```

To view detailed information about a specific policy, including the KMS key, use the GetSecurityPolicy command.

Updating encryption policies

If you update the KMS key in an encryption policy, the change only applies to the newly created collections that match the configured name or pattern. It doesn't affect existing collections that have KMS keys already assigned.

The same applies to policy matching rules. If you add, modify, or delete a rule, the change only applies to newly created collections. Existing collections don't lose their assigned KMS key if you modify a policy's rules so that it no longer matches a collection's name.

To update an encryption policy in the OpenSearch Serverless console, choose **Encryption policies**, select the policy to modify, and choose **Edit**. Make your changes and choose **Save**.

To update an encryption policy using the OpenSearch Serverless API, use the UpdateSecurityPolicy operation. The following request updates an encryption policy with a new policy JSON document:

```
aws opensearchserverless update-security-policy \
    --name sales-inventory \
    --type encryption \
    --policy-version 2 \
    --policy file://my-new-policy.json
```

Deleting encryption policies

When you delete an encryption policy, any collections that are currently using the KMS key defined in the policy are not affected. To delete a policy in the OpenSearch Serverless console, select the policy and choose **Delete**.

You can also use the DeleteSecurityPolicy operation:

```
aws opensearchserverless delete-security-policy --name my-policy --type encryption
```

Encryption in transit

Within OpenSearch Serverless, all paths in a collection are encrypted in transit using Transport Layer Security 1.2 (TLS) with an industry-standard AES-256 cipher. Access to all APIs and

Dashboards for Opensearch is also through TLS 1.2 . TLS is a set of industry-standard cryptographic protocols used for encrypting information that is exchanged over the network.

Network access for Amazon OpenSearch Serverless

The network settings for an Amazon OpenSearch Serverless collection determine whether the collection is accessible over the internet from public networks, or whether it must be accessed privately.

Private access can apply to one or both of the following:

- OpenSearch Serverless-managed VPC endpoints
- Supported AWS services such as Amazon Bedrock

You can configure network access separately for a collection's *OpenSearch* endpoint and its corresponding *OpenSearch Dashboards* endpoint.

Network access is the isolation mechanism for allowing access from different source networks. For example, if a collection's OpenSearch Dashboards endpoint is publicly accessible but the OpenSearch API endpoint isn't, a user can access the collection data only through Dashboards when connecting from a public network. If they try to call the OpenSearch APIs directly from a public network, they'll be blocked. Network settings can be used for such permutations of source to resource type. Amazon OpenSearch Serverless supports both IPv4 and IPv6 connectivity.

Topics

Network policies

Network policies let you manage many collections at scale by automatically assigning network access settings to collections that match the rules defined in the policy.

In a network policy, you specify a series of *rules*. These rule define access permissions to collection endpoints and OpenSearch Dashboards endpoints. Each rule consists of an access type (public or private) and a resource type (collection and/or OpenSearch Dashboards endpoint). For each resource type (`collection` and `dashboard`), you specify a series of rules that define which collection(s) the policy will apply to.

In this sample policy, the first rule specifies VPC endpoint access to both the collection endpoint and the Dashboards endpoint for all collections beginning with the term `marketing*`. It also specifies Amazon Bedrock access.

> **Note**
>
> Private access to AWS services such as Amazon Bedrock *only* applies to the collection's OpenSearch endpoint, not to the OpenSearch Dashboards endpoint. Even if the `ResourceType` is `dashboard`, AWS services cannot be granted access to OpenSearch Dashboards.

The second rule specifies public access to the `finance` collection, but only for the collection endpoint (no Dashboards access).

```
[
   {
      "Description":"Marketing access",
      "Rules":[
         {
            "ResourceType":"collection",
            "Resource":[
               "collection/marketing*"
            ]
         },
         {
            "ResourceType":"dashboard",
            "Resource":[
               "collection/marketing*"
```

```
            ]
         }
      ],
      "AllowFromPublic":false,
      "SourceVPCEs":[
         "vpce-050f79086ee71ac05"
      ],
      "SourceServices":[
         "bedrock.amazonaws.com"
      ],
   },
   {
      "Description":"Sales access",
      "Rules":[
         {
            "ResourceType":"collection",
            "Resource":[
               "collection/finance"
            ]
         }
      ],
      "AllowFromPublic":true
   }
]
```

This policy provides public access only to OpenSearch Dashboards for collections beginning with "finance". Any attempts to directly access the OpenSearch API will fail.

```
[
  {
    "Description": "Dashboards access",
    "Rules": [
      {
        "ResourceType": "dashboard",
        "Resource": [
          "collection/finance*"
        ]
      }
    ],
    "AllowFromPublic": true
  }
]
```

Network policies can apply to existing collections as well as future collections. For example, you can create a collection and then create a network policy with a rule that matches the collection name. You don't need to create network policies before you create collections.

Considerations

Consider the following when you configure network access for your collections:

- If you plan to configure VPC endpoint access for a collection, you must first create at least one OpenSearch Serverless-managed VPC endpoint.
- Private access to AWS services only applies to the collection's OpenSearch endpoint, not to the OpenSearch Dashboards endpoint. Even if the `ResourceType` is `dashboard`, AWS services cannot be granted access to OpenSearch Dashboards.
- If a collection is accessible from public networks, it's also accessible from all OpenSearch Serverless-managed VPC endpoints and all AWS services.
- Multiple network policies can apply to a single collection. For more information, see the section called “Policy precedence”.

Permissions required to configure network policies

Network access for OpenSearch Serverless uses the following AWS Identity and Access Management (IAM) permissions. You can specify IAM conditions to restrict users to network policies associated with specific collections.

- `aoss:CreateSecurityPolicy` – Create a network access policy.
- `aoss:ListSecurityPolicies` – List all network policies in the current account.
- `aoss:GetSecurityPolicy` – View a network access policy specification.
- `aoss:UpdateSecurityPolicy` – Modify a given network access policy, and change the VPC ID or public access designation.
- `aoss:DeleteSecurityPolicy` – Delete a network access policy (after it's detached from all collections).

The following identity-based access policy allows a user to view all network policies, and update policies with the resource pattern `collection/application-logs`:

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "aoss:UpdateSecurityPolicy"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aoss:collection": "application-logs"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "aoss:ListSecurityPolicies",
                "aoss:GetSecurityPolicy"
            ],
            "Resource": "*"
        }
    ]
}
```

Note

In addition, OpenSearch Serverless requires the `aoss:APIAccessAll` and `aoss:DashboardsAccessAll` permissions for collection resources. For more information, see the section called "Using OpenSearch API operations".

Policy precedence

There can be situations where network policy rules overlap, within or across policies. When this happens, a rule that specifies public access overrides a rule that specifies private access for any collections that are common to *both* rules.

For example, in the following policy, both rules assign network access to the `finance` collection, but one rule specifies VPC access while the other specifies public access. In this situation, public access overrides VPC access *only for the finance collection* (because it exists in both rules), so the

finance collection will be accessible from public networks. The sales collection will have VPC access from the specified endpoint.

```
[
   {
      "Description":"Rule 1",
      "Rules":[
         {
            "ResourceType":"collection",
            "Resource":[
               "collection/sales",
               "collection/finance"
            ]
         }
      ],
      "AllowFromPublic":false,
      "SourceVPCEs":[
         "vpce-050f79086ee71ac05"
      ]
   },
   {
      "Description":"Rule 2",
      "Rules":[
         {
            "ResourceType":"collection",
            "Resource":[
               "collection/finance"
            ]
         }
      ],
      "AllowFromPublic":true
   }
]
```

If multiple VPC endpoints from different rules apply to a collection, the rules are additive and the collection will be accessible from all specified endpoints. If you set `AllowFromPublic` to `true` but also provide one or more `SourceVPCEs` or `SourceServices`, OpenSearch Serverless ignores the VPC endpoints and service identifiers, and the associated collections will have public access.

Creating network policies (console)

Network policies can apply to existing policies as well as future policies. We recommend that you create network policies before you start creating collections.

To create an OpenSearch Serverless network policy

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. On the left navigation panel, expand **Serverless** and choose **Network policies**.
3. Choose **Create network policy**.
4. Provide a name and description for the policy.
5. Provide one or more *rules*. These rules define access permissions for your OpenSearch Serverless collections and their OpenSearch Dashboards endpoints.

   Each rule contains the following elements:

Element	Description
Rule name	A name that describes the contents of the rule. For example, "VPC access for marketing team".
Access type	Choose either public or private access. Then, select one or both of the following: • **VPC endpoints for access** – Specify one or more OpenSearch Serverless-managed VPC endpoints–managed VPC endpoints. • **AWS service private access** – Select one or more supported AWS services.
Resource type	Select whether to provide access to OpenSearch endpoints (which allows making calls to the OpenSearch API), to OpenSearch Dashboards (which allows access to visualizations and the user interface for OpenSearch plugins), or both.

Element	Description
	Note AWS service private access only applies to the collection's OpenSearch endpoint, not to the OpenSearch Dashboards endpoint. Even if you select **OpenSearch Dashboards**, AWS services can only be granted endpoint access.

For each resource type that you select, you can choose existing collections to apply the policy settings to, and/or create one or more resource patterns. Resource patterns consist of a prefix and a wildcard (*), and define which collections the policy settings will apply to.

For example, if you include a pattern called `Marketing*`, any new or existing collections whose names start with "Marketing" will have the network settings in this policy automatically applied to them. A single wildcard (*) applies the policy to all current and future collections.

In addition, you can specify the name of a *future* collection without a wildcard, such as `Finance`. OpenSearch Serverless will apply the policy settings to any newly created collection with that exact name.

6. When you're satisfied with your policy configuration, choose **Create**.

Creating network policies (AWS CLI)

To create a network policy using the OpenSearch Serverless API operations, you specify rules in JSON format. The CreateSecurityPolicy request accepts both inline policies and .json files. All collections and patterns must take the form `collection/<collection name|pattern>`.

Note

The resource type `dashboards` only allows permission to OpenSearch Dashboards, but in order for OpenSearch Dashboards to function, you must also allow collection access from the same sources. See the second policy below for an example.

To specify private access, include one or both of the following elements:

- `SourceVPCEs` – Specify one or more OpenSearch Serverless–managed VPC endpoints.
- `SourceServices` – Specify the identifier of one or more supported AWS services. Currently, the following service identifiers are supported:
  - `bedrock.amazonaws.com` – Amazon Bedrock

The following sample network policy provides private access, to a VPC endpoint and Amazon Bedrock, to collection endpoints only for collections beginning with the prefix `log*`. Authenticated users can't sign in to OpenSearch Dashboards; they can only access the collection endpoint programmatically.

```
[
   {
      "Description":"Private access for log collections",
      "Rules":[
         {
            "ResourceType":"collection",
            "Resource":[
               "collection/log*"
            ]
         }
      ],
      "AllowFromPublic":false,
      "SourceVPCEs":[
         "vpce-050f79086ee71ac05"
      ],
      "SourceServices":[
         "bedrock.amazonaws.com"
      ],
   }
]
```

The following policy provides public access to the OpenSearch endpoint *and* OpenSearch Dashboards for a single collection named `finance`. If the collection doesn't exist, the network settings will be applied to the collection if and when it's created.

```
[
   {
      "Description":"Public access for finance collection",
      "Rules":[
         {
            "ResourceType":"dashboard",
            "Resource":[
               "collection/finance"
            ]
         },
         {
            "ResourceType":"collection",
            "Resource":[
               "collection/finance"
            ]
         }
      ],
      "AllowFromPublic":true
   }
]
```

The following request creates the above network policy:

```
aws opensearchserverless create-security-policy \
    --name sales-inventory \
    --type network \
    --policy "[{\"Description\":\"Public access for finance collection\",\"Rules
\":[{\"ResourceType\":\"dashboard\",\"Resource\":[\"collection\/finance\"]},
{\"ResourceType\":\"collection\",\"Resource\":[\"collection\/finance\"]}],
\"AllowFromPublic\":true}]"
```

To provide the policy in a JSON file, use the format `--policy file://my-policy.json`

Viewing network policies

Before you create a collection, you might want to preview the existing network policies in your account to see which one has a resource pattern that matches your collection's name. The following ListSecurityPolicies request lists all network policies in your account:

```
aws opensearchserverless list-security-policies --type network
```

The request returns information about all configured network policies. To view the pattern rules defined in the one specific policy, find the policy information in the contents of the `securityPolicySummaries` element in the response. Note the `name` and `type` of this policy and use these properties in a GetSecurityPolicy request to receive a response with the following policy details:

```
{
    "securityPolicyDetail": [
        {
            "type": "network",
            "name": "my-policy",
            "policyVersion": "MTY2MzY5MTY1MDA3Ml8x",
            "policy": "[{\"Description\":\"My network policy rule\",\"Rules\":
[{\"ResourceType\":\"dashboard\",\"Resource\":[\"collection/*\"]}],\"AllowFromPublic
\":true}]",
            "createdDate": 1663691650072,
            "lastModifiedDate": 1663691650072
        }
    ]
}
```

To view detailed information about a specific policy, use the GetSecurityPolicy command.

Updating network policies

When you modify the VPC endpoints or public access designation for a network, all associated collections are impacted. To update a network policy in the OpenSearch Serverless console, expand **Network policies**, select the policy to modify, and choose **Edit**. Make your changes and choose **Save**.

To update a network policy using the OpenSearch Serverless API, use the UpdateSecurityPolicy command. You must include a policy version in the request. You can retrieve the policy version by using the `ListSecurityPolicies` or `GetSecurityPolicy` commands. Including the most recent policy version ensures that you don't inadvertently override a change made by someone else.

The following request updates a network policy with a new policy JSON document:

```
aws opensearchserverless update-security-policy \
    --name sales-inventory \
    --type network \
    --policy-version MTY2MzY5MTY1MDA3Ml8x \
    --policy file://my-new-policy.json
```

Deleting network policies

Before you can delete a network policy, you must detach it from all collections. To delete a policy in the OpenSearch Serverless console, select the policy and choose **Delete**.

You can also use the DeleteSecurityPolicy command:

```
aws opensearchserverless delete-security-policy --name my-policy --type network
```

Data access control for Amazon OpenSearch Serverless

With data access control in Amazon OpenSearch Serverless, you can allow users to access collections and indexes, regardless of their access mechanism or network source. You can provide access to IAM roles and SAML identities.

You manage access permissions through *data access policies*, which apply to collections and index resources. Data access policies help you manage collections at scale by automatically assigning access permissions to collections and indexes that match a specific pattern. Multiple data access policies can apply to a single resource. Note that you must have a data access policy for your collection in order to access your OpenSearch Dashboards URL.

Topics

- Data access policies versus IAM policies
- IAM permissions required to configure data access policies
- Policy syntax
- Supported policy permissions
- Sample datasets on OpenSearch Dashboards
- Creating data access policies (console)
- Creating data access policies (AWS CLI)
- Viewing data access policies

- Updating data access policies
- Deleting data access policies
- Cross-account data access

Data access policies versus IAM policies

Data access policies are logically separate from AWS Identity and Access Management (IAM) policies. IAM permissions control access to the serverless API operations, such as `CreateCollection` and `ListAccessPolicies`. Data access policies control access to the OpenSearch operations that OpenSearch Serverless supports, such as `PUT <index>` or `GET _cat/indices`.

The IAM permissions that control access to data access policy API operations, such as `aoss:CreateAccessPolicy` and `aoss:GetAccessPolicy` (described in the next section), don't affect the permission specified in a data access policy.

For example, suppose an IAM policy denies a user from creating data access policies for `collection-a`, but allows them to create data access policies for all collections (*):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "aoss:CreateAccessPolicy"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aoss:collection": "collection-a"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "aoss:CreateAccessPolicy"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

If the user creates a data access policy that allows certain permission to *all* collections (`collection/*` or `index/*/*`) the policy will apply to all collections, including collection A.

> **Important**
>
> Being granted permissions within a data access policy is not sufficient to access data in your OpenSearch Serverless collection. An associated principal must *also* be granted access to the IAM permissions `aoss:APIAccessAll` and `aoss:DashboardsAccessAll`. Both permissions grant full access to collection resources, while the Dashboards permission also provides access to OpenSearch Dashboards. If a principal doesn't have both of these IAM permissions, they will receive 403 errors when attempting to send requests to the collection. For more information, see the section called “Using OpenSearch API operations”.

IAM permissions required to configure data access policies

Data access control for OpenSearch Serverless uses the following IAM permissions. You can specify IAM conditions to restrict users to specific access policy names.

- `aoss:CreateAccessPolicy` – Create an access policy.
- `aoss:ListAccessPolicies` – List all access policies.
- `aoss:GetAccessPolicy` – See details about a specific access policy.
- `aoss:UpdateAccessPolicy` – Modify an access policy.
- `aoss:DeleteAccessPolicy` – Delete an access policy.

The following identity-based access policy allows a user to view all access policies, and update policies that contain the resource pattern `collection/logs`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
```

```
                "aoss:ListAccessPolicies",
                "aoss:GetAccessPolicy"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "aoss:UpdateAccessPolicy"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aoss:collection": [
                        "logs"
                    ]
                }
            }
        }
    ]
}
```

> **Note**
>
> In addition, OpenSearch Serverless requires the `aoss:APIAccessAll` and `aoss:DashboardsAccessAll` permissions for collection resources. For more information, see the section called “Using OpenSearch API operations”.

Policy syntax

A data access policy includes a set of rules, each with the following elements:

Element	Description
`ResourceType`	The type of resource (collection or index) that the permissions apply to. Alias and template permissions are at the collection level, while permissions for creating, modifying, and searching data are at the index level. For more information, see Supported policy permissions.

Element	Description
`Resource`	A list of resource names and/or patterns. Patterns are prefixes followed by a wildcard (*), which allow the associated permissions to apply to multiple resources. • Collections take the format `collection/` *`<name|pattern>`* . • Indexes take the format `index/`*`<collection-name|pattern>`* */*`<index-name|pattern/>` .
`Permission`	A list of permissions to grant for the specified resources. For a complete list of permissions and the API operations they allow, see the section called "Supported OpenSearch API operations and permissions".
`Principal`	A list of one or more principals to grant access to. Principals can be IAM role ARNs or SAML identities. These principals must be within the current AWS account. Data access policies don't directly support cross-account access, but you can include a role in your policy that a user from a different AWS account can assume in the collection-owning account. For more information, see the section called "Cross-account data access".

The following example policy grants alias and template permissions to the collection called `autopartsinventory`, as well as any collections that begin with the prefix `sales*`. It also grants read and write permissions to all indexes within the `autopartsinventory` collection, and any indexes in the `salesorders` collection that begin with the prefix `orders*`.

```
[
   {
      "Description": "Rule 1",
      "Rules":[
         {
            "ResourceType":"collection",
            "Resource":[
               "collection/autopartsinventory",
               "collection/sales*"
            ],
            "Permission":[
               "aoss:CreateCollectionItems",
               "aoss:UpdateCollectionItems",
```

```
                "aoss:DescribeCollectionItems"
            ]
        },
        {
            "ResourceType":"index",
            "Resource":[
                "index/autopartsinventory/*",
                "index/salesorders/orders*"
            ],
            "Permission":[
                "aoss:*"
            ]
        }
    ],
    "Principal":[
        "arn:aws:iam::123456789012:user/Dale",
        "arn:aws:iam::123456789012:role/RegulatoryCompliance",
        "saml/123456789012/myprovider/user/Annie",
        "saml/123456789012/anotherprovider/group/Accounting"
    ]
  }
]
```

You can't explicitly deny access within a policy. Therefore, all policy permissions are additive. For example, if one policy grants a user `aoss:ReadDocument`, and another policy grants `aoss:WriteDocument`, the user will have *both* permissions. If a third policy grants the same user `aoss:*`, then the user can perform *all* actions on the associated index; more restrictive permissions don't override less restrictive ones.

Supported policy permissions

The following permissions are supported in data access policies. For the OpenSearch API operations that each permission allows, see the section called “Supported OpenSearch API operations and permissions”.

Collection permissions

- `aoss:CreateCollectionItems`
- `aoss:DeleteCollectionItems`
- `aoss:UpdateCollectionItems`
- `aoss:DescribeCollectionItems`

- aoss:*

Index permissions

- aoss:ReadDocument
- aoss:WriteDocument
- aoss:CreateIndex
- aoss:DeleteIndex
- aoss:UpdateIndex
- aoss:DescribeIndex
- aoss:*

Sample datasets on OpenSearch Dashboards

OpenSearch Dashboards provides sample datasets that come with visualizations, dashboards, and other tools to help you explore Dashboards before you add your own data. To create indexes from this sample data, you need a data access policy that provides permissions to the dataset that you want to work with. The following policy uses a wildcard (*) to provide permissions to all three sample datasets.

```
[
  {
    "Rules": [
      {
        "Resource": [
          "index/<collection-name>/opensearch_dashboards_sample_data_*"
        ],
        "Permission": [
          "aoss:CreateIndex",
          "aoss:DescribeIndex",
          "aoss:ReadDocument"
        ],
        "ResourceType": "index"
      }
    ],
    "Principal": [
      "arn:aws:iam::<account-id>:user/<user>"
    ]
```

```
  }
]
```

Creating data access policies (console)

You can create a data access policy using the visual editor, or in JSON format. Any new collections that match one of the patterns defined in the policy will be assigned the corresponding permissions when you create the collection.

To create an OpenSearch Serverless data access policy

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. In the left navigation pane, expand **Serverless** and choose **Data access control**.
3. Choose **Create access policy**.
4. Provide a name and description for the policy.
5. Provide a name for the first rule in your policy. For example, "Logs collection access".
6. Choose **Add principals** and select one or more IAM roles or SAML users and groups to provide data access to.

   > **Note**
   >
   > In order to select principals from the dropdown menus, you must have the `iam:ListUsers` and `iam:ListRoles` permissions (for IAM principals) and `aoss:ListSecurityConfigs` permission (for SAML identities).

7. Choose **Grant** and select the alias, template, and index permissions to grant the associated principals. For a full list of permissions and the access they allow, see the section called "Supported OpenSearch API operations and permissions".
8. (Optional) Configure additional rules for the policy.
9. Choose **Create**. There might be about a minute of lag time between when you create the policy and when the permissions are enforced. If it takes more than 5 minutes, contact Support.

> **Important**
>
> If your policy only includes index permissions (and no collection permissions), you might still see a message for matching collections stating `Collection cannot be accessed`

yet. Configure data access policies so that users can access the data within this collection. You can ignore this warning. Allowed principals can still perform their assigned index-related operations on the collection.

Creating data access policies (AWS CLI)

To create a data access policy using the OpenSearch Serverless API, use the `CreateAccessPolicy` command. The command accepts both inline policies and .json files. Inline policies must be encoded as a JSON escaped string.

The following request creates a data access policy:

```
aws opensearchserverless create-access-policy \
    --name marketing \
    --type data \
    --policy "[{\"Rules\":[{\"ResourceType\":\"collection\",\"Resource\":
[\"collection/autopartsinventory\",\"collection/sales*\"],\"Permission\":
[\"aoss:UpdateCollectionItems\"]},{\"ResourceType\":\"index\",\"Resource\":
[\"index/autopartsinventory/*\",\"index/salesorders/orders*\"],\"Permission
\":[\"aoss:ReadDocument\",\"aoss:DescribeIndex\"]}],\"Principal\":
[\"arn:aws:iam::123456789012:user/Shaheen\"]}]"
```

To provide the policy within a .json file, use the format `--policy file://my-policy.json`.

The principals included in the policy can now use the OpenSearch operations that they were granted access to.

Viewing data access policies

Before you create a collection, you might want to preview the existing data access policies in your account to see which one has a resource pattern that matches your collection's name. The following ListAccessPolicies request lists all data access policies in your account:

```
aws opensearchserverless list-access-policies --type data
```

The request returns information about all configured data access policies. To view the pattern rules defined in the one specific policy, find the policy information in the contents of the `accessPolicySummaries` element in the response. Note the `name` and `type` of this policy and

use these properties in a GetAccessPolicy request to receive a response with the following policy details:

```
{
    "accessPolicyDetails": [
        {
            "type": "data",
            "name": "my-policy",
            "policyVersion": "MTY2NDA1NDE4MDg1OF8x",
            "description": "My policy",
            "policy": "[{\"Rules\":[{\"ResourceType\":\"collection\",
\"Resource\":[\"collection/autopartsinventory\",\"collection/sales*\"],
\"Permission\":[\"aoss:UpdateCollectionItems\"]},{\"ResourceType\":\"index\",
\"Resource\":[\"index/autopartsinventory/*\",\"index/salesorders/orders*\"],
\"Permission\":[\"aoss:ReadDocument\",\"aoss:DescribeIndex\"]}],\"Principal\":
[\"arn:aws:iam::123456789012:user/Shaheen\"]}]",
            "createdDate": 1664054180858,
            "lastModifiedDate": 1664054180858
        }
    ]
}
```

You can include resource filters to limit the results to policies that contain specific collections or indexes:

```
aws opensearchserverless list-access-policies --type data --resource
 "index/autopartsinventory/*"
```

To view details about a specific policy, use the GetAccessPolicy command.

Updating data access policies

When you update a data access policy, all associated collections are impacted. To update a data access policy in the OpenSearch Serverless console, choose **Data access control**, select the policy to modify, and choose **Edit**. Make your changes and choose **Save**.

To update a data access policy using the OpenSearch Serverless API, send an `UpdateAccessPolicy` request. You must include a policy version, which you can retrieve using the `ListAccessPolicies` or `GetAccessPolicy` commands. Including the most recent policy version ensures that you don't inadvertently override a change made by someone else.

The following UpdateAccessPolicy request updates a data access policy with a new policy JSON document:

```
aws opensearchserverless update-access-policy \
    --name sales-inventory \
    --type data \
    --policy-version MTY2NDA1NDE4MDg1OF8x \
    --policy file://my-new-policy.json
```

There might be a few minutes of lag time between when you update the policy and when the new permissions are enforced.

Deleting data access policies

When you delete a data access policy, all associated collections lose the access that is defined in the policy. Make sure that your IAM and SAML users have the appropriate access to the collection before you delete a policy. To delete a policy in the OpenSearch Serverless console, select the policy and choose **Delete**.

You can also use the DeleteAccessPolicy command:

```
aws opensearchserverless delete-access-policy --name my-policy --type data
```

Cross-account data access

While you can't create a data access policy with cross-account identity or cross-account collections, you can still set up cross-account access with the assume role option. For example, if *account-a* owns a collection that *account-b* needs access to, the user from *account-b* can assume a role in *account-a*. The role must have the IAM permissions `aoss:APIAccessAll` and `aoss:DashboardsAccessAll`, and be included in the data access policy on *account-a*.

Access Amazon OpenSearch Serverless using an interface endpoint (AWS PrivateLink)

You can use AWS PrivateLink to create a private connection between your VPC and Amazon OpenSearch Serverless. You can access OpenSearch Serverless as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to access OpenSearch Serverless. For more

information on VPC network access, see Network connectivity patterns for Amazon OpenSearch Serverless.

You establish this private connection by creating an *interface endpoint*, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you specify for the interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for OpenSearch Serverless.

For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Topics

DNS resolution of collection endpoints

When you create a VPC endpoint, the service creates a new Amazon Route 53 private hosted zone and attaches it to the VPC. This private hosted zone consists of a record to resolve the wildcard DNS record for OpenSearch Serverless collections (`*.aoss.us-east-1.amazonaws.com`) to the interface addresses used for the endpoint. You only need one OpenSearch Serverless VPC endpoint in a VPC to access any and all collections and Dashboards in each AWS Region. Every VPC with an endpoint for OpenSearch Serverless has its own private hosted zone attached.

OpenSearch Serverless also creates a public Route 53 wildcard DNS record for all collections in the Region. The DNS name resolves to the OpenSearch Serverless public IP addresses. Clients in VPCs that don't have an OpenSearch Serverless VPC endpoint or clients in public networks can use the public Route 53 resolver and access the collections and Dashboards with those IP addresses. The IP address type (IPv4, IPv6, or Dualstack) of VPC endpoint is determined based on the subnets provided when you Create an interface endpoint for OpenSearch Serverless.

> **Note**
>
> OpenSearch Serverless creates an additonal Amazon Route 53 private hosted zone (`<region>.opensearch.amazonaws.com`) for an OpenSearch Service domain resolution. You can update your existing IPv4 VPC endpoint to Dualstack by using the update-vpc-endpoint command in the AWS CLI.

The DNS resolver address for a given VPC is the second IP address of the VPC CIDR. Any client in the VPC needs to use that resolver to get the VPC endpoint address for any collection. The resolver uses private hosted zone created by OpenSearch Serverless. It's sufficient to use that resolver for all collections in any account. It's also possible to use the VPC resolver for some collection endpoints and the public resolver for others, although it's not typically necessary.

VPCs and network access policies

To grant network permission to OpenSearch APIs and Dashboards for your collections, you can use OpenSearch Serverless network access policies. You can control this network access either from your VPC endpoint(s) or the public internet. Since your network policy only controls traffic permissions, you must also set up a data access policy that specifies permission to operate on the data in a collection and its indices. Think of an OpenSearch Serverless VPC endpoint as an access point to the service, a network access policy as the network-level access point to collections and Dashboards, and a data access policy as the access point for fine-grained access control for any operation on data in the collection.

Since you can specify multiple VPC endpoint IDs in a network policy, we recommend that you create a VPC endpoint for every VPC that needs to access a collection. These VPCs can belong to different AWS accounts than the account that owns the OpenSearch Serverless collection and network policy. We don’t recommend that you create a VPC-to-VPC peering or other proxying solution between two accounts so that one account's VPC can use another account's VPC endpoint. This is less secure and cost effective than each VPC having its own endpoint. The first VPC will not be easily visible to the other VPC’s admin, who has set up access to that VPC's endpoint in the network policy.

VPCs and endpoint policies

Amazon OpenSearch Serverless supports endpoint policies for VPCs. An endpoint policy is an IAM resource-based policy that you attach to a VPC endpoint to control which AWS principals can

use the endpoint to access your AWS service. For more information, see Control access to VPC endpoints using endpoint policies.

To use an endpoint policy, you must first create an interface endpoint. You can create an interface endpoint using either the OpenSearch Serverless console or the OpenSearch Serverless API. After you create your interface endpoint, you will need to add the endpoint policy to the endpoint. For more information, see Access Amazon OpenSearch Serverless using an interface endpoint (AWS PrivateLink).

> **Note**
>
> You can't define an endpoint policy directly in the OpenSearch Service console.

An endpoint policy does not override or replace other identity-based policies, resource-based policies, network policies, or data access policies you may have configured. For more information on updating endpoint policies, see Control access to VPC endpoints using endpoint policies.

By default, an endpoint policy grants full access to your VPC endpoint.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

Although the default VPC endpoint policy grants full endpoint access, you can configure a VPC endpoint policy to allow access to specific roles and users. To do this, see the following example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
```

```
                "AWS": [
                    "123456789012",
                    "987654321098"
                ]
            },
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

You can specify an OpenSearch Serverless collection to be included as a conditional element in your VPC endpoint policy. To do this, see the following example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aoss:collection": [
                        "coll-abc"
                    ]
                }
            }
        }
    ]
}
```

Support for `aoss:CollectionId` is supported.

```
Condition": {
         "StringEquals": {
               "aoss:CollectionId": "collection-id"
          }
}
```

You can use SAML identities in your VPC endpoint policy to determine VPC endpoint access. You must use a wildcard (`*`) in the principal section of your VPC endpoint policy. To do this, see the following example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:SamlGroups": [
                        "saml/123456789012/idp123/group/football",
                        "saml/123456789012/idp123/group/soccer",
                        "saml/123456789012/idp123/group/cricket"
                    ]
                }
            }
        }
    ]
}
```

Additionally, you can configure your endpoint policy to include a specific SAML principal policy. To do this, see the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aoss:SamlPrincipal": [
                        "saml/123456789012/idp123/user/user1234"]
                }
            }
```

```
            }
        ]
    }
```

For more information on using SAML authentication with Amazon OpenSearch Serverless, see SAML authentication for Amazon OpenSearch Serverless.

You can also include IAM and SAML users in the same VPC endpoint policy. To do this, see the following example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aoss:SamlGroups": [
                        "saml/123456789012/idp123/group/football",
                        "saml/123456789012/idp123/group/soccer",
                        "saml/123456789012/idp123/group/cricket"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "123456789012"
                ]
            },
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

You can also access an Amazon OpenSearch Serverless collection from Amazon EC2 via interface VPC endpoints. For more information on this see, Access an OpenSearch Serverless collection from Amazon EC2 (via interface VPC endpoints).

Considerations

Before you set up an interface endpoint for OpenSearch Serverless, consider the following:

- OpenSearch Serverless supports making calls to all supported OpenSearch API operations (not configuration API operations) through the interface endpoint.
- After you create an interface endpoint for OpenSearch Serverless, you still need to include it in network access policies in order for it to access serverless collections.
- By default, full access to OpenSearch Serverless is allowed through the interface endpoint. You can associate a security group with the endpoint network interfaces to control traffic to OpenSearch Serverless through the interface endpoint.
- A single AWS account can have a maximum of 50 OpenSearch Serverless VPC endpoints.
- If you enable public internet access to your collection's API or Dashboards in a network policy, your collection is accessible by any VPC and by the public internet.
- If you're on-premises and outside of the VPC, you can't use a DNS resolver for the OpenSearch Serverless VPC endpoint resolution directly. If you need VPN access, the VPC needs a DNS proxy resolver for external clients to use. Route 53 provides an inbound endpoint option that you can use to resolve DNS queries to your VPC from your on-premises network or another VPC.
- The private hosted zone that OpenSearch Serverless creates and attaches to the VPC is managed by the service, but it shows up in your Amazon Route 53 resources and is billed to your account.
- For other considerations, see Considerations in the *AWS PrivateLink Guide*.

Permissions required

VPC access for OpenSearch Serverless uses the following AWS Identity and Access Management (IAM) permissions. You can specify IAM conditions to restrict users to specific collections.

- `aoss:CreateVpcEndpoint` – Create a VPC endpoint.
- `aoss:ListVpcEndpoints` – List all VPC endpoints.
- `aoss:BatchGetVpcEndpoint` – See details about a subset of VPC endpoints.
- `aoss:UpdateVpcEndpoint` – Modify a VPC endpoint.
- `aoss:DeleteVpcEndpoint` – Delete a VPC endpoint.

In addition, you need the following Amazon EC2 and Route 53 permissions in order to create a VPC endpoint.

- `ec2:CreateTags`
- `ec2:CreateVpcEndpoint`
- `ec2:DeleteVpcEndPoints`
- `ec2:DescribeSecurityGroups`
- `ec2:DescribeSubnets`
- `ec2:DescribeVpcEndpoints`
- `ec2:DescribeVpcs`
- `ec2:ModifyVpcEndPoint`
- `route53:AssociateVPCWithHostedZone`
- `route53:ChangeResourceRecordSets`
- `route53:CreateHostedZone`
- `route53:DeleteHostedZone`
- `route53:GetChange`
- `route53:GetHostedZone`
- `route53:ListHostedZonesByName`
- `route53:ListHostedZonesByVPC`
- `route53:ListResourceRecordSets`

Create an interface endpoint for OpenSearch Serverless

You can create an interface endpoint for OpenSearch Serverless using either the console or the OpenSearch Serverless API.

To create an interface endpoint for an OpenSearch Serverless collection

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. In the left navigation pane, expand **Serverless** and choose **VPC endpoints**.
3. Choose **Create VPC endpoint**.
4. Provide a name for the endpoint.
5. For **VPC**, select the VPC that you'll access OpenSearch Serverless from.

6. For **Subnets**, select one subnet that you'll access OpenSearch Serverless from.
   - Endpoint's IP address and DNS type is based on subnet type
     - Dualstack: If all subnets have both IPv4 and IPv6 address ranges
     - IPv6: If all subnets are IPv6 only subnets
     - IPv4: If all subnets have IPv4 address ranges
7. For **Security groups**, select the security groups to associate with the endpoint network interfaces. This is a critical step where you limit the ports, protocols, and sources for inbound traffic that you're authorizing into your endpoint. Make sure that the security group rules allow the resources that will use the VPC endpoint to communicate with OpenSearch Serverless to communicate with the endpoint network interface.
8. Choose **Create endpoint**.

To create a VPC endpoint using the OpenSearch Serverless API, use the `CreateVpcEndpoint` command.

> **Note**
>
> After you create an endpoint, note its ID (for example, `vpce-050f79086ee71ac05`. In order to provide the endpoint access to your collections, you must include this ID in one or more network access policies.

Next step: Grant the endpoint access to a collection

After you create an interface endpoint, you must provide it access to collections through network access policies. For more information, see the section called "Network access".

SAML authentication for Amazon OpenSearch Serverless

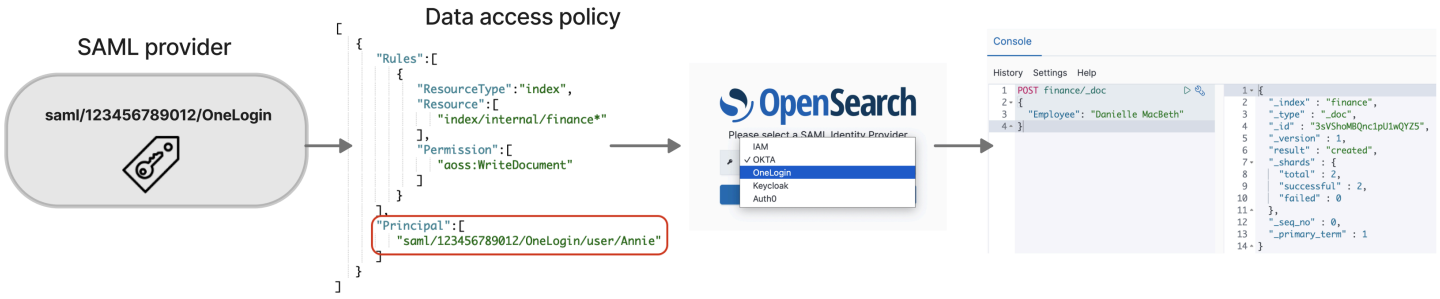
With SAML authentication for Amazon OpenSearch Serverless, you can use your existing identity provider to offer single sign-on (SSO) for the OpenSearch Dashboards endpoints of serverless collections.

SAML authentication lets you use third-party identity providers to sign in to OpenSearch Dashboards to index and search data. OpenSearch Serverless supports providers that use the SAML 2.0 standard, such as IAM Identity Center, Okta, Keycloak, Active Directory Federation Services (AD

FS), and Auth0. You can configure IAM Identity Center to synchronize users and groups from other identity sources like Okta, OneLogin, and Microsoft Entra ID. For a list of identity sources supported by IAM Identity Center and steps to configure them, see Getting started tutorials in the *IAM Identity Center User Guide*.

> **Note**
>
> SAML authentication is only for accessing OpenSearch Dashboards through a web browser. Authenticated users can only make requests to the OpenSearch API operations through **Dev Tools** in OpenSearch Dashboards. Your SAML credentials do *not* let you make direct HTTP requests to the OpenSearch API operations.

To set up SAML authentication, you first configure a SAML identity provider (IdP). You then include one or more users from that IdP in a data access policy. This policy grants it certain permissions to collections and/or indexes. A user can then sign in to OpenSearch Dashboards and perform the actions that are allowed in the data access policy.



Topics

- Considerations
- Permissions required
- Creating SAML providers (console)
- Accessing OpenSearch Dashboards
- Granting SAML identities access to collection data
- Creating SAML providers (AWS CLI)
- Viewing SAML providers
- Updating SAML providers
- Deleting SAML providers

Considerations

Consider the following when configuring SAML authentication:

- Signed and encrypted requests are not supported.
- Encrypted assertions are not supported.
- IdP-initiated authentication and sign-out are not supported.
- Service Control Policies (SCP) will not be applicable or evaluated in case of non-IAM identities (like SAML in Amazon OpenSearch Serverless & SAML and basic internal user authorization for Amazon OpenSearch Service).

Permissions required

SAML authentication for OpenSearch Serverless uses the following AWS Identity and Access Management (IAM) permissions:

- `aoss:CreateSecurityConfig` – Create a SAML provider.
- `aoss:ListSecurityConfig` – List all SAML providers in the current account.
- `aoss:GetSecurityConfig` – View SAML provider information.
- `aoss:UpdateSecurityConfig` – Modify a given SAML provider configuration, including the XML metadata.
- `aoss:DeleteSecurityConfig` – Delete a SAML provider.

The following identity-based access policy allows a user to manage all IdP configurations:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "aoss:CreateSecurityConfig",
                "aoss:DeleteSecurityConfig",
                "aoss:GetSecurityConfig",
                "aoss:UpdateSecurityConfig",
                "aoss:ListSecurityConfigs"
            ],
            "Effect": "Allow",
```

```
            "Resource": "*"
        }
    ]
}
```

Note that the `Resource` element must be a wildcard.

Creating SAML providers (console)

These steps explain how to create SAML providers. This enables SAML authentication with service provider (SP)-initiated authentication for OpenSearch Dashboards. IdP-initiated authentication is not supported.

To enable SAML authentication for OpenSearch Dashboards

1. Sign in to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. On the left navigation panel, expand **Serverless** and choose **SAML authentication**.
3. Choose **Add SAML provider**.
4. Provide a name and description for the provider.

   > **Note**
   >
   > The name that you specify is publicly accessible and will appear in a dropdown menu when users sign in to OpenSearch Dashboards. Make sure that the name is easily recognizable and doesn't reveal sensitive information about your identity provider.

5. Under **Configure your IdP**, copy the assertion consumer service (ACS) URL.
6. Use the ACS URL that you just copied to configure your identity provider. Terminology and steps vary by provider. Consult your provider's documentation.

   In Okta, for example, you create a "SAML 2.0 web application" and specify the ACS URL as the **Single Sign On URL**, **Recipient URL**, and **Destination URL**. For Auth0, you specify it in **Allowed Callback URLs**.
7. Provide the audience restriction if your IdP has a field for it. The audience restriction is a value within the SAML assertion that specifies who the assertion is intended for. For OpenSearch Serverless, specify `aws:opensearch:<aws account id>`. For example, `aws:opensearch:123456789012`.

The name of the audience restriction field varies by provider. For Okta it's **Audience URI (SP Entity ID)**. For IAM Identity Center it's **Application SAML audience**.

8. If you're using IAM Identity Center, you also need to specify the following attribute mapping: `Subject=${user:name}`, with a format of `unspecified`.
9. After you configure your identity provider, it generates an IdP metadata file. This XML file contains information about the provider, such as a TLS certificate, single sign-on endpoints, and the identity provider's entity ID.

   Copy the text in the IdP metadata file and paste it under **Provide metadata from your IdP** field. Alternately, choose **Import from XML file** and upload the file. The metadata file should look something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<md:EntityDescriptor entityID="entity-id"
 xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata">
  <md:IDPSSODescriptor WantAuthnRequestsSigned="false"
 protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <md:KeyDescriptor use="signing">
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:X509Data>
          <ds:X509Certificate>tls-certificate</ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>s
    </md:KeyDescriptor>
    <md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</
md:NameIDFormat>
    <md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress</
md:NameIDFormat>
    <md:SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
POST" Location="idp-sso-url"/>
    <md:SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
Redirect" Location="idp-sso-url"/>
  </md:IDPSSODescriptor>
</md:EntityDescriptor>
```

10. Keep the **Custom user ID attribute** field empty to use the `NameID` element of the SAML assertion for the username. If your assertion doesn't use this standard element and instead includes the username as a custom attribute, specify that attribute here. Attributes are case-sensitive. Only a single user attribute is supported.

The following example shows an override attribute for `NameID` in the SAML assertion:

```
<saml2:Attribute Name="UserId" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
  <saml2:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="xs:string">annie</saml2:AttributeValue>
</saml2:Attribute>
```

11. (Optional) Specify a custom attribute in the **Group attribute** field, such as `role` or `group`. Only a single group attribute is supported. There's no default group attribute. If you don't specify one, your data access policies can only contain user principals.

    The following example shows a group attribute in the SAML assertion:

    ```
    <saml2:Attribute Name="department"
     NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml2:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="xs:string">finance</saml2:AttributeValue>
    </saml2:Attribute>
    ```

12. By default, OpenSearch Dashboards signs users out after 24 hours. You can configure this value to any number between 1 and 12 hours (15 and 720 minutes) by specifying the **OpenSearch Dashboards timeout**. If you try to set the timeout equal to or less than 15 minutes, your session will be reset to one hour.
13. Choose **Create SAML provider**.

Accessing OpenSearch Dashboards

After you configure a SAML provider, all users and groups associated with that provider can navigate to the OpenSearch Dashboards endpoint. The Dashboards URL has the format *`collection-endpoint`*`/_dashboards/` *for all collections*.

If you have SAML enabled, selecting the link in the AWS Management Console directs you to the IdP selection page, where you can sign in using your SAML credentials. First, use the dropdown to select an identity provider:

Then sign in using your IdP credentials.

If you don't have SAML enabled, selecting the link in the AWS Management Console directs you to log in as an IAM user or role, with no option for SAML.

Granting SAML identities access to collection data

After you create a SAML provider, you still need to grant the underlying users and groups access to the data within your collections. You grant access through data access policies. Until you provide users access, they won't be able to read, write, or delete any data within your collections.

To grant access, create a data access policy and specify your SAML user and/or group IDs in the `Principal` statement:

```
[
   {
```

```
      "Rules":[
       ...
      ],
      "Principal":[
         "saml/987654321098/myprovider/user/Shaheen",
         "saml/987654321098/myprovider/group/finance"
      ]
   }
]
```

You can grant access to collections, indexes, or both. If you want different users to have different permissions, create multiple rules. For a list of available permissions, see Supported policy permissions. For information about how to format an access policy, see Policy syntax.

Creating SAML providers (AWS CLI)

To create a SAML provider using the OpenSearch Serverless API, send a CreateSecurityConfig request:

```
aws opensearchserverless create-security-config \
    --name myprovider \
    --type saml \
    --saml-options file://saml-auth0.json
```

Specify `saml-options`, including the metadata XML, as a key-value map within a .json file. The metadata XML must be encoded as a JSON escaped string.

```
{
   "sessionTimeout": 70,
   "groupAttribute": "department",
   "userAttribute": "userid",
   "metadata": "<EntityDescriptor xmlns=\"urn:oasis:names:tc:SAML:2.0:metadata
\" ... ... ... IDPSSODescriptor>\r\n<\/EntityDescriptor>"
}
```

Viewing SAML providers

The following ListSecurityConfigs request lists all SAML providers in your account:

```
aws opensearchserverless list-security-configs --type saml
```

The request returns information about all existing SAML providers, including the full IdP metadata that your identity provider generates:

```
{
   "securityConfigDetails": [
      {
         "configVersion": "MTY2NDA1MjY4NDQ5M18x",
         "createdDate": 1664054180858,
         "description": "Example SAML provider",
         "id": "saml/123456789012/myprovider",
         "lastModifiedDate": 1664054180858,
         "samlOptions": {
            "groupAttribute": "department",
            "metadata": "<EntityDescriptor xmlns=\"urn:oasis:names:tc:SAML:2.0:metadata
\" ... ... ... IDPSSODescriptor>\r\n<\/EntityDescriptor>",
            "sessionTimeout": 120,
            "userAttribute": "userid"
         }
      }
   ]
}
```

To view details about a specific provider, including the `configVersion` for future updates, send a `GetSecurityConfig` request.

Updating SAML providers

To update a SAML provider using the OpenSearch Serverless console, choose **SAML authentication**, select your identity provider, and choose **Edit**. You can modify all fields, including the metadata and custom attributes.

To update a provider through the OpenSearch Serverless API, send an UpdateSecurityConfig request and include the identifier of the policy to be updated. You must also include a configuration version, which you can retrieve using the `ListSecurityConfigs` or `GetSecurityConfig` commands. Including the most recent version ensures that you don't inadvertently override a change made by someone else.

The following request updates the SAML options for a provider:

```
aws opensearchserverless update-security-config \
    --id saml/123456789012/myprovider \
```

```
    --type saml \
    --saml-options file://saml-auth0.json \
    --config-version MTY2NDA1MjY4NDQ5M18x
```

Specify your SAML configuration options as a key-value map within a .json file.

> **Important**
>
> **Updates to SAML options are *not* incremental**. If you don't specify a value for a parameter in the `SAMLOptions` object when you make an update, the existing values will be overridden with empty values. For example, if the current configuration contains a value for `userAttribute`, and then you make an update and don't include this value, the value is removed from the configuration. Make sure you know what the existing values are before you make an update by calling the `GetSecurityConfig` operation.

Deleting SAML providers

When you delete a SAML provider, any references to associated users and groups in your data access policies are no longer functional. To avoid confusion, we suggest that you remove all references to the endpoint in your access policies before you delete the endpoint.

To delete a SAML provider using the OpenSearch Serverless console, choose **Authentication**, select the provider, and choose **Delete**.

To delete a provider through the OpenSearch Serverless API, send a DeleteSecurityConfig request:

```
aws opensearchserverless delete-security-config --id saml/123456789012/myprovider
```

Compliance validation for Amazon OpenSearch Serverless

Third-party auditors assess the security and compliance of Amazon OpenSearch Serverless as part of multiple AWS compliance programs. These programs include SOC, PCI, and HIPAA.

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security Compliance & Governance – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- HIPAA Eligible Services Reference – Lists HIPAA eligible services. Not all AWS services are HIPAA eligible.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Tagging Amazon OpenSearch Serverless collections

Tags let you assign arbitrary information to an Amazon OpenSearch Serverless collection so you can categorize and filter on that information. A *tag* is a metadata label that you assign or that AWS assigns to an AWS resource.

Each tag consists of a *key* and a *value*. For tags that you assign, you define the key and value. For example, you might define the key as `stage` and the value for one resource as `test`.

With tags, you can identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you could assign the same tag to an OpenSearch Serverless collection that you assign to an Amazon OpenSearch Service domain.

In OpenSearch Serverless, the primary resource is a collection. You can use the OpenSearch Service console, the AWS CLI, the OpenSearch Serverless API operations, or the AWS SDKs to add, manage, and remove tags from a collection.

Permissions required

OpenSearch Serverless uses the following AWS Identity and Access Management Access Analyzer (IAM) permissions for tagging collections:

- `aoss:TagResource`
- `aoss:ListTagsForResource`
- `aoss:UntagResource`

Tagging collections (console)

The console is the simplest way to tag a collection.

To create a tag (console)

1. Sign in to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Expand **Serverless** in the left navigation pane and choose **Collections**.
3. Select the collection that you want to add tags to, and go to the **Tags** tab.
4. Choose **Manage** and **Add new tag**.
5. Enter a tag key and an optional value.
6. Choose **Save**.

To delete a tag, follow the same steps and choose **Remove** on the **Manage tags** page.

For more information about using the console to work with tags, see Tag Editor in the *AWS Management Console Getting Started Guide*.

Tagging collections (AWS CLI)

To tag a collection using the AWS CLI, send a TagResource request:

```
aws opensearchserverless tag-resource
  --resource-arn arn:aws:aoss:us-east-1:123456789012:collection/my-collection
  --tags Key=service,Value=aoss Key=source,Value=logs
```

View the existing tags for a collection with the ListTagsForResource command:

```
aws opensearchserverless list-tags-for-resource
  --resource-arn arn:aws:aoss:us-east-1:123456789012:collection/my-collection
```

Remove tags from a collection using the UntagResource command:

```
aws opensearchserverless untag-resource
  --resource-arn arn:aws:aoss:us-east-1:123456789012:collection/my-collection
  --tag-keys service
```

Supported operations and plugins in Amazon OpenSearch Serverless

Amazon OpenSearch Serverless supports a variety of OpenSearch plugins, as well as a subset of the indexing, search, and metadata API operations available in OpenSearch. You can include the permissions in the left column of the table within data access policies in order to limit access to certain operations.

Topics

- Supported OpenSearch API operations and permissions
- Supported OpenSearch plugins

Supported OpenSearch API operations and permissions

The following table lists the API operations that OpenSearch Serverless supports, along with their corresponding data access policy permissions:

Data access policy permission	OpenSearch API operations	Description and caveats
`aoss:CreateIndex`	PUT <index>	Create indexes. For more information, see Create index. **Note** This permission also applies to creating indexes with the sample data on OpenSearch Dashboards.
`aoss:DescribeIndex`	• GET <index> • GET <index>/_mapping • GET <index>/_mappings • GET <index>/_setting • GET <index>/_setting/<setting> • GET <index>/_settings • GET <index>/_settings/<setting> • GET _cat/indices • GET _mapping • GET _mappings • GET _resolve/index/<index> • HEAD <index>	Describe indexes. For more information, see the following resources: • Get index • Get a mapping • Get settings • Index exists • CAT indices (Response does not include `health` or `status` fields.)
`aoss:WriteDocument`	• DELETE <index>/_doc/<id> • POST <index>/_bulk • POST <index>/_create/<id> (for search collection types only) • POST <index>/_doc	Write and update documents. For more information, see the following resources: • Bulk

Data access policy permission	OpenSearch API operations	Description and caveats
	• POST <index>/_update/<id> (for search collection types only) • POST _bulk • PUT <index>/_create/<id> (for search collection types only) • PUT <index>/_doc/<id> (for search collection types only)	• Index data **Note** Some operations are only allowed for collections of type SEARCH. For more informati on, see the section called “Choosing a collection type”.

Data access policy permission	OpenSearch API operations	Description and caveats
`aoss:ReadDocument`	• DELETE /_search/point_in_time/_all • DELETE /_search/point_in_time • GET <index>/_analyz • GET /_search/point_in_time/_all • GET <index>/_doc/<id> • GET <index>/_explain/<id> • GET <index>/_mget • GET <index>/_source/<id> • GET <index>/_count • GET <index>/_field_caps • GET <index>/_msearch • GET <index>/_rank_eval • GET <index>/_search • GET <index>/_validate/<query> • GET _analyze • GET _field_caps • GET _mget • GET _search • HEAD <index>/_doc/<id> • HEAD <index>/_source/<id> • POST /_plugins/_sql • POST /_plugins/_ppl • POST /_plugins/_sql/_explain • POST /_plugins/_ppl/_explain • POST /_plugins/_ppl/_close • POST <index>/_analyze	Read documents. For more information, see the following resources: • Perform text analysis • Get document • Count • Query DSL • Ranking evaluation • Analyze API • Explain

Data access policy permission	OpenSearch API operations	Description and caveats
	• POST /<target_indexes>/_search/point_in_time • POST <index>/_explain/<id> • POST <index>/_count • POST <index>/_field_caps • POST <index>/_rank_eval • POST <index>/_search • POST _analyze • POST _field_caps • POST _search	
`aoss:DeleteIndex`	DELETE <target>	Delete indexes. For more information, see Delete index.
`aoss:UpdateIndex`	• POST _mapping • POST <index>/_mapping/ • POST <index>/_mappings/ • POST <index>/_setting • POST <index>/_settings • POST _setting • POST _settings • PUT _mapping • PUT <index>/_mapping • PUT <index>/_mappings/ • PUT <index>/_setting • PUT <index>/_settings • PUT _setting • PUT _settings	Update index settings. For more information, see the following resources: • Mapping • Update settings

Data access policy permission	OpenSearch API operations	Description and caveats
`aoss:CreateCollectionItems`	POST _aliases	Create index aliases. For more information, see Create aliases.
`aoss:DescribeCollectionItems`	• GET <index>/_alias/<alias> • GET _alias • GET _alias/<alias> • GET _cat/aliases • GET _cat/templates • GET _cat/templates/<template_name> • GET _component_template • GET _component_template/<component-template> • GET _index_template • GET _index_template/<index-template> • HEAD _alias/<alias> • HEAD _component_template/<component-template> • HEAD _index_template/<name> • HEAD <index>/_alias/<alias>	Describe aliases and index templates. For more information, see the following resources: • Manage aliases • Index templates

Data access policy permission	OpenSearch API operations	Description and caveats
`aoss:UpdateCollectionItems`	• POST <index>/_alias/<alias> • POST <index>/_aliases/<alias> • POST _component_template/<component-template> • POST _index_template/<index-template> • PUT <index>/_alias/<alias> • PUT <index>/_aliases/<alias> • PUT _component_template/<component-template> • PUT _index_template/<index-template>	Update aliases and index templates. For more information, see the following resources: • Index aliases • Index templates
`aoss:DeleteCollectionItems`	• DELETE <index>/_alias/<alias> • DELETE _component_template/<component-template> • DELETE _index_template/<index-template> • DELETE <index>/_aliases/<alias>	Delete aliases and index templates. For more information, see the following resources: • Delete aliases • Delete a template

Supported OpenSearch plugins

OpenSearch Serverless collections come prepackaged with the following plugins from the OpenSearch community. Serverless automatically deploys and manages plugins for you.

Analysis plugins

- ICU Analysis
- Japanese (kuromoji) Analysis
- Korean (Nori) Analysis
- Phonetic Analysis

- Smart Chinese Analysis
- Stempel Polish Analysis
- Ukrainian Analysis

Mapper plugins

- Mapper Size
- Mapper Murmur3
- Mapper Annotated Text

Scripting plugins

- Painless
- Expression
- Mustache

In addition, OpenSearch Serverless includes all plugins that ship as modules.

Monitoring Amazon OpenSearch Serverless

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon OpenSearch Serverless and your other AWS solutions. AWS provides the following monitoring tools to watch OpenSearch Serverless, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors your AWS resources and the applications that you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify.

  For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account. It delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.

- *Amazon EventBridge* delivers a near real-time stream of system events that describe changes in your OpenSearch Service domains. You can create rules that watch for certain events, and trigger automated actions in other AWS services when these events occur. For more information, see the Amazon EventBridge User Guide.

Monitoring OpenSearch Serverless with Amazon CloudWatch

You can monitor Amazon OpenSearch Serverless using CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing.

You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the Amazon CloudWatch User Guide.

OpenSearch Serverless reports the following metrics in the `AWS/AOSS` namespace.

Metric	Description
`ActiveCollection`	Indicates whether a collection is active. A value of 1 means that the collection is in an `ACTIVE` state. This value is emitted upon successful creation of a collection and remains 1 until you delete the collection. The metric can't have a value of 0. **Relevant statistics**: Max **Dimensions**: `ClientId`, `CollectionId` , `CollectionName` **Frequency**: 60 seconds
`DeletedDocuments`	The total number of deleted documents. **Relevant statistics**: Average, Sum **Dimensions**: `ClientId`, `CollectionId` , `CollectionName` , `IndexId`, `IndexName`

Metric	Description
	Frequency: 60 seconds
IndexingOCU	The number of OpenSearch Compute Units (OCUs) used to ingest collection data. This metric applies at the account level. **Relevant statistics**: Sum **Dimensions**: ClientId **Frequency**: 60 seconds
IngestionDataRate	The indexing rate in GiB per second to a collection or index. This metric only applies to bulk indexing requests. **Relevant statistics**: Sum **Dimensions**: ClientId, CollectionId , CollectionName , IndexId, IndexName **Frequency**: 60 seconds
IngestionDocumentErrors	The total number of document errors during ingestion for a collection or index. After a successful bulk indexing request, writers process the request and emit errors for all failed documents within the request. **Relevant statistics**: Sum **Dimensions**: ClientId, CollectionId , CollectionName , IndexId, IndexName **Frequency**: 60 seconds

Metric	Description
`IngestionDocumentRate`	The rate per second at which documents are being ingested to a collection or index. This metric only applies to bulk indexing requests. **Relevant statistics**: Sum **Dimensions**: `ClientId`, `CollectionId` , `CollectionName` , `IndexId`, `IndexName` **Frequency**: 60 seconds
`IngestionRequestErrors`	The total number of bulk indexing request errors to a collection. OpenSearch Serverless emits this metric when a bulk indexing request fails for any reason, such as an authentication or availability issue. **Relevant statistics**: Sum **Dimensions**: `ClientId`, `CollectionId` , `CollectionName` **Frequency**: 60 seconds
`IngestionRequestLatency`	The latency, in seconds, for bulk write operations to a collection. **Relevant statistics**: Minimum, Maximum, Average **Dimensions**: `ClientId`, `CollectionId` , `CollectionName` **Frequency**: 60 seconds

Metric	Description
IngestionRequestRate	The total number of bulk write operations received by a collection. **Relevant statistics**: Minimum, Maximum, Average **Dimensions**: ClientId, CollectionId , CollectionName **Frequency**: 60 seconds
IngestionRequestSuccess	The total number of successful indexing operations to a collection. **Relevant statistics**: Sum **Dimensions**: ClientId, CollectionId , CollectionName **Frequency**: 60 seconds
SearchableDocuments	The total number of searchable documents in a collection or index. **Relevant statistics**: Sum **Dimensions**: ClientId, CollectionId , CollectionName , IndexId, IndexName **Frequency**: 60 seconds
SearchRequestErrors	The total number of query errors per minute for a collection. **Relevant statistics**: Sum **Dimensions**: ClientId, CollectionId , CollectionName **Frequency**: 60 seconds

Metric	Description
SearchRequestLatency	The average time, in milliseconds, that it takes to complete a search operation against a collection. **Relevant statistics**: Minimum, Maximum, Average **Dimensions**: ClientId, CollectionId , CollectionName **Frequency**: 60 seconds
SearchOCU	The number of OpenSearch Compute Units (OCUs) used to search collection data. This metric applies at the account level. **Relevant statistics**: Sum **Dimensions**: ClientId **Frequency**: 60 seconds
SearchRequestRate	The total number of search requests per minute to a collection. **Relevant statistics**: Average, Maximum, Sum **Dimensions**: ClientId, CollectionId , CollectionName **Frequency**: 60 seconds

Metric	Description
`StorageUsedInS3`	The amount, in bytes, of Amazon S3 storage used. OpenSearch Serverless stores indexed data in Amazon S3. You must select the period at one minute to get an accurate value. **Relevant statistics**: Sum **Dimensions**: `ClientId`, `CollectionId` , `CollectionName` , `IndexId`, `IndexName` **Frequency**: 60 seconds
`2xx, 3xx, 4xx, 5xx`	The number of requests to the collection that resulted in the given HTTP response code (2*xx*, 3*xx*, 4*xx*, 5*xx*). **Relevant statistics**: Sum **Dimensions**: `ClientId`, `CollectionId` , `CollectionName` **Frequency**: 60 seconds

Logging OpenSearch Serverless API calls using AWS CloudTrail

Amazon OpenSearch Serverless is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Serverless.

CloudTrail captures all API calls for OpenSearch Serverless as events. The calls captured include calls from the Serverless section of the OpenSearch Service console and code calls to the OpenSearch Serverless API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for OpenSearch Serverless. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**.

Using the information collected by CloudTrail, you can determine the request that was made to OpenSearch Serverless, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

OpenSearch Serverless information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in OpenSearch Serverless, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history.

For an ongoing record of events in your AWS account, including events for OpenSearch Serverless, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions.

The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

All OpenSearch Serverless actions are logged by CloudTrail and are documented in the OpenSearch Serverless API reference. For example, calls to the `CreateCollection`, `ListCollections`, and `DeleteCollection` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding OpenSearch Serverless log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries.

An event represents a single request from any source. It includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateCollection` action.


```
{
   "eventVersion":"1.08",
   "userIdentity":{
      "type":"AssumedRole",
      "principalId":"AIDACKCEVSQ6C2EXAMPLE",
      "arn":"arn:aws:iam::123456789012:user/test-user",
      "accountId":"123456789012",
      "accessKeyId":"access-key",
      "sessionContext":{
         "sessionIssuer":{
            "type":"Role",
            "principalId":"AIDACKCEVSQ6C2EXAMPLE",
            "arn":"arn:aws:iam::123456789012:role/Admin",
            "accountId":"123456789012",
            "userName":"Admin"
         },
         "webIdFederationData":{

         },
         "attributes":{
            "creationDate":"2022-04-08T14:11:34Z",
            "mfaAuthenticated":"false"
         }
      }
   },
   "eventTime":"2022-04-08T14:11:49Z",
   "eventSource":"aoss.amazonaws.com",
   "eventName":"CreateCollection",
   "awsRegion":"us-east-1",
   "sourceIPAddress":"AWS Internal",
```

```
   "userAgent":"aws-cli/2.1.30 Python/3.8.8 Linux/5.4.176-103.347.amzn2int.x86_64 exe/
x86_64.amzn.2 prompt/off command/aoss.create-collection",
   "errorCode":"HttpFailureException",
   "errorMessage":"An unknown error occurred",
   "requestParameters":{
      "accountId":"123456789012",
      "name":"test-collection",
      "description":"A sample collection",
      "clientToken":"d3a227d2-a2a7-49a6-8fb2-e5c8303c0718"
   },
   "responseElements": null,
   "requestID":"12345678-1234-1234-1234-987654321098",
   "eventID":"12345678-1234-1234-1234-987654321098",
   "readOnly":false,
   "eventType":"AwsApiCall",
   "managementEvent":true,
   "recipientAccountId":"123456789012",
   "eventCategory":"Management",
   "tlsDetails":{
      "clientProvidedHostHeader":"user.aoss-sample.us-east-1.amazonaws.com"
   }
}
```


Monitoring OpenSearch Serverless events using Amazon EventBridge

Amazon OpenSearch Service integrates with Amazon EventBridge to notify you of certain events that affect your domains. Events from AWS services are delivered to EventBridge in near real time. The same events are also sent to Amazon CloudWatch Events, the predecessor of Amazon EventBridge. You can write rules to indicate which events are of interest to you, and what automated actions to take when an event matches a rule. Examples of actions that you can automatically activate include the following:

- Invoking an AWS Lambda function
- Invoking an Amazon EC2 Run Command
- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine
- Notifying an Amazon SNS topic or an Amazon SQS queue

For more information, see Get started with Amazon EventBridge in the *Amazon EventBridge User Guide*.

Setting up notifications

You can use AWS User Notifications to receive notifications when an OpenSearch Serverless event occurs. An event is an indicator of a change in OpenSearch Serverless environment, such as when you reach the maximum limit of your OCU usage. Amazon EventBridge receives the event and routes a notification to the AWS Management Console Notifications Center and your chosen delivery channels. You receive a notification when an event matches a rule that you specify.

OpenSearch Compute Units (OCU) events

OpenSearch Serverless sends events to EventBridge when one of the following OCU-related events occur.

OCU usage approaching maximum limit

OpenSearch Serverless sends this event when your search or index OCU usage reaches 75% of your capacity limit. Your OCU usage is calculated based on your configured capacity limit and your current OCU consumption.

Example

The following is an example event of this type (search OCU):

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "OCU Utilization Approaching Max Limit",
  "source": "aws.aoss",
  "account": "123456789012",
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "eventTime" : 1678943345789,
    "description": "Your search OCU usage is at 75% and is approaching the configured
 maximum limit."
  }
}
```

The following is an example event of this type (index OCU):

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "OCU Utilization Approaching Max Limit",
  "source": "aws.aoss",
  "account": "123456789012",
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "eventTime" : 1678943345789,
    "description": "Your indexing OCU usage is at 75% and is approaching the configured
 maximum limit."
  }
```

OCU usage reached maximum limit

OpenSearch Serverless sends this event when your search or index OCU usage reaches 100% of your capacity limit. Your OCU usage is calculated based on your configured capacity limit and your current OCU consumption.

Example

The following is an example event of this type (search OCU):

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "OCU Utilization Reached Max Limit",
  "source": "aws.aoss",
  "account": "123456789012",
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "eventTime" : 1678943345789,
    "description": "Your search OCU usage has reached the configured maximum limit."
  }
}
```

The following is an example event of this type (index OCU):

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "OCU Utilization Reached Max Limit",
  "source": "aws.aoss",
  "account": "123456789012",
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "eventTime" : 1678943345789,
    "description": "Your indexing OCU usage has reached the configured maximum limit."
  }
}
```

Creating and managing Amazon OpenSearch Service domains

This chapter describes how to create and manage Amazon OpenSearch Service domains. A domain is the AWS-provisioned equivalent of an open source OpenSearch cluster. When you create a domain, you specify its settings, instance types, instance counts, and storage allocation. For more information about open source clusters, see Creating a cluster in the OpenSearch documentation.

Unlike the brief instructions in the Getting started tutorial, this chapter describes all options and provides relevant reference information. You can complete each procedure by using instructions for the OpenSearch Service console, the AWS Command Line Interface (AWS CLI), or the AWS SDKs.

Creating OpenSearch Service domains

This section describes how to create OpenSearch Service domains by using the OpenSearch Service console or by using the AWS CLI with the `create-domain` command.

Creating OpenSearch Service domains (console)

Use the following procedure to create an OpenSearch Service domain by using the console.

To create an OpenSearch Service domain (console)

1. Go to https://aws.amazon.com and choose **Sign In to the Console**.
2. Under **Analytics**, choose **Amazon OpenSearch Service**.
3. Choose **Create domain**.
4. For **Domain name**, enter a domain name. The name must meet the following criteria:
   - Unique to your account and AWS Region
   - Starts with a lowercase letter
   - Contains between 3 and 28 characters
   - Contains only lowercase letters a-z, the numbers 0-9, and the hyphen (-)
5. For the domain creation method, choose **Standard create**.
6. For **Templates**, choose the option that best matches the purpose of your domain:

- **Production** domains for workloads that need high-availability and performance. These domains use Multi-AZ (with or without standby) and dedicated master nodes for higher availability.
- **Dev/test** for development or testing. These domains can use Multi-AZ (with or without standby) or a single Availability Zone.

> **Important**
>
> Different deployment types present different options on subsequent pages. These steps include all options.

7. For **Deployment Option(s)**, choose **Domain with standby** to configure a 3-AZ domain, with nodes in one of the zones are reserved as standby. This option enforces a number of best practices, such as a specified data node count, master node count, instance type, replica count, and software update settings.
8. For **Version**, choose the version of OpenSearch or legacy Elasticsearch OSS to use. We recommend that you choose the latest version of OpenSearch. For more information, see the section called "Supported versions of OpenSearch and Elasticsearch".

   (Optional) If you chose an OpenSearch version for your domain, select **Enable compatibility mode** to make OpenSearch report its version as 7.10, which allows certain Elasticsearch OSS clients and plugins that check the version before connecting to continue working with the service.
9. For **Instance type**, choose an instance type for your data nodes. For more information, see the section called "Supported instance types".

   > **Note**
   >
   > Not all Availability Zones support all instance types. If you choose Multi-AZ with or without Standby, we recommend choosing current-generation instance types, such as R5 or I3.
10. For **Number of nodes**, choose the number of data nodes.

    For maximum values, see OpenSearch Service domain and instance quotas. Single-node clusters are fine for development and testing, but should not be used for production

workloads. For more guidance, see the section called “Sizing domains” and the section called “Configuring a multi-AZ domain”.

> **Note**
>
> (Optional)Dedicated coordinator nodes support all OpenSearch versions and ElasticSearch versions 6.8 through 7.10. Dedicated coordinator nodes are available for use with domains that have a dedicated cluster manager enabled. To enable dedicated coordinator nodes, you will select the instance type and count. As a best practice, you should keep the instance family for your dedicated coordinator node the same as your data nodes (Intel based instances or Graviton based instances).

11. For **Storage type**, select Amazon EBS. The volume types available in the list depend on the instance type that you've chosen. For guidance on creating especially large domains, see the section called “Petabyte scale”.
12. For **EBS** storage, configure the following additional settings. Some settings might not appear depending on the type of volume you choose.

Setting	Description
EBS volume type	Choose between General Purpose (SSD) - gp3 and General Purpose (SSD) - gp2, or the previous generation Provisioned IOPS (SSD), and Magnetic (standard).
EBS storage size per node	Enter the size of the EBS volume that you want to attach to each data node. EBS volume size is per node. You can calculate the total cluster size for the OpenSearch Service domain by multiplying the number of data nodes by the EBS volume size. The minimum and maximum size of an EBS volume depends on both the specified EBS volume type and the instance type that it's attached to. To learn more, see EBS volume size limits.
Provisioned IOPS	If you selected a Provisioned IOPS SSD volume type, enter the number of I/O operations per second (IOPS) that the volume can support.

13. (Optional) If you selected a gp3 volume type, expand **Advanced settings** and specify additional IOPS (up to 16,000 for every 3 TiB volume size provisioned per data node) and throughput (up to 1,000 MiB/s for every 3 TiB volume size provisioned per data node) beyond what is included with the price of storage, for an additional cost. For more information, see the Amazon OpenSearch Service pricing.
14. (Optional) To enable UltraWarm storage, choose **Enable UltraWarm data nodes**. Each instance type has a maximum amount of storage that it can address. Multiply that amount by the number of warm data nodes for the total addressable warm storage.
15. (Optional) To enable cold storage, choose **Enable cold storage**. You must enable UltraWarm to enable cold storage.
16. If you use Multi-AZ with Standby, three dedicated master nodes are aleady enabled. Choose the type of master nodes that you want. If you chose a Multi-AZ without Standby domain, select **Enable dedicated master nodes** and choose the type and number of master nodes that you want. Dedicated master nodes increase cluster stability and are required for domains that have instance counts greater than 10. We recommend three dedicated master nodes for production domains.

    > **Note**
    >
    > You can choose different instance types for your dedicated master nodes and data nodes. For example, you might select general purpose or storage-optimized instances for your data nodes, but compute-optimized instances for your dedicated master nodes.

17. (Optional) For domains running OpenSearch or Elasticsearch 5.3 and later, the **Snapshot configuration** is irrelevant. For more information about automated snapshots, see the section called “Creating index snapshots”.
18. If you want to use a custom endpoint rather than the standard one of `https://search-`*`mydomain`*`-`*`1a2a3a4a5a6a7a8a9a0a9a8a7a`*`.`*`us-east-1`*`.es.amazonaws.com`, choose **Enable custom endpoint** and provide a name and certificate. For more information, see the section called “Creating a custom endpoint”.
19. Under **Network**, choose either **VPC access** or **Public access**. If you choose **Public access**, skip to the next step. If you choose **VPC access**, make sure you meet the prerequisites, then configure the following settings:

Setting	Description
VPC	Choose the ID of the virtual private cloud (VPC) that you want to use. The VPC and domain must be in the same AWS Region, and you must select a VPC with tenancy set to **Default**. OpenSearch Service does not yet support VPCs that use dedicated tenancy.
Subnet	Choose a subnet. If you enabled Multi-AZ, you must choose two or three subnets. OpenSearch Service will place a VPC endpoint and *elastic network interfaces* in the subnets. You must reserve sufficient IP addresses for the network interfaces in the subnet(s). For more information, see Reserving IP addresses in a VPC subnet.
Security groups	Choose one or more VPC security groups that allow your required application to reach the OpenSearch Service domain on the ports (80 or 443) and protocols (HTTP or HTTPS) exposed by the domain. For more information, see the section called “VPC support”.
IAM Role	Keep the default role. OpenSearch Service uses this predefined role (also known as a *service-linked role*) to access your VPC and to place a VPC endpoint and network interfaces in the subnet of the VPC. For more information, see Service-linked role for VPC access.
IP Address Type	Choose either dual stack or IPv4 as your IP address type. Dual stack allows you to share domain resources across IPv4 and IPv6 address types, and is the recommended option. If you set your IP address type to dual stack, you can't change your address type later.

20. Enable or disable fine-grained access control:

    - If you want to use IAM for user management, choose **Set IAM ARN as master user** and specify the ARN for an IAM role.
    - If you want to use the internal user database, choose **Create master user** and specify a username and password.

Whichever option you choose, the master user can access all indexes in the cluster and all OpenSearch APIs. For guidance on which option to choose, see the section called "Key concepts".

If you disable fine-grained access control, you can still control access to your domain by placing it within a VPC, applying a restrictive access policy, or both. You must enable node-to-node encryption and encryption at rest to use fine-grained access control.

> **Note**
>
> We *strongly* recommend enabling fine-grained access control to protect the data on your domain. Fine-grained access control provides security at the cluster, index, document, and field levels.

21. (Optional) If you want to use SAML authentication for OpenSearch Dashboards, choose **Enable SAML authentication** and configure SAML options for the domain. For instructions, see the section called "SAML authentication for OpenSearch Dashboards".
22. (Optional) If you want to use Amazon Cognito authentication for OpenSearch Dashboards, choose **Enable Amazon Cognito authentication**. Then choose the Amazon Cognito user pool and identity pool that you want to use for OpenSearch Dashboards authentication. For guidance on creating these resources, see the section called "Amazon Cognito authentication for OpenSearch Dashboards".
23. For **Access policy**, choose an access policy or configure one of your own. If you choose to create a custom policy, you can configure it yourself or import one from another domain. For more information, see the section called "Identity and Access Management".

    > **Note**
    >
    > If you enabled VPC access, you can't use IP-based policies. Instead, you can use security groups to control which IP addresses can access the domain. For more information, see the section called "About access policies on VPC domains".

24. (Optional) To require that all requests to the domain arrive over HTTPS, select **Require HTTPS for all traffic to the domain**. To enable node-to-node encryption, select **Node-to-node encryption**. For more information, see the section called "Node-to-node encryption". To

enable encryption of data at rest, select **Enable encryption of data at rest**. These options are pre-selected if you chose the Multi-AZ with Standby deployment option.

25. (Optional) Select **Use AWS owned key** to have OpenSearch Service create an AWS KMS encryption key on your behalf (or use the one that it already created). Otherwise, choose your own KMS key. For more information, see the section called “Encryption at rest”.
26. For **Off-peak window**, select a start time to schedule service software updates and Auto-Tune optimizations that require a blue/green deployment. Off-peak updates help to minimize strain on a cluster's dedicated master nodes during high traffic periods.
27. For **Auto-Tune**, choose whether to allow OpenSearch Service to suggest memory-related configuration changes to your domain to improve speed and stability. For more information, see the section called “Auto-Tune”.

    (Optional) Select **Off-peak window** to schedule a recurring window during which Auto-Tune updates the domain.
28. (Optional) Select **Automatic software update** to enable automatic software updates.
29. (Optional) Add tags to describe your domain so you can categorize and filter on that information. For more information, see the section called “Tagging domains”.
30. (Optional) Expand and configure **Advanced cluster settings**. For a summary of these options, see the section called “Advanced cluster settings”.
31. Choose **Create**.

Creating OpenSearch Service domains (AWS CLI)

Instead of creating an OpenSearch Service domain by using the console, you can use the AWS CLI. For syntax, see Amazon OpenSearch Service in the AWS CLI command referencea.

Example commands

This first example demonstrates the following OpenSearch Service domain configuration:

- Creates an OpenSearch Service domain named *mylogs* with OpenSearch version 1.2
- Populates the domain with two instances of the `r6g.large.search` instance type
- Uses a 100 GiB General Purpose (SSD) gp3 EBS volume for storage for each data node
- Allows anonymous access, but only from a single IP address: 192.0.2.0/32

```
aws opensearch create-domain \
    --domain-name mylogs \
    --engine-version OpenSearch_1.2 \
    --cluster-config  InstanceType=r6g.large.search,InstanceCount=2 \
    --ebs-options
 EBSEnabled=true,VolumeType=gp3,VolumeSize=100,Iops=3500,Throughput=125 \
    --access-policies '{"Version": "2012-10-17", "Statement": [{"Action": "es:*",
 "Principal":"*","Effect": "Allow", "Condition": {"IpAddress":{"aws:SourceIp":
["192.0.2.0/32"]}}}]}'
```

The next example demonstrates the following OpenSearch Service domain configuration:

- Creates an OpenSearch Service domain named *mylogs* with Elasticsearch version 7.10
- Populates the domain with six instances of the `r6g.large.search` instance type
- Uses a 100 GiB General Purpose (SSD) gp2 EBS volume for storage for each data node
- Restricts access to the service to a single user, identified by the user's AWS account ID: 555555555555
- Distributes instances across three Availability Zones

```
aws opensearch create-domain \
    --domain-name mylogs \
    --engine-version Elasticsearch_7.10 \
    --cluster-config
 InstanceType=r6g.large.search,InstanceCount=6,ZoneAwarenessEnabled=true,ZoneAwarenessConfig={A
 \
    --ebs-options EBSEnabled=true,VolumeType=gp2,VolumeSize=100 \
    --access-policies '{"Version": "2012-10-17", "Statement": [ { "Effect": "Allow",
 "Principal": {"AWS": "arn:aws:iam::555555555555:root" }, "Action":"es:*", "Resource":
 "arn:aws:es:us-east-1:555555555555:domain/mylogs/*" } ] }'
```

The next example demonstrates the following OpenSearch Service domain configuration:

- Creates an OpenSearch Service domain named *mylogs* with OpenSearch version 1.0
- Populates the domain with ten instances of the `r6g.xlarge.search` instance type
- Populates the domain with three instances of the `r6g.large.search` instance type to serve as dedicated master nodes
- Uses a 100 GiB Provisioned IOPS EBS volume for storage, configured with a baseline performance of 1000 IOPS for each data node

- Restricts access to a single user and to a single subresource, the `_search` API

```
aws opensearch create-domain \
    --domain-name mylogs \
    --engine-version OpenSearch_1.0 \
    --cluster-config
 InstanceType=r6g.xlarge.search,InstanceCount=10,DedicatedMasterEnabled=true,DedicatedMasterTyp
 \
    --ebs-options EBSEnabled=true,VolumeType=io1,VolumeSize=100,Iops=1000 \
    --access-policies '{"Version": "2012-10-17", "Statement": [ { "Effect": "Allow",
 "Principal": { "AWS": "arn:aws:iam::555555555555:root" }, "Action": "es:*",
 "Resource": "arn:aws:es:us-east-1:555555555555:domain/mylogs/_search" } ] }'
```

> **Note**
>
> If you attempt to create an OpenSearch Service domain and a domain with the same name already exists, the CLI does not report an error. Instead, it returns details for the existing domain.

Creating OpenSearch Service domains (AWS SDKs)

The AWS SDKs (except the Android and iOS SDKs) support all the actions defined in the Amazon OpenSearch Service API Reference, including `CreateDomain`. For sample code, see the section called "Using the AWS SDKs". For more information about installing and using the AWS SDKs, see AWS Software Development Kits.

Creating OpenSearch Service domains (AWS CloudFormation)

OpenSearch Service is integrated with AWS CloudFormation, a service that helps you to model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes the OpenSearch domain you want to create, and CloudFormation provisions and configures the domain for you. For more information, including examples of JSON and YAML templates for OpenSearch domains, see the Amazon OpenSearch Service resource type reference in the *AWS CloudFormation User Guide*.

Configuring access policies

Amazon OpenSearch Service offers several ways to configure access to your OpenSearch Service domains. For more information, see the section called "Identity and Access Management" and the section called "Fine-grained access control".

The console provides preconfigured access policies that you can customize for the specific needs of your domain. You also can import access policies from other OpenSearch Service domains. For information about how these access policies interact with VPC access, see the section called "About access policies on VPC domains".

To configure access policies (console)

1. Go to https://aws.amazon.com, and then choose **Sign In to the Console**.
2. Under **Analytics**, choose **Amazon OpenSearch Service**.
3. In the navigation pane, under **Domains**, choose the domain you want to update.
4. Choose **Actions** and **Edit security configuration**.
5. Edit the access policy JSON, or import a preconfigured option.
6. Choose **Save changes**.

Advanced cluster settings

Use advanced options to configure the following:

Indices in request bodies

Specifies whether explicit references to indexes are allowed inside the body of HTTP requests. Setting this property to `false` prevents users from bypassing access control for subresources. By default, the value is `true`. For more information, see the section called "Advanced options and API considerations".

Fielddata cache allocation

Specifies the percentage of Java heap space that is allocated to field data. By default, this setting is 20% of the JVM heap.

> **Note**
>
> Many customers query rotating daily indices. We recommend that you begin benchmark testing with `indices.fielddata.cache.size` configured to 40% of the JVM heap for most of these use cases. For very large indices, you might need a large field data cache.

Max clause count

Specifies the maximum number of clauses allowed in a Lucene boolean query. The default is 1,024. Queries with more than the permitted number of clauses result in a `TooManyClauses` error. For more information, see the Lucene documentation.

Making configuration changes in Amazon OpenSearch Service

Amazon OpenSearch Service uses a *blue/green* deployment process when updating domains. A blue/green deployment creates an idle environment for domain updates that copies the production environment, and routes users to the new environment after those updates are complete. In a blue/green deployment, the blue environment is the current production environment. The green environment is the idle environment.

Data is migrated from the blue environment to the green environment. When the new environment is ready, OpenSearch Service switches over the environments to promote the green environment to be the new production environment. The switchover happens with no data loss. This practice minimizes downtime and maintains the original environment in the event that deployment to the new environment is unsuccessful.

Changes that usually cause blue/green deployments

The following operations cause blue/green deployments:

- Changing the instance type
- Enabling fine-grained access control
- Performing service software updates
- Enabling or disabling dedicated master nodes

- Enabling or disabling Multi-AZ without Standby
- Changing the storage type, volume type, or volume size
- Choosing different VPC subnets
- Adding or removing VPC security groups
- Adding or removing dedicated coordinator nodes
- Enabling or disabling Amazon Cognito authentication for OpenSearch Dashboards
- Choosing a different Amazon Cognito user pool or identity pool
- Modifying advanced settings
- Upgrading to a new OpenSearch version (OpenSearch Dashboards might be unavailable during some or all of the upgrade)
- Enabling encryption of data at rest or node-to-node encryption
- Enabling or disabling UltraWarm or cold storage
- Disabling Auto-Tune and rolling back its changes
- Associating an optional plugin to a domain and dissociating an optional plugin from a domain
- Increasing the dedicated master node count for Multi-AZ domains with two dedicated master nodes
- Decreasing the EBS volume size
- Changing EBS volume size, IOPS, or throughput, if the the last change you made is in progress or occurred less than 6 hours ago
- Enabling the publication of audit logs to CloudWatch.

For Multi-AZ with Standby domains, you can only make one change request at a time. If a change is already in progress, the new request is rejected. You can check the status of the current change with the `DescribeDomainChangeProgress` API.

Changes that usually don't cause blue/green deployments

In *most* cases, the following operations do not cause blue/green deployments:

- Modifying the access policy
- Modifying the custom endpoint
- Changing the Transport Layer Security (TLS) policy

- Changing the automated snapshot hour
- Enabling or disabling **Require HTTPS**
- Enabling Auto-Tune or disabling it without rolling back its changes
- If your domain has dedicated master nodes, changing the data node or UltraWarm node count
- If your domain has dedicated master nodes, changing the dedicated master instance type or count (except for Multi-AZ domains with two dedicated master nodes)
- Enabling or disabling the publication of error logs or slow logs to CloudWatch
- Disabling the publication of audit logs to CloudWatch
- Increasing the volume size up to 3 TiB per data node, changing the volume type, IOPS, or throughput
- Adding or removing tags

> **Note**
>
> There are some exceptions depending on your service software version. If you want to be sure that a change won't cause a blue/green deployment, perform a dry run before updating your domain, if this option is available. Some changes don't offer a dry run option. We generally recommend that you make changes to your cluster outside of peak traffic hours.

Determining whether a change will cause a blue/green deployment

You can test some types of planned configuration changes to determine whether they will cause a blue/green deployment, without having to commit to those changes. Before you initiate a configuration change, use the console or an API to run a validation check to ensure that your domain is eligible for an update.

Console

To validate a configuration change

1. Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/.
2. In the left navigation pane, choose **Domains**.

3. Select the domain you want to make a configuration change for. This opens the domain details page. Select the **Actions** dropdown menu and then choose **Edit cluster configuration**.
4. On the **Edit cluster configuration** page, you can make changes to the instance type, the number of nodes, and any other configurations. After you've confirmed your changes in the summary panel, choose **Run**.
5. Once your dry run is complete, the results automatically display at the bottom of the page, along with a dry run ID. These results notify you which category your change falls into:

   - Initiates a blue/green deployment
   - Doesn't require a blue/green deployment
   - Contains validation errors that you need to address before you can save your changes

   Note that each dry run overwrites the one before it. To look up the details of each dry run later on, make sure you save your dry run ID. Each dry run is available for 90 days, or until you make a configuration update.
6. To proceed with your configuration update, choose **Save changes**. Otherwise, choose **Cancel**. Either option takes you back to the **Cluster configuration** tab. On this tab, you can choose **Dry run details** to see the details of your latest dry run. This page also includes a side-by-side comparison between the configuration before the dry run and the dry run configuration.

API

You can perform a dry run validation through the configuration API. To test your changes with the API, set `DryRun` to `true`, and `DryRunMode` to `Verbose`. Verbose mode runs a validation check in addition to determining whether the change will initiate a blue/green deployment. For example, this UpdateDomainConfig request tests the deployment type that results from enabling UltraWarm:

```
POST https://es.us-east-1.amazonaws.com/2021-01-01/opensearch/domain/my-domain/
config
{
   "ClusterConfig": {
    "WarmCount": 3,
    "WarmEnabled": true,
    "WarmType": "ultrawarm1.large.search"
```

```
   },
   "DryRun": true,
   "DryRunMode": "Verbose"
}
```

The request runs a validation check and returns the type of deployment the change will cause but doesn't actually perform the update:

```
{
   "ClusterConfig": {
     ...
    },
   "DryRunResults": {
      "DeploymentType": "Blue/Green",
      "Message": "This change will require a blue/green deployment."
    }
}
```

Possible deployment types are:

- `Blue/Green` – The change will cause a blue/green deployment.
- `DynamicUpdate` – The change won't cause a blue/green deployment.
- `Undetermined` – The domain is still in a processing state, so the deployment type can't be determined.
- None – No configuration change.

If the validation fails, it returns a list of validation failures.

```
{
   "ClusterConfig":{
      "..."
   },
   "DryRunProgressStatus":{
      "CreationDate":"2023-01-12T01:14:33.847Z",
      "DryRunId":"db00ca39-48b2-4774-bbd3-252cf094d205",
      "DryRunStatus":"failed",
      "UpdateDate":"2023-01-12T01:14:33.847Z",
      "ValidationFailures":[
         {
            "Code":"Cluster.Index.WriteBlock",
```

```
            "Message":"Cluster has index write blocks."
         }
      ]
   }
}
```

If the status is still `pending`, you can use the dry run ID in your UpdateDomainConfig response in subsequent DescribeDryRunProgress calls to check the status of the validation.

```
GET https://es.us-east-1.amazonaws.com/2021-01-01/opensearch/domain/my-domain/
dryRun?dryRunId=my-dry-run-id
{
    "DryRunConfig": null,
    "DryRunProgressStatus": {
        "CreationDate": "2023-01-12T01:14:42.998Z",
        "DryRunId": "db00ca39-48b2-4774-bbd3-252cf094d205",
        "DryRunStatus": "succeeded",
        "UpdateDate": "2023-01-12T01:14:49.334Z",
        "ValidationFailures": null
    },
    "DryRunResults": {
        "DeploymentType": "Blue/Green",
        "Message": "This change will require a blue/green deployment."
    }
}
```

To run a dry run analysis without a validation check, set `DryRunMode` to `Basic` when you use the configuration API.

Python

The following Python code uses the UpdateDomainConfig API to perform a dry run validation check and, if the check succeeds, calls the same API without a dry run to start the update. If the check fails, the script prints out the error and stops.

```
import time
import boto3

client = boto3.client('opensearch')

response = client.UpdateDomainConfig(
    ClusterConfig={
        'WarmCount': 3,
```

```
        'WarmEnabled': True,
        'WarmCount': 123,
    },
    DomainName='test-domain',
    DryRun=True,
    DryRunMode='Verbose'
)

dry_run_id = response.DryRunProgressStatus.DryRunId

retry_count = 0

while True:

    if retry_count == 5:
        print('An error occured')
        break

    dry_run_progress_response = client.DescribeDryRunProgress('test-domain',
 dry_run_id)
    dry_run_status = dry_run_progress_response.DryRunProgressStatus.DryRunStatus

    if dry_run_status == 'succeeded':
        client.UpdateDomainConfig(
            ClusterConfig={
            'WarmCount': 3,
            'WarmEnabled': True,
            'WarmCount': 123,
        })
        break

    elif dry_run_status == 'failed':
        validation_failures_list =
 dry_run_progress_response.DryRunProgressStatus.ValidationFailures
        for item in validation_failures_list:
            print(f"Code: {item['Code']}, Message: {item['Message']}")
        break

    retry_count += 1
    time.sleep(30)
```

Initiating and tracking a configuration change

> **Note**
>
> You can request one configuration change at a time. You can also group multiple configuration changes in a single request. Wait for the status of your domain to become `Active` before requesting any additional configuration changes.

You can view the **Domain Processing Status** and **Config Change Status** fields in the Amazon OpenSearch Service console to track domain and configuration changes. You can also track domain and configuration changes through the `DomainProcessingStatus` and `ConfigChangeStatus` parameters in the API responses. For more information, see the DomainStatus data type in the OpenSearch Service API reference.

Domain processing status visibility: You can easily determine the configuration status of a domain by looking at the **Domain Processing Status** field in the console. Similarly, the `DomainProcessingStatus` API parameter can be used to identify the status. The following values are processing statuses for a domain:

- `Active`: No configuration change is in progress. You can submit a new configuration change request.
- `Creating`: Domain is being created.
- `Modifying`: Configuration changes, such as the addition of new data nodes, EBS, gp3, IOPS provisioning, or setting up KMS keys, are in progress.

  > **Note**
  >
  > You might see the status as `Modifying` in situations where a domain requires shard movement to complete the configuration changes. For backwards compatibility, the behavior of the `Processing` parameter is kept unchanged in the API responses, and is set to false as soon as core configuration changes are complete, without waiting for shard movement completion.

- `Upgrading Engine Version`: An engine version upgrade is in progress.
- `Updating Service Software`: A service software update is in progress.
- `Deleting`: The domain is being deleted.

- `Isolated`: The domain is suspended.

Configuration status visibility: Configuration changes can be initiated by the operator (e.g. new data node addition, instance type change) or by the service (e.g. Auto-Tune and off-peak hour updates). You can find the status of the latest configuration change details in the **Configuration Change Status** field of the Amazon OpenSearch Service console, and in the `ConfigChangeStatus` API response. The following values indicate the configuration status of a domain:

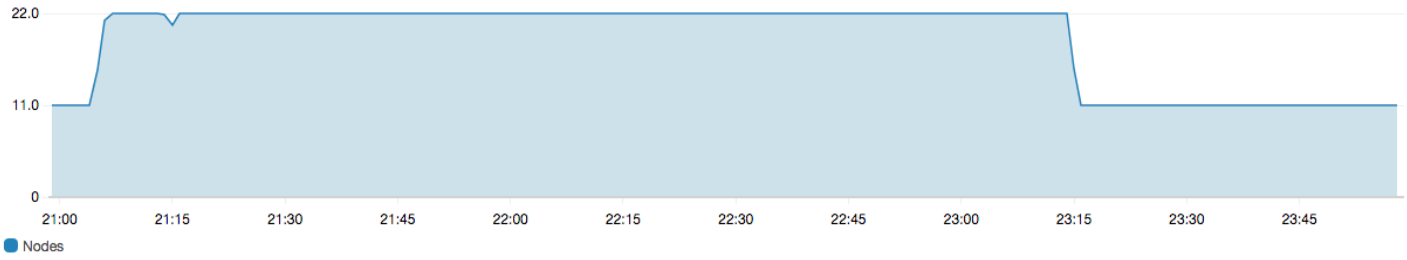
- `Pending`: A configuration change request has been submitted.
- `Initializing`: Service is initializing a configuration change request.
- `Validating`: Service is validating the requested changes and resources required.
- `Awaiting user inputs`: Applies when operator expects some configuration changes such as instance type change to proceed further. You are able to edit configuration changes.
- `Applying changes`: Service is applying requested configuration changes.
- `Cancelled`: Configuration change is cancelled. If you receive the validation failed status, you can click **Cancel** in the console or call the `CancelDomainConfigChange` API operation. If you do this, all the applied changes are rolled back.
- `Completed`: Requested configuration changes have been completed with success.
- `Validation Failed`: Requested changes failed validation. No configuration changes are applied.

> **Note**
>
> Validation failures could be the result of red indexes present in your domain, unavailability of a chosen instance type, or low disk space. For a list of validation errors, see the section called "Troubleshooting validation errors". During a validation failure event, you can cancel, retry, or edit configuration changes.

API Summary: You can use the `DescribeDomain`, `DescribeDomainChangeProgress`, and `DescribeDomainConfig` API operations to get detailed configuration update statuses. In addition, you can use `CancelDomainConfigChange` to cancel the updates in the event of validation failures. For more information, see the OpenSearch Service API documentation

When the configuration changes are complete, the domain state changes back to `Active`.

You can review the cluster health and Amazon CloudWatch metrics and see that the number of nodes in the cluster temporarily increases—often doubling—while the domain update occurs. In the following illustration, you can see the number of nodes doubling from 11 to 22 during a configuration change and returning to 11 when the update is complete.



This temporary increase can strain the cluster's dedicated master nodes, which suddenly might have many more nodes to manage. It can also increase search and indexing latencies as OpenSearch Service copies data from the old cluster to the new one. It's important to maintain sufficient capacity on the cluster to handle the overhead that is associated with these blue/green deployments.

> **Important**
>
> You do *not* incur any additional charges during configuration changes and service maintenance. You're billed only for the number of nodes that you request for your cluster. For specifics, see the section called “Charges for configuration changes”.

To prevent overloading dedicated master nodes, you can monitor usage with the Amazon CloudWatch metrics. For recommended maximum values, see the section called “Recommended CloudWatch alarms”.

Stages of a configuration change

After you initiate a configuration change, OpenSearch Service goes through a series of steps to update your domain. You can view the progress of the configuration change under **Configuration change status** in the console. The exact steps that an update goes through depends on the type of change you're making. You can also monitor a configuration change using the DescribeDomainChangeProgress API operation.

The following are possible stages an update can go through during a configuration change:

Stage name	Description
Validation	Validating that the domain is eligible for an update, and surfacing validation issues if necessary.
Creating a new environment	Completing the necessary prerequisites and creating required resources to start the blue/green deployment.
Provisioning new nodes	Creating a new set of instances in the new environment.
Traffic routing on new nodes	Redirecting traffic to the newly created data nodes.
Traffic routing on old nodes	Disabling traffic on the old data nodes.

Stage name	Description
Preparing nodes for removal	Preparing to remove nodes. This step only happens when you're downscaling your domain (for example, from 8 nodes to 6 nodes).
Copying shards to new nodes	Moving shards from the old nodes to the new nodes.
Terminating nodes	Terminating and deleting old nodes after shards are removed.
Deleting older resources	Deleting resources associated with the old environme nt (e.g. load balancer).

Stage name	Description
Dynamic update	Displayed when the update does not require a blue/green deployment and can be dynamically applied.
Applying dedicated master related changes	Displayed when the dedicated master instance type or count is changed.
Applying volume related changes	Displayed when volume size, type, IOPS and throughput are changed.

Performance impact of blue/green deployments

During blue/green deployment your Amazon OpenSearch Service cluster is available for incoming search and indexing requests. However, you might experience the following performance issues:

- Temporary increase in usage on leader nodes as clusters have more nodes to manage.
- Increased search and indexing latency as OpenSearch Service copies data from old nodes to new nodes.

- Increased rejections for incoming requests as the cluster load increases during blue/green deployments.
- To avoid latency issues and request rejections, you should run blue/green deployments when the cluster is healthy and there's low network traffic.

Charges for configuration changes

If you change the configuration for a domain, OpenSearch Service creates a new cluster as described in the section called “Configuration changes”. During the migration of old to new, you incur the following charges:

- If you change the instance type, you're charged for both clusters for the first hour. After the first hour, you're only charged for the new cluster. EBS volumes aren't charged twice because they're part of your cluster, so their billing follows instance billing.

  Example: You change the configuration from three `m3.xlarge` instances to four `m4.large` instances. For the first hour, you're charged for both clusters (3 * `m3.xlarge` + 4 * `m4.large`). After the first hour, you're charged only for the new cluster (4 * `m4.large`).
- If you don't change the instance type, you're charged only for the largest cluster for the first hour. After the first hour, you're charged only for the new cluster.

  Example: You change the configuration from six `m3.xlarge` instances to three `m3.xlarge` instances. For the first hour, you're charged for the largest cluster (6 * `m3.xlarge`). After the first hour, you're charged only for the new cluster (3 * `m3.xlarge`).

Troubleshooting validation errors

When you initiate a configuration change or perform an OpenSearch or Elasticsearch version upgrade, OpenSearch Service first performs a series of validation checks to ensure that your domain is eligible for an update. If any of these checks fail, you receive a notification in the console containing the specific issues that you must fix before updating your domain. The following table lists the possible domain issues that OpenSearch Service might surface, and steps to resolve them.

Issue	Error code	Troubleshooting steps
Security group not found	`SecurityGroupNotFound`	The security group associated with your OpenSearch Service domain does not exist. To resolve this issue, create a security group with the specified name.
Subnet not found	`SubnetNotFound`	The subnet associated with your OpenSearch Service domain does not exist. To resolve this issue, create a subnet in your VPC.
Service-linked role not configured	`SLRNotConfigured`	The service-linked role for OpenSearch Service is not configured. The service-linked role is predefined by OpenSearch Service and includes all the permissions the service requires to call other AWS services on your behalf. If the role doesn't exist, you might need to create it manually.
Not enough IP addresses	`InsufficientFreeIPsForSubnets`	One or more of your VPC subnets don't have enough IP addresses to update your domain. To calculate how many IP addresses you need, see the section called “Reserving IP addresses in a VPC subnet”.
Cognito user pool doesn't exist	`CognitoUserPoolNotFound`	OpenSearch Service can't find the Amazon Cognito user pool. Confirm that you created one and have the correct ID. To find the ID, you can use the Amazon Cognito console or the following AWS CLI command: `aws cognito-idp list-user-pools --max-results 60 --region us-east-1`
Cognito identity pool doesn't exist	`CognitoIdentityPoolNotFound`	OpenSearch Service can't find the Cognito identity pool. Confirm that you created one and have the correct ID. To find the ID, you can use the Amazon Cognito console or the following AWS CLI command: `aws cognito-identity list-identity-pools --max-results 60 --region us-east-1`

Issue	Error code	Troubleshooting steps
Cognito domain not found for user pool	`CognitoDomainNotFound`	The user pool does not have a domain name. You can configure one using the Amazon Cognito console or the following AWS CLI command: `aws cognito-idp create-user-pool-domain --domain my-domain --user-pool-id id`
Cognito role not configured	`CognitoRoleNotConfigured`	The IAM role that grants OpenSearch Service permission to configure the Amazon Cognito user and identity pools, and use them for authentication, is not configured. Configure the role with an appropriate permission set and trust relationship. You can use the console, which creates the default CognitoAccessForAmazonOpenSearch role for you, or you can manually configure a role using the AWS CLI or the AWS SDK.
Unable to describe user pool	`UserPoolNotDescribable`	The specified Amazon Cognito role doesn't have permission to describe the user pool associated with your domain. Make sure the role permissions policy allows the `cognito-identity:DescribeUserPool` action. See the section called "About the CognitoAccessForAmazonOpenSearch role" for the full permissions policy.
Unable to describe identity pool	`IdentityPoolNotDescribable`	The specified Amazon Cognito role doesn't have permission to describe the identity pool associated with your domain. Make sure the role permissions policy allows the `cognito-identity:DescribeIdentityPool` action. See the section called "About the CognitoAccessForAmazonOpenSearch role" for the full permissions policy.

Issue	Error code	Troubleshooting steps
Unable to describe user and identity pool	`CognitoPoolsNotDescribable`	The specified Amazon Cognito role doesn't have permission to describe the user and identity pools associated with your domain. Make sure the role permissions policy allows the `cognito-identity:DescribeIdentityPool` and `cognito-identity:DescribeUserPool` actions. See the section called "About the CognitoAccessForAmazonOpenSearch role" for the full permissions policy.
KMS key not enabled	`KMSKeyNotEnabled`	The AWS Key Management Service (AWS KMS) key used to encrypt your domain is disabled. Re-enable the key immediately.
Custom certificate not in ISSUED state	`InvalidCertificate`	If your domain uses a custom endpoint, you secure it by either generating an SSL certificate in AWS Certificate Manager (ACM) or importing one of your own. The certificate status must be **Issued**. If you receive this error, check the status of your certificate in the ACM console. If the status is Expired, Failed, Inactive, or Pending validation, see the ACM troubleshooting documentation to resolve the issue.
Not enough capacity to launch chosen instance type	`InsufficientInstanceCapacity`	The requested instance type capacity is not available. For example, you might have requested five `i3.16xlarge.search` nodes, but OpenSearch Service doesn't have enough `i3.16xlarge.search` hosts available, so the request can't be fulfilled. Check the supported instance types in OpenSearch Service and choose a different instance type.
Red indexes in cluster	`RedCluster`	One or more indexes in your cluster have a red status, leading to an overall red cluster status. To troubleshoot and remediate this issue, see the section called "Red cluster status".
Memory circuit breaker, too many requests	`TooManyRequests`	There are too many search and write requests to your domain, so OpenSearch Service can't update its configuration. You can reduce the number of requests, scale instances vertically up to 64 GiB of RAM, or scale horizontally by adding instances.

Issue	Error code	Troubleshooting steps
New configuration can't hold data (low disk space)	`InsufficientStorageCapacity`	The configured storage size can't hold all of the data on your domain. To resolve this issue, choose a larger volume, delete unused indexes, or increase the number of nodes in the cluster to immediately free up disk space.

Shards pinned to specific nodes	`ShardMovementBlocked`	One or more indexes in your domain are attached to specific nodes and can't be reassigned. This most likely happened because you configured shard allocation filtering, which lets you specify which nodes are allowed to host the shards of a particular index.
		
		To resolve this issue, remove shard allocation filters from all affected indexes:
		
		`PUT my-index/_settings`
		`{`
		`  "settings": {`
		`  "index.routing.allocation.require._name": null`
		`  }`
		`}`

New configuration can't hold all shards (shard count)	`TooManyShards`	The shard count on your domain is too high, which prevents OpenSearch Service from moving them to the new configuration. To resolve this issue, scale your domain horizonally by adding nodes of the same configuration type as your current cluster nodes. Note that the maximum EBS volume size depends on the node's instance type. To prevent this issue in the future, see the section called “Choosing the number of shards” and define a sharding strategy that is appropriate for your use case.

Issue	Error code	Troubleshooting steps
The subnet associated with your domain does not support IPv4 addresses	`ResultCodeIPv4BlockNotExists`	To resolve this issue, create a subnet or update the existing subnet in your VPC according to the configured IP address type of the domain. If your domain uses an **IPv4 only** address type, use an IPv4-only subnet. If your domain uses **Dual-stack mode**, use a dual-stack subnet.
The subnet associated with your domain does not support IPv6 addresses	`ResultCodeIPv6BlockNotExists`	To resolve this issue, create a subnet or update the existing subnet in your VPC according to the configured IP address type of the domain. If your domain uses an **IPv4 only** address type, use an IPv4-only subnet. If your domain uses **Dual-stack mode**, use a dual-stack subnet.

Service software updates in Amazon OpenSearch Service

> **Note**
>
> For explanations of the changes and additions made in each *major* (non-patch) service software update, see the release notes.

Amazon OpenSearch Service regularly releases service software updates that add features or otherwise improve your domains. The **Notifications** panel in the console is the easiest way to see if an update is available or to check the status of an update. Each notification includes details about the service software update. All service software updates use blue/green deployments to minimize downtime.

Service software updates differ from OpenSearch *version* upgrades. For information about upgrading to a later version of OpenSearch, see the section called “Upgrading domains”.

Optional versus required updates

OpenSearch Service has two broad categories of service software updates:

Optional updates

Optional service software updates generally include enhancements and support for new features or functionality. Optional updates aren't enforced on your domains, and there's no hard deadline to install them. The availability of the update is communicated through email and a console notification. You can choose to apply the update immediately or reschedule it for a more appropriate date and time. You can also schedule it during the domain's off-peak window. The majority of software updates are optional.

Regardless of whether or not you schedule an update, if you make a change on the domain that causes a blue/green deployment, OpenSearch Service automatically updates your service software for you.

You can configure your domain to automatically apply optional updates during off-peak hours. When this option is turned on, OpenSearch Service waits at least 13 days from when an optional update is available and then schedules the update after 72 hours (three days). You receive a console notification when the update is scheduled and you can choose to reschedule it for a later date.

To turn on automatic software updates, select **Enable automatic software update** when you create or update your domain. To configure the same setting using the AWS CLI, set `--software-update-options` to `true` when you create or update your domain.

Required updates

Required service software updates generally include critical security fixes or other mandatory updates to ensure the continued integrity and functionality of your domain. Examples of required updates are Log4j Common Vulnerabilities and Exposures (CVEs) and enforcement of Instance Metadata Service Version 2 (IMDSv2). The number of mandatory updates in a year is usually less than three.

OpenSearch Service automatically schedules these updates and notifies you 72 hours (three days) before the scheduled update through email and a console notification. You can choose to apply the update immediately or reschedule it for a more appropriate date and time *within the allowed timeframe.* You can also schedule it during the domain's next off-peak window. If you take no

action on a required update and you don't make any domain changes that cause a blue/green deployment, OpenSearch Service can initiate the update at any time beyond the specified deadline (typically 14 days from availability), within the domain's off-peak window.

Regardless of when the update is scheduled for, if you make a change on the domain that causes a blue/green deployment, OpenSearch Service automatically updates your domain for you.

Patch updates

Service software versions that end in "-P" and a number, such as R20211203-*P4*, are patch releases. Patches are likely to include performance improvements, minor bug fixes, and security fixes or posture improvements. Patch releases do not include new features or breaking changes, and they generally don't have a direct or noticeable impact on users. The service software notification tells you if a patch release is optional or mandatory.

Considerations

Consider the following when deciding whether to update your domain:

- Manually updating your domain lets you take advantage of new features more quickly. When you choose **Update**, OpenSearch Service places the request in a queue and begins the update when it has time.
- When you initiate a service software update, OpenSearch Service sends a notification when the update starts and when it completes.
- Software updates use blue/green deployments to minimize downtime. Updates can temporarily strain a cluster's dedicated master nodes, so make sure to maintain sufficient capacity to handle the associated overhead.
- Updates typically complete within minutes, but can also take several hours or even days if your system is experiencing heavy load. Consider updating your domain during the configured off-peak window to avoid long update periods.

Starting a service software update

You can request a service software update through the OpenSearch Service console, the AWS CLI, or one of the SDKs.

Console

To request a service software update

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Select the domain name to open its configuration.
3. Choose **Actions**, **Update** and select one of the following options:

   - **Apply update now** - Immediately schedules the action to happen in the current hour *if there's capacity available*. If capacity isn't available, we provide other available time slots to choose from.
   - **Schedule it in off-peak window** – Only available if the off-peak window is enabled for the domain. Schedules the update to take place during the domain's configured off-peak window. There's no guarantee that the update will happen during the next immediate window. Depending on capacity, it might happen in subsequent days. For more information, see the section called “Off-peak windows”.
   - **Schedule for specific date and time** – Schedules the update to take place at a specific date and time. If the time that you specify is unavailable for capacity reasons, you can select a different time slot.

   If you schedule the update for a later date (within or outside the domain's off-peak window), you can reschedule it at any time. For instructions, see the section called “Rescheduling actions”.
4. Choose **Confirm**.

AWS CLI

Send a start-service-software-update AWS CLI request to initiate a service software update. This example adds the update to the queue immediately:

```
aws opensearch start-service-software-update \
  --domain-name my-domain \
  --schedule-at "NOW"
```

Response:

```
{
```

```
    "ServiceSoftwareOptions": {
        "CurrentVersion": "R20220928-P1",
        "NewVersion": "R20220928-P2",
        "UpdateAvailable": true,
        "Cancellable": true,
        "UpdateStatus": "PENDING_UPDATE",
        "Description": "",
        "AutomatedUpdateDate": "1969-12-31T16:00:00-08:00",
        "OptionalDeployment": true
    }
}
```

Tip

After you request an update, you have a narrow window of time in which you can cancel it. The duration of this `PENDING_UPDATE` state can vary greatly and depends on your AWS Region and the number of concurrent updates that OpenSearch Service is performing. To cancel an update, use the console or `cancel-service-software-update` AWS CLI command.

If the request fails with a `BaseException`, it means that the time you specified isn't available for capacity reasons, and you must specify a different time. OpenSearch Service provides alternate available slot suggestions in the response.

AWS SDKs

This sample Python script uses the describe_domain and start_service_software_update methods from the AWS SDK for Python (Boto3) to check whether a domain is eligible for a service software update and if so, starts the update. You must provide a value for `domain_name`.

```
import boto3
from botocore.config import Config
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default region.

my_config = Config(
    # Optionally lets you specify a Region other than your default.
```

```
    region_name='us-east-1'
)

domain_name = ''  # The name of the domain to check and update

client = boto3.client('opensearch', config=my_config)


def getUpdateStatus(client):
    """Determines whether the domain is eligible for an update"""
    response = client.describe_domain(
        DomainName=domain_name
    )
    sso = response['DomainStatus']['ServiceSoftwareOptions']
    if sso['UpdateStatus'] == 'ELIGIBLE':
        print('Domain [' + domain_name + '] is eligible for a service software update
 from version ' +
              sso['CurrentVersion'] + ' to version ' + sso['NewVersion'])
        updateDomain(client)
    else:
        print('Domain is not eligible for an update at this time.')


def updateDomain(client):
    """Starts a service software update for the eligible domain"""
    response = client.start_service_software_update(
        DomainName=domain_name
    )
    print('Updating domain [' + domain_name + '] to version ' +
          response['ServiceSoftwareOptions']['NewVersion'] + '...')
    waitForUpdate(client)


def waitForUpdate(client):
    """Waits for the domain to finish updating"""
    response = client.describe_domain(
        DomainName=domain_name
    )
    status = response['DomainStatus']['ServiceSoftwareOptions']['UpdateStatus']
    if status == 'PENDING_UPDATE' or status == 'IN_PROGRESS':
        time.sleep(30)
        waitForUpdate(client)
    elif status == 'COMPLETED':
        print('Domain [' + domain_name +
```

```
                '] successfully updated to the latest software version')
    else:
        print('Domain is not currently being updated.')

def main():
    getUpdateStatus(client)
```

Scheduling software updates during off-peak windows

Each OpenSearch Service domain created after February 16, 2023 has a daily 10-hour window between 10:00 P.M. and 8:00 A.M. local time that we consider the off-peak window. OpenSearch Service uses this window to schedule service software updates for the domain. Off-peak updates help to minimize strain on a cluster's dedicated master nodes during higher traffic periods. OpenSearch Service can't initiate updates outside of this 10-hour window without your consent.

- For *optional* updates, OpenSearch Service notifies you of the update's availability and prompts you to schedule the update during an upcoming off-peak window.
- For *required* updates, OpenSearch Service automatically schedules the update during an upcoming off-peak window and notifies you three days ahead of time. You can reschedule the update (for within or outside the off-peak window), but only within the required timeframe for the update to be completed.

For each domain, you can choose to override the default 10:00 P.M. start time with a custom time. For instructions, see the section called “Configuring a custom off-peak window”.

Console

To schedule an update during an upcoming off-peak window

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Select the domain name to open its configuration.
3. Choose **Actions**, **Update**.
4. Select **Schedule it in off-peak window**.
5. Choose **Confirm**.

You can view the scheduled action on the **Off-peak window** tab and reschedule it at any time. See the section called “Viewing scheduled actions”.

CLI

To schedule an update during an upcoming off-peak window using the AWS CLI, send a StartServiceSoftwareUpdate request and specify `OFF_PEAK_WINDOW` for the `--schedule-at` parameter:

```
aws opensearch start-service-software-update \
  --domain-name my-domain \
  --schedule-at "OFF_PEAK_WINDOW"
```

Monitoring service software updates

OpenSearch Service sends a notification when a service software update is available, required, started, completed, or failed. You can view these notifications on the **Notifications** panel of the OpenSearch Service console. The notification severity is `Informational` if the update is optional and `High` if it's required.

OpenSearch Service also sends service software events to Amazon EventBridge. You can use EventBridge to configure rules that send an email or perform a specific action when an event is received. For an example walkthrough, see the section called "Tutorial: Sending SNS alerts for available updates".

To see the format of each service software event sent to Amazon EventBridge, see the section called "Service software update events".

When domains are ineligible for an update

Your domain is ineligible for a service software update if it's in any of the following states:

State	Description
Domain in processing	The domain is in the middle of a configuration change. Check update eligibility after the operation completes.
Red cluster status	One or more indexes in the cluster is red. For troubleshooting steps, see the section called "Red cluster status".
High error rate	The OpenSearch cluster is returning a large number of 5*xx* errors when attempting to process requests. This problem is usually the result of

State	Description
	too many simultaneous read or write requests. Consider reducing traffic to the cluster or scaling your domain.
Split brain	*Split brain* means your OpenSearch cluster has more than one master node and has split into two clusters that never will rejoin on their own. You can avoid split brain by using the recommended number of dedicated master nodes. For help recovering from split brain, contact Support.
Amazon Cognito integration issue	Your domain uses authentication for OpenSearch Dashboards, and OpenSearch Service can't find one or more Amazon Cognito resources. This problem usually occurs if the Amazon Cognito user pool is missing. To correct the issue, recreate the missing resource and configure the OpenSearch Service domain to use it.
Other service issue	Issues with OpenSearch Service itself might cause your domain to display as ineligible for an update. If none of the previous conditions apply to your domain and the problem persists for more than a day, contact Support.

Defining off-peak windows for Amazon OpenSearch Service

When you create an Amazon OpenSearch Service domain, you define a daily 10-hour window that's considered *off-peak* hours. OpenSearch Service uses this window to schedule service software updates and Auto-Tune optimizations that require a blue/green deployment during comparatively lower traffic times, whenever possible. Blue/green refers to the process of creating a new environment for domain updates and routing users to the new environment after those updates are complete.

Although blue/green deployments are non-disruptive, to minimize any potential performance impact while resources are being consumed for a blue/green deployment, we recommend that you schedule these deployments during the domain's configured off-peak window. Updates such as node replacements, or those that need to be deployed to the domain immediately, don't use the off-peak window.

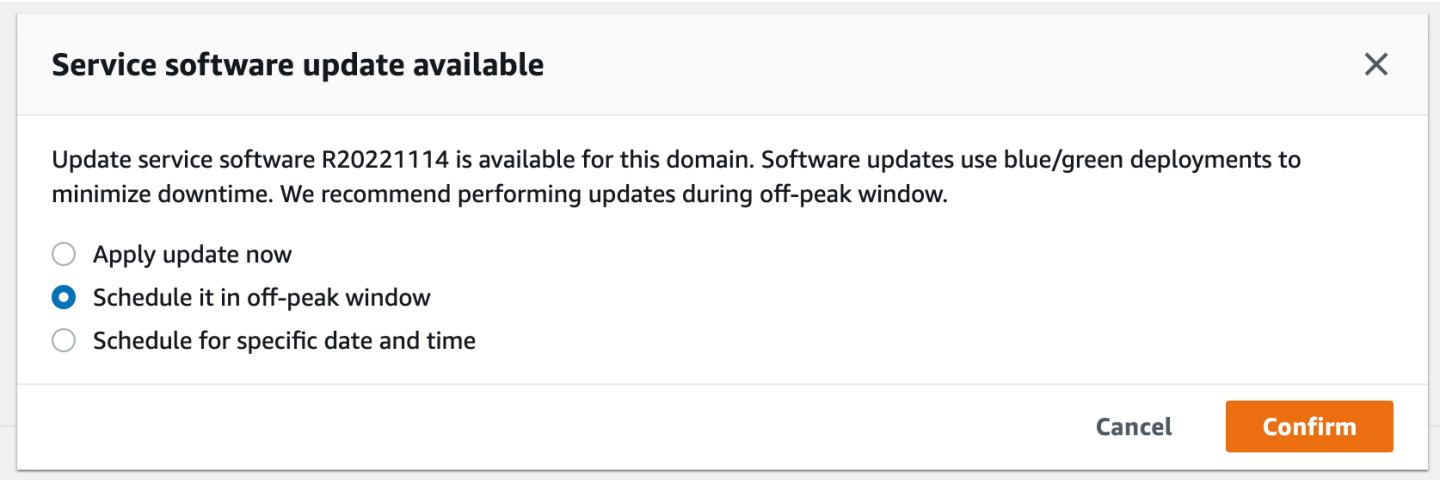
You can modify the start time for the off-peak window, but you can't modify the length of the window.

> **Note**
>
> Off-peak windows were introduced on February 16, 2023. All domains created before this date have the off-peak window disabled by default. You must manually enable and configure the off-peak window for these domains. All domains created *after* this date will have the off-peak window enabled by default. You can't disable the off-peak window for a domain after it's enabled.

Off-peak service software updates

OpenSearch Service has two broad categories of service software updates—*optional* and *required*. Both types require blue/green deployments. Optional updates aren't enforced on your domains, while required updates are automatically installed if you take no action before the specified deadline (typically two weeks from availability). For more information, see the section called "Optional versus required updates".

When you initiate an *optional* update, you have the choice to apply the update immediately, schedule it for a subsequent off-peak window, or specify a custom date and time to apply it.



For *required* updates, OpenSearch Service automatically schedules a date and time during off-peak hours to perform the update. You receive a notification three days before the scheduled update, and you can choose to reschedule it for a later date and time within the required deployment period. For instructions, see the section called "Rescheduling actions".

Off-peak Auto-Tune optimizations

Previously, Auto-Tune used maintenance windows to schedule changes that required a blue/green deployment. Domains that already had Auto-Tune and maintenance windows enabled prior to the introduction of off-peak windows will continue to use maintenance windows for these updates, unless you migrate them to use the off-peak window.

We recommend that you migrate your domains to use the off-peak window, as it's used to schedule other activities on the domain such as service softwate updates. For instructions, see the section called “Migrating from Auto-Tune maintenance windows”. You can't revert back to using maintenance windows after you migrate your domain to the off-peak window.

All domains created after February 16, 2023 will use the off-peak window, rather than legacy maintenance windows, to schedule blue/green deployments. You can't disable the off-peak window for a domain. For a list of Auto-Tune optimizations that require blue/green deployments, see the section called “Types of changes”.

Enabling the off-peak window

Any domains created before February 16, 2023 (when off-peak windows were introduced) have the feature disabled by default. You must manually enable it for these domains. You can't disable the off-peak window after it's enabled.

Console

To enable the off-peak window for a domain

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Select the name of the domain to open its configuration.
3. Navigate to the **Off-peak window** tab and choose **Edit**.
4. Specify a custom start time in Coordinated Universal Time (UTC). For example, to configure a start time of 11:30 P.M. in the US West (Oregon) Region, specify **07:30**.
5. Choose **Save changes**.

CLI

To modify the off-peak window using the AWS CLI, send an UpdateDomainConfig request:

```
aws opensearch update-domain-config \
  --domain-name my-domain \
  --off-peak-window-options 'Enabled=true,
 OffPeakWindow={WindowStartTime={Hours=02,Minutes=00}}'
```

If you don't specify a custom window start time, it defaults to 00:00 UTC.

Configuring a custom off-peak window

You specify a custom off-peak window for your domain in Coordinated Universal Time (UTC). For example, if your want the off-peak window to start at 11:00 P.M. for a domain in the US East (N. Virginia) Region, you'd specify 04:00 UTC.

Console

To modify the off-peak window for a domain

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Select the name of the domain to open its configuration.
3. Navigate to the **Off-peak window** tab. You can view the configured off-peak window and a list of upcoming scheduled actions for the domain.
4. Choose **Edit** and specify a new start time in UTC. For example, to configure a start time of 9:00 PM in the US East (N. Virginia) Region, specify **02:00 UCT**.
5. Choose **Save changes**.

CLI

To configure a custom off-peak window using the AWS CLI, send an UpdateDomainConfig request and specify the hour and minute in 24-hour time format.

For example, the following request changes the window start time to 2:00 A.M. UTC:

```
aws opensearch update-domain-config \
  --domain-name my-domain \
  --off-peak-window-options 'OffPeakWindow={WindowStartTime={Hours=02,Minutes=00}}'
```

If you don't specify a window start time, it defaults to 10:00 P.M. local time for the AWS Region that the domain is created in.

Viewing scheduled actions

You can view all actions that are currently scheduled, in progress, or pending for each of your domains. Actions can have a severity of `HIGH`, `MEDIUM`, and `LOW`.

Actions can have the following statuses:

- `Pending update` – The action is in the queue to be processed.
- `In progress` – The action is currently in progress.
- `Failed` – The action failed to complete.
- `Completed` – The action has completed successfully.
- `Not eligible` – Only for service software updates. The update can't proceed because the cluster is in an unhealthy state.
- `Eligible` – Only for service software updates. The domain is eligible for an update.

Console

The OpenSearch Service console displays all scheduled actions within the domain configuration, along with each action's severity and current status.

To view scheduled actions for a domain

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Select the name of the domain to open its configuration.
3. Navigate to the **Off-peak window** tab.
4. Under **Scheduled actions**, view all actions that are currently scheduled, in progress, or pending for the domain.

CLI

To view scheduled actions using the AWS CLI, send a ListScheduledActions request:

```
aws opensearch list-scheduled-actions \
  --domain-name my-domain
```

Response:

```
{
    "ScheduledActions": [
        {
            "Cancellable": true,
            "Description": "The Deployment type is : BLUE_GREEN.",
            "ID": "R20220721-P13",
            "Mandatory": false,
            "Severity": "HIGH",
            "ScheduledBy": "CUSTOMER",
            "ScheduledTime": 1.673871601E9,
            "Status": "PENDING_UPDATE",
            "Type": "SERVICE_SOFTWARE_UPDATE",
        },
        {
            "Cancellable": true,
            "Description": "Amazon Opensearch will adjust the young generation JVM
 arguments on your domain to improve performance",
            "ID": "Auto-Tune",
            "Mandatory": true,
            "Severity": "MEDIUM",
            "ScheduledBy": "SYSTEM",
            "ScheduledTime": 1.673871601E9,
            "Status": "PENDING_UPDATE",
            "Type": "JVM_HEAP_SIZE_TUNING",
        }
    ]
}
```


Rescheduling actions

OpenSearch Service notifies you of scheduled service software updates and Auto-Tune optimizations. You can choose to apply the change immediately, or reschedule it for a later date and time.

> **Note**
>
> OpenSearch Service can schedule the action within an hour of the time you select. For exmple, if you choose to apply an update at 5 P.M., it can be applied between 5 and 6 P.M.

Console

To reschedule an action

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Select the name of the domain to open its configuration.
3. Navigate to the **Off-peak window** tab.
4. Under **Scheduled actions**, select the action and choose **Reschedule**.
5. Choose one of the following options:

   - **Apply update now** - Immediately schedules the action to happen in the current hour *if there's capacity available*. If capacity isn't available, we provide other available time slots to choose from.
   - **Schedule it in off-peak window** - Marks the action to be picked up during an upcoming off-peak window. There's no guarantee that the change will be implemented during the immediate next window. Depending on capacity, it might happen in subsequent days.
   - **Reschedule this update** - Lets you specify a custom date and time to apply the change. If the time that you specify is unavailable for capacity reasons, you can select a different time slot.
   - **Cancel scheduled update** - Cancels the update. This option is only available for optional service software updates. It's not available for Auto-Tune actions or mandatory software updates.
6. Choose **Save changes**.

CLI

To reschedule an action using the AWS CLI, send an UpdateScheduledAction request. To retrieve the action ID, send a `ListScheduledActions` request.

The following request reschedules a service software update for a specific date and time:

```
aws opensearch update-scheduled-action \
  --domain-name my-domain \
  --action-id R20220721-P13 \
  --action-type "SERVICE_SOFTWARE_UPDATE" \
  --desired-start-time 1677348395000 \
  --schedule-at TIMESTAMP
```

Response:

```
{
   "ScheduledAction": {
      "Cancellable": true,
      "Description": "Cluster status is updated.",
      "Id": "R20220721-P13",
      "Mandatory": false,
      "ScheduledBy": "CUSTOMER",
      "ScheduledTime": 1677348395000,
      "Severity": "HIGH",
      "Status": "PENDING_UPDATE",
      "Type": "SERVICE_SOFTWARE_UPDATE"
   }
}
```

If the request fails with a `SlotNotAvailableException`, it means that the time you specified isn't available for capacity reasons, and you must specify a different time. OpenSearch Service provides alternate available slot suggestions in the response.

Migrating from Auto-Tune maintenance windows

If a domain was created before February 16, 2023, it could use maintenance windows to schedule Auto-Tune optimizations that require a blue/green deployment. You can migrate your existing Auto-Tune domains to use the off-peak window instead.

> **Note**
>
> You can't revert back to using maintenance windows after you migrate your domain to use off-peak windows.

Console

To migrate a domain to use the off-peak window

1. Within the Amazon OpenSearch Service console, select the name of the domain to open its configuration.
2. Go to the **Auto-Tune** tab and choose **Edit**.
3. Select **Migrate to off-peak window**.

4. For **Start time (UTC)**, provide a daily start time for the off-peak window in Universal Coordinated Time (UTC).
5. Choose **Save changes**.

CLI

To migrate from a Auto-Tune maintenance window to the off-peak window using the AWS CLI, send an UpdateDomainConfig request:

```
aws opensearch update-domain-config \
  --domain-name my-domain \
  --auto-tune-options
 DesiredState=ENABLED,UseOffPeakWindow=true,MaintenanceSchedules=[]
```

The off-peak window must be turned on in order for you to migrate a domain from the Auto-Tune maintenance window to the off-peak window. You can enable the off-peak window in a separate request or in the same request. For instructions, see the section called “Enabling the off-peak window”.

Notifications in Amazon OpenSearch Service

Notifications in Amazon OpenSearch Service contain important information about the performance and health of your domains. OpenSearch Service notifies you about service software updates, Auto-Tune enhancements, cluster health events, and domain errors. Notifications are available for all versions of OpenSearch and Elasticsearch OSS.

You can view notifications in the **Notifications** panel of the OpenSearch Service console. All notifications for OpenSearch Service are also surfaced in Amazon EventBridge. For a full list of notifications and sample events, see the section called “Monitoring events”.

Getting started with notifications

Notifications are enabled automatically when you create a domain. Go to the **Notifications** panel of the OpenSearch Service console to monitor and acknowledge notifications. Each notification includes information such as the time it was posted, the domain it relates to, a severity and status level, and a brief explanation. You can view historical notifications for up to 90 days in the console.

After accessing the **Notifications** panel or acknowledging a notification, you might receive an error message about not having permissions to perform `es:ListNotifications` or

`es:UpdateNotificationStatus`. To resolve this problem, give your user or role the following permissions in IAM:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "es:UpdateNotificationStatus",
      "es:ListNotifications"
    ],
    "Resource": "arn:aws:es:*:123456789012:domain/*"
  }]
}
```

The IAM console throws an error ("IAM does not recognize one or more actions.") that you can safely ignore. You can also restrict the `es:UpdateNotificationStatus` action to certain domains. To learn more, see the section called “Policy element reference”.

Notification severities

Notifications in OpenSearch Service can be *informational*, which relate to any action you've already taken or the operations of your domain, or *actionable*, which require you to take specific actions such as applying a mandatory security patch. Each notification has a severity associated with it, which can be `Informational`, `Low`, `Medium`, `High`, or `Critical`. The following table summarizes each severity:

Severity	Description	Examples
`Informational`	Information related to the operation of your domain.	• Service software update available • Auto-Tune started
`Low`	A recommended action, but has no adverse impact on domain availability or performance if no action is taken.	• Auto-Tune cancelled • High shard count warning
`Medium`	There might be an impact if the recommended action	• Service software update failed

Severity	Description	Examples
	is not taken, but comes with an extended time window for the action to be taken.	• Shard count limit exceeded
`High`	Urgent action is required to avoid adverse impact.	• Service software update required • KMS key inaccessible
`Critical`	Immediate action is required to avoid adverse impact, or to recover from it.	None currently available

Sample EventBridge event

The following example shows an OpenSearch Service notification event sent to Amazon EventBridge. The notification has a severity of `Informational` because the update is optional:

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Amazon OpenSearch Service Software Update Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Service Software Update",
    "status": "Available",
    "severity": "Informational",
    "description": "Service software update [R20200330-p1] available."
  }
}
```

Configuring a multi-AZ domain in Amazon OpenSearch Service

To prevent data loss and minimize Amazon OpenSearch Service cluster downtime in the event of a service disruption, you can distribute nodes across two or three *Availability Zones* in the same Region, a configuration known as Multi-AZ. Availability Zones are isolated locations within each AWS Region.

For domains that run production workloads, we recommend the Multi-AZ with Standby deployment option, which creates the following configuration:

- The domain deployed across three zones.
- Current-generation instance types for dedicated master nodes and data nodes.
- Three dedicated master nodes and three (or a multiple of three) data nodes.
- At least two replicas for each index in your domain, or a multiple of three copies of data (including both primary nodes and replicas).

The rest of this section provides explanations for and context around these configurations.

Multi-AZ with Standby

Multi-AZ with Standby is a deployment option for Amazon OpenSearch Service domains that offers 99.99% availability, consistent performance for production workloads, and simplified domain configuration and management. When you use Multi-AZ with Standby, domains are resilient to infrastructure failures, with no impact to performance or availability. This deployment option achieves this standard by mandating a number of best practices, such as a specified data node count, master node count, instance type, replica count, software update settings, and Auto-Tune turned on.

When you use Multi-AZ with Standby, OpenSearch Service creates a domain across three Availability Zones, with each zone containing a complete copy of data and with the data equally distributed in each of the zones. Your domain reserves nodes in one of these zones as standby, which means that they don't serve search requests. When OpenSearch Service detects a failure in the underlying infrastructure, it automatically activates the standby nodes in less than a minute. The domain continues to serve indexing and search requests, and any impact is limited to the time it takes to perform the failover. There is no redistribution of data or resources, which results in unaffected cluster performance and no risk of degraded availability. Multi-AZ with Standby is available at no extra cost.

You have two options to create a domain with standby on the AWS Management Console. First, you can create a domain with the **Easy create** creation method, and OpenSearch Service will automatically use a predetermined configuration, which includes the following:

- Three Availability Zones, with one acting as a standby
- Three dedicated master node and data nodes
- Auto-Tune enabled on the domain
- GP3 storage for the data nodes

You can also choose the **Standard create** creation method and select **Domain with standby** as your deployment option. This allows you to customize your domain while still mandating key features of standby, such as three zones and three master nodes. We recommend choosing a data node count that's a multiple of three (the number of Availability Zones).

Once you've created your domain, you can navigate to the domain details pages and, in the **Cluster configuration** tab, confirm that *3-AZ with standby* appears under Availability Zone(s).

If you have problems migrating an existing domain to Multi-AZ with Standby, see Error migrating to Multi-AZ with Standby in the troubleshooting guide.

Limitations

When you set up a domain with Multi-AZ with Standby, consider the following limitations:

- The total number of shards on a node can't exceed 1000, the total number of shards on a cluster can't exceed 75000, and the size of a single shard can't exceed 65 GB.
- Multi-AZ with Standby only works with the `m5`, `c5`, `r5`, `r6g`, `r7g`, `c6g`, `m6g`, `r6gd` and `i3` instance types. For more information on supported instances, see Supported instance types.
- You can only use Provisioned IOPs SSD, General Purpose SSD (GP3), or instance-backed storage with standby.
- If you enable UltraWarm on a Multi-AZ with Standby domain, the number of warm nodes must be a multiple of the number of Availability Zones being used.

Multi-AZ without Standby

OpenSearch Service still supports multi-AZ without Standby, which offers 99.9% availability. Nodes are distributed across Availability Zone(s), and availability depends on the number of Availability

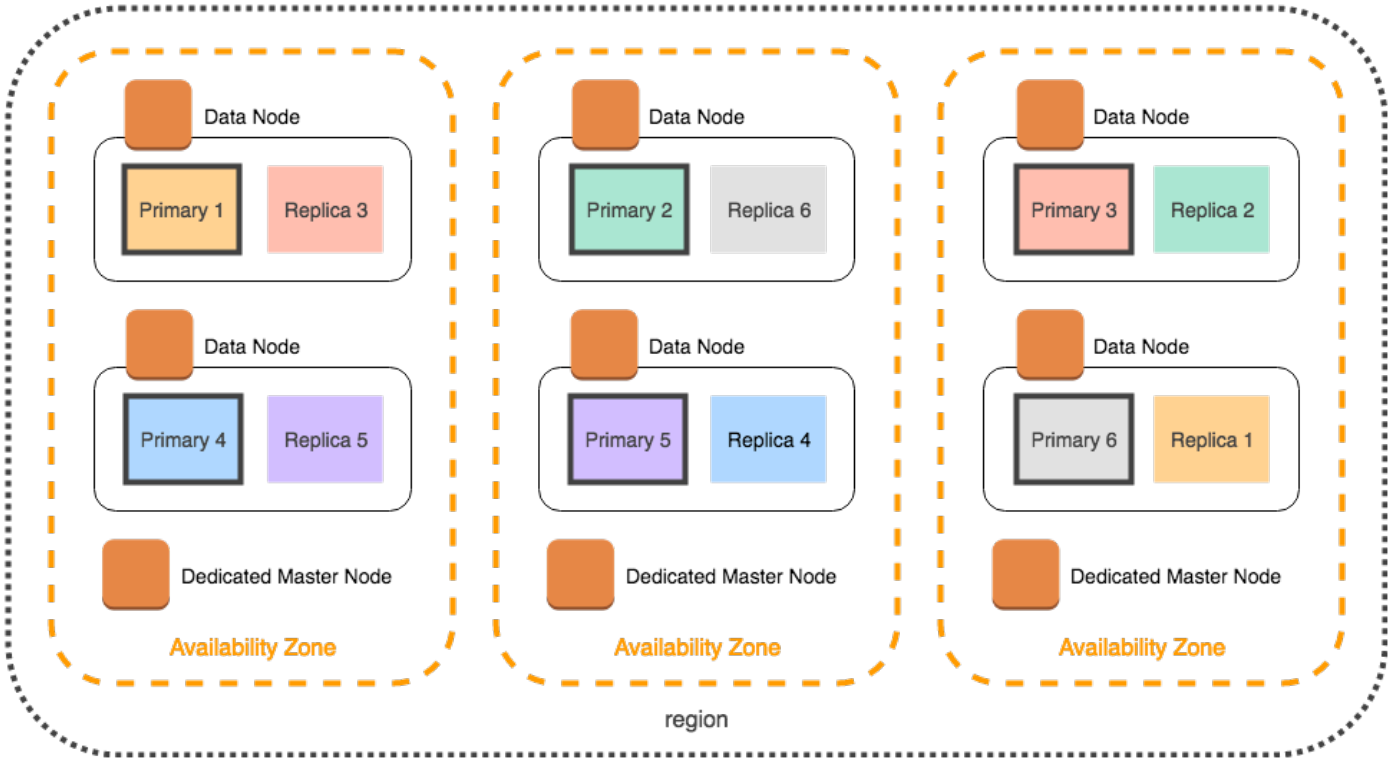
Zones and copies of data. Whereas with standby you have to configure your domain with best practices, without standby you can choose your own number of Availability Zones, nodes, and replicas. We don't recommend this option unless you have existing workflows that would be disrupted by creating domains with standby.

If you choose this option, we still recommend that you select three Availability Zones in order to remain resilient to node, disk, and single-AZ failures. When a failure occurs, the cluster redistributes data across the remaining resources to maintain availability and redundancy. This data movement increases resource usage on the cluster, and can have an impact on the performance. If the cluster isn't sized properly, it can experience degraded availability, which largely defeats the purpose of multi-AZ.

The only way to configure a domain without standby on the AWS Management Console is to choose the **Standard create** creation method, and select **Domain without standby** as your deployment option.

Shard distribution

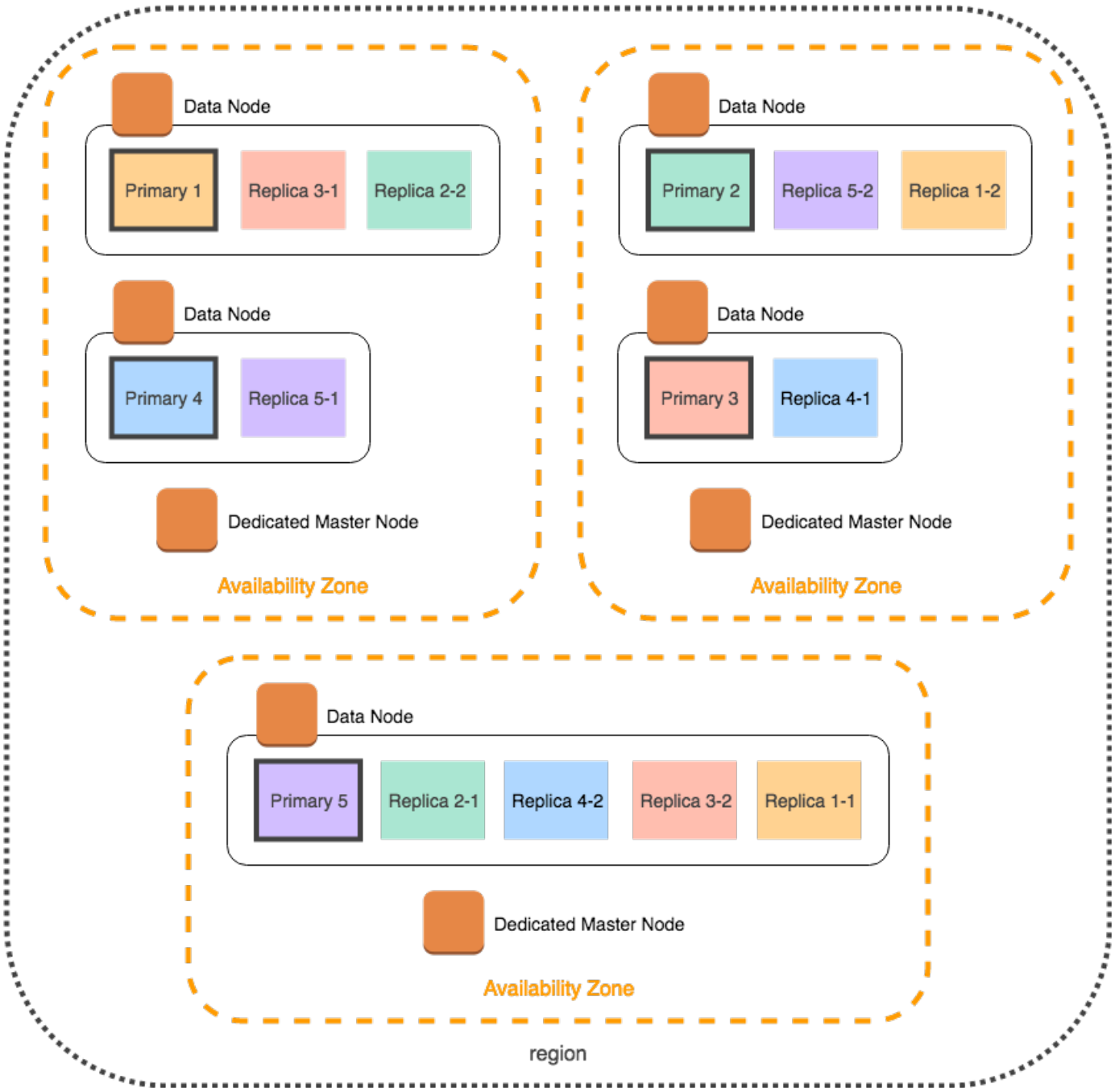
If you enable multi-AZ without Standby, you should create at least one replica for each index in your cluster. Without replicas, OpenSearch Service can't distribute copies of your data to other Availability Zones. Fortunately, the default configuration for any index is a replica count of 1. As the following diagram shows, OpenSearch Service makes a best effort to distribute primary shards and their corresponding replica shards to different zones.



In addition to distributing shards by Availability Zone, OpenSearch Service distributes them by node. Still, certain domain configurations can result in imbalanced shard counts. Consider the following domain:

- 5 data nodes
- 5 primary shards
- 2 replicas
- 3 Availability Zones

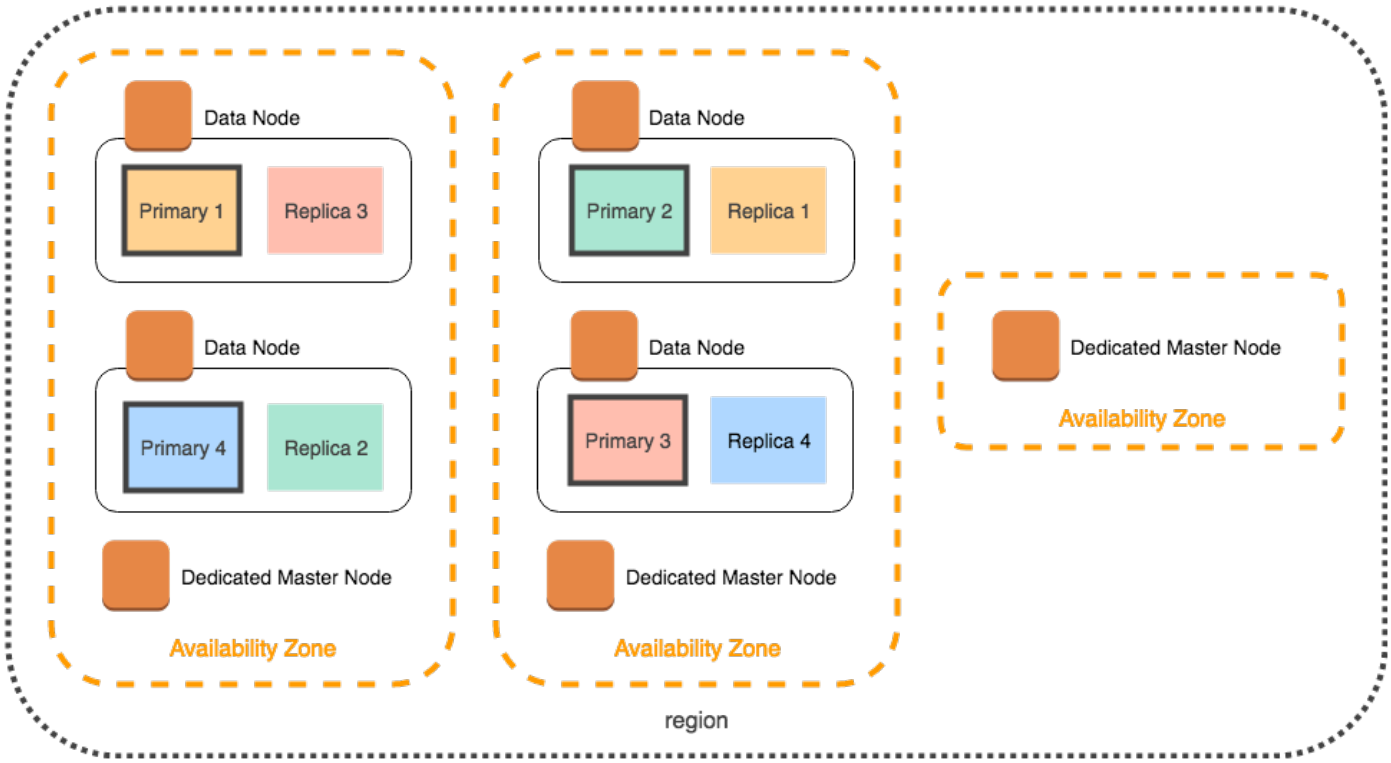
In this situation, OpenSearch Service has to overload one node in order to distribute the primary and replica shards across the zones, as shown in the following diagram.



To avoid these kinds of situations, which can strain individual nodes and hurt performance, we recommend that you choose multi-AZ with Standby, or choose an instance count that is a multiple of three when you plan to have two or more replicas per index.

Dedicated master node distribution

Even if you select two Availability Zones when configuring your domain, OpenSearch Service automatically distributes dedicated master nodes across three Availability Zones. This distribution helps prevent cluster downtime if a zone experiences a service disruption. If you use the recommended three dedicated master nodes and one Availability Zone goes down, your cluster still has a quorum (2) of dedicated master nodes and can elect a new master. The following diagram demonstrates this configuration.



If you choose an older-generation instance type that is not available in three Availability Zones, the following scenarios apply:

- If you chose three Availability Zones for the domain, OpenSearch Service throws an error. Choose a different instance type, and try again.
- If you chose two Availability Zones for the domain, OpenSearch Service distributes the dedicated master nodes across two zones.

Availability zone disruptions

Availability Zone disruptions are rare, but do occur. The following table lists different Multi-AZ configurations and behaviors during a disruption. The last row in the table applies to Multi-AZ with Standby, while all other rows have configurations that only apply to Multi-AZ without Standby.

Number of Availability Zones in a region	Number of Availability Zones you chose	Number of dedicated master nodes	Behavior if one Availability Zone experiences a disruption
2 or more	2	0	Downtime. Your cluster loses half of its data nodes and must replace at least one in the remaining Availability Zone before it can elect a master.
2	2	3	50/50 chance of downtime. OpenSearch Service distributes two dedicated master nodes into one Availability Zone and one into the other: • If the Availability Zone with one dedicated master node experiences a disruption, the two dedicated master nodes in the remaining Availability Zone can elect a master. • If the Availability Zone with two dedicated master nodes experiences a disruption, the cluster is unavailable until the remaining Availability Zone recovers.
3 or more	2	3	No downtime. OpenSearch Service automatic ally distributes the dedicated master nodes across three Availability Zones, so the remaining two dedicated master nodes can elect a master.

Number of Availability Zones in a region	Number of Availability Zones you chose	Number of dedicated master nodes	Behavior if one Availability Zone experiences a disruption
3 or more	3	0	No downtime. Roughly two-thirds of your data nodes are still available to elect a master.
3 or more	3	3	No downtime. The remaining two dedicated master nodes can elect a master.

In *all* configurations, regardless of the cause, node failures can cause the cluster's remaining data nodes to experience a period of increased load while OpenSearch Service automatically configures new nodes to replace the now-missing ones.

For example, in the event of an Availability Zone disruption in a three-zone configuration, two-thirds as many data nodes have to process just as many requests to the cluster. As they process these requests, the remaining nodes are also replicating shards onto new nodes as they come online, which can further impact performance. If availability is critical to your workload, consider adding resources to your cluster to alleviate this concern.

> **Note**
>
> OpenSearch Service manages Multi-AZ domains transparently, so you can't manually simulate Availability Zone disruptions.

Launching your Amazon OpenSearch Service domains within a VPC

You can launch AWS resources, such as Amazon OpenSearch Service domains, into a *virtual private cloud* (VPC). A VPC is a virtual network that's dedicated to your AWS account. It's logically isolated from other virtual networks in the AWS Cloud. Placing an OpenSearch Service domain within a VPC enables 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.

> **Note**
>
> If you place your OpenSearch Service domain within a VPC, your computer must be able to connect to the VPC. This connection often takes the form of a VPN, transit gateway, managed network, or proxy server. You can't directly access your domains from outside the VPC.

VPC versus public domains

The following are some of the ways VPC domains differ from public domains. Each difference is described later in more detail.

- Because of their logical isolation, domains that reside within a VPC have an extra layer of security compared to domains that use public endpoints.
- While public domains are accessible from any internet-connected device, VPC domains require some form of VPN or proxy.
- Compared to public domains, VPC domains display less information in the console. Specifically, the **Cluster health** tab does not include shard information, and the **Indices** tab isn't present.
- The domain endpoints take different forms (`https://search-domain-name` vs. `https://vpc-domain-name`).
- You can't apply IP-based access policies to domains that reside within a VPC because security groups already enforce IP-based access policies.

Limitations

Operating an OpenSearch Service domain within a VPC has the following limitations:

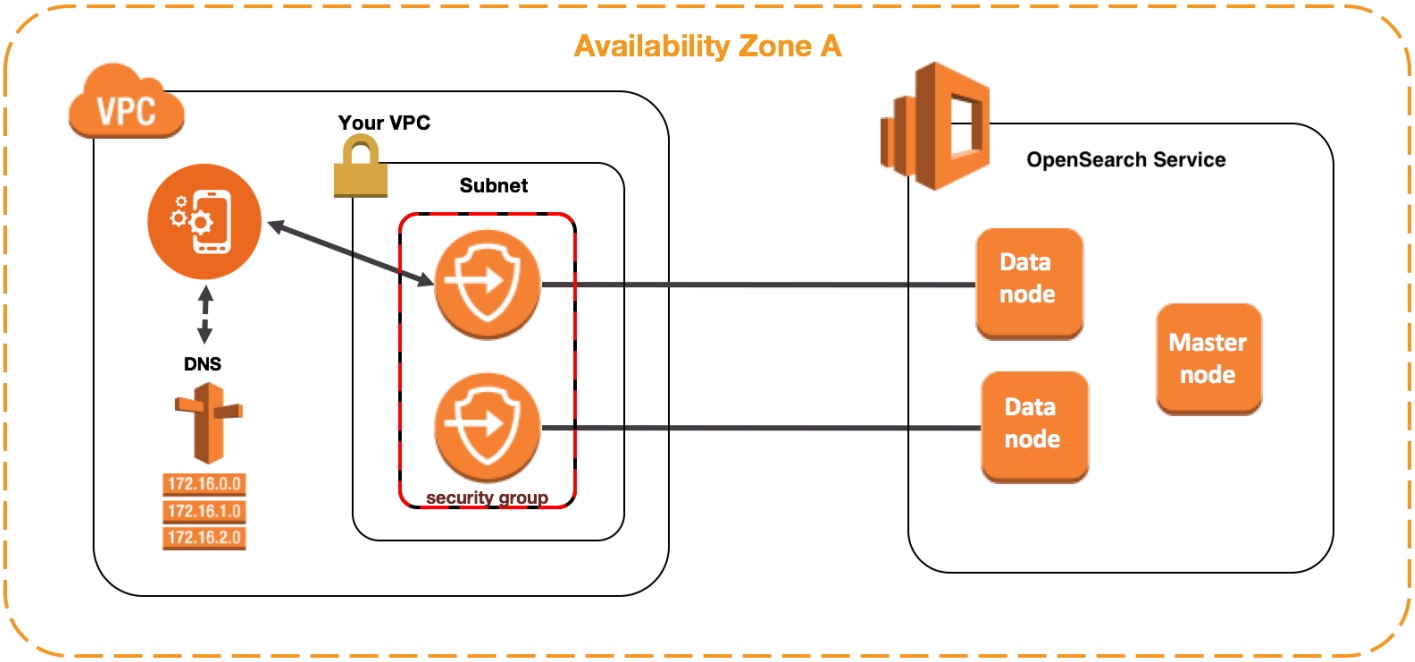
- If you launch a new domain within a VPC, you can't later switch it to use a public endpoint. The reverse is also true: If you create a domain with a public endpoint, you can't later place it within a VPC. Instead, you must create a new domain and migrate your data.
- You can either launch your domain within a VPC or use a public endpoint, but you can't do both. You must choose one or the other when you create your domain.
- You can't launch your domain within a VPC that uses dedicated tenancy. You must use a VPC with tenancy set to **Default**.

- After you place a domain within a VPC, you can't move it to a different VPC, but you can change the subnets and security group settings.
- To access the default installation of OpenSearch Dashboards for a domain that resides within a VPC, users must have access to the VPC. This process varies by network configuration, but likely involves connecting to a VPN or managed network or using a proxy server or transit gateway. To learn more, see the section called “About access policies on VPC domains”, the Amazon VPC User Guide, and the section called “Controlling access to Dashboards ”.

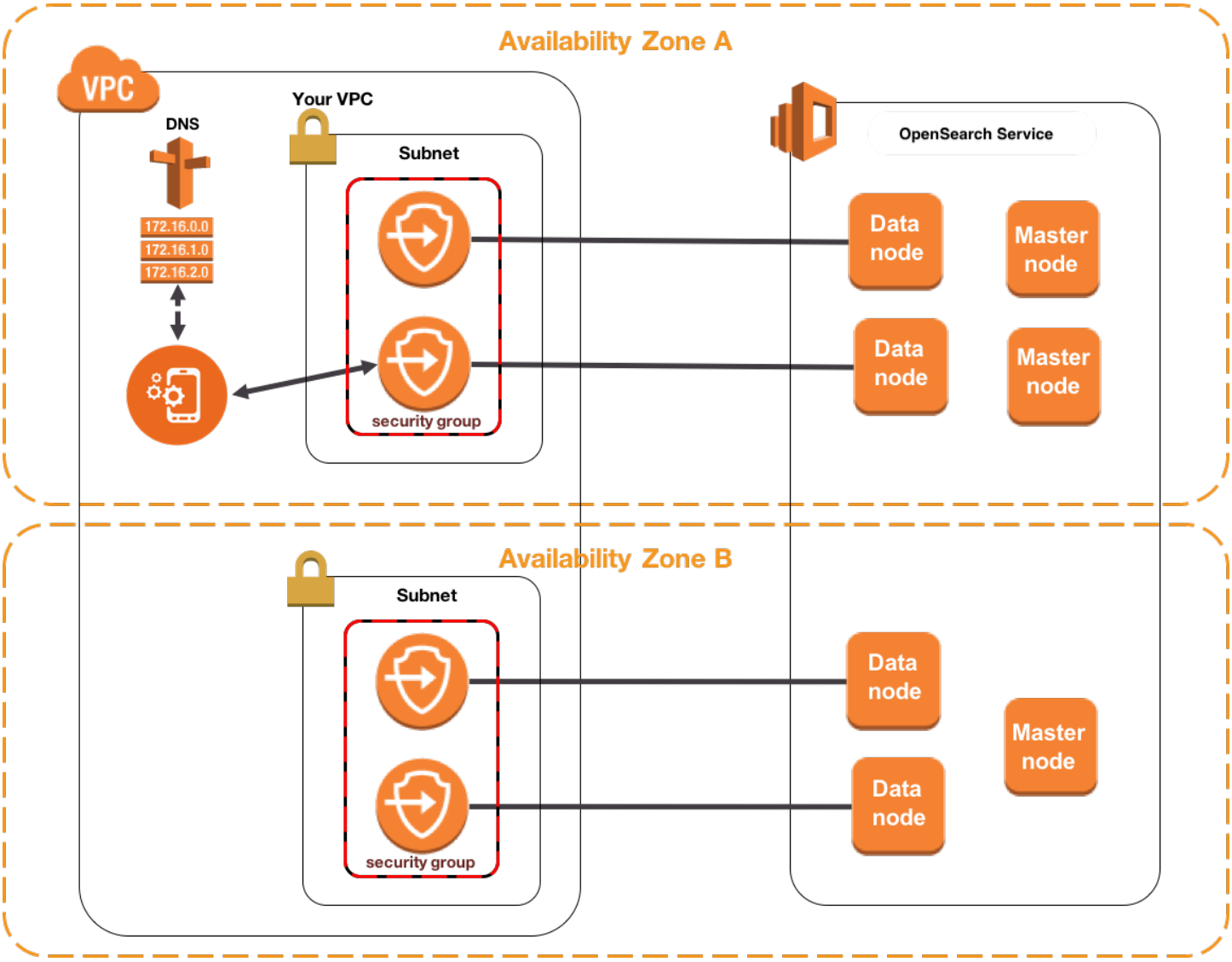
Architecture

To support VPCs, OpenSearch Service places an endpoint into one, two, or three subnets of your VPC. If you enable multiple Availability Zones for your domain, each subnet must be in a different Availability Zone in the same region. If you only use one Availability Zone, OpenSearch Service places an endpoint into only one subnet.

The following illustration shows the VPC architecture for one Availability Zone:



The following illustration shows the VPC architecture for two Availability Zones:



OpenSearch Service also places an *elastic network interface* (ENI) in the VPC for each of your data nodes. OpenSearch Service assigns each ENI a private IP address from the IPv4 address range of your subnet. The service also assigns a public DNS hostname (which is the domain endpoint) for the IP addresses. You must use a public DNS service to resolve the endpoint (which is a DNS hostname) to the appropriate IP addresses for the data nodes:

- If your VPC uses the Amazon-provided DNS server by setting the `enableDnsSupport` option to `true` (the default value), resolution for the OpenSearch Service endpoint will succeed.
- If your VPC uses a private DNS server and the server can reach the public authoritative DNS servers to resolve DNS hostnames, resolution for the OpenSearch Service endpoint will also succeed.

Because the IP addresses might change, you should resolve the domain endpoint periodically so that you can always access the correct data nodes. We recommend that you set the DNS resolution interval to one minute. If you're using a client, you should also ensure that the DNS cache in the client is cleared.

Migrating from public access to VPC access

When you create a domain, you specify whether it should have a public endpoint or reside within a VPC. Once created, you cannot switch from one to the other. Instead, you must create a new domain and either manually reindex or migrate your data. Snapshots offer a convenient means of migrating data. For information about taking and restoring snapshots, see the section called "Creating index snapshots".

About access policies on VPC domains

Placing your OpenSearch Service domain within a VPC provides an inherent, strong layer of security. When you create a domain with public access, the endpoint takes the following form:

```
https://search-domain-name-identifier.region.es.amazonaws.com
```

As the "public" label suggests, this endpoint is accessible from any internet-connected device, though you can (and should) control access to it. If you access the endpoint in a web browser, you might receive a `Not Authorized` message, but the request reaches the domain.

When you create a domain with VPC access, the endpoint *looks* similar to a public endpoint:

```
https://vpc-domain-name-identifier.region.es.amazonaws.com
```

If you try to access the endpoint in a web browser, however, you might find that the request times out. To perform even basic `GET` requests, your computer must be able to connect to the VPC. This connection often takes the form of a VPN, transit gateway, managed network, or proxy server. For details on the various forms it can take, see Examples for VPC in the *Amazon VPC User Guide*. For a development-focused example, see the section called "Testing VPC domains".

In addition to this connectivity requirement, VPCs let you manage access to the domain through security groups. For many use cases, this combination of security features is sufficient, and you might feel comfortable applying an open access policy to the domain.

Operating with an open access policy does *not* mean that anyone on the internet can access the OpenSearch Service domain. Rather, it means that if a request reaches the OpenSearch Service

domain and the associated security groups permit it, the domain accepts the request. The only exception is if you're using fine-grained access control or an access policy that specifies IAM roles. In these situations, for the domain to accept a request, the security groups must permit it *and* it must be signed with valid credentials.

> **Note**
>
> Because security groups already enforce IP-based access policies, you can't apply IP-based access policies to OpenSearch Service domains that reside within a VPC. If you use public access, IP-based policies are still available.

Before you begin: prerequisites for VPC access

Before you can enable a connection between a VPC and your new OpenSearch Service domain, you must do the following:

- **Create a VPC**

  To create your VPC, you can use the Amazon VPC console, the AWS CLI, or one of the AWS SDKs. For more information, see Working with VPCs in the *Amazon VPC User Guide*. If you already have a VPC, you can skip this step.

- **Reserve IP addresses**

  OpenSearch Service enables the connection of a VPC to a domain by placing network interfaces in a subnet of the VPC. Each network interface is associated with an IP address. You must reserve a sufficient number of IP addresses in the subnet for the network interfaces. For more information, see Reserving IP addresses in a VPC subnet.

Testing VPC domains

The enhanced security of a VPC can make connecting to your domain and running basic tests a challenge. If you already have an OpenSearch Service VPC domain and would rather not create a VPN server, try the following process:

1. For your domain's access policy, choose **Only use fine-grained access control**. You can always update this setting after you finish testing.

2. Create an Amazon Linux Amazon EC2 instance in the same VPC, subnet, and security group as your OpenSearch Service domain.

   Because this instance is for testing purposes and needs to do very little work, choose an inexpensive instance type like `t2.micro`. Assign the instance a public IP address and either create a new key pair or choose an existing one. If you create a new key, download it to your `~/.ssh` directory.

   To learn more about creating instances, see Getting started with Amazon EC2 Linux instances.

3. Add an internet gateway to your VPC.

4. In the route table for your VPC, add a new route. For **Destination**, specify a CIDR block that contains your computer's public IP address. For **Target**, specify the internet gateway you just created.

   For example, you might specify `123.123.123.123/32` for just your computer or `123.123.123.0/24` for a range of computers.

5. For the security group, specify two inbound rules:

Type	Protocol	Port Range	Source
SSH (22)	TCP (6)	22	*your-cidr-block*
HTTPS (443)	TCP (6)	443	*your-security-group-id*

   The first rule lets you SSH into your EC2 instance. The second allows the EC2 instance to communicate with the OpenSearch Service domain over HTTPS.

6. From the terminal, run the following command:

```
ssh -i ~/.ssh/your-key.pem ec2-user@your-ec2-instance-public-ip -N -L
 9200:vpc-domain-name.region.es.amazonaws.com:443
```

   This command creates an SSH tunnel that forwards requests to https://localhost:9200 to your OpenSearch Service domain through the EC2 instance. Specifying port 9200 in the command simulates a local OpenSearch install, but use whichever port you'd like. OpenSearch Service only accepts connections over port 80 (HTTP) or 443 (HTTPS).

The command provides no feedback and runs indefinitely. To stop it, press `Ctrl + C`.

7. Navigate to https://localhost:9200/_dashboards/ in your web browser. You might need to acknowledge a security exception.

   Alternately, you can send requests to https://localhost:9200 using curl, Postman, or your favorite programming language.

> **Tip**
>
> If you encounter curl errors due to a certificate mismatch, try the `--insecure` flag.

Reserving IP addresses in a VPC subnet

OpenSearch Service connects a domain to a VPC by placing network interfaces in a subnet of the VPC (or multiple subnets of the VPC if you enable multiple Availability Zones). Each network interface is associated with an IP address. Before you create your OpenSearch Service domain, you must have a sufficient number of IP addresses available in each subnet to accommodate the network interfaces.

Here's the basic formula: The number of IP addresses that OpenSearch Service reserves in each subnet is three times the number of data nodes, divided by the number of Availability Zones.

Examples

- If a domain has nine data nodes across three Availability Zones, the IP count per subnet is 9 * 3 / 3 = 9.
- If a domain has eight data nodes across two Availability Zones, the IP count per subnet is 8 * 3 / 2 = 12.
- If a domain has six data nodes in one Availability Zone, the IP count per subnet is 6 * 3 / 1 = 18.

When you create the domain, OpenSearch Service reserves the IP addresses, uses some for the domain, and reserves the rest for blue/green deployments. You can see the network interfaces and their associated IP addresses in the **Network Interfaces** section of the Amazon EC2 console. The **Description** column shows which OpenSearch Service domain the network interface is associated with.

> **Tip**
>
> We recommend that you create dedicated subnets for the OpenSearch Service reserved IP addresses. By using dedicated subnets, you avoid overlap with other applications and services and ensure that you can reserve additional IP addresses if you need to scale your cluster in the future. To learn more, see Creating a subnet in your VPC.

You may also consider provisioning dedicated coordinator nodes to reduce the number of private IP address reservations required for your VPC domain. OpenSearch attaches an elastic network interface (ENI) to your dedicated coordinator nodes instead of your data nodes. Dedicated coordinator nodes usually represent around 10% of total data nodes. As a result, a smaller number of private IP addresses will be reserved for VPC domains.

Service-linked role for VPC access

A service-linked role is a unique type of IAM role that delegates permissions to a service so that it can create and manage resources on your behalf. OpenSearch Service requires a service-linked role to access your VPC, create the domain endpoint, and place network interfaces in a subnet of your VPC.

OpenSearch Service automatically creates the role when you use the OpenSearch Service console to create a domain within a VPC. For this automatic creation to succeed, you must have permissions for the `iam:CreateServiceLinkedRole` action. To learn more, see Service-linked role permissions in the *IAM User Guide*.

After OpenSearch Service creates the role, you can view it (`AWSServiceRoleForAmazonOpenSearchService`) using the IAM console.

For full information on this role's permissions and how to delete it, see the section called “Using service-linked roles”.

Creating index snapshots in Amazon OpenSearch Service

Snapshots in Amazon OpenSearch Service are backups of a cluster's indexes and state. *State* includes cluster settings, node information, index settings, and shard allocation.

OpenSearch Service snapshots come in the following forms:

- **Automated snapshots** are only for cluster recovery. You can use them to restore your domain in the event of red cluster status or data loss. For more information, see Restoring snapshots below. OpenSearch Service stores automated snapshots in a preconfigured Amazon S3 bucket at no additional charge.
- **Manual snapshots** are for cluster recovery *or* for moving data from one cluster to another. You have to initiate manual snapshots. These snapshots are stored in your own Amazon S3 bucket and standard S3 charges apply. If you have a snapshot from a self-managed OpenSearch cluster, you can use that snapshot to migrate to an OpenSearch Service domain. For more information, see Migrating to Amazon OpenSearch Service.

All OpenSearch Service domains take automated snapshots, but the frequency differs in the following ways:

- For domains running OpenSearch or Elasticsearch 5.3 and later, OpenSearch Service takes hourly automated snapshots and retains up to 336 of them for 14 days. Hourly snapshots are less disruptive because of their incremental nature. They also provide a more recent recovery point in case of domain problems.
- For domains running Elasticsearch 5.1 and earlier, OpenSearch Service takes daily automated snapshots during the hour you specify, retains up to 14 of them, and doesn't retain any snapshot data for more than 30 days.

If your cluster enters red status, all automated snapshots fail while the cluster status persists. If you don't correct the problem within two weeks, you can permanently lose the data in your cluster. For troubleshooting steps, see the section called “Red cluster status”.

Prerequisites

To create snapshots manually, you need to work with IAM and Amazon S3. Make sure you meet the following prerequisites before you attempt to take a snapshot:

Prerequisite	Description
S3 bucket	Create an S3 bucket to store manual snapshots for your OpenSearch Service domain. For instructions, see Create a Bucket in the *Amazon Simple Storage Service User Guide*.

Prerequisite	Description
	Remember the name of the bucket to use it in the following places: • The `Resource` statement of the IAM policy attached to your IAM role • The Python client used to register a snapshot repository (if you use this method) **Important** Do not apply an S3 Glacier lifecycle rule to this bucket. Manual snapshots don't support the S3 Glacier storage class.

Prerequisite	Description
IAM role	Create an IAM role to delegate permissions to OpenSearch Service. For instructions, see Creating an IAM role (console) in the *IAM User Guide*. The rest of this chapter refers to this role as `TheSnapshotRole` . **Attach an IAM policy** Attach the following policy to `TheSnapshotRole` to allow access to the S3 bucket: `{ "Version": "2012-10-17", "Statement": [{ "Action": [ "s3:ListBucket" ], "Effect": "Allow", "Resource": [ "arn:aws:s3::: s3-bucket-name " ] }, { "Action": [ "s3:GetObject", "s3:PutObject", "s3:DeleteObject" ], "Effect": "Allow", "Resource": [ "arn:aws:s3::: s3-bucket-name /*" ] } ] }` For instructions to attach a policy to a role, see Adding IAM Identity Permissions in the *IAM User Guide*. **Edit the trust relationship**

Prerequisite	Description
	Edit the trust relationship of `TheSnapshotRole` to specify OpenSearch Service in the `Principal` statement as shown in the following example: `{ "Version": "2012-10-17", "Statement": [{ "Sid": "", "Effect": "Allow", "Principal": { "Service": "es.amazonaws.com" }, "Action": "sts:AssumeRole" }] }` For instructions to edit the trust relationship, see Modifying a role trust policy in the *IAM User Guide*.

Prerequisite	Description

Permissions	In order to register the snapshot repository, you need to be able to pass `TheSnapshotRole` to OpenSearch Service. You also need access to the `es:ESHttpPut` action. To grant both of these permissions, attach the following policy to the IAM role whose credentials are being used to sign the request:
	
	`{`
	`  "Version": "2012-10-17",`
	`  "Statement": [`
	`  {`
	`  "Effect": "Allow",`
	`  "Action": "iam:PassRole",`
	`  "Resource": "arn:aws:iam::  123456789012 :role/TheSnapshotRole "`
	`  },`
	`  {`
	`  "Effect": "Allow",`
	`  "Action": "es:ESHttpPut",`
	`  "Resource": "arn:aws:es:  region:123456789012 :domain/domain-name /*"`
	`  }`
	`  ]`
	`}`
	
	If your user or role doesn't have `iam:PassRole` permissions to pass `TheSnapshotRole` , you might encounter the following common error when you try to register a repository in the next step:
	
	`$ python register-repo.py`
	`{"Message":"User: arn:aws:iam::  123456789012 :user/MyUserAccount is not authorized to perform: iam:PassRole on resource: arn:aws:iam:: 123456789012 :role/TheSnapshotRole "}`

Registering a manual snapshot repository

You need to register a snapshot repository with OpenSearch Service before you can take manual index snapshots. This one-time operation requires that you sign your AWS request with credentials that are allowed to access `TheSnapshotRole`, as described in the section called “Prerequisites”.

Step 1: Map the snapshot role in OpenSearch Dashboards (if using fine-grained access control)

Fine-grained access control introduces an additional step when registering a repository. Even if you use HTTP basic authentication for all other purposes, you need to map the `manage_snapshots` role to your IAM role that has `iam:PassRole` permissions to pass `TheSnapshotRole`.

1. Navigate to the OpenSearch Dashboards plugin for your OpenSearch Service domain. You can find the Dashboards endpoint on your domain dashboard on the OpenSearch Service console.
2. From the main menu choose **Security**, **Roles**, and select the **manage_snapshots** role.
3. Choose **Mapped users**, **Manage mapping**.
4. Add the ARN of the role that has permissions to pass `TheSnapshotRole`. Put role ARNs under **Backend roles**.

   ```
   arn:aws:iam::123456789123:role/role-name
   ```

5. Select **Map** and confirm the user or role shows up under **Mapped users**.

Step 2: Register a repository

The following **Snapshots** tab demonstrates how to register a snapshot directory. For options specific to encrypting a manual snapshot and registering a snapshot after migrating to a new domain, see the relevant tabs.

Snapshots

To register a snapshot repository, send a PUT request to the OpenSearch Service domain endpoint. You can use curl, the sample Python client, Postman, or some other method to send a signed request to register the snapshot repository. Note that you can't use a PUT request in the OpenSearch Dashboards console to register the repository.

The request takes the following format:

```
PUT domain-endpoint/_snapshot/my-snapshot-repo-name
{
  "type": "s3",
  "settings": {
    "bucket": "s3-bucket-name",
    "base_path": "my/snapshot/directory",
    "region": "region",
    "role_arn": "arn:aws:iam::123456789012:role/TheSnapshotRole"
  }
}
```

Note

Repository names cannot start with "cs-". Additionally, you shouldn't write to the same repository from multiple domains. Only one domain should have write access to the repository.

If your domain resides within a virtual private cloud (VPC), your computer must be connected to the VPC for the request to successfully register the snapshot repository. Accessing a VPC varies by network configuration, but likely involves connecting to a VPN or corporate network. To check that you can reach the OpenSearch Service domain, navigate to `https://your-vpc-domain.region.es.amazonaws.com` in a web browser and verify that you receive the default JSON response.

When your Amazon S3 bucket is in another AWS Region than your OpenSearch domain, add the parameter `"endpoint": "s3.amazonaws.com"` to the request.

Encrypted snapshots

You currently can't use AWS Key Management Service (KMS) keys to encrypt manual snapshots, but you can protect them using server-side encryption (SSE).

To turn on SSE with S3-managed keys for the bucket you use as a snapshot repository, add `"server_side_encryption": true` to the `"settings"` block of the PUT request. For more information, see Protecting data using server-side encryption with Amazon S3-managed encryption keys in the *Amazon Simple Storage Service User Guide*.

Alternatively, you can use AWS KMS keys for server-side encryption on the S3 bucket that you use as a snapshot repository. If you use this approach, make sure to provide `TheSnapshotRole`

permission to the AWS KMS key used to encrypt the S3 bucket. For more information, see Key policies in AWS KMS.

Domain migration

Registering a snapshot repository is a one-time operation. However, to migrate from one domain to another, you have to register the same snapshot repository on the old domain and the new domain. The repository name is arbitrary.

Consider the following guidelines when migrating to a new domain or registering the same repository with multiple domains:

- When registering the repository on the new domain, add `"readonly": true` to the `"settings"` block of the PUT request. This setting prevents you from accidentally overwriting data from the old domain. Only one domain should have write access to the repository.
- If you're migrating data to a domain in a different AWS Region, (for example, from an old domain and bucket located in us-east-2 to a new domain in us-west-2), replace `"region": "region"` with `"endpoint": "s3.amazonaws.com"` in the PUT statement and retry the request.

Using the sample Python client

The Python client is easier to automate than a simple HTTP request and has better reusability. If you choose to use this method to register a snapshot repository, save the following sample Python code as a Python file, such as `register-repo.py`. The client requires the AWS SDK for Python (Boto3), requests and requests-aws4auth packages. The client contains commented-out examples for other snapshot operations.

Update the following variables in the sample code: `host`, `region`, `path`, and `payload`.

```
import boto3
import requests
from requests_aws4auth import AWS4Auth

host = '' # domain endpoint
region = '' # e.g. us-west-1
service = 'es'
credentials = boto3.Session().get_credentials()
awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region, service,
 session_token=credentials.token)
```

```
# Register repository

path = '/_snapshot/my-snapshot-repo-name' # the OpenSearch API endpoint
url = host + path

payload = {
  "type": "s3",
  "settings": {
    "bucket": "s3-bucket-name",
    "base_path": "my/snapshot/directory",
    "region": "us-west-1",
    "role_arn": "arn:aws:iam::123456789012:role/snapshot-role"
  }
}

headers = {"Content-Type": "application/json"}

r = requests.put(url, auth=awsauth, json=payload, headers=headers)

print(r.status_code)
print(r.text)

# # Take snapshot
#
# path = '/_snapshot/my-snapshot-repo-name/my-snapshot'
# url = host + path
#
# r = requests.put(url, auth=awsauth)
#
# print(r.text)
#
# # Delete index
#
# path = 'my-index'
# url = host + path
#
# r = requests.delete(url, auth=awsauth)
#
# print(r.text)
#
# # Restore snapshot (all indexes except Dashboards and fine-grained access control)
#
# path = '/_snapshot/my-snapshot-repo-name/my-snapshot/_restore'
```

```
# url = host + path
#
# payload = {
#   "indices": "-.kibana*,-.opendistro_security,-.opendistro-*",
#   "include_global_state": False
# }
#
# headers = {"Content-Type": "application/json"}
#
# r = requests.post(url, auth=awsauth, json=payload, headers=headers)
#
# print(r.text)
#
# # Restore snapshot (one index)
#
# path = '/_snapshot/my-snapshot-repo-name/my-snapshot/_restore'
# url = host + path
#
# payload = {"indices": "my-index"}
#
# headers = {"Content-Type": "application/json"}
#
# r = requests.post(url, auth=awsauth, json=payload, headers=headers)
#
# print(r.text)
```

Taking manual snapshots

Snapshots are not instantaneous. They take time to complete and don't represent perfect point-in-time views of the cluster. While a snapshot is in progress, you can still index documents and make other requests to the cluster, but new documents and updates to existing documents generally aren't included in the snapshot. The snapshot includes primary shards as they existed when OpenSearch initiated the snapshot. Depending on the size of your snapshot thread pool, different shards might be included in the snapshot at slightly different times. For snapshot best practices, see the section called “Improve snapshot performance”.

Snapshot storage and performance

OpenSearch snapshots are incremental, meaning they only store data that changed since the last successful snapshot. This incremental nature means the difference in disk usage between frequent and infrequent snapshots is often minimal. In other words, taking hourly snapshots for a week (for a total of 168 snapshots) might not use much more disk space than taking a single snapshot

at the end of the week. Also, the more frequently you take snapshots, the less time they take to complete. For example, daily snapshots can take 20-30 minutes to complete, whereas hourly snapshots might complete within a few minutes. Some OpenSearch users take snapshots as often as every half hour.

Take a snapshot

You specify the following information when you create a snapshot:

- The name of your snapshot repository
- A name for the snapshot

The examples in this chapter use curl, a common HTTP client, for convenience and brevity. To pass a username and password to your curl request, see the Getting started tutorial.

If your access policies specify users or roles, you must sign your snapshot requests. For curl, you can use the `--aws-sigv4` option with version 7.75.0 or later. You can also use the commented-out examples in the sample Python client to make signed HTTP requests to the same endpoints that the curl commands use.

To take a manual snapshot, perform the following steps:

1. You can't take a snapshot if one is currently in progress. To check, run the following command:

   ```
   curl -XGET 'domain-endpoint/_snapshot/_status'
   ```

2. Run the following command to take a manual snapshot:

   ```
   curl -XPUT 'domain-endpoint/_snapshot/repository-name/snapshot-name'
   ```

   To include or exclude certain indexes and specify other settings, add a request body. For the request structure, see Take snapshots in the OpenSearch documentation.

> **Note**
>
> The time required to take a snapshot increases with the size of the OpenSearch Service domain. Long-running snapshot operations sometimes encounter the following error: `504 GATEWAY_TIMEOUT`. You can typically ignore these errors and wait for the operation to

complete successfully. Run the following command to verify the state of all snapshots of your domain:

```
curl -XGET 'domain-endpoint/_snapshot/repository-name/_all?pretty'
```

Restoring snapshots

Before you restore a snapshot, make sure that the destination domain does not use Multi-AZ with Standby. Having standby enabled causes the restore operation to fail.

Warning

If you use index aliases, you should either cease write requests to an alias or switch the alias to another index prior to deleting its index. Halting write requests helps avoid the following scenario:

1. You delete an index, which also deletes its alias.
2. An errant write request to the now-deleted alias creates a new index with the same name as the alias.
3. You can no longer use the alias due to a naming conflict with the new index. If you switched the alias to another index, specify `"include_aliases": false` when you restore from a snapshot.

To restore a snapshot

1. Identify the snapshot you want to restore. Ensure that all settings for this index, such as custom analyzer packages or allocation requirement settings, are compatible with the domain. To see all snapshot repositories, run the following command:

   ```
   curl -XGET 'domain-endpoint/_snapshot?pretty'
   ```

   After you identify the repository, run the following command to see all snapshots:

   ```
   curl -XGET 'domain-endpoint/_snapshot/repository-name/_all?pretty'
   ```

> **Note**
>
> Most automated snapshots are stored in the `cs-automated` repository. If your domain encrypts data at rest, they're stored in the `cs-automated-enc` repository. If you don't see the manual snapshot repository you're looking for, make sure you registered it to the domain.

2. (Optional) Delete or rename one or more indexes in the OpenSearch Service domain if you have naming conflicts between indexes on the cluster and indexes in the snapshot. You can't restore a snapshot of your indexes to an OpenSearch cluster that already contains indexes with the same names.

   You have the following options if you have index naming conflicts:

   - Delete the indexes on the existing OpenSearch Service domain and then restore the snapshot.
   - Rename the indexes as you restore them from the snapshot and reindex them later. To learn how to rename indexes, see this example request in the OpenSearch documentation.
   - Restore the snapshot to a different OpenSearch Service domain (only possible with manual snapshots).

   The following command deletes all existing indexes in a domain:

   ```
   curl -XDELETE 'domain-endpoint/_all'
   ```

   However, if you don't plan to restore all indexes, you can just delete one:

   ```
   curl -XDELETE 'domain-endpoint/index-name'
   ```

3. To restore a snapshot, run the following command:

   ```
   curl -XPOST 'domain-endpoint/_snapshot/repository-name/snapshot-name/_restore'
   ```

   Due to special permissions on the OpenSearch Dashboards and fine-grained access control indexes, attempts to restore all indexes might fail, especially if you try to restore from an

automated snapshot. The following example restores just one index, `my-index`, from `2020-snapshot` in the `cs-automated` snapshot repository:

```
curl -XPOST 'domain-endpoint/_snapshot/cs-automated/2020-snapshot/_restore' \
-d '{"indices": "my-index"}' \
-H 'Content-Type: application/json'
```

Alternately, you might want to restore all indexes *except* the Dashboards and fine-grained access control indexes:

```
curl -XPOST 'domain-endpoint/_snapshot/cs-automated/2020-snapshot/_restore' \
-d '{"indices": "-.kibana*,-.opendistro*"}' \
-H 'Content-Type: application/json'
```

You can restore a snapshot without deleting its data by using the `rename_pattern` and `rename_replacement` parameters. For more information on these parameters, see the Restore Snapshot API request fields and example request in the OpenSearch documentation.

Note

If not all primary shards were available for the indexes involved, a snapshot might have a `state` of `PARTIAL`. This value indicates that data from at least one shard wasn't stored successfully. You can still restore from a partial snapshot, but you might need to use older snapshots to restore any missing indexes.

Deleting manual snapshots

To delete a manual snapshot, run the following command:

```
DELETE _snapshot/repository-name/snapshot-name
```

Automating snapshots with Snapshot Management

You can set up a Snapshot Management (SM) policy in OpenSearch Dashboards to automate periodic snapshot creation and deletion. SM can snapshot of a group of indices, whereas Index State Management can only take one snapshot per index. To use SM in OpenSearch Service, you

need to register your own Amazon S3 repository. For instructions to register your repository, see Registering a manual snapshot repository.

Prior to SM, OpenSearch Service offered a free, automated snapshot feature that's still turned on by default. This feature sends snapshots into the service-maintained `cs-*` repository. To deactivate the feature, reach out to Support.

For more information on the SM feature, see Snapshot management in the OpenSearch documentation.

SM doesn't currently support snapshot creation on multiple index types. For example, if you try to create snapshot on multiple indices with `*` and some indices are in the warm tier, the snapshot creation will fail. If you need your snapshot to contain multiple index types, use the ISM snapshot action until SM supports this option.

Configure permissions

If you're upgrading to 2.5 from a previous OpenSearch Service domain version, the snapshot management security permissions might not be defined on the domain. Non-admin users must be mapped to this role in order to use snapshot management on domains using fine-grained access control. To manually create the snapshot management role, perform the following steps:

1. In OpenSearch Dashboards, go to **Security** and choose **Permissions**.
2. Choose **Create action group** and configure the following groups:

Group name	Permissions
`snapshot_management_full_access`	• `cluster:admin/opensearch/snapshot_management/*` • `cluster:admin/opensearch/notifications/feature/publish` • `cluster:admin/repository/*` • `cluster:admin/snapshot/*`
`snapshot_management_read_access`	• `cluster:admin/opensearch/snapshot_management/policy/get` • `cluster:admin/opensearch/snapshot_management/policy/search`

Group name	Permissions
	• `cluster:admin/opensearch/snapshot_management/policy/explain` • `cluster:admin/repository/get` • `cluster:admin/snapshot/get`

3. Choose **Roles** and **Create role**.
4. Name the role **snapshot_management_role**.
5. For **Cluster permissions**, select `snapshot_management_full_access` or `snapshot_management_read_access`.
6. Choose **Create**.
7. After you create the role, map it to any user or backend role that will manage snapshots.

Considerations

Consider the following when you configure snapshot management:

- One policy is allowed per repository.
- Up to 400 snapshots are allowed for one policy.
- This feature won't run if your domain has a red status, is under high JVM pressure (85% or above), or has a stuck snapshot function. When the overall indexing and searching performance of your cluster is impacted, SM may also be impacted.
- A snapshot operation only starts after the previous operation finishes, so that no concurrent snapshot operations are activated by one policy.
- Multiple policies with the same schedule can cause a resource spike. If the policies' snapshotted indices overlap, the shard-level snapshot operations can only run sequentially, which can cause a cascaded performance problem. If the policies share a repository, there will be spike of write operations to that repository.
- We recommend that you schedule your snapshot operations automation to no more than once per hour, unless you have a special use case.

Automating snapshots with Index State Management

You can use the Index State Management (ISM) `snapshot` operation to automatically trigger snapshots of indexes based on changes in their age, size, or number of documents. ISM is best when you need one snapshot per index. If you need to snapshot of a group of indices, see Automating snapshots with Snapshot Management.

To use SM in OpenSearch Service, you need to register your own Amazon S3 repository. For an example ISM policy using the `snapshot` operation, see Sample Policies.

Using Curator for snapshots

If ISM doesn't work for index and snapshot management, you can use Curator instead. It offers advanced filtering functionality that can help simplify management tasks on complex clusters. Use pip to install Curator:

```
pip install elasticsearch-curator
```

You can use Curator as a command line interface (CLI) or Python API. If you use the Python API, you must use version 7.13.4 or earlier of the legacy elasticsearch-py client. It doesn't support the opensearch-py client.

If you use the CLI, export your credentials at the command line and configure `curator.yml` as follows:

```
client:
  hosts: search-my-domain.us-west-1.es.amazonaws.com
  port: 443
  use_ssl: True
  aws_region: us-west-1
  aws_sign_request: True
  ssl_no_validate: False
  timeout: 60

logging:
  loglevel: INFO
```

Upgrading Amazon OpenSearch Service domains

> **Note**
>
> OpenSearch and Elasticsearch version upgrades differ from service software updates. For information on updating the service software for your OpenSearch Service domain, see the section called "Service software updates".

Amazon OpenSearch Service offers in-place upgrades for domains that run OpenSearch 1.0 or later, or Elasticsearch 5.1 or later. If you use services like Amazon Data Firehose or Amazon CloudWatch Logs to stream data to OpenSearch Service, check that these services support the newer version of OpenSearch before migrating.

Supported upgrade paths

Currently, OpenSearch Service supports the following upgrade paths:

From version	To version
OpenSearch 1.3 or 2.*x*	OpenSearch 2.*x* OpenSearch 2.17 will enable concurrent segment search by default with auto mode if the domain satisfies the following conditions: • No previous concurrent search settings are explicitly set. • All data instances (hot and warm) are of instance type 2.xl or more. • The average p90 cpu utilization on data instances (hot and warm) for more than 1 week is below 45%. For more details about concurrent segment search settings here, see Concurrent segment search. Version 2.3 has the following breaking changes: • The `type` parameter was removed from all OpenSearch API endpoints in version 2.0. For more information, see the breaking changes.

From version	To version
	• If your domain contains any indexes (hot, UltraWarm, or cold) that were originally created in Elasticsearch 6.8, those indexes are not compatible with OpenSearch 2.3. Before you upgrade to version 2.3, you must reindex the incompatible indexes. For incompatible UltraWarm or cold indexes, migrate them to hot storage, reindex the data, and then migrate them back to warm or cold storage. Alternately, you can delete the indexes if you no longer need them. If you accidentally upgrade your domain to version 2.3 without performing these steps first, you won't be able to migrate the incompatible indexes out of their current storage tier. Your only option is to delete them.
OpenSearch 1.*x*	OpenSearch 1.*x*
Elasticsearch 7.*x*	Elasticsearch 7.*x* or OpenSearch 1.*x* **Important** OpenSearch 1.*x* introduces numerous breaking changes. For details, see *Amazon OpenSearch Service rename*.

From version	To version
Elasticsearch 6.8	Elasticsearch 7.*x* or OpenSearch 1.*x* **Important** Elasticsearch 7.0 and OpenSearch 1.0 include numerous breaking changes. Before initiating an in-place upgrade, we recommend taking a manual snapshot of the 6.*x* domain, restoring it on a test 7.*x* or OpenSearch 1.*x* domain, and using that test domain to identify potential upgrade issues. For breaking changes in OpenSearch 1.0, see *Amazon OpenSearch Service rename*. Like Elasticsearch 6.*x*, indexes can only contain one mapping type, but that type must now be named `_doc`. As a result, certain APIs no longer require a mapping type in the request body (such as the `_bulk` API). For new indexes, self-hosted Elasticsearch 7.*x* and OpenSearch 1.*x* have a default shard count of one. OpenSearch Service domains on Elasticsearch 7.*x* and later retain the previous default of five.
Elasticsearch 6.*x*	Elasticsearch 6.*x*
Elasticsearch 5.6	Elasticsearch 6.*x* **Important** Indexes created in version 6.*x* no longer support multiple mapping types. Indexes created in version 5.*x* still support multiple mapping types when restored into a 6.*x* cluster. Check that your client code creates only a single mapping type per index. To minimize downtime during the upgrade from Elasticsearch 5.6 to 6.*x*, OpenSearch Service reindexes the `.kibana` index to `.kibana-6` , deletes `.kibana`, creates an alias named `.kibana`, and maps the new index to the new alias.

From version	To version
Elasticsearch 5.*x*	Elasticsearch 5.*x*

The upgrade process consists of three steps:

1. **Pre-upgrade checks** – OpenSearch Service checks for issues that can block an upgrade and doesn't proceed to the next step unless these checks succeed.
2. **Snapshot** – OpenSearch Service takes a snapshot of the OpenSearch or Elasticsearch cluster and doesn't proceed to the next step unless the snapshot succeeds. If the upgrade fails, OpenSearch Service uses this snapshot to restore the cluster to its original state. For more information see the section called “Can't downgrade after upgrade”.
3. **Upgrade** – OpenSearch Service starts the upgrade, which can take from 15 minutes to several hours to complete. OpenSearch Dashboards might be unavailable during some or all of the upgrade.

Upgrading a domain (console)

The upgrade process is irreversible and can't be paused or cancelled. During an upgrade, you can't make configuration changes to the domain. Before starting an upgrade, double-check that you want to proceed. You can use these same steps to perform the pre-upgrade check without actually starting an upgrade.

If the cluster has dedicated master nodes, OpenSearch upgrades complete without downtime. Otherwise, the cluster might be unresponsive for several seconds post-upgrade while it elects a master node.

To upgrade a domain to a later version of OpenSearch or Elasticsearch

1. Take a manual snapshot of your domain. This snapshot serves as a backup that you can restore on a new domain if you want to return to using the prior OpenSearch version.
2. Go to https://aws.amazon.com and choose **Sign In to the Console**.
3. Under **Analytics**, choose **Amazon OpenSearch Service**.
4. In the navigation pane, under **Domains**, choose the domain that you want to upgrade.

5. Choose **Actions** and **Upgrade**.
6. Select the version to upgrade to. If you're upgrading to an OpenSearch version, the **Enable compatibility mode** option appears. If you enable this setting, OpenSearch reports its version as 7.10 to allow Elasticsearch OSS clients and plugins like Logstash to continue working with Amazon OpenSearch Service. You can disable this setting later
7. Choose **Upgrade**.
8. Check the **Status** on the domain dashboard to monitor the status of the upgrade.

Upgrading a domain (CLI)

You can use the following operations to identify the correct version of OpenSearch or Elasticsearch for your domain, start an in-place upgrade, perform the pre-upgrade check, and view progress:

- `get-compatible-versions` (`GetCompatibleVersions`)
- `upgrade-domain` (`UpgradeDomain`)
- `get-upgrade-status` (`GetUpgradeStatus`)
- `get-upgrade-history` (`GetUpgradeHistory`)

For more information, see the AWS CLI command reference and Amazon OpenSearch Service API Reference.

Upgrading a domain (SDK)

This sample uses the OpenSearchService low-level Python client from the AWS SDK for Python (Boto) to check if a domain is eligible for upgrade to a specific version, upgrades it, and continuously checks the upgrade status.

```
import boto3
from botocore.config import Config
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default Region.

DOMAIN_NAME = ''  # The name of the domain to upgrade
TARGET_VERSION = ''  # The version you want to upgrade the domain to. For example,
 OpenSearch_1.1
```

```
my_config = Config(
    # Optionally lets you specify a Region other than your default.
    region_name='us-east-1'
)
client = boto3.client('opensearch', config=my_config)


def check_versions():
    """Determine whether domain is eligible for upgrade"""
    response = client.get_compatible_versions(
        DomainName=DOMAIN_NAME
    )
    compatible_versions = response['CompatibleVersions']
    for i in range(len(compatible_versions)):
        if TARGET_VERSION in compatible_versions[i]["TargetVersions"]:
            print('Domain is eligible for upgrade to ' + TARGET_VERSION)
            upgrade_domain()
            print(response)
        else:
            print('Domain not eligible for upgrade to ' + TARGET_VERSION)


def upgrade_domain():
    """Upgrades the domain"""
    response = client.upgrade_domain(
        DomainName=DOMAIN_NAME,
        TargetVersion=TARGET_VERSION
    )
    print('Upgrading domain to ' + TARGET_VERSION + '...' + response)
    time.sleep(5)
    wait_for_upgrade()


def wait_for_upgrade():
    """Get the status of the upgrade"""
    response = client.get_upgrade_status(
        DomainName=DOMAIN_NAME
    )
    if (response['UpgradeStep']) == 'UPGRADE' and (response['StepStatus']) ==
 'SUCCEEDED':
        print('Domain successfully upgraded to ' + TARGET_VERSION)
    elif (response['StepStatus']) == 'FAILED':
        print('Upgrade failed. Please try again.')
```

```
    elif (response['StepStatus']) == 'SUCCEEDED_WITH_ISSUES':
        print('Upgrade succeeded with issues')
    elif (response['StepStatus']) == 'IN_PROGRESS':
        time.sleep(30)
        wait_for_upgrade()


def main():
    check_versions()


if __name__ == "__main__":
    main()
```

Troubleshooting validation failures

When you initiate an OpenSearch or Elasticsearch version upgrade, OpenSearch Service first performs a series of validation checks to ensure that your domain is eligible for an upgrade. If any of these checks fail, you receive a notification containing the specific issues that you must fix before upgrading your domain. For a list of potential issues and steps to resolve them, see the section called “Troubleshooting validation errors”.

Troubleshooting an upgrade

In-place upgrades require healthy domains. Your domain might be ineligible for an upgrade or fail to upgrade for a wide variety of reasons. The following table shows the most common issues.

Issue	Description
Optional plugin not supported	When you upgrade a domain with optional plugins, OpenSearch Service automatically upgrades the plugins as well. Therefore, the target version for your domain must also support these optional plugins. If the domain has an optional plugin installed that is not available for the target version, the upgrade request fails.
Too many shards per node	OpenSearch, as well as 7.*x* versions of Elasticsearch, have a default setting of no more than 1,000 shards per node. If a node in your current cluster exceeds this setting, OpenSearch Service doesn't allow

Issue	Description
	you to upgrade. See the section called "Exceeded maximum shard limit" for troubleshooting options.
Domain in processing	The domain is in the middle of a configuration change. Check upgrade eligibility after the operation completes.
Red cluster status	One or more indexes in the cluster is red. For troubleshooting steps, see the section called "Red cluster status".
High error rate	The cluster is returning a large number of 5*xx* errors when attemptin g to process requests. This problem is usually the result of too many simultaneous read or write requests. Consider reducing traffic to the cluster or scaling your domain.
Split brain	*Split brain* means that your cluster has more than one master node and has split into two clusters that never will rejoin on their own. You can avoid split brain by using the recommended number of dedicated master nodes. For help recovering from split brain, contact Support.
Master node not found	OpenSearch Service can't find the cluster's master node. If your domain uses multi-AZ, an Availability Zone failure might have caused the cluster to lose quorum and be unable to elect a new master node. If the issue does not self-resolve, contact Support.
Too many pending tasks	The master node is under heavy load and has many pending tasks. Consider reducing traffic to the cluster or scaling your domain.
Impaired storage volume	The disk volume of one or more nodes isn't functioning properly. This issue often occurs alongside other issues, like a high error rate or too many pending tasks. If it occurs in isolation and doesn't self-resolve, contact Support.
KMS key issue	The KMS key that is used to encrypt the domain is either inaccessible or missing. For more information, see the section called "Monitoring domains that encrypt data at rest".

Issue	Description
Snapshot in progress	The domain is currently taking a snapshot. Check upgrade eligibili ty after the snapshot finishes. Also check that you can list manual snapshot repositories, list snapshots within those repositories, and take manual snapshots. If OpenSearch Service is unable to check whether a snapshot is in progress, upgrades can fail.
Snapshot timeout or failure	The pre-upgrade snapshot took too long to complete or failed. Check cluster health, and try again. If the problem persists, contact Support.
Incompatible indexes	One or more indexes is incompatible with the target version. This problem can occur if you migrated the indexes from an older version of OpenSearch or Elasticsearch. Reindex the indexes and try again.
High disk usage	Disk usage for the cluster is above 90%. Delete data or scale the domain, and try again.
High JVM usage	JVM memory pressure is above 75%. Reduce traffic to the cluster or scale the domain, and try again.
OpenSearch Dashboards alias problem	`.dashboards` is already configured as an alias and maps to an incompatible index, likely one from an earlier version of OpenSearch Dashboards. Reindex and try again.
Red Dashboards status	OpenSearch Dashboards status is red. Try using Dashboards when the upgrade completes. If the red status persists, resolve it manually, and try again.
Cross-cluster compatibility	You can only upgrade if cross-cluster compatibility is maintained between the source and destination domains after the upgrade. During the upgrade process, any incompatible connections are identified. To proceed, either upgrade the remote domain or delete the incompatible connections. Note that if replication is active on the domain, you can't resume it once you delete the connection.

Issue	Description
Other OpenSearch Service service issue	Issues with OpenSearch Service itself might cause your domain to display as ineligible for an upgrade. If none of the preceding condition s apply to your domain and the problem persists for more than a day, contact Support.

Using a snapshot to migrate data

In-place upgrades are the easier, faster, and more reliable way to upgrade a domain to a later OpenSearch or Elasticsearch version. Snapshots are a good option if you need to migrate from a pre-5.1 version of Elasticsearch or want to migrate to an entirely new cluster.

The following table shows how to use snapshots to migrate data to a domain that uses a different OpenSearch or Elasticsearch version. For more information about taking and restoring snapshots, see the section called “Creating index snapshots”.

From version	To version	Migration process

OpenSearch 1.3 or 2.*x*	OpenSearch 2.*x*	1. Review breaking changes for OpenSearch 2.3 to see if you need to make adjustments to your indexes or applications.
		2. Create a manual snapshot of the 1.3 or 2.*x* domain.
		3. Create a 2.*x* domain that's a higher version than your original 1.3 or 2.*x* domain.
		4. Restore the snapshot from the original domain to the 2.*x* domain. During the operation, you might need to restore your `.opensearch` index under a new name:
		`POST _snapshot/ <repository-name> /<snapshot-name>/_restore`
		`{`
		`  "indices": "*",`
		`  "ignore_unavailable": true,`
		`  "rename_pattern": ".opensearch",`

From version	To version	Migration process
		<pre> "rename_replacement": ".backup-opensearc h" }</pre> Then you can reindex `.backup-opensearch` on the new domain and alias it to `.opensearch` . Note that the `_restore` REST call doesn't include `include_global_state` because the default in `_restore` is false. As a result, the test domain won't include any index templates and won't have the full state from the backup. 5. If you no longer need your original domain, delete it. Otherwise, you continue to incur charges for the domain.

From version	To version	Migration process

OpenSearch 1.*x*	OpenSearch 1.*x*	1. Create a manual snapshot of the 1.*x* domain.
		2. Create a 1.*x* domain that's a higher version than your original 1.*x* domain.
		3. Restore the snapshot from the original domain to the new 1.*x* domain. During the operation, you might need to restore your `.opensearch` index under a new name:
		`POST _snapshot/ <repository-name> /<snapshot-name>/_restore`
		`{`
		`  "indices": "*",`
		`  "ignore_unavailable": true,`
		`  "rename_pattern": ".opensearch",`
		`  "rename_replacement": ".backup-opensearch"`
		`}`
		Then you can reindex `.backup-opensearch` on the new domain and alias it to `.opensearch` . Note that the `_restore` REST call doesn't include `include_global_state` because the default in `_restore` is false. As a result, the test domain won't include any index templates and won't have the full state from the backup.
		4. If you no longer need your original domain, delete it. Otherwise, you continue to incur charges for the domain.

From version	To version	Migration process

Elasticsearch 6.*x* or 7.*x*	OpenSearch 1.*x*	1. Review breaking changes for OpenSearch 1.0 to see if you need to make adjustments to your indexes or applications.
		2. Create a manual snapshot of the Elasticsearch 7.*x* or 6.*x* domain.
		3. Create an OpenSearch 1.*x* domain.
		4. Restore the snapshot from the Elasticsearch domain to the OpenSearch domain. During the operation, you might need to restore your `.elasticsearch` index under a new name:
		`POST _snapshot/ <repository-name> /<snapshot-name>/_restore`
		`{`
		`  "indices": "*",`
		`  "ignore_unavailable": true,`
		`  "rename_pattern": ".elasticsearch",`
		`  "rename_replacement": ".backup-opensearch"`
		`}`
		Then you can reindex `.backup-opensearch` on the new domain and alias it to `.elasticsearch`. Note that the `_restore` REST call doesn't include `include_global_state` because the default in `_restore` is false. As a result, the test domain won't include any index templates and won't have the full state from the backup.
		5. If you no longer need your original domain, delete it. Otherwise, you continue to incur charges for the domain.

From version	To version	Migration process

Elasticsearch 6.*x*	Elasticsearch 7.*x*	1. Review breaking changes for 7.0 to see if you need to make adjustments to your indexes or applications.
		2. Create a manual snapshot of the 6.*x* domain.
		3. Create a 7.*x* domain.
		4. Restore the snapshot from the original domain to the 7.*x* domain. During the operation, you likely need to restore the `.opensearch` index under a new name:
		`POST _snapshot/ <repository-name> /<snapshot-name>/_restore`
		`{`
		`  "indices": "*",`
		`  "ignore_unavailable": true,`
		`  "rename_pattern": ".elasticsearch",`
		`  "rename_replacement": ".backup-elasticsearch"`
		`}`
		Then you can reindex `.backup-elasticsearch` on the new domain and alias it to `.elasticsearch`. Note that the `_restore` REST call doesn't include `include_global_state` because the default in `_restore` is false. As a result, the test domain won't include any index templates and won't have the full state from the backup.
		5. If you no longer need your original domain, delete it. Otherwise, you continue to incur charges for the domain.

From version	To version	Migration process
Elasticsearch 6.*x*	Elasticsearch 6.8	1. Create a manual snapshot of the 6.*x* domain. 2. Create a 6.8 domain. 3. Restore the snapshot from the original domain to the 6.8 domain. 4. If you no longer need your original domain, delete it. Otherwise, you continue to incur charges for the domain.
Elasticsearch 5.*x*	Elasticsearch 6.*x*	1. Review breaking changes for 6.0 to see if you need to make adjustments to your indices or applications. 2. Create a manual snapshot of the 5.*x* domain. 3. Create a 6.*x* domain. 4. Restore the snapshot from the original domain to the 6.*x* domain. 5. If you no longer need your 5.*x* domain, delete it. Otherwise, you continue to incur charges for the domain.
Elasticsearch 5.*x*	Elasticsearch 5.6	1. Create a manual snapshot of the 5.*x* domain. 2. Create a 5.6 domain. 3. Restore the snapshot from the original domain to the 5.6 domain. 4. If you no longer need your original domain, delete it. Otherwise, you continue to incur charges for the domain.

From version	To version	Migration process
Elasticsearch 2.3	Elasticsearch 6.*x*	Elasticsearch 2.3 snapshots are not compatible with 6.*x*. To migrate your data directly from 2.3 to 6.*x*, you must manually recreate your indexes in the new domain. Alternately, you can follow the 2.3 to 5.*x* steps in this table, perform `_reindex` operations in the new 5.*x* domain to convert your 2.3 indexes to 5.*x* indexes, and then follow the 5.*x* to 6.*x* steps.
Elasticsearch 2.3	Elasticsearch 5.*x*	1. Review breaking changes for 5.0 to see if you need to make adjustments to your indexes or applications. 2. Create a manual snapshot of the 2.3 domain. 3. Create a 5.*x* domain. 4. Restore the snapshot from the 2.3 domain to the 5.*x* domain. 5. If you no longer need your 2.3 domain, delete it. Otherwise, you continue to incur charges for the domain.
Elasticsearch 1.5	Elasticsearch 5.*x*	Elasticsearch 1.5 snapshots are not compatible with 5.*x*. To migrate your data from 1.5 to 5.*x*, you must manually recreate your indexes in the new domain. **Important** 1.5 snapshots *are* compatible with 2.3, but OpenSearch Service 2.3 domains do not support the `_reindex` operation. Because you cannot reindex them, indexes that originated in a 1.5 domain still fail to restore from 2.3 snapshots to 5.*x* domains.

From version	To version	Migration process
Elasticsearch 1.5	Elasticsearch 2.3	1. Use the migration plugin to find out if you can directly upgrade to version 2.3. You might need to make changes to your data before migration. a. In a web browser, open `http://domain-endpoint /_plugin/migration/` . b. Choose **Run checks now**. c. Review the results and, if needed, follow the instructions to make changes to your data. 2. Create a manual snapshot of the 1.5 domain. 3. Create a 2.3 domain. 4. Restore the snapshot from the 1.5 domain to the 2.3 domain. 5. If you no longer need your 1.5 domain, delete it. Otherwise, you continue to incur charges for the domain.

Creating a custom endpoint for Amazon OpenSearch Service

Creating a custom endpoint for your Amazon OpenSearch Service domain makes it easier for you to refer to your OpenSearch and OpenSearch Dashboards URLs. You can include your company's branding or just use a shorter, easier-to-remember endpoint than the standard one.

If you ever need to switch to a new domain, just update your DNS to point to the new URL and continue using the same endpoint as before.

You secure custom endpoints by either generating a certificate in AWS Certificate Manager (ACM) or importing one of your own.

Custom endpoints for new domains

You can enable a custom endpoint for a new OpenSearch Service domain using the OpenSearch Service console, AWS CLI, or configuration API.

To customize your endpoint (console)

1. From the OpenSearch Service console, choose **Create domain** and provide a name for the domain.
2. Under **Custom endpoint**, select **Enable custom endpoint**.
3. For **Custom hostname**, enter your preferred custom endpoint hostname. The hostname should be a fully qualified domain name (FQDN), such as www.yourdomain.com or example.yourdomain.com.

   > **Note**
   >
   > If you don't have a wildcard certificate you must obtain a new certificate for your custom endpoint's subdomains.

4. For **AWS certificate**, choose the SSL certificate to use for your domain. If no certificates are available, you can import one into ACM or use ACM to provision one. For more information, see Issuing and Managing Certificates in the *AWS Certificate Manager User Guide*.

   > **Note**
   >
   > The certificate must have the custom endpoint name and be in the same account as your OpenSearch Service domain. The certificate status should be ISSUED.

   - Follow the rest of the steps to create your domain and choose **Create**.
   - Select the domain when it's finished processing to view your custom endpoint.

   To use the CLI or configuration API, use the `CreateDomain` and `UpdateDomainConfig` operations. For more information, see the AWS CLI Command Reference and Amazon OpenSearch Service API Reference.

Custom endpoints for existing domains

To add a custom endpoint to an existing OpenSearch Service domain, choose **Edit** and perform steps 2–4 above.

CNAME mapping

After you enable a custom endpoint for your OpenSearch Service domain, you can create a CNAME mapping in Amazon Route 53 (or your preferred DNS service provider). Creating a CNAME mapping will enable you to route traffic to your custom endpoint and its subdomains. Without this mapping, you won't be able to route traffic to your custom endpoint. For steps to create this mapping in Route 53, see Configuring DNS routing for a new domain and Creating a new hosted zone for a subdomain. For other providers, consult their documentation.

Create a CNAME record that points the custom endpoint to the automatically generated domain endpoint. If your domain is dual stack, you can point your CNAME record to either of the two service generated endpoints. The dual stack capabilty of your custom endpoint depends on the service generated endpoint that you point the CNAME record to. The custom endpoint hostname is the *name* of the CNAME record, and the domain endpoint hostname is the *value* of the CNAME record.

If you use SAML authentication for OpenSearch Dashboards, you must update your IdP with the new SSO URL.

You can use Amazon Route 53 to create an alias record type to point your domain's custom endpoint to a dual stack search endpoint. To create an alias record type, you must configure your domain to use the dual stack IP address type. You can do this using the Route 53 API.

To create an alias record type using the Route 53 API, specify the alias target of your domain. You can find the alias target of your domain in the **Hosted Zone (dual stack)** field in the custom endpoint section of the OpenSearch Service console or by using the `DescribeDomain` API and copying the value of the `DomainEndpointV2HostedZoneId`.

Auto-Tune for Amazon OpenSearch Service

Auto-Tune in Amazon OpenSearch Service uses performance and usage metrics from your OpenSearch cluster to suggest memory-related configuration changes, including queue and cache sizes and Java virtual machine (JVM) settings on your nodes. These optional changes improve cluster speed and stability.

Some changes deploy immediately, while others are scheduled during your domain's off-peak window. You can revert to the default OpenSearch Service settings at any time. As Auto-Tune gathers and analyzes performance metrics for your domain, you can view its recommendations in the OpenSearch Service console on the **Notifications** page.

Auto-Tune is available in commercial AWS Regions on domains running any OpenSearch version, or Elasticsearch 6.7 or later, with a supported instance type.

Types of changes

Auto-Tune has two broad categories of changes:

- Nondisruptive changes that it applies as the cluster runs.
- Changes that require a blue/green deployment, which it applies during the domain's off-peak window.

Based on your domain's performance metrics, Auto-Tune can suggest adjustments to the following settings:

Change type	Category	Description
JVM heap size	Blue/green	By default, OpenSearch Service uses 50% of an instance's RAM for the JVM heap, up to a heap size of 32 GiB. Increasing this percentage gives OpenSearch more memory, but leaves less for the operating system and other processes. Larger values can decrease the number of garbage collection pauses, but increase the length of those pauses.
JVM young generation settings	Blue/green	JVM "young generation" settings affect the frequency of minor garbage collections. More frequent minor collections can decrease the number of major collections and pauses.
Queue size	Nondisruptive	By default, the search queue size is `1000` and the write queue size is `10000`. Auto-Tune automatically scales the search and write queues if additional heap is available to handle requests.
Cache size	Nondisruptive	The *field cache* monitors on-heap data structures, so it's important to monitor the cache's use. Auto-Tune scales the field data cache size to avoid out of memory and circuit breaker issues. The *shard request cache* is managed at the node level and has a default maximum size of 1% of the heap. Auto-Tune scales the

Change type	Category	Description
		shard request cache size to accept more search and index requests than what the configured cluster can handle.
Request size	Nondisruptive	By default, when the aggregated size of in-flight requests surpasses 10% of total JVM (2% for `t2` instance types and 1% for `t3.small`), OpenSearch throttles all new `_search` and `_bulk` requests until the existing requests complete. Auto-Tune automatically tunes this threshold, typically between 5-15%, based on the amount of JVM that is currently occupied on the system. For example, if JVM memory pressure is high, Auto-Tune might reduce the threshold to 5%, at which point you might see more rejections until the cluster stabilizes and the threshold increases.

Enabling or disabling Auto-Tune

OpenSearch Service enables Auto-Tune by default on new domains. To enable or disable Auto-Tune on existing domains, we recommend using the console, which simplifies the process. Enabling Auto-Tune doesn't cause a blue/green deployment.

You currently can't enable or disable Auto-Tune using AWS CloudFormation.

Console

To enable Auto-Tune on an existing domain

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. In the navigation pane, under **Domains**, choose the domain name to open the cluster configuration.
3. Choose **Turn on** if Auto-Tune isn't already enabled.
4. Optionally, select **Off-peak window** to schedule optimizations that require a blue/green deployment during the domain's configured off-peak window. For more information, see the section called “Scheduling Auto-Tune enhancements”.
5. Choose **Save changes**.

CLI

To enable Auto-Tune using the AWS CLI, send an UpdateDomainConfig request:

```
aws opensearch update-domain-config \
  --domain-name my-domain \
  --auto-tune-options DesiredState=ENABLED
```

Scheduling Auto-Tune enhancements

Prior to February 16, 2023, Auto-Tune used *maintenance windows* to schedule changes that required a blue/green deployment. Maintenance windows are now deprecated in favor of the off-peak window, which is a daily 10-hour time block during which your domain typically experiences low traffic. You can modify the default start time for the off-peak window, but you can't modify the length.

Any domains that had Auto-Tune maintenance windows enabled before the introduction of off-peak windows on February 16, 2023 can continue to use legacy maintenance windows with no interruption. However, we recommend that you migrate your existing domains to use the off-peak window for domain maintenance instead. For instructions, see the section called “Migrating from Auto-Tune maintenance windows”.

Console

To schedule Auto-Tune actions the off-peak window

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. In the navigation pane, under **Domains**, choose the domain name to open the cluster configuration.
3. Go to the **Auto-Tune** tab and choose **Edit**.
4. Choose **Turn on** if Auto-Tune isn't already enabled.
5. Under **Schedule optimizations during off-peak window**, select **Off-peak window**.
6. Choose **Save changes**.

CLI

To configure your domain to schedule Auto-Tune actions during the configured off-peak window, include `UseOffPeakWindow` in the UpdateDomainConfig request:

```
aws opensearch update-domain-config \
  --domain-name my-domain \
  --auto-tune-options
 DesiredState=ENABLED,UseOffPeakWindow=true,MaintenanceSchedules=null
```

Monitoring Auto-Tune changes

You can monitor Auto-Tune statistics in Amazon CloudWatch. For a full list of metrics, see the section called "Auto-Tune metrics".

OpenSearch Service sends Auto-Tune events to Amazon EventBridge. You can use EventBridge to configure rules that send an email or perform a specific action when an event is received. To see the format of each Auto-Tune event sent to EventBridge, see the section called "Auto-Tune events".

Tagging Amazon OpenSearch Service domains

Tags let you assign arbitrary information to an Amazon OpenSearch Service domain so you can categorize and filter on that information. A tag is a key-value pair that you define and associate with an OpenSearch Service domain. You can use these tags to track costs by grouping expenses for similarly tagged resources. AWS doesn't apply any semantic meaning to your tags. Tags are interpreted strictly as character strings. All tags have the following elements:

Tag Element	Description	Required
Tag key	The tag key is the name of the tag. Key must be unique to the OpenSearch Service domain to which they're attached. For a list of basic restrictions on tag keys and values, see User-Defined Tag Restrictions.	Yes
Tag value	The tag value is the string value of the tag. Tag values can be `null` and don't have to be unique in a tag set. For example, you can have a key-value pair in a tag set of project/Trinity and cost-center/Trinity. For a list of basic restrictions on tag keys and values, see User-Defined Tag Restrictions.	No

Each OpenSearch Service domain has a tag set, which contains all the tags assigned to that OpenSearch Service domain. AWS doesn't automatically assign any tags to OpenSearch Service domains. A tag set can contain between 0 and 50 tags. If you add a tag to a domain with the same key as an existing tag, the new value overwrites the old value.

Tagging examples

You can use a key to define a category, and the value could be an item in that category. For example, you could define a tag key of `project` and a tag value of `Salix`, indicating that the OpenSearch Service domain is assigned to the Salix project. You could also use tags to designate OpenSearch Service domains as being used for test or production by using a key such as `environment=test` or `environment=production`. Try to use a consistent set of tag keys to make it easier to track metadata that is associated with OpenSearch Service domains.

You also can use tags to organize your AWS bill to reflect your own cost structure. To do this, sign up to get your AWS account bill with tag key values included. Then, organize your billing information according to resources with the same tag key values to see the cost of combined resources. For example, you can tag several OpenSearch Service domains with key-value pairs, and then organize your billing information to see the total cost for each domain across several services. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management* documentation.

> **Note**
>
> Tags are cached for authorization purposes. Because of this, additions and updates to tags on OpenSearch Service domains might take several minutes before they're available.

Tagging domains (console)

The console is the simplest way to tag a domain.

To create a tag (console)

1. Go to https://aws.amazon.com, and then choose **Sign In to the Console**.
2. Under **Analytics**, choose **Amazon OpenSearch Service**.
3. Select the domain you want to add tags to and go to the **Tags** tab.
4. Choose **Manage** and **Add new tag**.

5. Enter a tag key and an optional value.
6. Choose **Save**.

To delete a tag, follow the same steps and choose **Remove** on the **Manage tags** page.

For more information about using the console to work with tags, see Tag Editor in the *AWS Management Console Getting Started Guide*.

Tagging domains (AWS CLI)

You can create resource tags using the AWS CLI with the **--add-tags** command.

Syntax

```
add-tags --arn=<domain_arn> --tag-list Key=<key>,Value=<value>
```

Parameter	Description
`--arn`	Amazon resource name for the OpenSearch Service domain to which the tag is attached.
`--tag-list`	Set of space-separated key-value pairs in the following format: `Key=<key>,Value=<value>`

Example

The following example creates two tags for the *logs* domain:

```
aws opensearch add-tags --arn arn:aws:es:us-east-1:379931976431:domain/logs --tag-list
 Key=service,Value=OpenSearch Key=instances,Value=m3.2xlarge
```

You can remove tags from an OpenSearch Service domain using the **--remove-tags** command.

Syntax

```
remove-tags --arn=<domain_arn> --tag-keys Key=<key>,Value=<value>
```

Parameter	Description
`--arn`	Amazon Resource Name (ARN) for the OpenSearch Service domain to which the tag is attached.
`--tag-keys`	Set of space-separated key-value pairs that you want to remove from the OpenSearch Service domain.

Example

The following example removes two tags from the *logs* domain that were created in the preceding example:

```
aws opensearch remove-tags --arn arn:aws:es:us-east-1:379931976431:domain/logs --tag-keys service instances
```

You can view the existing tags for an OpenSearch Service domain with the **--list-tags** command:

Syntax

```
list-tags --arn=<domain_arn>
```

Parameter	Description
`--arn`	Amazon Resource Name (ARN) for the OpenSearch Service domain to which the tags are attached.

Example

The following example lists all resource tags for the *logs* domain:

```
aws opensearch list-tags --arn arn:aws:es:us-east-1:379931976431:domain/logs
```

Tagging domains (AWS SDKs)

The AWS SDKs (except the Android and iOS SDKs) support all the actions defined in the Amazon OpenSearch Service API Reference, including the `AddTags`, `ListTags`, and `RemoveTags`

operations. For more information about installing and using the AWS SDKs, see AWS Software Development Kits.

Python

This example uses the OpenSearchService low-level Python client from the AWS SDK for Python (Boto) to add a tag to a domain, list the tag attached to the domain, and remove a tag from the domain. You must provide values for DOMAIN_ARN, TAG_KEY, and TAG_VALUE.

```
import boto3
from botocore.config import Config  # import configuration

DOMAIN_ARN = ''  # ARN for the domain. i.e "arn:aws:es:us-east-1:123456789012:domain/
my-domain
TAG_KEY = ''  # The name of the tag key. i.e 'Smileyface'
TAG_VALUE = ''  # The value assigned to the tag. i.e 'Practicetag'

# defines the configurations parameters such as region

my_config = Config(region_name='us-east-1')
client = boto3.client('opensearch', config=my_config)


# defines the client variable

def addTags():
    """Adds tags to the domain"""

    response = client.add_tags(ARN=DOMAIN_ARN,
                               TagList=[{'Key': TAG_KEY,
                                         'Value': TAG_VALUE}])

    print(response)


def listTags():
    """List tags that have been added to the domain"""

    response = client.list_tags(ARN=DOMAIN_ARN)
    print(response)


def removeTags():
```

```
    """Remove tags that have been added to the domain"""

    response = client.remove_tags(ARN=DOMAIN_ARN, TagKeys=[TAG_KEY])

    print('Tag removed')
    return response
```

Performing administrative actions on Amazon OpenSearch Service domains

Amazon OpenSearch Service offers several administrative options that provide granular control if you need to troubleshoot issues with your domain. These options include the ability to restart the OpenSearch process on a data node and the ability to restart a data node.

OpenSearch Service monitors node health parameters and, when there are anomalies, takes corrective actions to keep domains stable. With the administrative options to restart the OpenSearch process on a node, and restart a node itself, you have control over some of these mitigation actions.

You can use the AWS Management Console, AWS CLI, or the AWS SDK to perform these actions. The following sections cover how to perform these actions with the console.

Restart the OpenSearch process on a node

To restart the OpenSearch process on a node

1. Navigate to the OpenSearch Service console at https://console.aws.amazon.com/aos/.
2. In the left navigation pane, choose **Domains**. Choose the name of the domain that you want to work with.
3. After the domain details page opens, navigate to the **Instance health** tab.
4. Under **Data nodes**, select the button next to the node that you want to restart the process on.
5. Select the **Actions** dropdown and choose **Restart OpenSearch/Elasticsearch process**.
6. Choose **Confirm** on the modal.
7. To see the status of the action that you initiated, select the name of the node. After the node details page opens, choose the **Events** tab under the name of the node to see a list of events associated with that node.

Reboot a data node

To reboot a data node

1. Navigate to the OpenSearch Service console at https://console.aws.amazon.com/aos/.
2. In the left navigation pane, choose **Domains**. Choose the name of the domain that you want to work with.
3. After the domain details page opens, navigate to the **Instance health** tab.
4. Under **Data nodes**, select the button next to the node that you want to restart the process on.
5. Select the **Actions** dropdown and choose **Reboot node**.
6. Choose **Confirm** on the modal.
7. To see the status of the action that you initiated, select the name of the node. After the node details page opens, choose the **Events** tab under the name of the node to see a list of events associated with that node.

Restart the Dashboard or Kibana process on a node

To restart the Dashboard or Kibana process on a node

1. Navigate to the OpenSearch Service console at https://console.aws.amazon.com/aos/.
2. In the left navigation pane, choose **Domains**. Choose the name of the domain that you want to work with.
3. After the domain details page opens, navigate to the **Instance health** tab.
4. Under **Data nodes**, select the button next to the node that you want to restart the process on.
5. Select the **Actions** dropdown and choose **Restart Dashboard/Kibana process**.
6. Choose **Confirm** on the modal.
7. To see the status of the action that you initiated, select the name of the node. After the node details page opens, choose the **Events** tab under the name of the node to see a list of events associated with that node.

Limitations

Administrative options have the following limitations:

- Administrative options are supported on Elasticsearch versions 7.x and higher.

- Administrative options don't support domains with Multi-AZ with Standby enabled.
- The OpenSearch and Elasticsearch process restart and the data node reboot are supported on domains with three or more data nodes.
- The Dashboards and Kibana process support is supported on domains with two or more data nodes.
- To restart the OpenSearch process on a node or reboot a node, the domain must not be in red state and all indexes must have replicas configured.

Working with Amazon OpenSearch Service direct queries

You can use Amazon OpenSearch Service *direct query* to analyze data in Amazon CloudWatch Logs, Amazon S3, and Amazon Security Lake. OpenSearch Service provides a zero-ETL integration as a way to analyze your log data using OpenSearch SQL or OpenSearch Piped Processing Language (PPL) without incurring the friction of building ingestion pipelines or switching between analytics tools. This approach eliminates the need for data movement or duplication, allowing you to analyze your data where it rests using OpenSearch Discover. When you want to switch from querying data at rest to actively monitoring with dashboards or alerts, you can build indexed views on the data and ingest it into an OpenSearch Service index.

To get started, you configure a data source in the OpenSearch Service console. For Amazon S3, you use the domain's connections, while for CloudWatch Logs and Security Lake, you use connected data sources under Central Management in the console. Amazon S3 and Security Lake both use tables in AWS Glue Data Catalog to represent your data structure, including schema, file type, and partitioning. For Amazon S3, you create these tables within OpenSearch Query Workbench using CREATE TABLE SQL statements. For Amazon Security Lake, the tables in AWS Glue are already set up during the Security Lake setup process. CloudWatch Logs similarly has pre-configured log groups.

After you set up your data source, you sign in to Discover, where you can select your data source and choose the relevant tables (for Amazon S3 and Security Lake) or log groups (for CloudWatch Logs). From there, you can start querying your data directly.

To use advanced analytics features of OpenSearch Service for data monitoring, such as building dashboards and full-text search, you ingest data from your direct query data source by creating an indexed view on the data. You can create indexed views using common SQL indexing techniques, such as skipping indexes, materialized views, and covering indexes (where supported). To help you get started quickly building dashboards, you can use pre-built templates for common log types like VPC Flow Logs, AWS CloudTrail logs, and AWS WAF logs.

Direct query pricing

When you use OpenSearch Service direct queries, you'll incur separate charges for OpenSearch Service and the resources used to process and store your data on Amazon S3, Amazon CloudWatch Logs, and Amazon Security Lake. As you run direct queries, you'll see charges for OpenSearch Compute Units (OCUs) per hour, listed as DirectQuery OCU usage type on your bill.

Direct queries are of two types—interactive and indexed view queries.

- *Interactive queries* are used to populate the data selector and perform analytics on your data in S3, CloudWatch Logs, or Security Lake.

  For Amazon S3 direct queries, when you run a new query from Discover, OpenSearch Service starts a new session that lasts for a minimum of three minutes. OpenSearch Service keeps this session active to ensure that subsequent queries run quickly.

  For CloudWatch Logs and Security Lake queries, OpenSearch Service handles each query with a separate pre-warmed job, without maintaining an extended session.
- *Indexed view queries* use compute to maintain indexed views in OpenSearch Service. These queries usually take longer because they ingest a varying amount of data into a named index. By indexing data, you can make future interactive queries run faster or unlock advanced analytics capabilities such as dashboards or alerts, which require an index to reference.

  For Amazon S3 data sources, the indexed data is stored in a domain based on an instance type purchased. For CloudWatch Logs and Security Lake connected data sources, the indexed data is stored in an OpenSearch Serverless collection where you are charged for data indexed (IndexingOCU), data searched (SearchOCU), and data stored in GB.

For more information, see the Direct Query and Serverless sections within Amazon OpenSearch Service Pricing.

Direct query limitations

General limitations

The following limitations apply to OpenSearch Service direct queries.

- Some data types aren't supported. Supported data types are limited to Parquet, CSV, and JSON.
- If the structure of your data changes over time, you will need to update your indexed views or out-of-the-box integrations to account for the data structure changes.
- AWS CloudFormation templates aren't supported yet.
- OpenSearch SQL and OpenSearch PPL statements have different limitations when working with OpenSearch indexes compared to using direct query. Direct query supports advanced commands such as JOINs, subqueries, and lookups, while support for these commands on OpenSearch

indexes is limited or nonexistent. For more information, see the section called “Supported SQL and PPL commands”.

Limitations for Amazon S3

If you’re direct querying data in Amazon S3, the following additional limitations apply:

- Direct query for S3 is only available on OpenSearch Service domains running OpenSearch version 2.13 or later, and requires access to AWS Glue Data Catalog. Existing AWS Glue Data Catalog tables must be recreated using SQL in OpenSearch Query Workbench.
- Direct query for S3 requires you to specify a checkpoint bucket on Amazon S3. This bucket maintains the state of your indexed views, including the last refresh time and the most recently ingested data.
- Your OpenSearch domain and AWS Glue Data Catalog must be in the same AWS account. Your S3 bucket can be in a different account (requires condition to be added to your IAM policy), but must be in the same AWS Region as your domain.
- OpenSearch Service direct queries with S3 only support Spark tables generated from Query Workbench. Tables generated within AWS Glue Data Catalog or Athena are not supported by Spark streaming, which is needed to maintain indexed views.
- OpenSearch instance types have networked payload limitations of either 10 MiB or 100 MiB, depending on the specific instance type you choose.

Limitations for Amazon CloudWatch Logs

If you’re direct querying data in CloudWatch Logs, the following additional limitations apply:

- The direct query integration with CloudWatch Logs is only available on OpenSearch Service collections and the OpenSearch user interface.
- OpenSearch Serverless collections have networked payload limitations of 100 MiB.
- CloudWatch Logs supports VPC Flow, CloudTrail, and AWS WAF dashboard integrations installed from the console.

Limitations for Amazon Security Lake

If you’re direct querying data in Security Lake, the following additional limitations apply:

- The direct query integration with Security Lake is only available on OpenSearch Service collections and the OpenSearch user interface.
- OpenSearch Serverless collections have networked payload limitations of 100 MiB.
- Table management for Security Lake is performed in Lake Formation.
- Security Lake only supports materialized views as indexed views. Covering indexes are not supported.

Important recommendations for getting started with direct query

General information

We recommend you do the following when using direct query:

- Use the `COALESCE SQL` function to handle missing columns and ensure results are returned.

- Use limits on your queries to ensure you aren't pulling too much data back.
- If you plan to analyze the same dataset many times, create an indexed view to fully ingest and index the data into OpenSearch and drop it when you have completed the analysis.
- Drop acceleration jobs and indexes when they're no longer needed.
- Queries containing field names that are identical but differ only in case (such as `field1` and `FIELD1`) are not supported.

  For example, the following queries are not supported:

  ```
  Select AWSAccountId, AwsAccountId from LogGroup
  Select a.@LogStream, b.@logStream from Table A INNER Join Table B ona.id = b.id
  ```

  However, the following query is supported because the field name (@logStream) is identical in both log groups:

  ```
  Select a.@logStream, b.@logStream from Table A INNER Join Table B on a.id = b.id
  ```

- Functions and expressions must operate on field names and be part of a `SELECT` statement with a log group specified in the `FROM` clause.

For example, this query is not supported:

```
SELECT cos(10) FROM LogGroup
```

This query is supported:

```
SELECT cos(field1) FROM LogGroup
```

Information for Amazon S3

If you're using Amazon OpenSearch Service to direct query data in Amazon S3, we also recommend the following:

- Ingest data into Amazon S3 using partition formats of year, month, day, hour to speed up queries.
- When you build skipping indexes, use Bloom filters for fields with high cardinality and min/max indexes for fields with large value ranges. For high-cardinality fields, consider using a value-based approach to improve query efficiency.
- Use Index State Management to maintain storage for materialized views and covering indexes.

Information for CloudWatch Logs

If you're using Amazon OpenSearch Service to direct query data in CloudWatch Logs, we also recommend the following:

- When searching multiple log groups in one query, use the appropriate syntax. For more information, see the section called "Multi-log group functions".
- When using SQL or PPL commands, enclose certain fields in backticks to successfully query them. Backticks are needed for fields with special characters (non-alphabetic and non-numeric). For example, enclose `@message`, `Operation.Export,` and `Test::Field` in backticks. You don't need to enclose columns with purely alphabetic names in backticks.

  Example query with simple fields:

```
SELECT SessionToken, Operation, StartTime  FROM `LogGroup-A`
```

```
LIMIT 1000;
```

Similar query with backticks appended:

```
SELECT `@SessionToken`, `@Operation`, `@StartTime` FROM `LogGroup-A`
LIMIT 1000;
```

Information for Security Lake

If you're using Amazon OpenSearch Service to direct query data in Security Lake, we also recommend the following:

- Check your Security Lake status and ensure that it's running smoothly without any problems. For detailed troubleshooting steps, see Troubleshooting data lake status in the Amazon Security Lake User Guide.
- Verify your query access:
  - If you're querying Security Lake from a different account than the Security Lake delegated administrator account, set up a subscriber with query access in Security Lake.
  - If you're querying Security Lake from the same account, check for any messages in Security Lake about registering your managed S3 buckets with LakeFormation.
- Explore the query templates and pre-built dashboards to jumpstart your analysis.
- Familiarize yourself with Open Cybersecurity Schema Framework (OCSF) and Security Lake:
  - Review schema mapping examples for AWS sources in the OCSF GitHub repository
  - Learn how to query Security Lake effectively by visiting Security Lake queries for AWS source version 2 (OCSF 1.1.0)
  - Improve query performance by using partitions: `accountid`, `region`, and `time_dt`
- Get comfortable with SQL syntax, which Security Lake supports for querying. For more information, see the section called "Supported SQL commands".

Direct query quotas

Your account has the following quotas related to OpenSearch Service direct queries.

Quotas for Amazon S3

Each time you initiate a query to an Amazon S3 data source, OpenSearch Service opens a *session* and keeps it alive for at least three minutes. This reduces query latency by removing session start-up time in subsequent queries.

Description	Maximum	Can override
Connections per domain	10	Yes
Data sources per domain	20	Yes
Indexes per domain	5	Yes
Concurrent sessions per data source	10	Yes
Maximum OCU per query	60	Yes
Maximum query execution time (minutes)	30	Yes
Maximum OCUs per acceleration	20	Yes
Maximum ephemeral storage	20	Yes

Quotas for CloudWatch Logs

Note

If you're looking to perform direct queries using CloudWatch Logs Insights, make sure that you refer to the section called “Additional information for CloudWatch Logs Insights users using OpenSearch SQL”.

Description	Value	Soft limit?	Notes
Account-level TPS limit across direct query APIs	3 TPS	Yes	
Maximum number of data sources	20	Yes	Limit is per AWS account.
Maximum auto-refr eshing indexes or materialized views	30	Yes	Limit is per data source.
Maximum concurrent queries	15	Yes	Limit applies to queries in pending or running state. Includes interactive queries (for example, data retrieval commands like SELECT) and index queries (for example, operations like CREATE/ALTER/DROP).
Maximum concurrent OCU per query	512	Yes	OpenSearch Compute Units (OCU). Limit based on 15 executors and 1 driver, each with 16 vCPU and 32 GB memory. Represents concurrent processing power.
Maximum query execution time in minutes	60	No	Limit applies to OpenSearch PPL/SQL queries in CloudWatch Logs Insights.
Period for purging stale query IDs	90 days	Yes	This is the time period after which OpenSearch Service purges query metadata for older entries. For example, calling GetDirectQuery or GetDirect QueryResult fails for queries older than 90 days.

Quotas for Security Lake

Description	Value	Soft limit?	Notes
Account-level TPS limit across direct query APIs	3 TPS	Yes	
Maximum number of data sources	20	Yes	Limit is per AWS account.
Maximum auto-refr eshing indexes or materialized views	30	Yes	Limit applies per data source. Only includes indices and materialized views (MVs) with auto-refresh set to true.
Maximum concurrent queries	30	Yes	Limit applies to queries in pending or running state. Includes interactive queries (for example, data retrieval commands like `SELECT`) and index queries (for example, operations like `CREATE/ALTER/DROP`).
Maximum concurrent OCU per query	512	Yes	OpenSearch Compute Units (OCU). Limit based on 15 executors and 1 driver, each with 16 vCPU and 32 GB memory. Represents concurrent processing power.
Maximum query execution time in minutes	30	No	Applies only to interactive queries (for example, data retrieval commands like `SELECT`). For `REFRESH` queries, the limit is 6 hours.
Period for purging stale query IDs	90 days	Yes	This is the time period after which OpenSearch Service purges query metadata for older entries. For example, calling GetDirectQuery or GetDirect QueryResult fails for queries older than 90 days.

Supported AWS Regions

The following AWS Regions are supported for OpenSearch Service direct queries in Amazon S3, CloudWatch Logs, and Security Lake:

Available AWS Regions for Amazon S3

- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (Stockholm)
- US East (N. Virginia)
- US East (Ohio)
- US West (Oregon)

Available AWS Regions for CloudWatch Logs

- Asia Pacific (Mumbai)
- Asia Pacific (Hong Kong)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)

- Europe (Stockholm)
- US East (N. Virginia)
- US East (Ohio)
- US West (Oregon)
- Europe (Paris)
- Europe (London)
- South America (Sao Paulo)

Available AWS Regions for Security Lake

- Asia Pacific (Mumbai)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (Stockholm)
- US East (N. Virginia)
- US East (Ohio)
- US West (Oregon)
- Europe (Paris)
- Europe (London)
- South America (Sao Paulo)

Directly querying Amazon S3 data in OpenSearch Service

This section will walk you through the process of creating and configuring a data source integration in Amazon OpenSearch Service, enabling you to efficiently query and analyze your data stored in Amazon S3.

In the following pages, you'll learn how to set up an Amazon S3 direct-query data source, navigate the necessary prerequisites, and follow step-by-step procedures using both the AWS Management Console and the OpenSearch Service API. It also covers important next steps, including mapping AWS Glue Data Catalog roles and configuring access controls in OpenSearch Dashboards.

Topics

Creating an Amazon S3 data source integration in OpenSearch Service

You can create a new Amazon S3 direct-query data source for OpenSearch Service through the AWS Management Console or the API. Each new data source uses the AWS Glue Data Catalog to manage tables that represent Amazon S3 buckets.

Topics

Prerequisites

Before you get started, make sure that you have reviewed the following documentation:

- the section called “Limitations for Amazon S3”
- the section called “Information for Amazon S3”
- the section called “Quotas for Amazon S3”

Before you can create a data source, you must have the following resources in your AWS account:

- **An OpenSearch domain with version 2.13 or later.** This is the foundation for setting up the direct query integration. For instructions on setting this up, see the section called “ Creating OpenSearch Service domains”.

- **One or more S3 buckets.** You'll need to specify the buckets containing data that you want to query, and a bucket to store your query checkpoints in. For instructions on creating an S3 bucket, see Creating a bucket in the Amazon S3 user guide.
- **(Optional) One or more AWS Glue tables.** Querying data on Amazon S3 requires that you have tables setup in AWS Glue Data Catalog to point to the S3 data. You must create the tables using OpenSearch Query Workbench. Existing Hive tables are not compatible.

  If this is the first time you're setting up an Amazon S3 data source, you must create an admin data source to configure all of your AWS Glue Data Catalog tables. You can do this by installing OpenSearch out-of-the-box integrations or by using OpenSearch Query Workbench to create custom SQL tables for advanced use cases. For examples on creating tables for VPC, CloudTrail, and AWS WAF logs, see the documentation on GitHub for VPC, CloudTrail, and AWS WAF. After you create your tables, you can create new Amazon S3 data sources and restrict access to limited tables.
- **(Optional) A manually created IAM role.** You can use this role to manage access to your data source. Alternatively, you can have OpenSearch Service create a role for you automatically with the required permissions. If you choose to use a manually created IAM role, follow the guidance in the section called "Required permissions for manually created IAM roles".

Procedure

You can set up a direct-query data source on a domain with the AWS Management Console or the OpenSearch Service API.

To set up a data source using the AWS Management Console

1. Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/.
2. In the left navigation pane, choose **Domains**.
3. Select the domain that you want to set up a new data source for. This opens the domain details page.
4. Choose the **Connections** tab below the general domain details and find the **Direct query** section.
5. Choose **Configure data source**.
6. Enter a name and an optional description for your new data source.
7. Choose **Amazon S3 with AWS Glue Data Catalog**.

8. Under **IAM permission access settings**, choose how to manage access.

   a. If you want to automatically create a role for this data source, follow these steps:

      i. Select **Create a new role**.

      ii. Enter a name for the IAM role.

      iii. Select one or more S3 buckets that contain data you want to query.

      iv. Select an checkpoint S3 bucket to store query checkpoints in.

      v. Select one or more AWS Glue databases or tables to define which data can be queried. If tables haven't been created yet, provide access to the default database.

   b. If you want to use an existing role that you manage yourself, follow these steps:

      i. Select **Use an existing role.**

      ii. Select an existing role from the drop-down menu.

> **Note**
>
> When using your own role, you must ensure it has all necessary permissions by attaching required policies from the IAM console. For more information, refer to the sample policy in the section called "Required permissions for manually created IAM roles".

9. Choose **Configure**. This opens the data source details screen with an OpenSearch Dashboards URL. You can navigate to this URL to complete the next steps.

OpenSearch Service API

Use the AddDataSource API operation to create a new data source in your domain.

```
POST https://es.region.amazonaws.com/2021-01-01/opensearch/domain/domain-name/
dataSource

{
   "DataSourceType": {
        "S3GlueDataCatalog": {
            "RoleArn": "arn:aws:iam::account-id:role/role-name"
        }
    }
```

```
    "Description": "data-source-description",
    "Name": "my-data-source"
}
```

Next steps

Visit OpenSearch Dashboards

After you create a data source, OpenSearch Service provides you with an OpenSearch Dashboards link. You can use this to configure access control, define tables, install out-of-the-box integrations, and query your data.

For more information, see the section called “Configuring an S3 data source”.

Map the AWS Glue Data Catalog role

If you have enabled fine-grained access control after creating a data source, you must map non-admin users to an IAM role with AWS Glue Data Catalog access in order to run direct queries. To manually create a back-end `glue_access` role that you can map to the IAM role, perform the following steps:

> **Note**
>
> Indexes are used for any queries against the data source. A user with read access to the request index for a given data source can read *all* queries against that data source. A user with read access to the result index can read results for *all* queries against that data source.

1. From the main menu in OpenSearch Dashboards, choose **Security**, **Roles**, and **Create roles**.
2. Name the role **glue_access**.
3. For **Cluster permissions**, select `indices:data/write/bulk*`, `indices:data/read/scroll`, `indices:data/read/scroll/clear`.
4. For **Index**, enter the following indexes you want to grant the user with the role access to:
   - `.query_execution_request_<name of data source>`
   - `query_execution_result_<name of data source>`
   - `.async-query-scheduler`
   - `flint_*`

5. For **Index permissions**, select `indices_all`.
6. Choose **Create**.
7. Choose **Mapped users**, **Manage mapping**.
8. Under **Backend roles**, add the ARN of the AWS Glue role that needs permission to call your domain.

```
arn:aws:iam::account-id:role/role-name
```

9. Select **Map** and confirm the role shows up under **Mapped users**.

For more information on mapping roles, see the section called “Mapping roles to users”.

Additional resources

Required permissions for manually created IAM roles

When creating a data source for your domain, you choose an IAM role to manage access to your data. You have two options:

1. Create a new IAM role automatically
2. Use an existing IAM role that you created manually

If you use a manually created role, you need to attach the correct permissions to the role. The permissions must allow access to the specific data source, and allow OpenSearch Service to assume the role. This is required so that the OpenSearch Service can securely access and interact with your data.

The following sample policy demonstrates the least-privilege permissions required to create and manage a data source. If you have broader permissions, such as `s3:*` or the `AdminstratorAccess` policy, these permissions encompasses the least-privilege permissions in the sample policy.

In the following sample policy, replace the *`placeholder text`* with your own information.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"HttpActionsForOpenSearchDomain",
```

```
            "Effect":"Allow",
            "Action":"es:ESHttp*",
            "Resource":"arn:aws:es:region:account:domain/<domain_name>/*"
        },
        {
            "Sid":"AmazonOpenSearchS3GlueDirectQueryReadAllS3Buckets",
            "Effect":"Allow",
            "Action":[
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:ListBucket"
            ],
            "Condition":{
                "StringEquals":{
                    "aws:ResourceAccount":"account"
                }
            },
            "Resource":"*"
        },
        {
            "Sid":"AmazonOpenSearchDirectQueryGlueCreateAccess",
            "Effect":"Allow",
            "Action":[
                "glue:CreateDatabase",
                "glue:CreatePartition",
                "glue:CreateTable",
                "glue:BatchCreatePartition"
            ],
            "Resource":"*"
        },
        {
            "Sid":"AmazonOpenSearchS3GlueDirectQueryModifyAllGlueResources",
            "Effect":"Allow",
            "Action":[
                "glue:DeleteDatabase",
                "glue:DeletePartition",
                "glue:DeleteTable",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:GetTable",
                "glue:GetTableVersions",
                "glue:GetTables",
```

```
            "glue:UpdateDatabase",
            "glue:UpdatePartition",
            "glue:UpdateTable",
            "glue:BatchGetPartition",
            "glue:BatchDeletePartition",
            "glue:BatchDeleteTable"
         ],
         "Resource":[
            "arn:aws:glue:us-east-1:account:table/*",
            "arn:aws:glue:us-east-1:account:database/*",
            "arn:aws:glue:us-east-1:account:catalog",
            "arn:aws:es:region:account:domain/domain_name"
         ],
         "Condition":{
            "StringEquals":{
               "aws:ResourceAccount":"account"
            }
         }
      },
      {
         "Sid":"ReadAndWriteActionsForS3CheckpointBucket",
         "Effect":"Allow",
         "Action":[
            "s3:ListMultipartUploadParts",
            "s3:DeleteObject",
            "s3:GetObject",
            "s3:PutObject",
            "s3:GetBucketLocation",
            "s3:ListBucket"
         ],
         "Condition":{
            "StringEquals":{
               "aws:ResourceAccount":"account"
            }
         },
         "Resource":[
            "arn:aws:s3:::amzn-s3-demo-bucket",
            "arn:aws:s3:::amzn-s3-demo-bucket/*"
         ]
      }
   ]
}
```

To support Amazon S3 buckets in different accounts, you will need to include a condition to the Amazon S3 policy and add the appropriate account.

In the following sample condition, replace the *placeholder text* with your own information.

```
"Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "{{accountId}}"
                }
```

The role must also have the following trust policy, which specifies the target ID.

```
{
    "Version":"2012-10-17",
    "Statement":[
       {
          "Effect":"Allow",
          "Principal":{
             "Service": "directquery.opensearchservice.amazonaws.com"
          },
          "Action":"sts:AssumeRole"
       }
     ]
}
```

For instructions to create the role, see Creating a role using custom trust policies.

If you have fine-grained access control enabled in OpenSearch Service, a new OpenSearch fine-grained access control role will automatically be created for your data source. The name of the new fine-grained access control role will be `AWSOpenSearchDirectQuery` *<name of data source>*.

By default, the role has access to direct query data source indexes only. Although you can configure the role to limit or grant access to your data source, it is recommended you not adjust the access of this role. **If you delete the data source, this role will be deleted**. This will remove access for any other users if they are mapped to the role.

Configuring and querying an S3 data source in OpenSearch Dashboards

Now that you've created your data source, you can configure security settings, define your Amazon S3 tables, or set up accelerated data indexing. This section walks you through various use cases with your data source in OpenSearch Dashboards before you query your data.

To configure the following sections, you must first navigate to your data source in OpenSearch Dashboards. In the left-hand navigation, under **Management**, choose **Data sources**. Under **Manage data sources**, select the name of the data source that you created in the console.

Create Spark Tables using Query Workbench

Direct queries from OpenSearch Service to Amazon S3 use Spark tables within the AWS Glue Data Catalog. You can create tables from within the Query Workbench without having to leave OpenSearch Dashboards.

To manage existing databases and tables in your data source, or to create new tables that you want to use direct queries on, choose **Query Workbench** from the left navigation and select the Amazon S3 data source from the data source drop down.

To set up a table for VPC Flow logs stored in S3 in Parquet format, run the following query:

```
CREATE TABLE
datasourcename.gluedatabasename.vpclogstable (version INT, account_id STRING,
 interface_id STRING,
srcaddr STRING, dstaddr STRING, srcport INT, dstport INT, protocol INT, packets
 BIGINT,
bytes BIGINT, start BIGINT, end BIGINT, action STRING, log_status STRING,
`aws-account-id` STRING, `aws-service` STRING, `aws-region` STRING, year STRING,
month STRING, day STRING, hour STRING)

USING parquet PARTITIONED BY (aws-account-id, aws-service, aws-region, year, month,
day, hour)

LOCATION "s3://accountnum-vpcflow/AWSLogs"
```

After creating the table, run the following query to ensure that it's compatible with direct queries:

```
MSCK REPAIR TABLE  datasourcename.databasename.vpclogstable
```

Setup integrations for popular AWS log types

You can integrate AWS log types stored in Amazon S3 with OpenSearch Service. Use OpenSearch Dashboards to install integrations that create AWS Glue Data Catalog tables, saved queries, and dashboards. These integrations use indexed views to keep dashboards updated.

For instructions to install an integration, see Installing an integration asset in the OpenSearch documentation.

When you select an integration, make sure it has the `S3 Glue` tag.

When you set up the integration, specify **S3 Connection** for the connection type. Then, select the data source for the integration, the Amazon S3 location of the data, the checkpoint to manage acceleration indexing, and the assets required for your use case.

> **Note**
>
> Make sure the S3 bucket for your checkpoint has write permissions for the checkpoint location. Without these permissions, the integration's accelerations will fail.

Set up access control

On the details page for your data source, find the **Access controls** section and choose **Edit**. If the domain has fine-grained access control enabled, choose **Restricted** and select which roles you want to provide with access to the new data source. You can also choose **Admin only** if you only want the administrator to have access to the data source.

> **Important**
>
> Indexes are used for any queries against the data source. A user with read access to the request index for a given data source can read *all* queries against that data source. A user with read access to the result index can read results for *all* queries against that data source.

Querying S3 data in OpenSearch Discover

After you set up your tables and configure your desired optional query acceleration, you can start analyzing your data. To query your data, select your data source from the drop-down menu. If

you're using Amazon S3 and OpenSearch Dashboards, go to Discover and select the data source name.

If you're using a skipping index or haven't created an index, you can use SQL or PPL to query your data. If you've configured a materialized view or a covering index, you already have an index and can use Dashboards Query Language (DQL) throughout Dashboards. You can also use PPL with the Observability plugin, and SQL with the Query Workbench plugin. Currently, only the Observability and Query Workbench plugins support PPL and SQL. For querying data using the OpenSearch Service API, refer to the async API documentation.

> **Note**
>
> Not all SQL and PPL statements, commands and functions are supported. For a list of supported commands, see the section called "Supported SQL and PPL commands".
> If you've created a materialized view or covering index, you can use DQL to query your data given that you've indexed it within.

Troubleshooting

There might be instances when results don't return as expected. If you experience any issues, make sure that you're following the the section called "Recommendations".

Directly querying Amazon CloudWatch Logs data in OpenSearch Service

This section will walk you through the process of creating and configuring a data source integration in Amazon OpenSearch Service, enabling you to efficiently query and analyze your data stored in CloudWatch Logs.

In the following pages, you'll learn how to set up a CloudWatch Logs direct-query data source, navigate the necessary prerequisites, and follow step-by-step procedures using the AWS Management Console.

Topics

- Creating an Amazon CloudWatch Logs data source integration in OpenSearch Service
- Configuring and querying a CloudWatch Logs data source in OpenSearch Dashboards

Creating an Amazon CloudWatch Logs data source integration in OpenSearch Service

If you use Amazon OpenSearch Serverless for your observability needs, you can now analyze your Amazon CloudWatch Logs without copying or ingesting the data into OpenSearch Service. This capability leverages direct query for querying data, similar to analyzing data in Amazon S3 from OpenSearch Service. You can get started by creating a new connected data source from the AWS Management Console.

You can create a new data source to analyze CloudWatch Logs data without having to build Amazon OpenSearch Serverless to directly query operational logs in CloudWatch Logs. This enables you to analyze your accessed operational data that rests outside of OpenSearch Service. By querying across OpenSearch Service and CloudWatch Logs, you can start analyzing logs in CloudWatch Logs and then move back to monitoring data sources in OpenSearch without having to switch tools.

To use this feature, you create a CloudWatch Logs direct query data source for OpenSearch Service through the AWS Management Console.

Topics

Prerequisites

Before you get started, make sure that you have reviewed the following documentation:

- the section called “Limitations for Amazon CloudWatch Logs”
- the section called “Information for CloudWatch Logs”
- the section called “Quotas for CloudWatch Logs”

Before you can create a data source, you must have the following resources in your AWS account:

- **Enable CloudWatch Logs.** Configure CloudWatch Logs to collect logs on the same AWS account as your OpenSearch resource. For instructions, see Getting started with CloudWatch Logs in the Amazon CloudWatch Logs user guide.
- **One or more CloudWatch log groups.** You can specify the log groups containing data that you want to query. For instructions on creating a log group, see Create a log group in CloudWatch Logs in the Amazon CloudWatch Logs user guide.
- **(Optional) A manually created IAM role.** You can use this role to manage access to your data source. Alternatively, you can have OpenSearch Service create a role for you automatically with the required permissions. If you choose to use a manually created IAM role, follow the guidance in the section called "Required permissions for manually created IAM roles".

Procedure

You can set up a collection-level query data source with the AWS Management Console.

To set up a collection-level data source using the AWS Management Console

1. Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/.
2. In the left navigation pane, go to **Central management** and choose **Connected data sources**.
3. Choose **Connect**.
4. Choose **CloudWatch** as the data source type.
5. Choose **Next**.
6. Under **Data connection details**, enter a name and an optional description.
7. Under **IAM roles**, choose how to manage access to the log groups.
    a. If you want to automatically create a role for this data source, follow these steps:
        i. Select **Create a new role**.
        ii. Enter a name for the IAM role.
        iii. Select one or more log groups to define which data can be queried.
    b. If you want to use an existing role that you manage yourself, follow these steps:
        i. Select **Use an existing role.**
        ii. Select an existing role from the drop-down menu.

> **Note**
>
> When using your own role, you must ensure it has all necessary permissions by attaching required policies from the IAM console. For more information, see the section called "Required permissions for manually created IAM roles".

8. (Optional) Under **Tags**, add tags to your data source.
9. Choose **Next**.
10. Under **Set up OpenSearch**, choose how to set up OpenSearch.
    a. Use the default settings:
       - Review the default resource names and data retention settings. We suggest you use custom names.

         When you use the default settings, a new OpenSearch application and Essentials workspace is created for you at no additional cost. OpenSearch enables you to analyze multiple data sources. It includes workspaces, which provide a tailored experiences for popular use cases. Workspaces support access control, enabling you to create private spaces for your use cases and share them only with your collaborators.

    b. Use customized settings:
       i. Choose **Customize**.
       ii. Edit the collection name and the data retention settings as needed.
       iii. Select the OpenSearch application and workspace that you want to use.
11. Choose **Next**.
12. Review your choices and choose **Edit** if you need to make any changes.
13. Choose **Connect** to set up the data source. Stay on this page while your data source is created. When it's ready, you'll be taken to the data source details page.

Next steps

Visit OpenSearch Dashboards

After you create a data source, OpenSearch Service provides you with an OpenSearch Dashboards URL. You use this to configure access control, define tables, set up log-type based dashboards for popular log types, and query your data using SQL or PPL.

For more information, see the section called “Configuring a CloudWatch Logs data source”.

Additional resources

Required permissions for manually created IAM roles

When creating a data source, you choose an IAM role to manage access to your data. You have two options:

1. Create a new IAM role automatically
2. Use an existing IAM role that you created manually

If you use a manually created role, you need to attach the correct permissions to the role. The permissions must allow access to the specific data source, and allow OpenSearch Service to assume the role. This is required so that the OpenSearch Service can securely access and interact with your data.

The following sample policy demonstrates the least-privilege permissions required to create and manage a data source. If you have broader permissions, such as `logs:*` or the `AdminstratorAccess` policy, these permissions encompasses the least-privilege permissions in the sample policy.

In the following sample policy, replace the *`placeholder text`* with your own information.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonOpenSearchDirectQueryAllLogsAccess",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups",
```

```
                "logs:StartQuery",
                "logs:GetLogGroupFields"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "accountId"
                }
            },
            "Resource": [
                "arn:aws:logs:region:accountId:log-group:*"
            ]
        }
    ]
}

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonOpenSearchDirectQueryServerlessAccess",
            "Effect": "Allow",
            "Action": [
                "aoss:APIAccessAll",
                "aoss:DashboardsAccessAll"
            ],
            "Resource": [
                "arn:aws:aoss:region:accountId:collection/ARN/*",
                "arn:aws:aoss:region:accountId:collection/ARN"
            ]
        }
    ]
}
```

The role must also have the following trust policy, which specifies the target ID.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "TrustPolicyForAmazonOpenSearchDirectQueryService",
            "Effect": "Allow",
            "Principal": {
                "Service": "directquery.opensearchservice.amazonaws.com"
```

```
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn":
 "arn:aws:opensearch:region:accountId:datasource/rolename"
                }
            }
        }
    ]
}
```

For instructions to create the role, see Creating a role using custom trust policies.

By default, the role has access to direct query data source indexes only. Although you can configure the role to limit or grant access to your data source, it is recommended you not adjust the access of this role. **If you delete the data source, this role will be deleted**. This will remove access for any other users if they are mapped to the role.

Configuring and querying a CloudWatch Logs data source in OpenSearch Dashboards

Now that you've created your data source, you can get started with it OpenSearch Dashboards. This section walks you through various use cases with your data source in OpenSearch Dashboards.

Query log groups from the Discover page

In the OpenSearch Discover page, you can use the new direct query data source you configured to query your CloudWatch Logs log groups. To do this, choose **Explore logs**, then use the search bar to build your query using SQL or PPL. You can filter, sort, and visualize the data returned from your log groups. To understand what statements, commands, and limitations are supported for the CloudWatch Logs integration, see the section called “Supported SQL and PPL commands”.

Create a dashboard view for your data source

When you use OpenSearch Service, you can quickly analyze popular AWS log types using pre-built dashboard templates. For CloudWatch Logs there are templates for VPC, CloudTrail, and WAF logs. These templates allow you to quickly create a dashboard tailored to your specific data. They include dashboards tailored for that specific log type. This enables you to quickly get up and running with analyzing these popular AWS log sources, without having to build everything from scratch.

> **Note**
>
> Dashboards use indexed views, which ingest data from CloudWatch Logs using direct query OpenSearch Compute Units (OCUs) as well as serverless collection indexingOCUs, searchingOCUs, and storage.

Follow these steps to create a dashboard using one of these pre-built templates, so you can start exploring and analyzing your data right away.

To create a dashboard view

1. Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/.
2. From the left navigation pane, choose **Central management**, then **Connected data sources**.
3. Select the data source to open the details page.
4. Choose **Create dashboard**.
5. Choose which type of dashboard you want to create.
6. Enter a name for your dashboard.
7. Enter an optional description for your dashboard.
8. Select one or more log groups to view on your dashboard.
9. Choose how often you want to refresh the data in your dashboard.
10. Choose which OpenSearch workspace you want to use.
    a. To create a new workspace, select **Create new workspace** and enter a name.
    b. To use an existing workspace, select **Select existing workspace**.
11. Choose **Create dashboard**.

Querying CloudWatch Logs data in OpenSearch Discover

To query your data, select your data source from the drop-down menu. If you're using CloudWatch Logs, navigate to Discover from your Essentials workspace and start querying data using OpenSearch SQL or Piped Processing Language (PPL). For a list of supported commands, see the section called “Supported SQL and PPL commands”.

> **Note**
>
> If you've created a materialized view, you can use DQL to query your data given that you've indexed it within.

Troubleshooting

There might be instances when results don't return as expected. If you experience any issues, make sure that you're following the the section called "Recommendations".

Directly querying Amazon Security Lake data in OpenSearch Service

This section will walk you through the process of creating and configuring a data source integration in Amazon OpenSearch Service, enabling you to efficiently query and analyze your data stored in Security Lake.

In the following pages, you'll learn how to set up a Security Lake direct-query data source, navigate the necessary prerequisites, and follow step-by-step procedures using the AWS Management Console.

Topics

- Creating an Amazon Security Lake data source integration in OpenSearch Service
- Configuring and querying a Security Lake data source in OpenSearch Dashboards

Creating an Amazon Security Lake data source integration in OpenSearch Service

You can use Amazon OpenSearch Serverless to directly query security data in Amazon Security Lake. To do this, you create a data source that enables you to use OpenSearch zero-ETL capabilities on Security Lake data. When you create a data source you can directly search, gain insights from, and analyze data stored in Security Lake. You can accelerate your query performance and use advanced OpenSearch analytics on select Security Lake data sets using on-demand indexing.

Topics

Prerequisites

Before you get started, make sure that you have reviewed the following documentation:

- the section called “Limitations for Amazon Security Lake”
- the section called “Information for Security Lake”
- the section called “Quotas for Security Lake”

Before you can create a data source, take the following actions in Security Lake:

- **Enable Security Lake**. Configure Security Lake to collect logs on the same AWS Region as your OpenSearch resource. For instructions, see Getting started with Amazon Security Lake in the Amazon Security Lake user guide.
- **Set up Security Lake permissions**. Make sure you have accepted the service linked role permissions for resource management and the console does not show any issues under the **Issues** page. For more information, see Service-linked role for Security Lake in the Amazon Security Lake user guide.
- **Share Security Lake data sources**. When accessing OpenSearch within the same account as Security Lake, ensure that there is no message to register your Security Lake buckets with Lake Formation in the Security Lake console. For cross-account OpenSearch access, set up a Lake Formation query subscriber in the Security Lake console. Use the account associated with your OpenSearch resource as the subscriber. For more information, see Subscriber management in Security Lake in the Amazon Security Lake user guide.

In addition, you must also you must have the following resources in your AWS account:

- **(Optional) A manually created IAM role.** You can use this role to manage access to your data source. Alternatively, you can have OpenSearch Service create a role for you automatically with the required permissions. If you choose to use a manually created IAM role, follow the guidance in the section called “Required permissions for manually created IAM roles”.

Procedure

You can set up a data source to connect with a Security Lake database from within the AWS Management Console.

To set up a data source using the AWS Management Console

1. Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/.
2. In the left navigation pane, go to **Central management** and choose **Connected data sources**.
3. Choose **Connect**.
4. Choose **Security Lake** as the data source type.
5. Choose **Next**.
6. Under **Data connection details**, enter a name and an optional description.
7. Under **IAM permission access settings**, choose how to manage access to your data source.
   a. If you want to automatically create a role for this data source, follow these steps:
      i. Select **Create a new role**.
      ii. Enter a name for the IAM role.
      iii. Select one or more AWS Glue tables to define which data can be queried.
   b. If you want to use an existing role that you manage yourself, follow these steps:
      i. Select **Use an existing role.**
      ii. Select an existing role from the drop-down menu.

> **Note**
>
> When using your own role, you must ensure it has all necessary permissions by attaching required policies from the IAM console. For more information, see the section called “Required permissions for manually created IAM roles”.

8. (Optional) Under **Tags**, add tags to your data source.
9. Choose **Next**.
10. Under **Set up OpenSearch**, choose how to set up OpenSearch.

- Review the default resource names and data retention settings.

  When you use the default settings, a new OpenSearch application and Essentials workspace is created for you at no additional cost. OpenSearch enables you to analyze multiple data sources. It includes workspaces, which provide a tailored experiences for popular use cases. Workspaces support access control, enabling you to create private spaces for your use cases and share them only with your collaborators.

11. Use customized settings:

    a. Choose **Customize**.

    b. Edit the collection name and the data retention settings as needed.

    c. Select the OpenSearch application and workspace that you want to use.

12. Choose **Next**.
13. Review your choices and choose **Edit** if you need to make any changes.
14. Choose **Connect** to set up the data source. Stay on this page while your data source is created. When it's ready, you'll be taken to the data source details page.

Next steps

Visit OpenSearch Dashboards and create a dashboard

After you create a data source, OpenSearch Service provides you with an OpenSearch Dashboards URL. You use this to query your data using SQL or PPL. The Security Lake integration comes with pre-packaged query templates for SQL and PPL to get you get started analyzing your logs.

For more information, see the section called "Configuring a Security Lake data source".

Additional resources

Required permissions for manually created IAM roles

When creating a data source, you choose an IAM role to manage access to your data. You have two options:

1. Create a new IAM role automatically
2. Use an existing IAM role that you created manually

If you use a manually created role, you need to attach the correct permissions to the role. The permissions must allow access to the specific data source, and allow OpenSearch Service to assume the role. This is required so that the OpenSearch Service can securely access and interact with your data.

The following sample policy demonstrates the least-privilege permissions required to create and manage a data source. If you have broader permissions, such as the `AdminstratorAccess` policy, these permissions encompasses the least-privilege permissions in the sample policy.

In the following sample policy, replace the *`placeholder text`* with your own information.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonOpenSearchDirectQueryServerlessAccess",
            "Effect": "Allow",
            "Action": [
                "aoss:APIAccessAll",
                "aoss:DashboardsAccessAll"
            ],
            "Resource": "arn:aws:aoss:region:account:collection/collectionname/*"
        },
        {
            "Sid": "AmazonOpenSearchDirectQueryGlueAccess",
            "Effect": "Allow",
            "Action": [
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:GetTable",
                "glue:GetTableVersions",
                "glue:GetTables",
                "glue:SearchTables",
                "glue:BatchGetPartition"
            ],
            "Resource": [
                "arn:aws:glue:region:account:table/databasename/*",
                "arn:aws:glue:region:account:database/databasename",
                "arn:aws:glue:region:account:catalog",
                "arn:aws:glue:region:account:database/default"
            ]
```

```
        },
        {
            "Sid": "AmazonOpenSearchDirectQueryLakeFormationAccess",
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

The role must also have the following trust policy, which specifies the target ID.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "directquery.opensearchservice.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

For instructions to create the role, see Creating a role using custom trust policies.

By default, the role has access to direct query data source indexes only. Although you can configure the role to limit or grant access to your data source, it is recommended you not adjust the access of this role. **If you delete the data source, this role will be deleted**. This will remove access for any other users if they are mapped to the role.

Querying Security Lake data that's encrypted with a customer managed key

If the Security Lake bucket associated with the data connection is encrypted using server-side encryption with a customer managed AWS KMS key, you must add the LakeFormation service role to the key policy. This allows the service to access and read the data for your queries.

In the following sample policy, replace the *placeholder text* with your own information.

```
{
    "Sid": "Allow LakeFormation to access the key",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::account:role/aws-service-role/lakeformation.amazonaws.com/
AWSServiceRoleForLakeFormationDataAccess"
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
    ],
    "Resource": "*"
}
```

Configuring and querying a Security Lake data source in OpenSearch Dashboards

Now that you've created your data source, you can set it up in OpenSearch Dashboards.

This section walks you through various use cases with your data source in OpenSearch Dashboards before you query your data. To get started, you need to navigate to your data source in OpenSearch Dashboards. In the left-hand menu, under **Management**, choose **Data sources**. Then, select the name of the data source that you created earlier in the OpenSearch Service console.

Query Security Lake tables from Discover

If you have created tables based on your Security Lake logs, you can now query those tables directly from OpenSearch Discover. This enables you to seamlessly access and analyze data stored in Security Lake, directly from the familiar Discover interface. By querying Security Lake directly from Discover, you can avoid the need to manually extract, transform, and load the data into a separate search index. To quickly get started analyzing your logs, Discover includes a set of PPL and SQL saved queries.

Start by selecting the data source that you configured. Select the associated database and table you want to query, then use the search bar to write queries against your tables. To understand what statements, commands, and limitations are supported for the Security Lake integration, see the section called “Supported SQL and PPL commands”.

To take advantage of the pre-built queries that are available for Security Lake, go to **...** on the top right hand side of Discover, choose **Open Query** and then choose **Templates**. There are many pre-built queries available for log sources supported in Security Lake. Search for the templates that match your use case, copy the query to use in the search bar, and replace templated fields (such as Region and action) with your own information.

Accelerate data from Discover

To enhance performance and enable faster subsequent queries and analysis in OpenSearch, you can ingest the results of your query from Discover into an OpenSearch indexed view.

To create an indexed view

1. From Discover, choose **Create Indexed View**.
2. In the query editor, enter your desired query. You can create a new query here or use an existing one from your previous searches.
3. Specify a name for your new indexed view. Choose a descriptive name that will help you identify the view later.
4. Configure the data retention settings for your indexed view. You can specify how long the data should be kept in the index, allowing you to balance performance with storage costs.
5. Create the indexed view. After it's created, your indexed view will be available for faster querying and analysis.

If you've previously created indexed views, you can access them from Discover.

To use an existing index view

1. From Discover, choose **Select Indexed View** to see a list of your existing indexed views for Security Lake.
2. Choose the indexed view you want to use. This will apply the view to your current query, potentially significantly speeding up your data retrieval and analysis.

Create a dashboard view for your data source

When you use OpenSearch Service, you can analyze popular AWS log types using pre-built dashboard templates. For Security Lake there are templates for VPC, CloudTrail, and WAF logs. These templates allow you to create a dashboard tailored to your specific data. They include pre-

built queries and dashboards tailored for that specific log type. This enables you to quickly get up and running with analyzing these popular AWS log sources, without having to build everything from scratch.

> **Note**
>
> Dashboards use indexed views, which ingest data from Security Lake and contribute to direct query and collection compute.

Follow these steps to create a dashboard using one of these pre-built templates, so you can start exploring and analyzing your data right away.

To create a dashboard view

1. Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/.
2. From the left navigation pane, choose **Central management**, then **Connected data sources**.
3. Select the data source to open the details page.
4. Choose **Create dashboard**.
5. Choose which type of dashboard you want to create.
6. Enter a name for your dashboard.
7. Enter an optional description for your dashboard.
8. Select one or more AWS Glue tables to view on your dashboard.
9. Choose how often you want to refresh the data in your dashboard.
10. Choose which OpenSearch workspace you want to use.
    a. To create a new workspace, select **Create new workspace**.
    b. To use an existing workspace, select **Select existing workspace**.
11. Enter a name for your workspace.
12. Choose **Create dashboard**.

Troubleshooting

There might be instances when results don’t return as expected. If you experience any issues, make sure that you're following the the section called “Recommendations”.

Managing a data source in Amazon OpenSearch Service

Managing your data source is an important part of maintaining the reliability, availability, and performance of direct query data sources and your other AWS solutions. AWS provides the following tools to monitor, report when something is wrong, and take automatic actions when appropriate.

Topics

Monitoring with CloudWatch metrics data sources

You can monitor direct query using CloudWatch. CloudWatch collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing.

You can also set alarms to monitor certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see What is Amazon CloudWatch.

Amazon S3 reports the following metrics:

Metric	Description
AsyncQueryCreateAPI	The total number of requests made to the API for creating asynchronous queries. **Relevant statistics**: Average, Maximum, Sum **Dimensions**: `ClientId`,`DomainName` **Frequency**: 60 seconds
AsyncQueryGetApiRequestCount	The total number of requests made to the API for retrieving asynchronous query results.

Metric	Description
	Relevant statistics: Average, Maximum, Sum **Dimensions**: `ClientId`, `DomainName` **Frequency**: 60 seconds
AsyncQueryCancelApiRequestCount	The total number of requests made to the API for canceling asynchronous queries. **Relevant statistics**: Average, Maximum, Sum **Dimensions**: `ClientId`, `DomainName` **Frequency**: 60 seconds
AsyncQueryGetApiFailedRequestCusErrCount	The number of failed requests when retrieving asynchronous query results due to customer-related errors (e.g., invalid query ID). **Relevant statistics**: Average, Maximum, Sum **Dimensions**: `ClientId`, `DomainName` **Frequency**: 60 seconds
AsyncQueryCancelApiFailedRequestCusErrCount	The number of failed requests when retrieving asynchronous query results due to customer-related errors (e.g., invalid query ID). **Relevant statistics**: Average, Maximum, Sum **Dimensions**: `ClientId`,`DomainName` **Frequency**: 60 seconds

Metric	Description
AsyncQueryCancelApiFailedRequestSysErrCount	The number of failed requests when creating asynchronous queries due to customer-related errors. **Relevant statistics**: Average, Maximum, Sum **Dimensions**: `ClientId`, `DomainName` **Frequency**: 60 seconds
AsyncQueryGetApiFailedRequestSysErrCount	The number of failed requests when retrieving asynchronous query results due to system-related errors. **Relevant statistics**: Average, Maximum, Sum **Dimensions**: `ClientId`, `DomainName` **Frequency**: 60 seconds

CloudWatch Logs and Security Lake report the following metrics:

Metric	Description
DirectQueryRate	The rate of requests made against the data sources. **Relevant statistics**: Sum, Maximum, Minimum, Average **Dimensions**: `DataSourceName` **Frequency**: 60 seconds
DirectQueryLatency	The latency observed for running queries on the data sources. **Relevant statistics**: Average, P90, P99, Sum, Minimum, Maximum **Dimensions**: `DataSourceName`

Metric	Description
	Frequency: 60 seconds
FailedDirectQueries	The total number of query failures that are observed on the data source queries. **Relevant statistics**: Sum, Maximum, Minimum, Average **Dimensions**: `DataSourceName` **Frequency**: 60 seconds
DirectQueryConsumedOCU	The number of OCUs that are consumed for running the queries on the data sources. **Relevant statistics**: Average, P90, P99, Sum, Minimum, Maximum **Dimensions**: `DataSourceName` **Frequency**: 60 seconds

Enabling and disabling data sources

> **Note**
>
> The following information is only applicable to Amazon S3 data sources.

For circumstances when you want to halt direct query usage for a data source, you can opt to disable the data source. Disabling a data source will finish executing existing queries and halt all new queries from being executed.

Accelerations setup to boost query performance such as skipping indexes, materialized views, covering indexes will be set to manual once a data source is disabled. Once a data source is set to active after being disabled, user queries will run as expected. Accelerations which were previously setup and set to manual, will need to be manually configured to run on a schedule again.

Monitoring with AWS Budget

Amazon OpenSearch Service is populating OCU usage data at the account level into Billing and Cost Management's Cost Explorer. You can account for OCU usage at the account level and set thresholds and alerts when thresholds have been crossed.

The format of the usage type to filter on in Cost Explorer looks like RegionCode-DirectQueryOCU (OCU-hours). If you want to be notified when DirectQueryOCU (OCU-Hours) usage meets your threshold, you can create an AWS Budgets account and configure an alert based on the threshold you set. Optionally for Amazon S3, you can set up an Amazon SNS topic, which will turn off a data source in the event a threshold criteria is met.

> **Note**
>
> Usage data in AWS Budgets is not real-time and may be delayed up to 8 hours.

Deleting a data source

When you delete a data source, Amazon OpenSearch Service removes it from your domain or your collection. OpenSearch Service also removes indexes associated with the data source. Your transactional data isn't deleted from the other AWS service, but the other AWS service doesn't send new data to OpenSearch Service.

You can delete a data source integration using the AWS Management Console or the OpenSearch Service API.

AWS Management Console

To delete an Amazon S3 data source

1. Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/.
2. From the left navigation pane, choose **Domains**.
3. Select the domain that you want to delete a data source for. This opens the domain details page. Choose the **Connections** tab below the general information and find the **Direct query** section.
4. Select the data source you want to delete, choose **Delete**, and confirm deletion.

To delete a CloudWatch Logs or Security Lake data source

1. Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/.
2. From the left navigation pane, choose **Central management**, then **Connected data sources**.
3. Select the data source you want to delete, choose **Delete**, and confirm deletion.

OpenSearch Service API

To delete an Amazon S3 data source, use the DeleteDataSource API operation.

```
POST https://es.region.amazonaws.com/2021-01-01/opensearch/domain/domain-name/
dataSource/data-source-name
```

To delete a CloudWatch Logs or Security Lake data source, use the DeleteDirectQueryDataSource API operation.

Optimizing query performance for Amazon OpenSearch Service data sources

Query performance in Amazon OpenSearch Service might slow down when you're accessing external data sources. This can be due to factors like network latency, data transformation, or large data volumes. To improve performance, consider indexing select amounts of data depending on the use case:

- Speeding up direct queries on Amazon S3 (skipping index)
- Building dashboard visualizations on Amazon S3, CloudWatch Logs, and Security Lake (materialized views)
- Ingesting query results using indexed views for offline review or improved performance on Security Lake (materialized views)

For full documentation on accelerated queries, including example queries, see Optimize query performance using OpenSearch indexing in the open source documentation.

Topics

- Skipping indexes

- Materialized views
- Covering indexes

Skipping indexes

A skipping index ingests only the metadata of data stored in Amazon S3. When you query a table with a skipping index, the query planner uses the index to rewrite the query, efficiently identifying the location of the data without scanning all partitions and files. This approach helps narrow down the exact location of the stored data.

There are two ways to create a skipping index. The first way is to autogenerate the skipping index from within data source details. The second is to use Query Workbench to manually create the skipping index using a SQL statement.

To autogenerate a skipping index from your data source, go to **Dashboard management** and **Accelerate data**, then select your database and table (you might need to refresh to get the latest databases and tables). You can then choose **Generate** to autogenerate a skipping index, or manually select each field you want to index and specify the acceleration (skipping index type). Lastly, choose **Create acceleration** to create a reoccurring job that populates the new skipping index.

Skipping indexes are supported only for Amazon S3 data sources.

For more information about setting up skipping indexes using Query Workbench, see Skipping indexes in the OpenSearch documentation.

Materialized views

Materialized views use complex queries, such as aggregations, to support OpenSearch Dashboards visualizations. They ingest a subset of your data based on the query and store it in an OpenSearch index. You can then use this index to create visualizations.

Materialized views are supported for Amazon S3, CloudWatch Logs, and Security Lake data sources.

For more information about setting up materialized views using Query Workbench, see Materialized views in the OpenSearch documentation.

Covering indexes

A covering index ingests data from a specified column in a table, and OpenSearch creates a new index based on this data. You can use this new index for visualizations and other OpenSearch features, such as anomaly detection or geospatial analysis.

Covering indexes are supported only for Amazon S3 data sources.

For more information about setting up covering indexes, see Covering indexes in the OpenSearch documentation.

Supported SQL and PPL commands

OpenSearch SQL and OpenSearch Pipeline Processing Language (PPL) are languages for querying, analyzing, and processing data in OpenSearch, CloudWatch Logs Insights, and Security Lake. You can use OpenSearch SQL and OpenSearch PPL in OpenSearch Discover to query data within CloudWatch Logs, Amazon S3, or Security Lake. CloudWatch Logs Insights also supports both OpenSearch PPL and OpenSearch SQL query languages, in addition to Logs Insights QL, a purpose-built query language for analyzing CloudWatch Logs.

- **OpenSearch SQL**: OpenSearch SQL provides a familiar option if you're used to working with relational databases. OpenSearch SQL offers a subset of SQL functionality, making it a good choice for performing ad-hoc queries and data analysis tasks. With OpenSearch SQL, you can use commands such as SELECT, FROM, WHERE, GROUP BY, HAVING, and various other SQL commands and functions available in SQL. You can execute JOINs across tables (or log groups), correlate data across tables (or log groups) using subqueries, and use the rich set of JSON, mathematical, string, conditional, and other SQL functions to perform powerful analysis on log and security data.
- **OpenSearch PPL (Piped Processing Language):** With OpenSearch PPL, you can retrieve, query, and analyze data using piped-together commands, making it easier to understand and compose complex queries. Its syntax is based on Unix pipes, and enables chaining of commands to transform and process data. With PPL, you can filter and aggregate data, and use commands such as JOINs, subqueries, LOOKUP, and a rich set of math, string, date, conditional, and other functions for analysis.

Although most of the commands in OpenSearch PPL and OpenSearch SQL query languages are common across CloudWatch Logs and OpenSearch, there are some differences in which set of

commands and functions are supported in each of these services. For more details, see the tables on the following pages.

- the section called “Supported SQL commands”
  - the section called “Additional information for CloudWatch Logs Insights users using OpenSearch SQL”
  - the section called “General SQL restrictions”
- the section called “Supported PPL commands”
  - the section called “Additional information for CloudWatch Logs Insights users using OpenSearch PPL”

Supported OpenSearch SQL commands and functions

The following reference tables show which SQL commands are supported in OpenSearch Discover for querying data in Amazon S3, Security Lake, or CloudWatch Logs, and which SQL commands are supported in CloudWatch Logs Insights. The SQL syntax supported in CloudWatch Logs Insights and that supported in OpenSearch Discover for querying CloudWatch Logs are the same, and referenced as CloudWatch Logs in the following tables.

> **Note**
>
> OpenSearch also has SQL support for querying data that is ingested in OpenSearch and stored in indexes. This SQL dialect is different than the SQL used in direct query and is referred to as OpenSearch SQL on indexes.

Topics

- Commands
- Functions
- General SQL restrictions
- Additional information for CloudWatch Logs Insights users using OpenSearch SQL

Commands

> **Note**
>
> In the example commands column, replace *<tableName/logGroup>* as needed depending on which data source you're querying.
>
> - Example command: `SELECT Body , Operation FROM` **`<tableName/logGroup>`**
> - If you're querying Amazon S3 or Security Lake, use: `SELECT Body , Operation FROM` **`table_name`**
> - If you're querying CloudWatch Logs, use: `SELECT Body , Operation FROM` **`` `LogGroupA` ``**

Commanc	Descript on	CloudW: h Logs	Amazon S3	Security Lake	Example command
the section called "SELECT clause"	Displays projecte values.	Supporte	Supporte	Supporte	`SELECT method, status FROM <tableName/logGroup>`
the section called "WHERE clause"	Filters log events based on the providec field criteria.	Supporte	Supporte	Supporte	`SELECT * FROM <tableName/logGroup> WHERE status = 100`
the section called "GROUP	Groups log events based on	Supporte	Supporte	Supporte	`SELECT method, status, COUNT(*) AS request_count, SUM(bytes) AS total_bytes`

Commanc	Descript on	CloudW: h Logs	Amazon S3	Security Lake	Example command
BY clause"	category and finds the average based on stats.				FROM <tableName/logGroup> GROUP BY method, status
the section called "HAVING clause"	Filters the results based on groupinc conditio s.	Support	Support	Support	SELECT method, status, COUNT(*) AS request_count, SUM(bytes) AS total_bytes FROM <tableName/logGroup> GROUP BY method, status HAVING COUNT(*) > 5

Commanc	Descript on	CloudW h Logs	Amazon S3	Security Lake	Example command
the section called "ORDER BY clause"	Orders the results based on fields in the order clause. You can sort in either descendi g or ascendin order.	Support	Support	Support	`SELECT * FROM <tableName/logGroup> ORDER BY status DESC`

Commanc	Descript on	CloudW h Logs	Amazon S3	Security Lake	Example command

the section called "JOIN clause"
 	
(INNER | CROSS | LEFT OUTER)
	Joins the results for two tables based on commor fields.	Support (must use `Inner` and `Left Outer` keyword for join; Only only one JOIN operatio is supporte in a SELECT statemer)	Support (must use Inner, Left Outer, and Cross keyword for join)	Support (must use Inner, Left Outer, and Cross keyword for join)	`SELECT`
					`  A.Body,`
					`  B.Timestamp`
					`FROM`
					`  <tableNameA/logGroupA>  AS A`
					`INNER JOIN`
					`  <tableNameB/logGroupB>  AS B`
					`  ON A.`requestId` = B.`requestId``

the section called "LIMIT clause"	Restricts the results to first N rows.	Support	Support	Support	`SELECT`
					`  *`
					`FROM`
					`  <tableName/logGroup>`
					`LIMIT`
					`  10`

Commanc	Descript on	CloudW h Logs	Amazon S3	Security Lake	Example command
the section called “CASE clause”	Evaluate s conditio s and returns a value when the first conditio is met.	Support	Support	Support	SELECT method, status, CASE WHEN status BETWEEN 100 AND 199 THEN 'Informational' WHEN status BETWEEN 200 AND 299 THEN 'Success' WHEN status BETWEEN 300 AND 399 THEN 'Redirection' WHEN status BETWEEN 400 AND 499 THEN 'Client Error' WHEN status BETWEEN 500 AND 599 THEN 'Server Error' ELSE 'Unknown Status' END AS status_category, CASE method WHEN 'GET' THEN 'Read Operation' WHEN 'POST' THEN 'Create Operation' WHEN 'PUT' THEN 'Update Operation' WHEN 'PATCH' THEN 'Partial Update Operation' WHEN 'DELETE' THEN 'Delete Operation' ELSE 'Other Operation' END AS operation_type, bytes, datetime FROM *<tableName/logGroup>*

Commanc	Descript on	CloudW h Logs	Amazon S3	Security Lake	Example command

the section called “Common table expressio n”	Creates a named tempora result set within a SELECT, INSERT, UPDATE, DELETE, or MERGE statemei .	Not supporte	Support	Support	`WITH RequestStats AS (`
					`  SELECT`
					`  method,`
					`  status,`
					`  bytes,`
					`  COUNT(*) AS request_count`
					`  FROM`
					`  tableName`
					`  GROUP BY`
					`  method,`
					`  status,`
					`  bytes`
					`)`
					`SELECT`
					`  method,`
					`  status,`
					`  bytes,`
					`  request_count`
					`FROM`
					`  RequestStats`
					`WHERE`
					`  bytes > 1000`

the section called “EXPLAIN”	Displays the executio plan of a SQL statemei without actually executin it.	Not supporte	Support	Support	`EXPLAIN`
					`SELECT`
					`  k,`
					`  SUM(v)`
					`FROM`
					`  VALUES`
					`  (1, 2),`
					`  (1, 3) AS t(k, v)`
					`GROUP BY`
					`  k`

Commanc	Descript on	CloudW h Logs	Amazon S3	Security Lake	Example command
the section called “LATERAL SUBQUER clause”	Allows a subquer in the FROM clause to referenc columns from precedin items in the same FROM clause.	Not supporte	Support	Support	`SELECT * FROM tableName LATERAL ( SELECT * FROM t2 WHERE t1.c1 = t2.c1 )`
the section called “LATERAL VIEW clause”	Generat a virtual table by applying a table-gen erating function to each row of a base table.	Not supporte	Support	Support	`SELECT * FROM tableName LATERAL VIEW EXPLODE(ARRAY(30, 60)) tableName AS c_age LATERAL VIEW EXPLODE(ARRAY(40, 80)) AS d_age`

Commanc	Descript on	CloudW h Logs	Amazon S3	Security Lake	Example command
the section called "LIKE predicate " –	Matches a string against a pattern using wildcard characte s.	Support	Support	Support	`SELECT method, status, request, host FROM <tableName/logGroup> WHERE method LIKE 'D%'`

Commanc	Descript on	CloudW h Logs	Amazon S3	Security Lake	Example command
the section called "OFFSET"	Specifies the number of rows to skip before starting to return rows from the query.	Supporte when used in conjunct on with a LIMIT clause in a query. For example • Suppo : SELEC * FROM Table LIMIT 100 OFFSE 10 • Not suppo : SELEC * FROM Table	Supporte	Supporte	`SELECT method, status, bytes, datetime FROM <tableName/logGroup> ORDER BY datetime OFFSET 10`

Commanc	Descript on	CloudW h Logs	Amazon S3	Security Lake	Example command
		OFFSE 10			
the section called "PIVOT clause"	Transfor s rows into columns rotating data from a row-based format to a column-based format.	Not supporte	Support	Support	`SELECT * FROM ( SELECT method, status, bytes FROM <tableName/logGroup> ) AS SourceTable PIVOT ( SUM(bytes) FOR method IN ('GET', 'POST', 'PATCH', 'PUT', 'DELETE') ) AS PivotTable`

Commanc	Descript on	CloudW h Logs	Amazon S3	Security Lake	Example command

the section called "Set operators" –	Combine the results of two or more SELECT statemer s (e.g., UNION, INTERSE , EXCEPT)	Support	Support	Support	`SELECT method, status, bytes FROM <tableName/logGroup> WHERE status = '416'`
					
					`UNION`
					
					`SELECT method, status, bytes FROM <tableName/logGroup> WHERE bytes > 20000`

the section called "SORT BY clause"	Specifies the order in which to return the query results.	Support	Support	Support	`SELECT method, status, bytes FROM <tableName/logGroup> SORT BY bytes DESC`

Commanc	Descript on	CloudW: h Logs	Amazon S3	Security Lake	Example command
the section called "UNPIVOT	Transfor s columns into rows, rotating data from a column-based format to a row-based format.	Not supporte	Support	Support	`SELECT status, REPLACE(method, '_bytes', '') AS request_method, bytes, datetime FROM PivotedData UNPIVOT ( bytes FOR method IN ( GET_bytes, POST_bytes, PATCH_bytes, PUT_bytes, DELETE_bytes ) ) AS UnpivotedData`

Functions

Note

In the example commands column, replace *<tableName/logGroup>* as needed depending on which data source you're querying.

- Example command: `SELECT Body , Operation FROM` **`<tableName/logGroup>`**
- If you're querying Amazon S3 or Security Lake, use: `SELECT Body , Operation FROM` **`table_name`**
- If you're querying CloudWatch Logs, use: `SELECT Body , Operation FROM` **`` `LogGroupA` ``**

Availab SQL Gramm	Description	CloudW h Logs	Amazor S3	Security Lake	Example command
the section called "String functior "	Built-in functions that can manipulate and transform string and text data within SQL queries. For example, convertin g case, combining strings, extracting parts, and cleaning text.	Support	Support	Support	`SELECT UPPER(method) AS upper_met hod, LOWER(host) AS lower_host FROM <tableName/logGroup>`
the section called "Date and time functior "	Built-in functions for handling and transform ing date and timestamp data in queries. For example, **date_add**, **date_format**, **datediff**, and **current_date**.	Support	Support	Support	`SELECT TO_TIMESTAMP(datetime) AS timestamp, TIMESTAMP_SECONDS(UNIX_TIME STAMP(datetime)) AS from_seco nds, UNIX_TIMESTAMP(datetime) AS to_unix, FROM_UTC_TIMESTAMP(datetime, 'PST') AS to_pst, TO_UTC_TIMESTAMP(datetime, 'EST') AS from_est FROM <tableName/logGroup>`
the section	Built-in functions	S	Support	Support	`SELECT COUNT(*) AS total_records,`

Availab SQL Gramm	Description	CloudW h Logs	Amazor S3	Security Lake	Example command
called "Aggreg e functior "	that perform calculations on multiple rows to produce a single summarize d value. For example, **sum**, **count**, **avg**, **max**, and **min**.				`COUNT(DISTINCT method) AS unique_methods, SUM(bytes) AS total_bytes, AVG(bytes) AS avg_bytes, MIN(bytes) AS min_bytes, MAX(bytes) AS max_bytes FROM <tableName/logGroup>`
the section called "Conditi nal functior "	Built-in functions that perform actions based on specified conditions, or that evaluate expressions conditionally. For example, **CASE** and **IF**.	Support	Support	Support	`SELECT CASE WHEN method = 'GET' AND bytes < 1000 THEN 'Small Read' WHEN method = 'POST' AND bytes > 10000 THEN 'Large Write' WHEN status >= 400 OR bytes = 0 THEN 'Problem' ELSE 'Normal' END AS request_type FROM <tableName/logGroup>`

Availab SQL Gramm	Description	CloudW h Logs	Amazor S3	Security Lake	Example command
the section called "JSON functior "	Built-in functions for parsing, extracting, modifying, and querying JSON-form atted data within SQL queries (e.g., from_json , to_json, get_json_ object, json_tupl e) allowing manipulat ion of JSON structures in datasets.	Support	Support	Support	`SELECT FROM_JSON( @message, 'STRUCT< host: STRING, user-identifier: STRING, datetime: STRING, method: STRING, status: INT, bytes: INT >' ) AS parsed_json FROM <tableName/logGroup>`

Availab SQL Gramm	Description	CloudW h Logs	Amazor S3	Securit Lake	Example command
the section called "Array functior "	Built-in functions for working with array-type columns in SQL queries, allowing operations like accessing , modifying, and analyzing array data (e.g., size, explode, array_con tains).	Support	Support	Support	`SELECT scores, size(scores) AS length, array_contains(scores, 90) AS has_90 FROM <tableName/logGroup>`

Availab SQL Gramm	Description	CloudW h Logs	Amazor S3	Security Lake	Example command
the section called "Windo functior "	Built-in functions that perform calculations across a specified set of rows related to the current row (window), enabling operations like ranking, running totals, and moving averages (e.g., ROW_NUMBER, RANK, LAG, LEAD)	Support	Support	Support	`SELECT field1, field2, RANK() OVER (ORDER BY field2 DESC) AS field2Rank FROM <tableName/logGroup>`

Availab SQL Gramm	Description	CloudW h Logs	Amazor S3	Security Lake	Example command
the section called "Conver on functior "	Built-in functions for converting data from one type to another within SQL queries, enabling data type transform ations and format conversions (e.g., CAST, TO_DATE, TO_TIMEST AMP, BINARY)	Support	Support	Support	`SELECT CAST('123' AS INT) AS converted_number, CAST(123 AS STRING) AS converted_string FROM <tableName/logGroup>`
the section called "Predica e functior "	Built-in functions that evaluate conditions and return boolean values (true/ false) based on specified criteria or patterns (e.g., IN, LIKE, BETWEEN, IS NULL, EXISTS)	Support	Support	Support	`SELECT * FROM <tableName/logGroup> WHERE id BETWEEN 50000 AND 75000`

Availab SQL Gramm	Description	CloudW h Logs	Amazor S3	Security Lake	Example command
the section called "Map functior "	Applies a specified function to each element in a collection, transforming the data into a new set of values.	Not support	Support	Support	`SELECT MAP_FILTER( MAP( 'method', method, 'status', CAST(status AS STRING), 'bytes', CAST(bytes AS STRING) ), (k, v) -> k IN ('method', 'status') AND v != 'null' ) AS filtered_map FROM <tableName/logGroup> WHERE status = 100`
the section called "Mather ical functior "	Performs mathemati cal operation s on numeric data, such as calculati ng averages, sums, or trigonometric values.	Support	Support	Support	`SELECT bytes, bytes + 1000 AS added, bytes - 1000 AS subtracted, bytes * 2 AS doubled, bytes / 1024 AS kilobytes, bytes % 1000 AS remainder FROM <tableName/logGroup>`

Availab SQL Gramm	Description	CloudW h Logs	Amazor S3	Security Lake	Example command
the section called "Multi-log group functior "	Enables users to specify multiple log groups in a SQL SELECT statement	Support	Not applicab e	Not applicab e	`SELECT lg1.Column1, lg1.Column2 FROM `logGroups(logGroupIdentifier: ['LogGroup1', 'LogGroup2'])` AS lg1 WHERE lg1.Column3 = "Success"`
the section called "Genera r functior "	Creates an iterator object that yields a sequence of values, allowing for efficient memory usage in large data sets.	Not support	Support	Support	`SELECT explode(array(10, 20))`

General SQL restrictions

The following restrictions apply when using OpenSearch SQL with CloudWatch Logs, Amazon S3, and Security Lake.

1. You can only use one JOIN operation in a SELECT statement.
2. Only one level of nested subqueries is supported.
3. Multiple statement queries separated by semi-colons aren't supported.
4. Queries containing field names that are identical but differ only in case (such as field1 and FIELD1) are not supported.

   For example, the following queries are not supported:

```
Select AWSAccountId, awsaccountid from LogGroup
```

However, the following query is because the field name (@logStream) is identical in both log groups:

```
Select a.`@logStream`, b.`@logStream` from Table A INNER Join Table B on a.id = b.id
```

5. Functions and expressions must operate on field names and be part of a SELECT statement with a log group specified in the FROM clause.

   For example, this query is not supported:

   ```
   SELECT cos(10) FROM LogGroup
   ```

   This query is supported:

   ```
   SELECT cos(field1) FROM LogGroup
   ```

Additional information for CloudWatch Logs Insights users using OpenSearch SQL

If you're a CloudWatch Logs user, you can use OpenSearch SQL in the Logs Insights console, API or CLI. Most of the OpenSearch SQL commands such as SELECT, FROM, WHERE, GROUP BY, HAVING, JOINS, and nested queries are supported, including JSON, mathematics, string, conditional, and other functions. However, some commands and functions which aren't supported when querying CloudWatch Logs. For example, queries involving DDL and DML statements aren't supported because CloudWatch Logs only allows read operations. For a detailed list of query commands and functions supported in CloudWatch Logs, see the CloudWatch Logs columns in the above tables.

Multi-log group functions

CloudWatch Logs Insights supports the ability to query multiple log groups. To address this use case in SQL, you can use the `logGroups` command. This command is specific to querying data in CloudWatch Logs Insights involving one or more log groups. Using this syntax, you can query multiple log groups easily by specifying them in the command, instead of writing a query for each of the log groups and combining them with a `UNION` command.

Syntax:

```
`logGroups(
    logGroupIdentifier: ['LogGroup1','LogGroup2', ...'LogGroupn']
)
```

In this syntax, you can specify up to 50 log groups in the `logGroupIndentifier` parameter. To reference log groups in a monitoring account, use ARNs instead of `LogGroup` names.

Example query:

```
SELECT LG1.Column1, LG1.Column2 from `logGroups(
    logGroupIdentifier: ['LogGroup1', 'LogGroup2']
)` as LG1
WHERE LG1.Column1 = 'ABC'
```

The following syntax involving multiple log groups after the `FROM` statement is not supported when querying CloudWatch Logs:

```
SELECT Column1, Column2 FROM 'LogGroup1', 'LogGroup2', ...'LogGroupn'
WHERE Column1 = 'ABC'
```

Restrictions

When using SQL or PPL commands, enclose certain fields in backticks to successfully query them. Backticks are needed for fields with special characters (non-alphabetic and non-numeric). For example, enclose `@message`, `Operation.Export,` and `Test::Field` in backticks. You don't need to enclose columns with purely alphabetic names in backticks.

Example query with simple fields:

```
SELECT SessionToken, Operation, StartTime  FROM `LogGroup-A`
LIMIT 1000;
```

Same query with backticks appended:

```
SELECT `SessionToken`, `Operation`, `StartTime`  FROM `LogGroup-A`
LIMIT 1000;
```

For additional general restrictions that aren't specific to CloudWatch Logs, see the section called “General SQL restrictions”.

Sample queries and quotas

> **Note**
>
> The following applies to both CloudWatch Logs Insights users and OpenSearch users querying CloudWatch data.

For sample SQL queries that you can use in CloudWatch Logs, see **Saved and sample queries** in the Amazon CloudWatch Logs Insights console for examples.

For information about the limits that apply when querying CloudWatch Logs from OpenSearch Service, see CloudWatch Logs quotas in the Amazon CloudWatch Logs User Guide. Limits involve the number of CloudWatch Log groups you can query, the maximum concurrent queries that you can execute, the maximum query execution time, and the maximum number of rows returned in results. The limits are the same regardless of which language you use for querying CloudWatch Logs (namely, OpenSearch PPL, SQL, and Logs Insights).

SQL commands

Topics

- GROUP BY clause
- HAVING clause
- ORDER BY clause
- JOIN clause
- LIMIT clause
- CASE clause
- Common table expression
- EXPLAIN
- LATERAL SUBQUERY clause
- LATERAL VIEW clause
- LIKE predicate
- OFFSET
- PIVOT clause
- Set operators
- SORT BY clause
- UNPIVOT

String functions

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called "Supported SQL commands".

Function	Description
ascii(str)	Returns the numeric value of the first character of `str`.
base64(bin)	Converts the argument from a binary `bin` to a base 64 string.

Function	Description
bit_length(expr)	Returns the bit length of string data or number of bits of binary data.
btrim(str)	Removes the leading and trailing space characters from `str`.
btrim(str, trimStr)	Remove the leading and trailing `trimStr` characters from `str`.
char(expr)	Returns the ASCII character having the binary equivalent to `expr`. If n is larger than 256 the result is equivalent to chr(n % 256)
char_length(expr)	Returns the character length of string data or number of bytes of binary data. The length of string data includes the trailing spaces. The length of binary data includes binary zeros.
character_length(expr)	Returns the character length of string data or number of bytes of binary data. The length of string data includes the trailing spaces. The length of binary data includes binary zeros.
chr(expr)	Returns the ASCII character having the binary equivalent to `expr`. If n is larger than 256 the result is equivalent to chr(n % 256)
concat_ws(sep[, str | array(str)]+)	Returns the concatenation of the strings separated by `sep`, skipping null values.
contains(left, right)	Returns a boolean. The value is True if right is found inside left. Returns NULL if either input expression is NULL. Otherwise, returns False. Both left or right must be of STRING or BINARY type.

Function	Description
decode(bin, charset)	Decodes the first argument using the second argument character set.
decode(expr, search, result [, search, result] ... [, default])	Compares expr to each search value in order. If expr is equal to a search value, decode returns the corresponding result. If no match is found, then it returns default. If default is omitted, it returns null.
elt(n, input1, input2, ...)	Returns the n-th input, e.g., returns `input2` when n is 2.
encode(str, charset)	Encodes the first argument using the second argument character set.
endswith(left, right)	Returns a boolean. The value is True if left ends with right. Returns NULL if either input expression is NULL. Otherwise, returns False. Both left or right must be of STRING or BINARY type.
find_in_set(str, str_array)	Returns the index (1-based) of the given string (`str`) in the comma-delimited list (`str_array`). Returns 0, if the string was not found or if the given string (`str`) contains a comma.
format_number(expr1, expr2)	Formats the number `expr1` like '#,###,###.##', rounded to `expr2` decimal places. If `expr2` is 0, the result has no decimal point or fractional part. `expr2` also accept a user specified format. This is supposed to function like MySQL's FORMAT.
format_string(strfmt, obj, ...)	Returns a formatted string from printf-style format strings.

Function	Description
initcap(str)	Returns `str` with the first letter of each word in uppercase. All other letters are in lowercase . Words are delimited by white space.
instr(str, substr)	Returns the (1-based) index of the first occurrence of `substr` in `str`.
lcase(str)	Returns `str` with all characters changed to lowercase.
left(str, len)	Returns the leftmost `len`(`len` can be string type) characters from the string `str`,if `len` is less or equal than 0 the result is an empty string.
len(expr)	Returns the character length of string data or number of bytes of binary data. The length of string data includes the trailing spaces. The length of binary data includes binary zeros.
length(expr)	Returns the character length of string data or number of bytes of binary data. The length of string data includes the trailing spaces. The length of binary data includes binary zeros.
levenshtein(str1, str2[, threshold])	Returns the Levenshtein distance between the two given strings. If threshold is set and distance more than it, return -1.
locate(substr, str[, pos])	Returns the position of the first occurrence of `substr` in `str` after position `pos`. The given `pos` and return value are 1-based.
lower(str)	Returns `str` with all characters changed to lowercase.

Function	Description
lpad(str, len[, pad])	Returns `str`, left-padded with `pad` to a length of `len`. If `str` is longer than `len`, the return value is shortened to `len` characters or bytes. If `pad` is not specified, `str` will be padded to the left with space characters if it is a character string, and with zeros if it is a byte sequence.
ltrim(str)	Removes the leading space characters from `str`.
luhn_check(str)	Checks that a string of digits is valid according to the Luhn algorithm. This checksum function is widely applied on credit card numbers and government identification numbers to distinguish valid numbers from mistyped, incorrect numbers.
mask(input[, upperChar, lowerChar, digitChar, otherChar])	masks the given string value. The function replaces characters with 'X' or 'x', and numbers with 'n'. This can be useful for creating copies of tables with sensitive information removed.
octet_length(expr)	Returns the byte length of string data or number of bytes of binary data.
overlay(input, replace, pos[, len])	Replace `input` with `replace` that starts at `pos` and is of length `len`.
position(substr, str[, pos])	Returns the position of the first occurrence of `substr` in `str` after position `pos`. The given `pos` and return value are 1-based.
printf(strfmt, obj, ...)	Returns a formatted string from printf-style format strings.

Function	Description
regexp_count(str, regexp)	Returns a count of the number of times that the regular expression pattern `regexp` is matched in the string `str`.
regexp_extract(str, regexp[, idx])	Extract the first string in the `str` that match the `regexp` expression and corresponding to the regex group index.
regexp_extract_all(str, regexp[, idx])	Extract all strings in the `str` that match the `regexp` expression and corresponding to the regex group index.
regexp_instr(str, regexp)	Searches a string for a regular expression and returns an integer that indicates the beginning position of the matched substring . Positions are 1-based, not 0-based. If no match is found, returns 0.
regexp_replace(str, regexp, rep[, position])	Replaces all substrings of `str` that match `regexp` with `rep`.
regexp_substr(str, regexp)	Returns the substring that matches the regular expression `regexp` within the string `str`. If the regular expression is not found, the result is null.
repeat(str, n)	Returns the string which repeats the given string value n times.
replace(str, search[, replace])	Replaces all occurrences of `search` with `replace`.
right(str, len)	Returns the rightmost `len`(`len` can be string type) characters from the string `str`,if `len` is less or equal than 0 the result is an empty string.

Function	Description
rpad(str, len[, pad])	Returns `str`, right-padded with `pad` to a length of `len`. If `str` is longer than `len`, the return value is shortened to `len` character s. If `pad` is not specified, `str` will be padded to the right with space characters if it is a character string, and with zeros if it is a binary string.
rtrim(str)	Removes the trailing space characters from `str`.
sentences(str[, lang, country])	Splits `str` into an array of array of words.
soundex(str)	Returns Soundex code of the string.
space(n)	Returns a string consisting of n spaces.
split(str, regex, limit)	Splits `str` around occurrences that match `regex` and returns an array with a length of at most `limit`
split_part(str, delimiter, partNum)	Splits `str` by delimiter and return requested part of the split (1-based). If any input is null, returns null. if `partNum` is out of range of split parts, returns empty string. If `partNum` is 0, throws an error. If `partNum` is negative, the parts are counted backward from the end of the string. If the `delimiter` is an empty string, the `str` is not split.
startswith(left, right)	Returns a boolean. The value is True if left starts with right. Returns NULL if either input expression is NULL. Otherwise, returns False. Both left or right must be of STRING or BINARY type.

Function	Description
substr(str, pos[, len])	Returns the substring of `str` that starts at `pos` and is of length `len`, or the slice of byte array that starts at `pos` and is of length `len`.
substr(str FROM pos[FOR len]])	Returns the substring of `str` that starts at `pos` and is of length `len`, or the slice of byte array that starts at `pos` and is of length `len`.
substring(str, pos[, len])	Returns the substring of `str` that starts at `pos` and is of length `len`, or the slice of byte array that starts at `pos` and is of length `len`.
substring(str FROM pos[FOR len]])	Returns the substring of `str` that starts at `pos` and is of length `len`, or the slice of byte array that starts at `pos` and is of length `len`.
substring_index(str, delim, count)	Returns the substring from `str` before `count` occurrences of the delimiter `delim`. If `count` is positive, everything to the left of the final delimiter (counting from the left) is returned. If `count` is negative, everything to the right of the final delimiter (counting from the right) is returned. The function substring _index performs a case-sensitive match when searching for `delim`.
to_binary(str[, fmt])	Converts the input `str` to a binary value based on the supplied `fmt`. `fmt` can be a case-insensitive string literal of "hex", "utf-8", "utf8", or "base64". By default, the binary format for conversion is "hex" if `fmt` is omitted. The function returns NULL if at least one of the input parameters is NULL.

Function	Description
to_char(numberExpr, formatExpr)	Convert `numberExpr` to a string based on the `formatExpr`. Throws an exception if the conversion fails. The format can consist of the following characters, case insensitive: '0' or '9': Specifies an expected digit between 0 and 9. A sequence of 0 or 9 in the format string matches a sequence of digits in the input value, generating a result string of the same length as the corresponding sequence in the format string. The result string is left-padded with zeros if the 0/9 sequence comprises more digits than the matching part of the decimal value, starts with 0, and is before the decimal point. Otherwise, it is padded with spaces. '.' or 'D': Specifies the position of the decimal point (optional, only allowed once). ',' or 'G': Specifies the position of the grouping (thousands) separator (,). There must be a 0 or 9 to the left and right of each grouping separator. '

Function	Description
to_number(expr, fmt)	Convert string 'expr' to a number based on the string format 'fmt'. Throws an exception if the conversion fails. The format can consist of the following characters, case insensitive: '0' or '9': Specifies an expected digit between 0 and 9. A sequence of 0 or 9 in the format string matches a sequence of digits in the input string. If the 0/9 sequence starts with 0 and is before the decimal point, it can only match a digit sequence of the same size. Otherwise, if the sequence starts with 9 or is after the decimal point, it can match a digit sequence that has the same or smaller size. '.' or 'D': Specifies the position of the decimal point (optional, only allowed once). ',' or 'G': Specifies the position of the grouping (thousands) separator (,). There must be a 0 or 9 to the left and right of each grouping separator. 'expr' must match the grouping separator relevant for the size of the number. '

Function	Description
to_varchar(numberExpr, formatExpr)	Convert `numberExpr` to a string based on the `formatExpr`. Throws an exception if the conversion fails. The format can consist of the following characters, case insensitive: '0' or '9': Specifies an expected digit between 0 and 9. A sequence of 0 or 9 in the format string matches a sequence of digits in the input value, generating a result string of the same length as the corresponding sequence in the format string. The result string is left-padd ed with zeros if the 0/9 sequence comprises more digits than the matching part of the decimal value, starts with 0, and is before the decimal point. Otherwise, it is padded with spaces. '.' or 'D': Specifies the position of the decimal point (optional, only allowed once). ',' or 'G': Specifies the position of the grouping (thousands) separator (,). There must be a 0 or 9 to the left and right of each grouping separator. '
translate(input, from, to)	Translates the `input` string by replacing the characters present in the `from` string with the corresponding characters in the `to` string.
trim(str)	Removes the leading and trailing space characters from `str`.
trim(BOTH FROM str)	Removes the leading and trailing space characters from `str`.
trim(LEADING FROM str)	Removes the leading space characters from `str`.

Function	Description
trim(TRAILING FROM str)	Removes the trailing space characters from `str`.
trim(trimStr FROM str)	Remove the leading and trailing `trimStr` characters from `str`.
trim(BOTH trimStr FROM str)	Remove the leading and trailing `trimStr` characters from `str`.
trim(LEADING trimStr FROM str)	Remove the leading `trimStr` characters from `str`.
trim(TRAILING trimStr FROM str)	Remove the trailing `trimStr` characters from `str`.
try_to_binary(str[, fmt])	This is a special version of `to_binary` that performs the same operation, but returns a NULL value instead of raising an error if the conversion cannot be performed.
try_to_number(expr, fmt)	Convert string 'expr' to a number based on the string format `fmt`. Returns NULL if the string 'expr' does not match the expected format. The format follows the same semantics as the to_number function.
ucase(str)	Returns `str` with all characters changed to uppercase.
unbase64(str)	Converts the argument from a base 64 string `str` to a binary.
upper(str)	Returns `str` with all characters changed to uppercase.

Examples

```
-- ascii
SELECT ascii('222');
+----------+
|ascii(222)|
+----------+
|        50|
+----------+
SELECT ascii(2);
+--------+
|ascii(2)|
+--------+
|      50|
+--------+
-- base64
SELECT base64('Feathers');
+-----------------+
|base64(Feathers)|
+-----------------+
|     RmVhdGhlcnM=|
+-----------------+
SELECT base64(x'537061726b2053514c');
+----------------------------+
|base64(X'537061726B2053514C')|
+----------------------------+
|                 U3BhcmsgU1FM|
+----------------------------+
-- bit_length
SELECT bit_length('Feathers');
+---------------------+
|bit_length(Feathers)|
+---------------------+
|                   64|
+---------------------+
SELECT bit_length(x'537061726b2053514c');
+---------------------------------+
|bit_length(X'537061726B2053514C')|
+---------------------------------+
|                               72|
+---------------------------------+
-- btrim
SELECT btrim('    Feathers   ');
+---------------------+
|btrim(    Feathers   )|
```

```
+----------------------+
|              Feathers|
+----------------------+
SELECT btrim(encode('    Feathers   ', 'utf-8'));
+-----------------------------------+
|btrim(encode(    Feathers   , utf-8))|
+-----------------------------------+
|                           Feathers|
+-----------------------------------+
SELECT btrim('Feathers', 'Fe');
+--------------------+
|btrim(Alphabet, Al)|
+--------------------+
|               athers|
+--------------------+
SELECT btrim(encode('Feathers', 'utf-8'), encode('Al', 'utf-8'));
+-----------------------------------------------------+
|btrim(encode(Feathers, utf-8), encode(Al, utf-8))|
+-----------------------------------------------------+
|                                               athers|
+-----------------------------------------------------+
-- char
SELECT char(65);
+--------+
|char(65)|
+--------+
|       A|
+--------+
-- char_length
SELECT char_length('Feathers ');
+-----------------------+
|char_length(Feathers )|
+-----------------------+
|                      9 |
+-----------------------+
SELECT char_length(x'537061726b2053514c');
+---------------------------------+
|char_length(X'537061726B2053514C')|
+---------------------------------+
|                                9|
+---------------------------------+
SELECT CHAR_LENGTH('Feathers ');
+-----------------------+
|char_length(Feathers )|
```

```
+----------------------+
|                     9|
+----------------------+
SELECT CHARACTER_LENGTH('Feathers ');
+---------------------------+
|character_length(Feathers )|
+---------------------------+
|                          9|
+---------------------------+
-- character_length
SELECT character_length('Feathers ');
+---------------------------+
|character_length(Feathers )|
+---------------------------+
|                          9|
+---------------------------+
SELECT character_length(x'537061726b2053514c');
+--------------------------------------+
|character_length(X'537061726B2053514C')|
+--------------------------------------+
|                                     9|
+--------------------------------------+
SELECT CHAR_LENGTH('Feathers ');
+----------------------+
|char_length(Feathers )|
+----------------------+
|                     9|
+----------------------+
SELECT CHARACTER_LENGTH('Feathers ');
+---------------------------+
|character_length(Feathers )|
+---------------------------+
|                          9|
+---------------------------+
-- chr
SELECT chr(65);
+-------+
|chr(65)|
+-------+
|      A|
+-------+
-- concat_ws
SELECT concat_ws(' ', 'Fea', 'thers');
+----------------------+
```

```
|concat_ws( , Fea, thers)|
+-------------------------+
|               Feathers|
+-------------------------+
SELECT concat_ws('s');
+------------+
|concat_ws(s)|
+------------+
|            |
+------------+
SELECT concat_ws('/', 'foo', null, 'bar');
+----------------------------+
|concat_ws(/, foo, NULL, bar)|
+----------------------------+
|                     foo/bar|
+----------------------------+
SELECT concat_ws(null, 'Fea', 'thers');
+---------------------------+
|concat_ws(NULL, Fea, thers)|
+---------------------------+
|                       NULL|
+---------------------------+
-- contains
SELECT contains('Feathers', 'Fea');
+--------------------------+
|contains(Feathers, Fea)|
+--------------------------+
|                      true|
+--------------------------+
SELECT contains('Feathers', 'SQL');
+--------------------------+
|contains(Feathers, SQL)|
+--------------------------+
|                     false|
+--------------------------+
SELECT contains('Feathers', null);
+-------------------------+
|contains(Feathers, NULL)|
+-------------------------+
|                     NULL|
+-------------------------+
SELECT contains(x'537061726b2053514c', x'537061726b');
+-----------------------------------------------+
|contains(X'537061726B2053514C', X'537061726B')|
```

```
+-------------------------------------------+
|                                       true|
+-------------------------------------------+
-- decode
SELECT decode(encode('abc', 'utf-8'), 'utf-8');
+---------------------------------+
|decode(encode(abc, utf-8), utf-8)|
+---------------------------------+
|                              abc|
+---------------------------------+
SELECT decode(2, 1, 'Southlake', 2, 'San Francisco', 3, 'New Jersey', 4, 'Seattle',
 'Non domestic');
+-------------------------------------------------------------------------------+
|decode(2, 1, Southlake, 2, San Francisco, 3, New Jersey, 4, Seattle, Non domestic)|
+-------------------------------------------------------------------------------+
|                                                                  San Francisco|
+-------------------------------------------------------------------------------+
SELECT decode(6, 1, 'Southlake', 2, 'San Francisco', 3, 'New Jersey', 4, 'Seattle',
 'Non domestic');
+-------------------------------------------------------------------------------+
|decode(6, 1, Southlake, 2, San Francisco, 3, New Jersey, 4, Seattle, Non domestic)|
+-------------------------------------------------------------------------------+
|                                                                   Non domestic|
+-------------------------------------------------------------------------------+
SELECT decode(6, 1, 'Southlake', 2, 'San Francisco', 3, 'New Jersey', 4, 'Seattle');
+-----------------------------------------------------------------+
|decode(6, 1, Southlake, 2, San Francisco, 3, New Jersey, 4, Seattle)|
+-----------------------------------------------------------------+
|                                                             NULL|
+-----------------------------------------------------------------+
SELECT decode(null, 6, 'Fea', NULL, 'thers', 4, 'rock');
+-----------------------------------------+
|decode(NULL, 6, Fea, NULL, thers, 4, rock)|
+-----------------------------------------+
|                                    thers|
+-----------------------------------------+
-- elt
SELECT elt(1, 'scala', 'java');
+------------------+
|elt(1, scala, java)|
+------------------+
|             scala|
+------------------+
SELECT elt(2, 'a', 1);
```

```
+------------+
|elt(2, a, 1)|
+------------+
|           1|
+------------+
-- encode
SELECT encode('abc', 'utf-8');
+------------------+
|encode(abc, utf-8)|
+------------------+
|        [61 62 63]|
+------------------+
-- endswith
SELECT endswith('Feathers', 'ers');
+------------------------+
|endswith(Feathers, ers)|
+------------------------+
|                    true|
+------------------------+
SELECT endswith('Feathers', 'SQL');
+-------------------------+
|endswith(Feathers, SQL)|
+-------------------------+
|                     false|
+-------------------------+
SELECT endswith('Feathers', null);
+-------------------------+
|endswith(Feathers, NULL)|
+-------------------------+
|                     NULL|
+-------------------------+
SELECT endswith(x'537061726b2053514c', x'537061726b');
+-----------------------------------------------+
|endswith(X'537061726B2053514C', X'537061726B')|
+-----------------------------------------------+
|                                          false|
+-----------------------------------------------+
SELECT endswith(x'537061726b2053514c', x'53514c');
+---------------------------------------+
|endswith(X'537061726B2053514C', X'53514C')|
+---------------------------------------+
|                                   true|
+---------------------------------------+
-- find_in_set
```

```
SELECT find_in_set('ab','abc,b,ab,c,def');
+-------------------------------+
|find_in_set(ab, abc,b,ab,c,def)|
+-------------------------------+
|                              3|
+-------------------------------+
-- format_number
SELECT format_number(12332.123456, 4);
+------------------------------+
|format_number(12332.123456, 4)|
+------------------------------+
|                  12,332.1235|
+------------------------------+
SELECT format_number(12332.123456, '##################.###');
+---------------------------------------------------+
|format_number(12332.123456, ##################.###)|
+---------------------------------------------------+
|                                           12332.123|
+---------------------------------------------------+
-- format_string
SELECT format_string("Hello World %d %s", 100, "days");
+-------------------------------------------+
|format_string(Hello World %d %s, 100, days)|
+-------------------------------------------+
|                       Hello World 100 days|
+-------------------------------------------+
-- initcap
SELECT initcap('Feathers');
+------------------+
|initcap(Feathers)|
+------------------+
|         Feathers|
+------------------+
-- instr
SELECT instr('Feathers', 'ers');
+--------------------+
|instr(Feathers, ers)|
+--------------------+
|                   6|
+--------------------+
-- lcase
SELECT lcase('Feathers');
+---------------+
|lcase(Feathers)|
```

```
+---------------+
|       feathers|
+---------------+
-- left
SELECT left('Feathers', 3);
+------------------+
|left(Feathers, 3)|
+------------------+
|               Fea|
+------------------+
SELECT left(encode('Feathers', 'utf-8'), 3);
+---------------------------------+
|left(encode(Feathers, utf-8), 3)|
+---------------------------------+
|                       [RmVh]|
+---------------------------------+
-- len
SELECT len('Feathers ');
+---------------+
|len(Feathers )|
+---------------+
|              9|
+---------------+
SELECT len(x'537061726b2053514c');
+--------------------------+
|len(X'537061726B2053514C')|
+--------------------------+
|                         9|
+--------------------------+
SELECT CHAR_LENGTH('Feathers ');
+-----------------------+
|char_length(Feathers )|
+-----------------------+
|                      9|
+-----------------------+
SELECT CHARACTER_LENGTH('Feathers ');
+----------------------------+
|character_length(Feathers )|
+----------------------------+
|                           9|
+----------------------------+
-- length
SELECT length('Feathers ');
+------------------+
```

```
|length(Feathers )|
+-----------------+
|                9|
+-----------------+
SELECT length(x'537061726b2053514c');
+----------------------------+
|length(X'537061726B2053514C')|
+----------------------------+
|                           9|
+----------------------------+
SELECT CHAR_LENGTH('Feathers ');
+----------------------+
|char_length(Feathers )|
+----------------------+
|                     9|
+----------------------+
SELECT CHARACTER_LENGTH('Feathers ');
+---------------------------+
|character_length(Feathers )|
+---------------------------+
|                          9|
+---------------------------+
-- levenshtein
SELECT levenshtein('kitten', 'sitting');
+--------------------------+
|levenshtein(kitten, sitting)|
+--------------------------+
|                           3|
+--------------------------+
SELECT levenshtein('kitten', 'sitting', 2);
+-------------------------------+
|levenshtein(kitten, sitting, 2)|
+-------------------------------+
|                             -1|
+-------------------------------+
-- locate
SELECT locate('bar', 'foobarbar');
+-------------------------+
|locate(bar, foobarbar, 1)|
+-------------------------+
|                        4|
+-------------------------+
SELECT locate('bar', 'foobarbar', 5);
+-------------------------+
```

```
|locate(bar, foobarbar, 5)|
+-------------------------+
|                        7|
+-------------------------+
SELECT POSITION('bar' IN 'foobarbar');
+-------------------------+
|locate(bar, foobarbar, 1)|
+-------------------------+
|                        4|
+-------------------------+
-- lower
SELECT lower('Feathers');
+---------------+
|lower(Feathers)|
+---------------+
|       feathers|
+---------------+
-- lpad
SELECT lpad('hi', 5, '??');
+---------------+
|lpad(hi, 5, ??)|
+---------------+
|          ???hi|
+---------------+
SELECT lpad('hi', 1, '??');
+---------------+
|lpad(hi, 1, ??)|
+---------------+
|              h|
+---------------+
SELECT lpad('hi', 5);
+--------------+
|lpad(hi, 5,  )|
+--------------+
|            hi|
+--------------+
SELECT hex(lpad(unhex('aabb'), 5));
+---------------------------------+
|hex(lpad(unhex(aabb), 5, X'00'))|
+---------------------------------+
|                      000000AABB|
+---------------------------------+
SELECT hex(lpad(unhex('aabb'), 5, unhex('1122')));
+-----------------------------------+
```

```
|hex(lpad(unhex(aabb), 5, unhex(1122)))|
+--------------------------------------+
|                           112211AABB|
+--------------------------------------+
-- ltrim
SELECT ltrim('    Feathers   ');
+----------------------+
|ltrim(    Feathers   )|
+----------------------+
|           Feathers   |
+----------------------+
-- luhn_check
SELECT luhn_check('8112189876');
+----------------------+
|luhn_check(8112189876)|
+----------------------+
|                  true|
+----------------------+
SELECT luhn_check('79927398713');
+-----------------------+
|luhn_check(79927398713)|
+-----------------------+
|                   true|
+-----------------------+
SELECT luhn_check('79927398714');
+-----------------------+
|luhn_check(79927398714)|
+-----------------------+
|                  false|
+-----------------------+
-- mask
SELECT mask('abcd-EFGH-8765-4321');
+---------------------------------------+
|mask(abcd-EFGH-8765-4321, X, x, n, NULL)|
+---------------------------------------+
|                     xxxx-XXXX-nnnn-nnnn|
+---------------------------------------+
SELECT mask('abcd-EFGH-8765-4321', 'Q');
+---------------------------------------+
|mask(abcd-EFGH-8765-4321, Q, x, n, NULL)|
+---------------------------------------+
|                     xxxx-QQQQ-nnnn-nnnn|
+---------------------------------------+
SELECT mask('AbCD123-@$#', 'Q', 'q');
```

```
+--------------------------------+
|mask(AbCD123-@$#, Q, q, n, NULL)|
+--------------------------------+
|                     QqQQnnn-@$#|
+--------------------------------+
SELECT mask('AbCD123-@$#');
+--------------------------------+
|mask(AbCD123-@$#, X, x, n, NULL)|
+--------------------------------+
|                     XxXXnnn-@$#|
+--------------------------------+
SELECT mask('AbCD123-@$#', 'Q');
+--------------------------------+
|mask(AbCD123-@$#, Q, x, n, NULL)|
+--------------------------------+
|                     QxQQnnn-@$#|
+--------------------------------+
SELECT mask('AbCD123-@$#', 'Q', 'q');
+--------------------------------+
|mask(AbCD123-@$#, Q, q, n, NULL)|
+--------------------------------+
|                     QqQQnnn-@$#|
+--------------------------------+
SELECT mask('AbCD123-@$#', 'Q', 'q', 'd');
+--------------------------------+
|mask(AbCD123-@$#, Q, q, d, NULL)|
+--------------------------------+
|                     QqQQddd-@$#|
+--------------------------------+
SELECT mask('AbCD123-@$#', 'Q', 'q', 'd', 'o');
+-----------------------------+
|mask(AbCD123-@$#, Q, q, d, o)|
+-----------------------------+
|                  QqQQdddoooo|
+-----------------------------+
SELECT mask('AbCD123-@$#', NULL, 'q', 'd', 'o');
+--------------------------------+
|mask(AbCD123-@$#, NULL, q, d, o)|
+--------------------------------+
|                     AqCDdddoooo|
+--------------------------------+
SELECT mask('AbCD123-@$#', NULL, NULL, 'd', 'o');
+---------------------------------+
|mask(AbCD123-@$#, NULL, NULL, d, o)|
```

```
+----------------------------------+
|                       AbCDdddoooo|
+----------------------------------+
SELECT mask('AbCD123-@$#', NULL, NULL, NULL, 'o');
+-------------------------------------+
|mask(AbCD123-@$#, NULL, NULL, NULL, o)|
+-------------------------------------+
|                           AbCD123oooo|
+-------------------------------------+
SELECT mask(NULL, NULL, NULL, NULL, 'o');
+------------------------------+
|mask(NULL, NULL, NULL, NULL, o)|
+------------------------------+
|                           NULL|
+------------------------------+
SELECT mask(NULL);
+-------------------------+
|mask(NULL, X, x, n, NULL)|
+-------------------------+
|                     NULL|
+-------------------------+
SELECT mask('AbCD123-@$#', NULL, NULL, NULL, NULL);
+----------------------------------------+
|mask(AbCD123-@$#, NULL, NULL, NULL, NULL)|
+----------------------------------------+
|                             AbCD123-@$#|
+----------------------------------------+
-- octet_length
SELECT octet_length('Feathers');
+----------------------+
|octet_length(Feathers)|
+----------------------+
|                     8|
+----------------------+
SELECT octet_length(x'537061726b2053514c');
+----------------------------------+
|octet_length(X'537061726B2053514C')|
+----------------------------------+
|                                 9|
+----------------------------------+
-- overlay
SELECT overlay('Feathers' PLACING '_' FROM 6);
+---------------------------+
|overlay(Feathers, _, 6, -1)|
```

```
+---------------------------+
|                   Feathe_ers|
+---------------------------+
SELECT overlay('Feathers' PLACING 'ures' FROM 5);
+-------------------------------+
|overlay(Feathers, ures, 5, -1)|
+-------------------------------+
|                     Features  |
+-------------------------------+
-- position
SELECT position('bar', 'foobarbar');
+---------------------------+
|position(bar, foobarbar, 1)|
+---------------------------+
|                          4|
+---------------------------+
SELECT position('bar', 'foobarbar', 5);
+---------------------------+
|position(bar, foobarbar, 5)|
+---------------------------+
|                          7|
+---------------------------+
SELECT POSITION('bar' IN 'foobarbar');
+-------------------------+
|locate(bar, foobarbar, 1)|
+-------------------------+
|                        4|
+-------------------------+
-- printf
SELECT printf("Hello World %d %s", 100, "days");
+-----------------------------------+
|printf(Hello World %d %s, 100, days)|
+-----------------------------------+
|                Hello World 100 days|
+-----------------------------------+
-- regexp_count
SELECT regexp_count('Steven Jones and Stephen Smith are the best players', 'Ste(v|
ph)en');
+-------------------------------------------------------------------------------+
|regexp_count(Steven Jones and Stephen Smith are the best players, Ste(v|ph)en)|
+-------------------------------------------------------------------------------+
|                                                                              2|
+-------------------------------------------------------------------------------+
SELECT regexp_count('abcdefghijklmnopqrstuvwxyz', '[a-z]{3}');
```

```
+-------------------------------------------------+
|regexp_count(abcdefghijklmnopqrstuvwxyz, [a-z]{3})|
+-------------------------------------------------+
|                                                8|
+-------------------------------------------------+
-- regexp_extract
SELECT regexp_extract('100-200', '(\\d+)-(\\d+)', 1);
+--------------------------------------+
|regexp_extract(100-200, (\d+)-(\d+), 1)|
+--------------------------------------+
|                                   100|
+--------------------------------------+
-- regexp_extract_all
SELECT regexp_extract_all('100-200, 300-400', '(\\d+)-(\\d+)', 1);
+-----------------------------------------------------+
|regexp_extract_all(100-200, 300-400, (\d+)-(\d+), 1)|
+-----------------------------------------------------+
|                                          [100, 300]|
+-----------------------------------------------------+
-- regexp_instr
SELECT regexp_instr('user@opensearch.org', '@[^.]*');
+-----------------------------------------------+
|regexp_instr(user@opensearch.org, @[^.]*, 0)|
+-----------------------------------------------+
|                                              5|
+-----------------------------------------------+
-- regexp_replace
SELECT regexp_replace('100-200', '(\\d+)', 'num');
+--------------------------------------+
|regexp_replace(100-200, (\d+), num, 1)|
+--------------------------------------+
|                               num-num|
+--------------------------------------+
-- regexp_substr
SELECT regexp_substr('Steven Jones and Stephen Smith are the best players', 'Ste(v|
ph)en');
+-----------------------------------------------------------------------------+
|regexp_substr(Steven Jones and Stephen Smith are the best players, Ste(v|ph)en)|
+-----------------------------------------------------------------------------+
|                                                                       Steven|
+-----------------------------------------------------------------------------+
SELECT regexp_substr('Steven Jones and Stephen Smith are the best players', 'Jeck');
+-------------------------------------------------------------------------+
|regexp_substr(Steven Jones and Stephen Smith are the best players, Jeck)|
```

```
+------------------------------------------------------------------------------+
|                                                                          NULL|
+------------------------------------------------------------------------------+
-- repeat
SELECT repeat('123', 2);
+--------------+
|repeat(123, 2)|
+--------------+
|        123123|
+--------------+
-- replace
SELECT replace('ABCabc', 'abc', 'DEF');
+-------------------------+
|replace(ABCabc, abc, DEF)|
+-------------------------+
|                   ABCDEF|
+-------------------------+
-- right
SELECT right('Feathers', 3);
+-------------------+
|right(Feathers, 3)|
+-------------------+
|                ers|
+-------------------+
-- rpad
SELECT rpad('hi', 5, '??');
+---------------+
|rpad(hi, 5, ??)|
+---------------+
|          hi???|
+---------------+
SELECT rpad('hi', 1, '??');
+---------------+
|rpad(hi, 1, ??)|
+---------------+
|              h|
+---------------+
SELECT rpad('hi', 5);
+--------------+
|rpad(hi, 5,  )|
+--------------+
|         hi   |
+--------------+
SELECT hex(rpad(unhex('aabb'), 5));
```

```
+--------------------------------+
|hex(rpad(unhex(aabb), 5, X'00'))|
+--------------------------------+
|                      AABB000000|
+--------------------------------+
SELECT hex(rpad(unhex('aabb'), 5, unhex('1122')));
+-------------------------------------+
|hex(rpad(unhex(aabb), 5, unhex(1122)))|
+-------------------------------------+
|                           AABB112211|
+-------------------------------------+
-- rtrim
SELECT rtrim('    Feathers   ');
+----------------------+
|rtrim(    Feathers   )|
+----------------------+
|              Feathers|
+----------------------+
-- sentences
SELECT sentences('Hi there! Good morning.');
+--------------------------------------+
|sentences(Hi there! Good morning., , )|
+--------------------------------------+
|                  [[Hi, there], [Go...|
+--------------------------------------+
-- soundex
SELECT soundex('Miller');
+---------------+
|soundex(Miller)|
+---------------+
|           M460|
+---------------+
-- space
SELECT concat(space(2), '1');
+-------------------+
|concat(space(2), 1)|
+-------------------+
|                  1|
+-------------------+
-- split
SELECT split('oneAtwoBthreeC', '[ABC]');
+--------------------------------+
|split(oneAtwoBthreeC, [ABC], -1)|
+--------------------------------+
```

```
|             [one, two, three, ]|
+--------------------------------+
SELECT split('oneAtwoBthreeC', '[ABC]', -1);
+--------------------------------+
|split(oneAtwoBthreeC, [ABC], -1)|
+--------------------------------+
|             [one, two, three, ]|
+--------------------------------+
SELECT split('oneAtwoBthreeC', '[ABC]', 2);
+--------------------------------+
|split(oneAtwoBthreeC, [ABC], 2)|
+--------------------------------+
|              [one, twoBthreeC]|
+--------------------------------+
-- split_part
SELECT split_part('11.12.13', '.', 3);
+---------------------------+
|split_part(11.12.13, ., 3)|
+---------------------------+
|                         13|
+---------------------------+
-- startswith
SELECT startswith('Feathers', 'Fea');
+----------------------------+
|startswith(Feathers, Fea)|
+----------------------------+
|                        true|
+----------------------------+
SELECT startswith('Feathers', 'SQL');
+--------------------------+
|startswith(Feathers, SQL)|
+--------------------------+
|                     false|
+--------------------------+
SELECT startswith('Feathers', null);
+--------------------------+
|startswith(Feathers, NULL)|
+--------------------------+
|                      NULL|
+--------------------------+
SELECT startswith(x'537061726b2053514c', x'537061726b');
+-------------------------------------------------+
|startswith(X'537061726B2053514C', X'537061726B')|
+-------------------------------------------------+
```

```
|                                              true|
+-----------------------------------------------+
SELECT startswith(x'537061726b2053514c', x'53514c');
+-----------------------------------------+
|startswith(X'537061726B2053514C', X'53514C')|
+-----------------------------------------+
|                                    false|
+-----------------------------------------+
-- substr
SELECT substr('Feathers', 5);
+-------------------------------+
|substr(Feathers, 5, 2147483647)|
+-------------------------------+
|                          hers |
+-------------------------------+
SELECT substr('Feathers', -3);
+--------------------------------+
|substr(Feathers, -3, 2147483647)|
+--------------------------------+
|                             ers|
+--------------------------------+
SELECT substr('Feathers', 5, 1);
+-----------------------+
|substr(Feathers, 5, 1)|
+-----------------------+
|                      h|
+-----------------------+
SELECT substr('Feathers' FROM 5);
+---------------------------------+
|substring(Feathers, 5, 2147483647)|
+---------------------------------+
|                             hers |
+---------------------------------+
SELECT substr('Feathers' FROM -3);
+----------------------------------+
|substring(Feathers, -3, 2147483647)|
+----------------------------------+
|                                ers|
+----------------------------------+
SELECT substr('Feathers' FROM 5 FOR 1);
+-------------------------+
|substring(Feathers, 5, 1)|
+-------------------------+
|                        h|
```

```
+-------------------------+
-- substring
SELECT substring('Feathers', 5);
+---------------------------------+
|substring(Feathers, 5, 2147483647)|
+---------------------------------+
|                            hers |
+---------------------------------+
SELECT substring('Feathers', -3);
+---------------------------------+
|substring(Feathers, -3, 2147483647)|
+---------------------------------+
|                               ers|
+---------------------------------+
SELECT substring('Feathers', 5, 1);
+-------------------------+
|substring(Feathers, 5, 1)|
+-------------------------+
|                        h|
+-------------------------+
SELECT substring('Feathers' FROM 5);
+---------------------------------+
|substring(Feathers, 5, 2147483647)|
+---------------------------------+
|                            hers |
+---------------------------------+
SELECT substring('Feathers' FROM -3);
+---------------------------------+
|substring(Feathers, -3, 2147483647)|
+---------------------------------+
|                               ers|
+---------------------------------+
SELECT substring('Feathers' FROM 5 FOR 1);
+-------------------------+
|substring(Feathers, 5, 1)|
+-------------------------+
|                        h|
+-------------------------+
-- substring_index
SELECT substring_index('www.apache.org', '.', 2);
+-----------------------------------+
|substring_index(www.apache.org, ., 2)|
+-----------------------------------+
|                         www.apache|
```

```
+-------------------------------------+
-- to_binary
SELECT to_binary('abc', 'utf-8');
+---------------------+
|to_binary(abc, utf-8)|
+---------------------+
|           [61 62 63]|
+---------------------+
-- to_char
SELECT to_char(454, '999');
+-----------------+
|to_char(454, 999)|
+-----------------+
|              454|
+-----------------+
SELECT to_char(454.00, '000D00');
+------------------------+
|to_char(454.00, 000D00)|
+------------------------+
|                 454.00|
+------------------------+
SELECT to_char(12454, '99G999');
+----------------------+
|to_char(12454, 99G999)|
+----------------------+
|                12,454|
+----------------------+
SELECT to_char(78.12, '$99.99');
+----------------------+
|to_char(78.12, $99.99)|
+----------------------+
|                $78.12|
+----------------------+
SELECT to_char(-12454.8, '99G999D9S');
+----------------------------+
|to_char(-12454.8, 99G999D9S)|
+----------------------------+
|                   12,454.8-|
+----------------------------+
-- to_number
SELECT to_number('454', '999');
+-------------------+
|to_number(454, 999)|
+-------------------+
```

```
|                454|
+-------------------+
SELECT to_number('454.00', '000.00');
+-------------------------+
|to_number(454.00, 000.00)|
+-------------------------+
|                   454.00|
+-------------------------+
SELECT to_number('12,454', '99,999');
+-------------------------+
|to_number(12,454, 99,999)|
+-------------------------+
|                    12454|
+-------------------------+
SELECT to_number('$78.12', '$99.99');
+-------------------------+
|to_number($78.12, $99.99)|
+-------------------------+
|                    78.12|
+-------------------------+
SELECT to_number('12,454.8-', '99,999.9S');
+-------------------------------+
|to_number(12,454.8-, 99,999.9S)|
+-------------------------------+
|                       -12454.8|
+-------------------------------+
-- to_varchar
SELECT to_varchar(454, '999');
+-----------------+
|to_char(454, 999)|
+-----------------+
|              454|
+-----------------+
SELECT to_varchar(454.00, '000D00');
+-----------------------+
|to_char(454.00, 000D00)|
+-----------------------+
|                 454.00|
+-----------------------+
SELECT to_varchar(12454, '99G999');
+----------------------+
|to_char(12454, 99G999)|
+----------------------+
|                12,454|
```

```
+----------------------+
SELECT to_varchar(78.12, '$99.99');
+----------------------+
|to_char(78.12, $99.99)|
+----------------------+
|               $78.12|
+----------------------+
SELECT to_varchar(-12454.8, '99G999D9S');
+---------------------------+
|to_char(-12454.8, 99G999D9S)|
+---------------------------+
|                  12,454.8-|
+---------------------------+
-- translate
SELECT translate('AaBbCc', 'abc', '123');
+--------------------------+
|translate(AaBbCc, abc, 123)|
+--------------------------+
|                    A1B2C3|
+--------------------------+
-- try_to_binary
SELECT try_to_binary('abc', 'utf-8');
+-------------------------+
|try_to_binary(abc, utf-8)|
+-------------------------+
|              [61 62 63]|
+-------------------------+
select try_to_binary('a!', 'base64');
+-------------------------+
|try_to_binary(a!, base64)|
+-------------------------+
|                     NULL|
+-------------------------+
select try_to_binary('abc', 'invalidFormat');
+---------------------------------+
|try_to_binary(abc, invalidFormat)|
+---------------------------------+
|                             NULL|
+---------------------------------+
-- try_to_number
SELECT try_to_number('454', '999');
+----------------------+
|try_to_number(454, 999)|
+----------------------+
```

```
|                    454|
+-----------------------+
SELECT try_to_number('454.00', '000.00');
+----------------------------+
|try_to_number(454.00, 000.00)|
+----------------------------+
|                      454.00|
+----------------------------+
SELECT try_to_number('12,454', '99,999');
+----------------------------+
|try_to_number(12,454, 99,999)|
+----------------------------+
|                       12454|
+----------------------------+
SELECT try_to_number('$78.12', '$99.99');
+----------------------------+
|try_to_number($78.12, $99.99)|
+----------------------------+
|                       78.12|
+----------------------------+
SELECT try_to_number('12,454.8-', '99,999.9S');
+-----------------------------------+
|try_to_number(12,454.8-, 99,999.9S)|
+-----------------------------------+
|                          -12454.8|
+-----------------------------------+
-- ucase
SELECT ucase('Feathers');
+---------------+
|ucase(Feathers)|
+---------------+
|       FEATHERS|
+---------------+
-- unbase64
SELECT unbase64('U3BhcmsgU1FM');
+----------------------+
|unbase64(U3BhcmsgU1FM)|
+----------------------+
|  [53 70 61 72 6B 2...|
+----------------------+
-- upper
SELECT upper('Feathers');
+---------------+
|upper(Feathers)|
```

```
+---------------+
|       FEATHERS|
+---------------+
```

Date and time functions

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

Function	Description
add_months(start_date, num_months)	Returns the date that is `num_months` after `start_date` .
convert_timezone([sourceTz,]targetTz, sourceTs)	Converts the timestamp without time zone `sourceTs` from the `sourceTz` time zone to `targetTz`.
curdate()	Returns the current date at the start of query evaluation. All calls of curdate within the same query return the same value.
current_date()	Returns the current date at the start of query evaluation. All calls of current_date within the same query return the same value.
current_date	Returns the current date at the start of query evaluation.
current_timestamp()	Returns the current timestamp at the start of query evaluation. All calls of current_t imestamp within the same query return the same value.
current_timestamp	Returns the current timestamp at the start of query evaluation.

Function	Description
current_timezone()	Returns the current session local timezone.
date_add(start_date, num_days)	Returns the date that is num_days after start_date .
date_diff(endDate, startDate)	Returns the number of days from startDate to endDate.
date_format(timestamp, fmt)	Converts timestamp to a value of string in the format specified by the date format fmt.
date_from_unix_date(days)	Create date from the number of days since 1970-01-01.
date_part(field, source)	Extracts a part of the date/timestamp or interval source.
date_sub(start_date, num_days)	Returns the date that is num_days before start_date .
date_trunc(fmt, ts)	Returns timestamp ts truncated to the unit specified by the format model fmt.
dateadd(start_date, num_days)	Returns the date that is num_days after start_date .
datediff(endDate, startDate)	Returns the number of days from startDate to endDate.
datepart(field, source)	Extracts a part of the date/timestamp or interval source.
day(date)	Returns the day of month of the date/time stamp.
dayofmonth(date)	Returns the day of month of the date/time stamp.

Function	Description
dayofweek(date)	Returns the day of the week for date/time stamp (1 = Sunday, 2 = Monday, ..., 7 = Saturday).
dayofyear(date)	Returns the day of year of the date/time stamp.
extract(field FROM source)	Extracts a part of the date/timestamp or interval source.
from_unixtime(unix_time[, fmt])	Returns `unix_time` in the specified `fmt`.
from_utc_timestamp(timestamp, timezone)	Given a timestamp like '2017-07-14 02:40:00.0', interprets it as a time in UTC, and renders that time as a timestamp in the given time zone. For example, 'GMT+1' would yield '2017-07-14 03:40:00.0'.
hour(timestamp)	Returns the hour component of the string/timestamp.
last_day(date)	Returns the last day of the month which the date belongs to.
localtimestamp()	Returns the current timestamp without time zone at the start of query evaluation. All calls of localtimestamp within the same query return the same value.
localtimestamp	Returns the current local date-time at the session time zone at the start of query evaluation.
make_date(year, month, day)	Create date from year, month and day fields.
make_dt_interval([days[, hours[, mins[, secs]]]])	Make DayTimeIntervalType duration from days, hours, mins and secs.

Function	Description
make_interval([years[, months[, weeks[, days[, hours[, mins[, secs]]]]]]])	Make interval from years, months, weeks, days, hours, mins and secs.
make_timestamp(year, month, day, hour, min, sec[, timezone])	Create timestamp from year, month, day, hour, min, sec and timezone fields.
make_timestamp_ltz(year, month, day, hour, min, sec[, timezone])	Create the current timestamp with local time zone from year, month, day, hour, min, sec and timezone fields.
make_timestamp_ntz(year, month, day, hour, min, sec)	Create local date-time from year, month, day, hour, min, sec fields.
make_ym_interval([years[, months]])	Make year-month interval from years, months.
minute(timestamp)	Returns the minute component of the string/ timestamp.
month(date)	Returns the month component of the date/ timestamp.
months_between(timestamp1, timestamp2[, roundOff])	If `timestamp1` is later than `timestamp 2` , then the result is positive. If `timestamp 1` and `timestamp2` are on the same day of month, or both are the last day of month, time of day will be ignored. Otherwise, the difference is calculated based on 31 days per month, and rounded to 8 digits unless roundOff=false.
next_day(start_date, day_of_week)	Returns the first date which is later than `start_date` and named as indicated. The function returns NULL if at least one of the input parameters is NULL.

Function	Description
now()	Returns the current timestamp at the start of query evaluation.
quarter(date)	Returns the quarter of the year for date, in the range 1 to 4.
second(timestamp)	Returns the second component of the string/ timestamp.
session_window(time_column, gap_duration)	Generates session window given a timestamp specifying column and gap duration. See 'Types of time windows' in Structured Streaming guide doc for detailed explanation and examples.
timestamp_micros(microseconds)	Creates timestamp from the number of microseconds since UTC epoch.
timestamp_millis(milliseconds)	Creates timestamp from the number of milliseconds since UTC epoch.
timestamp_seconds(seconds)	Creates timestamp from the number of seconds (can be fractional) since UTC epoch.
to_date(date_str[, fmt])	Parses the `date_str` expression with the `fmt` expression to a date. Returns null with invalid input. By default, it follows casting rules to a date if the `fmt` is omitted.
to_timestamp(timestamp_str[, fmt])	Parses the `timestamp_str` expression with the `fmt` expression to a timestamp. Returns null with invalid input. By default, it follows casting rules to a timestamp if the `fmt` is omitted.

Function	Description
to_timestamp_ltz(timestamp_str[, fmt])	Parses the `timestamp_str` expression with the `fmt` expression to a timestamp with local time zone. Returns null with invalid input. By default, it follows casting rules to a timestamp if the `fmt` is omitted.
to_timestamp_ntz(timestamp_str[, fmt])	Parses the `timestamp_str` expression with the `fmt` expression to a timestamp without time zone. Returns null with invalid input. By default, it follows casting rules to a timestamp if the `fmt` is omitted.
to_unix_timestamp(timeExp[, fmt])	Returns the UNIX timestamp of the given time.
to_utc_timestamp(timestamp, timezone)	Given a timestamp like '2017-07-14 02:40:00.0', interprets it as a time in the given time zone, and renders that time as a timestamp in UTC. For example, 'GMT+1' would yield '2017-07-14 01:40:00.0'.
trunc(date, fmt)	Returns `date` with the time portion of the day truncated to the unit specified by the format model `fmt`.
try_to_timestamp(timestamp_str[, fmt])	Parses the `timestamp_str` expression with the `fmt` expression to a timestamp.
unix_date(date)	Returns the number of days since 1970-01-01.
unix_micros(timestamp)	Returns the number of microseconds since 1970-01-01 00:00:00 UTC.

Function	Description
unix_millis(timestamp)	Returns the number of milliseconds since 1970-01-01 00:00:00 UTC. Truncates higher levels of precision.
unix_seconds(timestamp)	Returns the number of seconds since 1970-01-01 00:00:00 UTC. Truncates higher levels of precision.
unix_timestamp([timeExp[, fmt]])	Returns the UNIX timestamp of current or specified time.
weekday(date)	Returns the day of the week for date/time stamp (0 = Monday, 1 = Tuesday, ..., 6 = Sunday).
weekofyear(date)	Returns the week of the year of the given date. A week is considered to start on a Monday and week 1 is the first week with >3 days.
window(time_column, window_duration[, slide_duration[, start_time]])	Bucketize rows into one or more time windows given a timestamp specifying column. Window starts are inclusive but the window ends are exclusive, e.g. 12:05 will be in the window [12:05,12:10) but not in [12:00,12:05). Windows can support microsecond precision. Windows in the order of months are not supported. See 'Window Operations on Event Time' in Structured Streaming guide doc for detailed explanation and examples.

Function	Description
window_time(window_column)	Extract the time value from time/session window column which can be used for event time value of window. The extracted time is (window.end - 1) which reflects the fact that the the aggregating windows have exclusive upper bound - [start, end) See 'Window Operations on Event Time' in Structured Streaming guide doc for detailed explanation and examples.
year(date)	Returns the year component of the date/time stamp.

Examples

```
-- add_months
SELECT add_months('2016-08-31', 1);
+-------------------------+
|add_months(2016-08-31, 1)|
+-------------------------+
|               2016-09-30|
+-------------------------+
-- convert_timezone
SELECT convert_timezone('Europe/Brussels', 'America/Los_Angeles',
 timestamp_ntz'2021-12-06 00:00:00');
+-------------------------------------------------------------------------------------------
+
|convert_timezone(Europe/Brussels, America/Los_Angeles, TIMESTAMP_NTZ '2021-12-06
 00:00:00')|
+-------------------------------------------------------------------------------------------
+
|                                                                                2021-12-05
 15:00:00|
+-------------------------------------------------------------------------------------------
+
SELECT convert_timezone('Europe/Brussels', timestamp_ntz'2021-12-05 15:00:00');
+-------------------------------------------------------------------------------------------
+
```

```
|convert_timezone(current_timezone(), Europe/Brussels, TIMESTAMP_NTZ '2021-12-05
 15:00:00')|
+-----------------------------------------------------------------------------------------------
+
|                                                                                2021-12-05
 07:00:00|
+-----------------------------------------------------------------------------------------------
+
-- curdate
SELECT curdate();
+--------------+
|current_date()|
+--------------+
|    2024-02-24|
+--------------+
-- current_date
SELECT current_date();
+--------------+
|current_date()|
+--------------+
|    2024-02-24|
+--------------+
SELECT current_date;
+--------------+
|current_date()|
+--------------+
|    2024-02-24|
+--------------+
-- current_timestamp
SELECT current_timestamp();
+--------------------+
| current_timestamp()|
+--------------------+
|2024-02-24 16:36:...|
+--------------------+
SELECT current_timestamp;
+--------------------+
| current_timestamp()|
+--------------------+
|2024-02-24 16:36:...|
+--------------------+
-- current_timezone
SELECT current_timezone();
+------------------+
```

```
|current_timezone()|
+------------------+
|        Asia/Seoul|
+------------------+
-- date_add
SELECT date_add('2016-07-30', 1);
+-----------------------+
|date_add(2016-07-30, 1)|
+-----------------------+
|             2016-07-31|
+-----------------------+
-- date_diff
SELECT date_diff('2009-07-31', '2009-07-30');
+---------------------------------+
|date_diff(2009-07-31, 2009-07-30)|
+---------------------------------+
|                                1|
+---------------------------------+
SELECT date_diff('2009-07-30', '2009-07-31');
+---------------------------------+
|date_diff(2009-07-30, 2009-07-31)|
+---------------------------------+
|                               -1|
+---------------------------------+
-- date_format
SELECT date_format('2016-04-08', 'y');
+--------------------------+
|date_format(2016-04-08, y)|
+--------------------------+
|                      2016|
+--------------------------+
-- date_from_unix_date
SELECT date_from_unix_date(1);
+----------------------+
|date_from_unix_date(1)|
+----------------------+
|            1970-01-02|
+----------------------+
-- date_part
SELECT date_part('YEAR', TIMESTAMP '2019-08-12 01:00:00.123456');
+-------------------------------------------------------+
|date_part(YEAR, TIMESTAMP '2019-08-12 01:00:00.123456')|
+-------------------------------------------------------+
|                                                   2019|
```

```
+-----------------------------------------------------+
SELECT date_part('week', timestamp'2019-08-12 01:00:00.123456');
+-----------------------------------------------------+
|date_part(week, TIMESTAMP '2019-08-12 01:00:00.123456')|
+-----------------------------------------------------+
|                                                   33|
+-----------------------------------------------------+
SELECT date_part('doy', DATE'2019-08-12');
+--------------------------------+
|date_part(doy, DATE '2019-08-12')|
+--------------------------------+
|                             224|
+--------------------------------+
SELECT date_part('SECONDS', timestamp'2019-10-01 00:00:01.000001');
+------------------------------------------------------------+
|date_part(SECONDS, TIMESTAMP '2019-10-01 00:00:01.000001')|
+------------------------------------------------------------+
|                                                    1.000001|
+------------------------------------------------------------+
SELECT date_part('days', interval 5 days 3 hours 7 minutes);
+-------------------------------------------------+
|date_part(days, INTERVAL '5 03:07' DAY TO MINUTE)|
+-------------------------------------------------+
|                                                5|
+-------------------------------------------------+
SELECT date_part('seconds', interval 5 hours 30 seconds 1 milliseconds 1 microseconds);
+-------------------------------------------------------------+
|date_part(seconds, INTERVAL '05:00:30.001001' HOUR TO SECOND)|
+-------------------------------------------------------------+
|                                                   30.001001|
+-------------------------------------------------------------+
SELECT date_part('MONTH', INTERVAL '2021-11' YEAR TO MONTH);
+-------------------------------------------------+
|date_part(MONTH, INTERVAL '2021-11' YEAR TO MONTH)|
+-------------------------------------------------+
|                                               11|
+-------------------------------------------------+
SELECT date_part('MINUTE', INTERVAL '123 23:55:59.002001' DAY TO SECOND);
+-----------------------------------------------------------------+
|date_part(MINUTE, INTERVAL '123 23:55:59.002001' DAY TO SECOND)|
+-----------------------------------------------------------------+
|                                                               55|
+-----------------------------------------------------------------+
-- date_sub
```

```
SELECT date_sub('2016-07-30', 1);
+-----------------------+
|date_sub(2016-07-30, 1)|
+-----------------------+
|             2016-07-29|
+-----------------------+
-- date_trunc
SELECT date_trunc('YEAR', '2015-03-05T09:32:05.359');
+----------------------------------------+
|date_trunc(YEAR, 2015-03-05T09:32:05.359)|
+----------------------------------------+
|                     2015-01-01 00:00:00|
+----------------------------------------+
SELECT date_trunc('MM', '2015-03-05T09:32:05.359');
+---------------------------------------+
|date_trunc(MM, 2015-03-05T09:32:05.359)|
+---------------------------------------+
|                    2015-03-01 00:00:00|
+---------------------------------------+
SELECT date_trunc('DD', '2015-03-05T09:32:05.359');
+---------------------------------------+
|date_trunc(DD, 2015-03-05T09:32:05.359)|
+---------------------------------------+
|                    2015-03-05 00:00:00|
+---------------------------------------+
SELECT date_trunc('HOUR', '2015-03-05T09:32:05.359');
+----------------------------------------+
|date_trunc(HOUR, 2015-03-05T09:32:05.359)|
+----------------------------------------+
|                     2015-03-05 09:00:00|
+----------------------------------------+
SELECT date_trunc('MILLISECOND', '2015-03-05T09:32:05.123456');
+-----------------------------------------------------+
|date_trunc(MILLISECOND, 2015-03-05T09:32:05.123456)|
+-----------------------------------------------------+
|                              2015-03-05 09:32:...|
+-----------------------------------------------------+
-- dateadd
SELECT dateadd('2016-07-30', 1);
+-----------------------+
|date_add(2016-07-30, 1)|
+-----------------------+
|             2016-07-31|
+-----------------------+
```

```
-- datediff
SELECT datediff('2009-07-31', '2009-07-30');
+--------------------------------+
|datediff(2009-07-31, 2009-07-30)|
+--------------------------------+
|                               1|
+--------------------------------+
SELECT datediff('2009-07-30', '2009-07-31');
+--------------------------------+
|datediff(2009-07-30, 2009-07-31)|
+--------------------------------+
|                              -1|
+--------------------------------+
-- datepart
SELECT datepart('YEAR', TIMESTAMP '2019-08-12 01:00:00.123456');
+-----------------------------------------------------------+
|datepart(YEAR FROM TIMESTAMP '2019-08-12 01:00:00.123456')|
+-----------------------------------------------------------+
|                                                       2019|
+-----------------------------------------------------------+
SELECT datepart('week', timestamp'2019-08-12 01:00:00.123456');
+-----------------------------------------------------------+
|datepart(week FROM TIMESTAMP '2019-08-12 01:00:00.123456')|
+-----------------------------------------------------------+
|                                                         33|
+-----------------------------------------------------------+
SELECT datepart('doy', DATE'2019-08-12');
+-----------------------------------+
|datepart(doy FROM DATE '2019-08-12')|
+-----------------------------------+
|                                224|
+-----------------------------------+
SELECT datepart('SECONDS', timestamp'2019-10-01 00:00:01.000001');
+------------------------------------------------------------+
|datepart(SECONDS FROM TIMESTAMP '2019-10-01 00:00:01.000001')|
+------------------------------------------------------------+
|                                                    1.000001|
+------------------------------------------------------------+
SELECT datepart('days', interval 5 days 3 hours 7 minutes);
+---------------------------------------------------+
|datepart(days FROM INTERVAL '5 03:07' DAY TO MINUTE)|
+---------------------------------------------------+
|                                                  5|
+---------------------------------------------------+
```

```
SELECT datepart('seconds', interval 5 hours 30 seconds 1 milliseconds 1 microseconds);
+-------------------------------------------------------------------+
|datepart(seconds FROM INTERVAL '05:00:30.001001' HOUR TO SECOND)|
+-------------------------------------------------------------------+
|                                                          30.001001|
+-------------------------------------------------------------------+
SELECT datepart('MONTH', INTERVAL '2021-11' YEAR TO MONTH);
+-----------------------------------------------------+
|datepart(MONTH FROM INTERVAL '2021-11' YEAR TO MONTH)|
+-----------------------------------------------------+
|                                                   11|
+-----------------------------------------------------+
SELECT datepart('MINUTE', INTERVAL '123 23:55:59.002001' DAY TO SECOND);
+-------------------------------------------------------------------+
|datepart(MINUTE FROM INTERVAL '123 23:55:59.002001' DAY TO SECOND)|
+-------------------------------------------------------------------+
|                                                                 55|
+-------------------------------------------------------------------+
-- day
SELECT day('2009-07-30');
+---------------+
|day(2009-07-30)|
+---------------+
|             30|
+---------------+
-- dayofmonth
SELECT dayofmonth('2009-07-30');
+----------------------+
|dayofmonth(2009-07-30)|
+----------------------+
|                    30|
+----------------------+
-- dayofweek
SELECT dayofweek('2009-07-30');
+---------------------+
|dayofweek(2009-07-30)|
+---------------------+
|                    5|
+---------------------+
-- dayofyear
SELECT dayofyear('2016-04-09');
+---------------------+
|dayofyear(2016-04-09)|
+---------------------+
```

```
|                  100|
+---------------------+
-- extract
SELECT extract(YEAR FROM TIMESTAMP '2019-08-12 01:00:00.123456');
+-----------------------------------------------------------+
|extract(YEAR FROM TIMESTAMP '2019-08-12 01:00:00.123456')|
+-----------------------------------------------------------+
|                                                       2019|
+-----------------------------------------------------------+
SELECT extract(week FROM timestamp'2019-08-12 01:00:00.123456');
+-----------------------------------------------------------+
|extract(week FROM TIMESTAMP '2019-08-12 01:00:00.123456')|
+-----------------------------------------------------------+
|                                                         33|
+-----------------------------------------------------------+
SELECT extract(doy FROM DATE'2019-08-12');
+----------------------------------+
|extract(doy FROM DATE '2019-08-12')|
+----------------------------------+
|                               224|
+----------------------------------+
SELECT extract(SECONDS FROM timestamp'2019-10-01 00:00:01.000001');
+-------------------------------------------------------------+
|extract(SECONDS FROM TIMESTAMP '2019-10-01 00:00:01.000001')|
+-------------------------------------------------------------+
|                                                     1.000001|
+-------------------------------------------------------------+
SELECT extract(days FROM interval 5 days 3 hours 7 minutes);
+---------------------------------------------------+
|extract(days FROM INTERVAL '5 03:07' DAY TO MINUTE)|
+---------------------------------------------------+
|                                                  5|
+---------------------------------------------------+
SELECT extract(seconds FROM interval 5 hours 30 seconds 1 milliseconds 1 microseconds);
+-----------------------------------------------------------------+
|extract(seconds FROM INTERVAL '05:00:30.001001' HOUR TO SECOND)|
+-----------------------------------------------------------------+
|                                                        30.001001|
+-----------------------------------------------------------------+
SELECT extract(MONTH FROM INTERVAL '2021-11' YEAR TO MONTH);
+---------------------------------------------------+
|extract(MONTH FROM INTERVAL '2021-11' YEAR TO MONTH)|
+---------------------------------------------------+
|                                                  11|
```

```
+-----------------------------------------------------+
SELECT extract(MINUTE FROM INTERVAL '123 23:55:59.002001' DAY TO SECOND);
+-----------------------------------------------------------------+
|extract(MINUTE FROM INTERVAL '123 23:55:59.002001' DAY TO SECOND)|
+-----------------------------------------------------------------+
|                                                               55|
+-----------------------------------------------------------------+
-- from_unixtime
SELECT from_unixtime(0, 'yyyy-MM-dd HH:mm:ss');
+-------------------------------------+
|from_unixtime(0, yyyy-MM-dd HH:mm:ss)|
+-------------------------------------+
|                  1970-01-01 09:00:00|
+-------------------------------------+
SELECT from_unixtime(0);
+-------------------------------------+
|from_unixtime(0, yyyy-MM-dd HH:mm:ss)|
+-------------------------------------+
|                  1970-01-01 09:00:00|
+-------------------------------------+
-- from_utc_timestamp
SELECT from_utc_timestamp('2016-08-31', 'Asia/Seoul');
+-----------------------------------------+
|from_utc_timestamp(2016-08-31, Asia/Seoul)|
+-----------------------------------------+
|                      2016-08-31 09:00:00|
+-----------------------------------------+
-- hour
SELECT hour('2009-07-30 12:58:59');
+-------------------------+
|hour(2009-07-30 12:58:59)|
+-------------------------+
|                       12|
+-------------------------+
-- last_day
SELECT last_day('2009-01-12');
+--------------------+
|last_day(2009-01-12)|
+--------------------+
|          2009-01-31|
+--------------------+
-- localtimestamp
SELECT localtimestamp();
+--------------------+
```

```
|    localtimestamp()|
+--------------------+
|2024-02-24 16:36:...|
+--------------------+
-- make_date
SELECT make_date(2013, 7, 15);
+----------------------+
|make_date(2013, 7, 15)|
+----------------------+
|            2013-07-15|
+----------------------+
SELECT make_date(2019, 7, NULL);
+------------------------+
|make_date(2019, 7, NULL)|
+------------------------+
|                    NULL|
+------------------------+
-- make_dt_interval
SELECT make_dt_interval(1, 12, 30, 01.001001);
+-------------------------------------+
|make_dt_interval(1, 12, 30, 1.001001)|
+-------------------------------------+
|                 INTERVAL '1 12:30...|
+-------------------------------------+
SELECT make_dt_interval(2);
+---------------------------------+
|make_dt_interval(2, 0, 0, 0.000000)|
+---------------------------------+
|               INTERVAL '2 00:00...|
+---------------------------------+
SELECT make_dt_interval(100, null, 3);
+--------------------------------------+
|make_dt_interval(100, NULL, 3, 0.000000)|
+--------------------------------------+
|                                  NULL|
+--------------------------------------+
-- make_interval
SELECT make_interval(100, 11, 1, 1, 12, 30, 01.001001);
+-----------------------------------------------+
|make_interval(100, 11, 1, 1, 12, 30, 1.001001)|
+-----------------------------------------------+
|                         100 years 11 mont...|
+-----------------------------------------------+
SELECT make_interval(100, null, 3);
```

```
+-----------------------------------------------+
|make_interval(100, NULL, 3, 0, 0, 0, 0.000000)|
+-----------------------------------------------+
|                                           NULL|
+-----------------------------------------------+
SELECT make_interval(0, 1, 0, 1, 0, 0, 100.000001);
+-----------------------------------------------+
|make_interval(0, 1, 0, 1, 0, 0, 100.000001)|
+-----------------------------------------------+
|                       1 months 1 days 1...|
+-----------------------------------------------+
-- make_timestamp
SELECT make_timestamp(2014, 12, 28, 6, 30, 45.887);
+-----------------------------------------------+
|make_timestamp(2014, 12, 28, 6, 30, 45.887)|
+-----------------------------------------------+
|                       2014-12-28 06:30:...|
+-----------------------------------------------+
SELECT make_timestamp(2014, 12, 28, 6, 30, 45.887, 'CET');
+-----------------------------------------------------+
|make_timestamp(2014, 12, 28, 6, 30, 45.887, CET)|
+-----------------------------------------------------+
|                             2014-12-28 14:30:...|
+-----------------------------------------------------+
SELECT make_timestamp(2019, 6, 30, 23, 59, 60);
+---------------------------------------+
|make_timestamp(2019, 6, 30, 23, 59, 60)|
+---------------------------------------+
|                     2019-07-01 00:00:00|
+---------------------------------------+
SELECT make_timestamp(2019, 6, 30, 23, 59, 1);
+--------------------------------------+
|make_timestamp(2019, 6, 30, 23, 59, 1)|
+--------------------------------------+
|                   2019-06-30 23:59:01|
+--------------------------------------+
SELECT make_timestamp(null, 7, 22, 15, 30, 0);
+-------------------------------------+
|make_timestamp(NULL, 7, 22, 15, 30, 0)|
+-------------------------------------+
|                                 NULL|
+-------------------------------------+
-- make_timestamp_ltz
SELECT make_timestamp_ltz(2014, 12, 28, 6, 30, 45.887);
```

```
+-----------------------------------------------+
|make_timestamp_ltz(2014, 12, 28, 6, 30, 45.887)|
+-----------------------------------------------+
|                           2014-12-28 06:30:...|
+-----------------------------------------------+
SELECT make_timestamp_ltz(2014, 12, 28, 6, 30, 45.887, 'CET');
+-------------------------------------------------------+
|make_timestamp_ltz(2014, 12, 28, 6, 30, 45.887, CET)|
+-------------------------------------------------------+
|                                2014-12-28 14:30:...|
+-------------------------------------------------------+
SELECT make_timestamp_ltz(2019, 6, 30, 23, 59, 60);
+-----------------------------------------+
|make_timestamp_ltz(2019, 6, 30, 23, 59, 60)|
+-----------------------------------------+
|                      2019-07-01 00:00:00|
+-----------------------------------------+
SELECT make_timestamp_ltz(null, 7, 22, 15, 30, 0);
+-----------------------------------------+
|make_timestamp_ltz(NULL, 7, 22, 15, 30, 0)|
+-----------------------------------------+
|                                     NULL|
+-----------------------------------------+
-- make_timestamp_ntz
SELECT make_timestamp_ntz(2014, 12, 28, 6, 30, 45.887);
+-----------------------------------------------+
|make_timestamp_ntz(2014, 12, 28, 6, 30, 45.887)|
+-----------------------------------------------+
|                           2014-12-28 06:30:...|
+-----------------------------------------------+
SELECT make_timestamp_ntz(2019, 6, 30, 23, 59, 60);
+-----------------------------------------+
|make_timestamp_ntz(2019, 6, 30, 23, 59, 60)|
+-----------------------------------------+
|                      2019-07-01 00:00:00|
+-----------------------------------------+
SELECT make_timestamp_ntz(null, 7, 22, 15, 30, 0);
+-----------------------------------------+
|make_timestamp_ntz(NULL, 7, 22, 15, 30, 0)|
+-----------------------------------------+
|                                     NULL|
+-----------------------------------------+
-- make_ym_interval
SELECT make_ym_interval(1, 2);
```

```
+----------------------+
|make_ym_interval(1, 2)|
+----------------------+
|  INTERVAL '1-2' YE...|
+----------------------+
SELECT make_ym_interval(1, 0);
+----------------------+
|make_ym_interval(1, 0)|
+----------------------+
|  INTERVAL '1-0' YE...|
+----------------------+
SELECT make_ym_interval(-1, 1);
+----------------------+
|make_ym_interval(-1, 1)|
+----------------------+
|   INTERVAL '-0-11' ...|
+----------------------+
SELECT make_ym_interval(2);
+----------------------+
|make_ym_interval(2, 0)|
+----------------------+
|  INTERVAL '2-0' YE...|
+----------------------+
-- minute
SELECT minute('2009-07-30 12:58:59');
+---------------------------+
|minute(2009-07-30 12:58:59)|
+---------------------------+
|                         58|
+---------------------------+
-- month
SELECT month('2016-07-30');
+-----------------+
|month(2016-07-30)|
+-----------------+
|                7|
+-----------------+
-- months_between
SELECT months_between('1997-02-28 10:30:00', '1996-10-30');
+-----------------------------------------------------+
|months_between(1997-02-28 10:30:00, 1996-10-30, true)|
+-----------------------------------------------------+
|                                           3.94959677|
+-----------------------------------------------------+
```

```
SELECT months_between('1997-02-28 10:30:00', '1996-10-30', false);
+-------------------------------------------------------+
|months_between(1997-02-28 10:30:00, 1996-10-30, false)|
+-------------------------------------------------------+
|                                   3.9495967741935485|
+-------------------------------------------------------+
-- next_day
SELECT next_day('2015-01-14', 'TU');
+------------------------+
|next_day(2015-01-14, TU)|
+------------------------+
|              2015-01-20|
+------------------------+
-- now
SELECT now();
+--------------------+
|               now()|
+--------------------+
|2024-02-24 16:36:...|
+--------------------+
-- quarter
SELECT quarter('2016-08-31');
+-------------------+
|quarter(2016-08-31)|
+-------------------+
|                  3|
+-------------------+
-- second
SELECT second('2009-07-30 12:58:59');
+---------------------------+
|second(2009-07-30 12:58:59)|
+---------------------------+
|                         59|
+---------------------------+
-- session_window
SELECT a, session_window.start, session_window.end, count(*) as cnt FROM VALUES ('A1',
 '2021-01-01 00:00:00'), ('A1', '2021-01-01 00:04:30'), ('A1', '2021-01-01 00:10:00'),
 ('A2', '2021-01-01 00:01:00') AS tab(a, b) GROUP by a, session_window(b, '5 minutes')
 ORDER BY a, start;
+---+-------------------+-------------------+---+
|  a|              start|                end|cnt|
+---+-------------------+-------------------+---+
| A1|2021-01-01 00:00:00|2021-01-01 00:09:30|  2|
| A1|2021-01-01 00:10:00|2021-01-01 00:15:00|  1|
```

```
| A2|2021-01-01 00:01:00|2021-01-01 00:06:00|  1|
+---+-------------------+-------------------+---+
SELECT a, session_window.start, session_window.end, count(*) as cnt FROM VALUES ('A1',
 '2021-01-01 00:00:00'), ('A1', '2021-01-01 00:04:30'), ('A1', '2021-01-01 00:10:00'),
 ('A2', '2021-01-01 00:01:00'), ('A2', '2021-01-01 00:04:30') AS tab(a, b) GROUP by a,
 session_window(b, CASE WHEN a = 'A1' THEN '5 minutes' WHEN a = 'A2' THEN '1 minute'
 ELSE '10 minutes' END) ORDER BY a, start;
+---+-------------------+-------------------+---+
|  a|              start|                end|cnt|
+---+-------------------+-------------------+---+
| A1|2021-01-01 00:00:00|2021-01-01 00:09:30|  2|
| A1|2021-01-01 00:10:00|2021-01-01 00:15:00|  1|
| A2|2021-01-01 00:01:00|2021-01-01 00:02:00|  1|
| A2|2021-01-01 00:04:30|2021-01-01 00:05:30|  1|
+---+-------------------+-------------------+---+
-- timestamp_micros
SELECT timestamp_micros(1230219000123123);
+----------------------------------+
|timestamp_micros(1230219000123123)|
+----------------------------------+
|              2008-12-26 00:30:...|
+----------------------------------+
-- timestamp_millis
SELECT timestamp_millis(1230219000123);
+-------------------------------+
|timestamp_millis(1230219000123)|
+-------------------------------+
|           2008-12-26 00:30:...|
+-------------------------------+
-- timestamp_seconds
SELECT timestamp_seconds(1230219000);
+-----------------------------+
|timestamp_seconds(1230219000)|
+-----------------------------+
|          2008-12-26 00:30:00|
+-----------------------------+
SELECT timestamp_seconds(1230219000.123);
+---------------------------------+
|timestamp_seconds(1230219000.123)|
+---------------------------------+
|             2008-12-26 00:30:...|
+---------------------------------+
-- to_date
SELECT to_date('2009-07-30 04:17:52');
```

```
+---------------------------+
|to_date(2009-07-30 04:17:52)|
+---------------------------+
|                 2009-07-30|
+---------------------------+
SELECT to_date('2016-12-31', 'yyyy-MM-dd');
+-------------------------------+
|to_date(2016-12-31, yyyy-MM-dd)|
+-------------------------------+
|                     2016-12-31|
+-------------------------------+
-- to_timestamp
SELECT to_timestamp('2016-12-31 00:12:00');
+---------------------------------+
|to_timestamp(2016-12-31 00:12:00)|
+---------------------------------+
|              2016-12-31 00:12:00|
+---------------------------------+
SELECT to_timestamp('2016-12-31', 'yyyy-MM-dd');
+-----------------------------------+
|to_timestamp(2016-12-31, yyyy-MM-dd)|
+-----------------------------------+
|                2016-12-31 00:00:00|
+-----------------------------------+
-- to_timestamp_ltz
SELECT to_timestamp_ltz('2016-12-31 00:12:00');
+-----------------------------------+
|to_timestamp_ltz(2016-12-31 00:12:00)|
+-----------------------------------+
|                 2016-12-31 00:12:00|
+-----------------------------------+
SELECT to_timestamp_ltz('2016-12-31', 'yyyy-MM-dd');
+---------------------------------------+
|to_timestamp_ltz(2016-12-31, yyyy-MM-dd)|
+---------------------------------------+
|                     2016-12-31 00:00:00|
+---------------------------------------+
-- to_timestamp_ntz
SELECT to_timestamp_ntz('2016-12-31 00:12:00');
+-----------------------------------+
|to_timestamp_ntz(2016-12-31 00:12:00)|
+-----------------------------------+
|                 2016-12-31 00:12:00|
+-----------------------------------+
```

```
SELECT to_timestamp_ntz('2016-12-31', 'yyyy-MM-dd');
+----------------------------------------+
|to_timestamp_ntz(2016-12-31, yyyy-MM-dd)|
+----------------------------------------+
|                     2016-12-31 00:00:00|
+----------------------------------------+
-- to_unix_timestamp
SELECT to_unix_timestamp('2016-04-08', 'yyyy-MM-dd');
+----------------------------------------+
|to_unix_timestamp(2016-04-08, yyyy-MM-dd)|
+----------------------------------------+
|                                1460041200|
+----------------------------------------+
-- to_utc_timestamp
SELECT to_utc_timestamp('2016-08-31', 'Asia/Seoul');
+----------------------------------------+
|to_utc_timestamp(2016-08-31, Asia/Seoul)|
+----------------------------------------+
|                     2016-08-30 15:00:00|
+----------------------------------------+
-- trunc
SELECT trunc('2019-08-04', 'week');
+-----------------------+
|trunc(2019-08-04, week)|
+-----------------------+
|             2019-07-29|
+-----------------------+
SELECT trunc('2019-08-04', 'quarter');
+---------------------------+
|trunc(2019-08-04, quarter)|
+---------------------------+
|                2019-07-01|
+---------------------------+
SELECT trunc('2009-02-12', 'MM');
+---------------------+
|trunc(2009-02-12, MM)|
+---------------------+
|           2009-02-01|
+---------------------+
SELECT trunc('2015-10-27', 'YEAR');
+-----------------------+
|trunc(2015-10-27, YEAR)|
+-----------------------+
|             2015-01-01|
```

```
+-----------------------+
-- try_to_timestamp
SELECT try_to_timestamp('2016-12-31 00:12:00');
+-------------------------------------+
|try_to_timestamp(2016-12-31 00:12:00)|
+-------------------------------------+
|                  2016-12-31 00:12:00|
+-------------------------------------+
SELECT try_to_timestamp('2016-12-31', 'yyyy-MM-dd');
+---------------------------------------+
|try_to_timestamp(2016-12-31, yyyy-MM-dd)|
+---------------------------------------+
|                     2016-12-31 00:00:00|
+---------------------------------------+
SELECT try_to_timestamp('foo', 'yyyy-MM-dd');
+---------------------------------+
|try_to_timestamp(foo, yyyy-MM-dd)|
+---------------------------------+
|                             NULL|
+---------------------------------+
-- unix_date
SELECT unix_date(DATE("1970-01-02"));
+---------------------+
|unix_date(1970-01-02)|
+---------------------+
|                    1|
+---------------------+
-- unix_micros
SELECT unix_micros(TIMESTAMP('1970-01-01 00:00:01Z'));
+--------------------------------+
|unix_micros(1970-01-01 00:00:01Z)|
+--------------------------------+
|                         1000000|
+--------------------------------+
-- unix_millis
SELECT unix_millis(TIMESTAMP('1970-01-01 00:00:01Z'));
+--------------------------------+
|unix_millis(1970-01-01 00:00:01Z)|
+--------------------------------+
|                            1000|
+--------------------------------+
-- unix_seconds
SELECT unix_seconds(TIMESTAMP('1970-01-01 00:00:01Z'));
+--------------------------------+
```

```
|unix_seconds(1970-01-01 00:00:01Z)|
+----------------------------------+
|                                 1|
+----------------------------------+
-- unix_timestamp
SELECT unix_timestamp();
+-----------------------------------------------------------+
|unix_timestamp(current_timestamp(), yyyy-MM-dd HH:mm:ss)|
+-----------------------------------------------------------+
|                                                 1708760216|
+-----------------------------------------------------------+
SELECT unix_timestamp('2016-04-08', 'yyyy-MM-dd');
+--------------------------------------+
|unix_timestamp(2016-04-08, yyyy-MM-dd)|
+--------------------------------------+
|                            1460041200|
+--------------------------------------+
-- weekday
SELECT weekday('2009-07-30');
+-------------------+
|weekday(2009-07-30)|
+-------------------+
|                  3|
+-------------------+
-- weekofyear
SELECT weekofyear('2008-02-20');
+----------------------+
|weekofyear(2008-02-20)|
+----------------------+
|                     8|
+----------------------+
-- window
SELECT a, window.start, window.end, count(*) as cnt FROM VALUES ('A1', '2021-01-01
 00:00:00'), ('A1', '2021-01-01 00:04:30'), ('A1', '2021-01-01 00:06:00'), ('A2',
 '2021-01-01 00:01:00') AS tab(a, b) GROUP by a, window(b, '5 minutes') ORDER BY a,
 start;
+---+-------------------+-------------------+---+
|  a|              start|                end|cnt|
+---+-------------------+-------------------+---+
| A1|2021-01-01 00:00:00|2021-01-01 00:05:00|  2|
| A1|2021-01-01 00:05:00|2021-01-01 00:10:00|  1|
| A2|2021-01-01 00:00:00|2021-01-01 00:05:00|  1|
+---+-------------------+-------------------+---+
```

```
SELECT a, window.start, window.end, count(*) as cnt FROM VALUES ('A1', '2021-01-01
 00:00:00'), ('A1', '2021-01-01 00:04:30'), ('A1', '2021-01-01 00:06:00'), ('A2',
 '2021-01-01 00:01:00') AS tab(a, b) GROUP by a, window(b, '10 minutes', '5 minutes')
 ORDER BY a, start;
+---+------------------+------------------+---+
|  a|             start|               end|cnt|
+---+------------------+------------------+---+
| A1|2020-12-31 23:55:00|2021-01-01 00:05:00|  2|
| A1|2021-01-01 00:00:00|2021-01-01 00:10:00|  3|
| A1|2021-01-01 00:05:00|2021-01-01 00:15:00|  1|
| A2|2020-12-31 23:55:00|2021-01-01 00:05:00|  1|
| A2|2021-01-01 00:00:00|2021-01-01 00:10:00|  1|
+---+------------------+------------------+---+
-- window_time
SELECT a, window.start as start, window.end as end, window_time(window), cnt FROM
 (SELECT a, window, count(*) as cnt FROM VALUES ('A1', '2021-01-01 00:00:00'), ('A1',
 '2021-01-01 00:04:30'), ('A1', '2021-01-01 00:06:00'), ('A2', '2021-01-01 00:01:00')
 AS tab(a, b) GROUP by a, window(b, '5 minutes') ORDER BY a, window.start);
+---+------------------+------------------+--------------------+---+
|  a|             start|               end| window_time(window)|cnt|
+---+------------------+------------------+--------------------+---+
| A1|2021-01-01 00:00:00|2021-01-01 00:05:00|2021-01-01 00:04:...|  2|
| A1|2021-01-01 00:05:00|2021-01-01 00:10:00|2021-01-01 00:09:...|  1|
| A2|2021-01-01 00:00:00|2021-01-01 00:05:00|2021-01-01 00:04:...|  1|
+---+------------------+------------------+--------------------+---+
-- year
SELECT year('2016-07-30');
+----------------+
|year(2016-07-30)|
+----------------+
|            2016|
+----------------+
```

Aggregate functions

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

Aggregate functions operate on values across rows to perform mathematical calculations such as sum, average, counting, minimum/maximum values, standard deviation, and estimation, as well as some non-mathematical operations.

Syntax

```
aggregate_function(input1 [, input2, ...]) FILTER (WHERE boolean_expression)
```

Parameters

- `boolean_expression` - Specifies any expression that evaluates to a result type boolean. Two or more expressions may be combined together using the logical operators (AND, OR).

Ordered-set aggregate functions

These aggregate functions use different syntax than the other aggregate functions so that to specify an expression (typically a column name) by which to order the values.

Syntax

```
{ PERCENTILE_CONT | PERCENTILE_DISC }(percentile) WITHIN GROUP (ORDER BY
 { order_by_expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [ , ... ] }) FILTER
 (WHERE boolean_expression)
```

Parameters

- `percentile` - The percentile of the value that you want to find. The percentile must be a constant between 0.0 and 1.0.
- `order_by_expression` - The expression (typically a column name) by which to order the values before aggregating them.
- `boolean_expression` - Specifies any expression that evaluates to a result type boolean. Two or more expressions may be combined together using the logical operators (AND, OR).

Examples

```
CREATE OR REPLACE TEMPORARY VIEW basic_pays AS SELECT * FROM VALUES
('Jane Doe','Accounting',8435),
('Akua Mansa','Accounting',9998),
('John Doe','Accounting',8992),
```

```
('Juan Li','Accounting',8870),
('Carlos Salazar','Accounting',11472),
('Arnav Desai','Accounting',6627),
('Saanvi Sarkar','IT',8113),
('Shirley Rodriguez','IT',5186),
('Nikki Wolf','Sales',9181),
('Alejandro Rosalez','Sales',9441),
('Nikhil Jayashankar','Sales',6660),
('Richard Roe','Sales',10563),
('Pat Candella','SCM',10449),
('Gerard Hernandez','SCM',6949),
('Pamela Castillo','SCM',11303),
('Paulo Santos','SCM',11798),
('Jorge Souza','SCM',10586)
AS basic_pays(employee_name, department, salary);
SELECT * FROM basic_pays;
+------------------+----------+------+
|    employee_name  |department|salary|
+------------------+----------+------+
| Arnav Desai       |Accounting|  6627|
| Jorge Souza       |       SCM| 10586|
| Jane Doe          |Accounting|  8435|
| Nikhil Jayashankar|     Sales|  6660|
| Diego Vanauf      |     Sales| 10563|
| Carlos Salazar    |Accounting| 11472|
| Gerard Hernandez  |       SCM|  6949|
| John Doe          |Accounting|  8992|
| Nikki Wolf        |     Sales|  9181|
| Paulo Santos      |       SCM| 11798|
| Saanvi Sarkar     |        IT|  8113|
| Shirley Rodriguez |        IT|  5186|
| Pat Candella      |       SCM| 10449|
| Akua Mansa        |Accounting|  9998|
| Pamela Castillo   |       SCM| 11303|
| Alejandro Rosalez |     Sales|  9441|
| Juan Li           |Accounting|  8870|
+------------------+----------+------+
SELECT
department,
percentile_cont(0.25) WITHIN GROUP (ORDER BY salary) AS pc1,
percentile_cont(0.25) WITHIN GROUP (ORDER BY salary) FILTER (WHERE employee_name LIKE
 '%Bo%') AS pc2,
percentile_cont(0.25) WITHIN GROUP (ORDER BY salary DESC) AS pc3,
```

```
percentile_cont(0.25) WITHIN GROUP (ORDER BY salary DESC) FILTER (WHERE employee_name
 LIKE '%Bo%') AS pc4,
percentile_disc(0.25) WITHIN GROUP (ORDER BY salary) AS pd1,
percentile_disc(0.25) WITHIN GROUP (ORDER BY salary) FILTER (WHERE employee_name LIKE
 '%Bo%') AS pd2,
percentile_disc(0.25) WITHIN GROUP (ORDER BY salary DESC) AS pd3,
percentile_disc(0.25) WITHIN GROUP (ORDER BY salary DESC) FILTER (WHERE employee_name
 LIKE '%Bo%') AS pd4
FROM basic_pays
GROUP BY department
ORDER BY department;
+----------+-------+--------+-------+--------+-----+-----+-----+-----+
|department|    pc1|     pc2|    pc3|     pc4|  pd1|  pd2|  pd3|  pd4|
+----------+-------+--------+-------+--------+-----+-----+-----+-----+
|Accounting|8543.75| 7838.25| 9746.5|10260.75| 8435| 6627| 9998|11472|
|        IT|5917.75|    NULL|7381.25|    NULL| 5186| NULL| 8113| NULL|
|     Sales|8550.75|    NULL| 9721.5|    NULL| 6660| NULL|10563| NULL|
|       SCM|10449.0|10786.25|11303.0|11460.75|10449|10449|11303|11798|
+----------+-------+--------+-------+--------+-----+-----+-----+-----+
```

Conditional functions

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

Function	Description
coalesce(expr1, expr2, ...)	Returns the first non-null argument if exists. Otherwise, null.
if(expr1, expr2, expr3)	If `expr1` evaluates to true, then returns `expr2`; otherwise returns `expr3`.
ifnull(expr1, expr2)	Returns `expr2` if `expr1` is null, or `expr1` otherwise.

Function	Description
nanvl(expr1, expr2)	Returns `expr1` if it's not NaN, or `expr2` otherwise.
nullif(expr1, expr2)	Returns null if `expr1` equals to `expr2`, or `expr1` otherwise.
nvl(expr1, expr2)	Returns `expr2` if `expr1` is null, or `expr1` otherwise.
nvl2(expr1, expr2, expr3)	Returns `expr2` if `expr1` is not null, or `expr3` otherwise.
CASE WHEN expr1 THEN expr2 [WHEN expr3 THEN expr4]* [ELSE expr5] END	When `expr1` = true, returns `expr2`; else when `expr3` = true, returns `expr4`; else returns `expr5`.

Examples

```
-- coalesce
SELECT coalesce(NULL, 1, NULL);
+----------------------+
|coalesce(NULL, 1, NULL)|
+----------------------+
|                     1|
+----------------------+
-- if
SELECT if(1 < 2, 'a', 'b');
+------------------+
|(IF((1 < 2), a, b))|
+------------------+
|                 a|
+------------------+
-- ifnull
SELECT ifnull(NULL, array('2'));
+---------------------+
|ifnull(NULL, array(2))|
+---------------------+
|                  [2]|
```

```
+----------------------+
-- nanvl
SELECT nanvl(cast('NaN' as double), 123);
+------------------------------+
|nanvl(CAST(NaN AS DOUBLE), 123)|
+------------------------------+
|                         123.0|
+------------------------------+
-- nullif
SELECT nullif(2, 2);
+------------+
|nullif(2, 2)|
+------------+
|        NULL|
+------------+
-- nvl
SELECT nvl(NULL, array('2'));
+-------------------+
|nvl(NULL, array(2))|
+-------------------+
|                [2]|
+-------------------+
-- nvl2
SELECT nvl2(NULL, 2, 1);
+----------------+
|nvl2(NULL, 2, 1)|
+----------------+
|               1|
+----------------+
-- when
SELECT CASE WHEN 1 > 0 THEN 1 WHEN 2 > 0 THEN 2.0 ELSE 1.2 END;
+-----------------------------------------------------------+
|CASE WHEN (1 > 0) THEN 1 WHEN (2 > 0) THEN 2.0 ELSE 1.2 END|
+-----------------------------------------------------------+
|                                                        1.0|
+-----------------------------------------------------------+
SELECT CASE WHEN 1 < 0 THEN 1 WHEN 2 > 0 THEN 2.0 ELSE 1.2 END;
+-----------------------------------------------------------+
|CASE WHEN (1 < 0) THEN 1 WHEN (2 > 0) THEN 2.0 ELSE 1.2 END|
+-----------------------------------------------------------+
|                                                        2.0|
+-----------------------------------------------------------+
SELECT CASE WHEN 1 < 0 THEN 1 WHEN 2 < 0 THEN 2.0 END;
+-------------------------------------------------+
```

```
|CASE WHEN (1 < 0) THEN 1 WHEN (2 < 0) THEN 2.0 END|
+--------------------------------------------------+
|                                              NULL|
+--------------------------------------------------+
```

JSON functions

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called "Supported SQL commands".

Function	Description
from_json (jsonStr, schema[, options])	Returns a struct value with the given `jsonStr` and `schema`.
get_json_ object(js on_txt, path)	Extracts a json object from `path`.
json_arra y_length(jsonArray)	Returns the number of elements in the outermost JSON array.
json_obje ct_keys(j son_object)	Returns all the keys of the outermost JSON object as an array.
json_tupl e(jsonStr, p1, p2, ..., pn)	Returns a tuple like the function get_json_object, but it takes multiple names. All the input parameters and output column types are string.

Function	Description
schema_of_json(json[, options])	Returns schema in the DDL format of JSON string.
to_json(expr[, options])	Returns a JSON string with a given struct value

Examples

```
-- from_json
SELECT from_json('{"a":1, "b":0.8}', 'a INT, b DOUBLE');
+---------------------------+
| from_json({"a":1, "b":0.8}) |
+---------------------------+
| {1, 0.8}                  |
+---------------------------+

SELECT from_json('{"time":"26/08/2015"}', 'time Timestamp', map('timestampFormat', 'dd/
MM/yyyy'));
+--------------------------------+
| from_json({"time":"26/08/2015"}) |
+--------------------------------+
| {2015-08-26 00:00...           |
+--------------------------------+

SELECT from_json('{"teacher": "Alice", "student": [{"name": "Bob", "rank": 1}, {"name":
 "Charlie", "rank": 2}]}', 'STRUCT<teacher: STRING, student: ARRAY<STRUCT<name: STRING,
 rank: INT>>>');
+-----------------------------------------------------------------------------------------------
+
| from_json({"teacher": "Alice", "student": [{"name": "Bob", "rank": 1}, {"name":
 "Charlie", "rank": 2}]}) |
+-----------------------------------------------------------------------------------------------
+
| {Alice, [{Bob, 1}...
                  |
+-----------------------------------------------------------------------------------------------
+
```

```
-- get_json_object
SELECT get_json_object('{"a":"b"}', '$.a');
+-------------------------------+
| get_json_object({"a":"b"}, $.a) |
+-------------------------------+
| b                             |
+-------------------------------+

-- json_array_length
SELECT json_array_length('[1,2,3,4]');
+----------------------------+
| json_array_length([1,2,3,4]) |
+----------------------------+
| 4                          |
+----------------------------+

SELECT json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4]');
+-------------------------------------------------+
| json_array_length([1,2,3,{"f1":1,"f2":[5,6]},4]) |
+-------------------------------------------------+
| 5                                               |
+-------------------------------------------------+

SELECT json_array_length('[1,2');
+-----------------------+
| json_array_length([1,2) |
+-----------------------+
| NULL                  |
+-----------------------+

-- json_object_keys
SELECT json_object_keys('{}');
+--------------------+
| json_object_keys({}) |
+--------------------+
| []                 |
+--------------------+

SELECT json_object_keys('{"key": "value"}');
+---------------------------------+
| json_object_keys({"key": "value"}) |
+---------------------------------+
| [key]                           |
```

```
+---------------------------------+

SELECT json_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}}');
+---------------------------------------------------------+
| json_object_keys({"f1":"abc","f2":{"f3":"a", "f4":"b"}}) |
+---------------------------------------------------------+
| [f1, f2]                                                |
+---------------------------------------------------------+

-- json_tuple
SELECT json_tuple('{"a":1, "b":2}', 'a', 'b');
+---+---+
| c0| c1|
+---+---+
|  1|  2|
+---+---+

-- schema_of_json
SELECT schema_of_json('[{"col":0}]');
+---------------------------+
| schema_of_json([{"col":0}]) |
+---------------------------+
| ARRAY<STRUCT<col:...      |
+---------------------------+

SELECT schema_of_json('[{"col":01}]', map('allowNumericLeadingZeros', 'true'));
+----------------------------+
| schema_of_json([{"col":01}]) |
+----------------------------+
| ARRAY<STRUCT<col:...       |
+----------------------------+

-- to_json
SELECT to_json(named_struct('a', 1, 'b', 2));
+---------------------------------+
| to_json(named_struct(a, 1, b, 2)) |
+---------------------------------+
| {"a":1,"b":2}                   |
+---------------------------------+

SELECT to_json(named_struct('time', to_timestamp('2015-08-26', 'yyyy-MM-dd')),
 map('timestampFormat', 'dd/MM/yyyy'));
+------------------------------------------------------------------+
| to_json(named_struct(time, to_timestamp(2015-08-26, yyyy-MM-dd))) |
```

```
+------------------------------------------------------------------+
| {"time":"26/08/20...                                             |
+------------------------------------------------------------------+

SELECT to_json(array(named_struct('a', 1, 'b', 2)));
+----------------------------------------+
| to_json(array(named_struct(a, 1, b, 2))) |
+----------------------------------------+
| [{"a":1,"b":2}]                        |
+----------------------------------------+

SELECT to_json(map('a', named_struct('b', 1)));
+-----------------------------------+
| to_json(map(a, named_struct(b, 1))) |
+-----------------------------------+
| {"a":{"b":1}}                     |
+-----------------------------------+

SELECT to_json(map(named_struct('a', 1),named_struct('b', 2)));
+-------------------------------------------------------+
| to_json(map(named_struct(a, 1), named_struct(b, 2))) |
+-------------------------------------------------------+
| {"[1]":{"b":2}}                                       |
+-------------------------------------------------------+

SELECT to_json(map('a', 1));
+------------------+
| to_json(map(a, 1)) |
+------------------+
| {"a":1}          |
+------------------+

SELECT to_json(array(map('a', 1)));
+-------------------------+
| to_json(array(map(a, 1))) |
+-------------------------+
| [{"a":1}]               |
+-------------------------+
```

Array functions

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

Function	Description
array(expr, ...)	Returns an array with the given elements.
array_append(array, element)	Add the element at the end of the array passed as first argument. Type of element should be similar to type of the elements of the array. Null element is also appended into the array. But if the array passed, is NULL output is NULL
array_compact(array)	Removes null values from the array.
array_contains(array, value)	Returns true if the array contains the value.
array_distinct(array)	Removes duplicate values from the array.
array_except(array1, array2)	Returns an array of the elements in array1 but not in array2, without duplicates.
array_insert(x, pos, val)	Places val into index pos of array x. Array indices start at 1. The maximum negative index is -1 for which the function inserts new element after the current last element. Index above array size appends the array, or prepends the array if index is negative, with 'null' elements.
array_intersect(array1, array2)	Returns an array of the elements in the intersection of array1 and array2, without duplicates.

Function	Description
array_join(array, delimiter[, nullReplacement])	Concatenates the elements of the given array using the delimiter and an optional string to replace nulls. If no value is set for nullRepla cement, any null value is filtered.
array_max(array)	Returns the maximum value in the array. NaN is greater than any non-NaN elements for double/float type. NULL elements are skipped.
array_min(array)	Returns the minimum value in the array. NaN is greater than any non-NaN elements for double/float type. NULL elements are skipped.
array_position(array, element)	Returns the (1-based) index of the first matching element of the array as long, or 0 if no match is found.
array_prepend(array, element)	Add the element at the beginning of the array passed as first argument. Type of element should be the same as the type of the elements of the array. Null element is also prepended to the array. But if the array passed is NULL output is NULL
array_remove(array, element)	Remove all elements that equal to element from array.
array_repeat(element, count)	Returns the array containing element count times.
array_union(array1, array2)	Returns an array of the elements in the union of array1 and array2, without duplicates.

Function	Description
arrays_overlap(a1, a2)	Returns true if a1 contains at least a non-null element present also in a2. If the arrays have no common element and they are both non-empty and either of them contains a null element null is returned, false otherwise.
arrays_zip(a1, a2, ...)	Returns a merged array of structs in which the N-th struct contains all N-th values of input arrays.
flatten(arrayOfArrays)	Transforms an array of arrays into a single array.
get(array, index)	Returns element of array at given (0-based) index. If the index points outside of the array boundaries, then this function returns NULL.
sequence(start, stop, step)	Generates an array of elements from start to stop (inclusive), incrementing by step. The type of the returned elements is the same as the type of argument expressions. Supported types are: byte, short, integer, long, date, timestamp. The start and stop expressions must resolve to the same type. If start and stop expressions resolve to the 'date' or 'timestamp' type then the step expressio n must resolve to the 'interval' or 'year-month interval' or 'day-time interval' type, otherwise to the same type as the start and stop expressions.
shuffle(array)	Returns a random permutation of the given array.

Function	Description
slice(x, start, length)	Subsets array x starting from index start (array indices start at 1, or starting from the end if start is negative) with the specified length.
sort_array(array[, ascendingOrder])	Sorts the input array in ascending or descending order according to the natural ordering of the array elements. NaN is greater than any non-NaN elements for double/fl oat type. Null elements will be placed at the beginning of the returned array in ascending order or at the end of the returned array in descending order.

Examples

```
-- array
SELECT array(1, 2, 3);
+--------------+
|array(1, 2, 3)|
+--------------+
|     [1, 2, 3]|
+--------------+
-- array_append
SELECT array_append(array('b', 'd', 'c', 'a'), 'd');
+---------------------------------+
|array_append(array(b, d, c, a), d)|
+---------------------------------+
|                   [b, d, c, a, d]|
+---------------------------------+
SELECT array_append(array(1, 2, 3, null), null);
+---------------------------------------+
|array_append(array(1, 2, 3, NULL), NULL)|
+---------------------------------------+
|                    [1, 2, 3, NULL, N...|
+---------------------------------------+
SELECT array_append(CAST(null as Array<Int>), 2);
+--------------------+
```

```
|array_append(NULL, 2)|
+---------------------+
|                 NULL|
+---------------------+
-- array_compact
SELECT array_compact(array(1, 2, 3, null));
+-----------------------------------+
|array_compact(array(1, 2, 3, NULL))|
+-----------------------------------+
|                          [1, 2, 3]|
+-----------------------------------+
SELECT array_compact(array("a", "b", "c"));
+----------------------------+
|array_compact(array(a, b, c))|
+----------------------------+
|                    [a, b, c]|
+----------------------------+
-- array_contains
SELECT array_contains(array(1, 2, 3), 2);
+---------------------------------+
|array_contains(array(1, 2, 3), 2)|
+---------------------------------+
|                             true|
+---------------------------------+
-- array_distinct
SELECT array_distinct(array(1, 2, 3, null, 3));
+--------------------------------------+
|array_distinct(array(1, 2, 3, NULL, 3))|
+--------------------------------------+
|                        [1, 2, 3, NULL]|
+--------------------------------------+
-- array_except
SELECT array_except(array(1, 2, 3), array(1, 3, 5));
+-------------------------------------------+
|array_except(array(1, 2, 3), array(1, 3, 5))|
+-------------------------------------------+
|                                        [2]|
+-------------------------------------------+
-- array_insert
SELECT array_insert(array(1, 2, 3, 4), 5, 5);
+-------------------------------------+
|array_insert(array(1, 2, 3, 4), 5, 5)|
+-------------------------------------+
|                     [1, 2, 3, 4, 5]|
```

```
+-----------------------------------+
SELECT array_insert(array(5, 4, 3, 2), -1, 1);
+-----------------------------------+
|array_insert(array(5, 4, 3, 2), -1, 1)|
+-----------------------------------+
|                     [5, 4, 3, 2, 1]|
+-----------------------------------+
SELECT array_insert(array(5, 3, 2, 1), -4, 4);
+-----------------------------------+
|array_insert(array(5, 3, 2, 1), -4, 4)|
+-----------------------------------+
|                     [5, 4, 3, 2, 1]|
+-----------------------------------+
-- array_intersect
SELECT array_intersect(array(1, 2, 3), array(1, 3, 5));
+-----------------------------------------------+
|array_intersect(array(1, 2, 3), array(1, 3, 5))|
+-----------------------------------------------+
|                                         [1, 3]|
+-----------------------------------------------+
-- array_join
SELECT array_join(array('hello', 'world'), ' ');
+---------------------------------+
|array_join(array(hello, world),  )|
+---------------------------------+
|                     hello world|
+---------------------------------+
SELECT array_join(array('hello', null ,'world'), ' ');
+---------------------------------------+
|array_join(array(hello, NULL, world),  )|
+---------------------------------------+
|                           hello world|
+---------------------------------------+
SELECT array_join(array('hello', null ,'world'), ' ', ',');
+-------------------------------------------+
|array_join(array(hello, NULL, world),  , ,)|
+-------------------------------------------+
|                            hello , world|
+-------------------------------------------+
-- array_max
SELECT array_max(array(1, 20, null, 3));
+-------------------------------+
|array_max(array(1, 20, NULL, 3))|
+-------------------------------+
```

```
|                              20|
+--------------------------------+
-- array_min
SELECT array_min(array(1, 20, null, 3));
+--------------------------------+
|array_min(array(1, 20, NULL, 3))|
+--------------------------------+
|                               1|
+--------------------------------+
-- array_position
SELECT array_position(array(312, 773, 708, 708), 708);
+-----------------------------------------------+
|array_position(array(312, 773, 708, 708), 708)|
+-----------------------------------------------+
|                                              3|
+-----------------------------------------------+
SELECT array_position(array(312, 773, 708, 708), 414);
+-----------------------------------------------+
|array_position(array(312, 773, 708, 708), 414)|
+-----------------------------------------------+
|                                              0|
+-----------------------------------------------+
-- array_prepend
SELECT array_prepend(array('b', 'd', 'c', 'a'), 'd');
+----------------------------------+
|array_prepend(array(b, d, c, a), d)|
+----------------------------------+
|                    [d, b, d, c, a]|
+----------------------------------+
SELECT array_prepend(array(1, 2, 3, null), null);
+---------------------------------------+
|array_prepend(array(1, 2, 3, NULL), NULL)|
+---------------------------------------+
|                     [NULL, 1, 2, 3, N...|
+---------------------------------------+
SELECT array_prepend(CAST(null as Array<Int>), 2);
+----------------------+
|array_prepend(NULL, 2)|
+----------------------+
|                  NULL|
+----------------------+
-- array_remove
SELECT array_remove(array(1, 2, 3, null, 3), 3);
+--------------------------------------+
```

```
|array_remove(array(1, 2, 3, NULL, 3), 3)|
+----------------------------------------+
|                            [1, 2, NULL]|
+----------------------------------------+
-- array_repeat
SELECT array_repeat('123', 2);
+--------------------+
|array_repeat(123, 2)|
+--------------------+
|          [123, 123]|
+--------------------+
-- array_union
SELECT array_union(array(1, 2, 3), array(1, 3, 5));
+-------------------------------------------+
|array_union(array(1, 2, 3), array(1, 3, 5))|
+-------------------------------------------+
|                               [1, 2, 3, 5]|
+-------------------------------------------+
-- arrays_overlap
SELECT arrays_overlap(array(1, 2, 3), array(3, 4, 5));
+-----------------------------------------------+
|arrays_overlap(array(1, 2, 3), array(3, 4, 5))|
+-----------------------------------------------+
|                                           true|
+-----------------------------------------------+
-- arrays_zip
SELECT arrays_zip(array(1, 2, 3), array(2, 3, 4));
+-----------------------------------------+
|arrays_zip(array(1, 2, 3), array(2, 3, 4))|
+-----------------------------------------+
|                     [{1, 2}, {2, 3}, ...|
+-----------------------------------------+
SELECT arrays_zip(array(1, 2), array(2, 3), array(3, 4));
+-------------------------------------------------+
|arrays_zip(array(1, 2), array(2, 3), array(3, 4))|
+-------------------------------------------------+
|                             [{1, 2, 3}, {2, 3...|
+-------------------------------------------------+
-- flatten
SELECT flatten(array(array(1, 2), array(3, 4)));
+----------------------------------------+
|flatten(array(array(1, 2), array(3, 4)))|
+----------------------------------------+
|                            [1, 2, 3, 4]|
```

```
+----------------------------------------+
-- get
SELECT get(array(1, 2, 3), 0);
+---------------------+
|get(array(1, 2, 3), 0)|
+---------------------+
|                    1|
+---------------------+
SELECT get(array(1, 2, 3), 3);
+---------------------+
|get(array(1, 2, 3), 3)|
+---------------------+
|                 NULL|
+---------------------+
SELECT get(array(1, 2, 3), -1);
+----------------------+
|get(array(1, 2, 3), -1)|
+----------------------+
|                  NULL|
+----------------------+
-- sequence
SELECT sequence(1, 5);
+---------------+
| sequence(1, 5)|
+---------------+
|[1, 2, 3, 4, 5]|
+---------------+
SELECT sequence(5, 1);
+---------------+
| sequence(5, 1)|
+---------------+
|[5, 4, 3, 2, 1]|
+---------------+
SELECT sequence(to_date('2018-01-01'), to_date('2018-03-01'), interval 1 month);
+------------------------------------------------------------------------+
|sequence(to_date(2018-01-01), to_date(2018-03-01), INTERVAL '1' MONTH)|
+------------------------------------------------------------------------+
|                                                    [2018-01-01, 2018...|
+------------------------------------------------------------------------+
SELECT sequence(to_date('2018-01-01'), to_date('2018-03-01'), interval '0-1' year to
 month);
+-----------------------------------------------------------------------------------+
|sequence(to_date(2018-01-01), to_date(2018-03-01), INTERVAL '0-1' YEAR TO MONTH)|
+-----------------------------------------------------------------------------------+
```

```
|                                                                                [2018-01-01, 2018...|
+-------------------------------------------------------------------------------------------------+
-- shuffle
SELECT shuffle(array(1, 20, 3, 5));
+----------------------------+
|shuffle(array(1, 20, 3, 5))|
+----------------------------+
|              [5, 1, 20, 3]|
+----------------------------+
SELECT shuffle(array(1, 20, null, 3));
+-----------------------------+
|shuffle(array(1, 20, NULL, 3))|
+-----------------------------+
|              [1, NULL, 20, 3]|
+-----------------------------+
-- slice
SELECT slice(array(1, 2, 3, 4), 2, 2);
+-----------------------------+
|slice(array(1, 2, 3, 4), 2, 2)|
+-----------------------------+
|                        [2, 3]|
+-----------------------------+
SELECT slice(array(1, 2, 3, 4), -2, 2);
+------------------------------+
|slice(array(1, 2, 3, 4), -2, 2)|
+------------------------------+
|                         [3, 4]|
+------------------------------+
-- sort_array
SELECT sort_array(array('b', 'd', null, 'c', 'a'), true);
+----------------------------------------+
|sort_array(array(b, d, NULL, c, a), true)|
+----------------------------------------+
|                      [NULL, a, b, c, d]|
+----------------------------------------+
```

Window functions

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

Window functions operate on a group of rows, referred to as a window, and calculate a return value for each row based on the group of rows. Window functions are useful for processing tasks such as calculating a moving average, computing a cumulative statistic, or accessing the value of rows given the relative position of the current row.

Syntax

```
window_function [ nulls_option ] OVER ( [ { PARTITION | DISTRIBUTE } BY
 partition_col_name = partition_col_val ( [ , ... ] ) ] { ORDER | SORT } BY expression
 [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [ , ... ] [ window_frame ] )
```

Parameters

- Ranking functions

  Syntax: `RANK | DENSE_RANK | PERCENT_RANK | NTILE | ROW_NUMBER`

  Analytic functions

  Syntax: `CUME_DIST | LAG | LEAD | NTH_VALUE | FIRST_VALUE | LAST_VALUE`

  Aggregate functions

  Syntax: `MAX | MIN | COUNT | SUM | AVG | ...`
- `nulls_option` - Specifies whether or not to skip null values when evaluating the window function. RESPECT NULLS means not skipping null values, while IGNORE NULLS means skipping. If not specified, the default is RESPECT NULLS.

  Syntax: `{ IGNORE | RESPECT } NULLS`

  Note: `Only LAG | LEAD | NTH_VALUE | FIRST_VALUE | LAST_VALUE` can be used with `IGNORE NULLS`.
- `window_frame` - Specifies which row to start the window on and where to end it.

  Syntax: `{ RANGE | ROWS } { frame_start | BETWEEN frame_start AND frame_end }`

  frame_start and frame_end have the following syntax:

  Syntax: `UNBOUNDED PRECEDING | offset PRECEDING | CURRENT ROW | offset FOLLOWING | UNBOUNDED FOLLOWING`

offset: specifies the offset from the position of the current row.

Note If frame_end is omitted, it defaults to CURRENT ROW.

Examples

```
CREATE TABLE employees (name STRING, dept STRING, salary INT, age INT);
INSERT INTO employees VALUES ("Lisa", "Sales", 10000, 35);
INSERT INTO employees VALUES ("Evan", "Sales", 32000, 38);
INSERT INTO employees VALUES ("Fred", "Engineering", 21000, 28);
INSERT INTO employees VALUES ("Alex", "Sales", 30000, 33);
INSERT INTO employees VALUES ("Tom", "Engineering", 23000, 33);
INSERT INTO employees VALUES ("Jane", "Marketing", 29000, 28);
INSERT INTO employees VALUES ("Jeff", "Marketing", 35000, 38);
INSERT INTO employees VALUES ("Paul", "Engineering", 29000, 23);
INSERT INTO employees VALUES ("Chloe", "Engineering", 23000, 25);
SELECT * FROM employees;
+-----+-----------+------+-----+
| name|       dept|salary|  age|
+-----+-----------+------+-----+
|Chloe|Engineering| 23000|   25|
| Fred|Engineering| 21000|   28|
| Paul|Engineering| 29000|   23|
|Helen|  Marketing| 29000|   40|
|  Tom|Engineering| 23000|   33|
| Jane|  Marketing| 29000|   28|
| Jeff|  Marketing| 35000|   38|
| Evan|      Sales| 32000|   38|
| Lisa|      Sales| 10000|   35|
| Alex|      Sales| 30000|   33|
+-----+-----------+------+-----+
SELECT name, dept, salary, RANK() OVER (PARTITION BY dept ORDER BY salary) AS rank FROM
 employees;
+-----+-----------+------+----+
| name|       dept|salary|rank|
+-----+-----------+------+----+
| Lisa|      Sales| 10000|   1|
| Alex|      Sales| 30000|   2|
| Evan|      Sales| 32000|   3|
| Fred|Engineering| 21000|   1|
|  Tom|Engineering| 23000|   2|
|Chloe|Engineering| 23000|   2|
```

```
| Paul|Engineering| 29000|   4|
|Helen|  Marketing| 29000|   1|
| Jane|  Marketing| 29000|   1|
| Jeff|  Marketing| 35000|   3|
+-----+-----------+------+----+
SELECT name, dept, salary, DENSE_RANK() OVER (PARTITION BY dept ORDER BY salary ROWS
 BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW) AS dense_rank FROM employees;
+-----+-----------+------+----------+
| name|       dept|salary|dense_rank|
+-----+-----------+------+----------+
| Lisa|      Sales| 10000|         1|
| Alex|      Sales| 30000|         2|
| Evan|      Sales| 32000|         3|
| Fred|Engineering| 21000|         1|
|  Tom|Engineering| 23000|         2|
|Chloe|Engineering| 23000|         2|
| Paul|Engineering| 29000|         3|
|Helen|  Marketing| 29000|         1|
| Jane|  Marketing| 29000|         1|
| Jeff|  Marketing| 35000|         2|
+-----+-----------+------+----------+
SELECT name, dept, age, CUME_DIST() OVER (PARTITION BY dept ORDER BY age
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cume_dist FROM employees;
+-----+-----------+------+------------------+
| name|       dept|age   |         cume_dist|
+-----+-----------+------+------------------+
| Alex|      Sales|    33|0.3333333333333333|
| Lisa|      Sales|    35|0.6666666666666666|
| Evan|      Sales|    38|               1.0|
| Paul|Engineering|    23|              0.25|
|Chloe|Engineering|    25|              0.75|
| Fred|Engineering|    28|              0.25|
|  Tom|Engineering|    33|               1.0|
| Jane|  Marketing|    28|0.3333333333333333|
| Jeff|  Marketing|    38|0.6666666666666666|
|Helen|  Marketing|    40|               1.0|
+-----+-----------+------+------------------+
SELECT name, dept, salary, MIN(salary) OVER (PARTITION BY dept ORDER BY salary) AS min
FROM employees;
+-----+-----------+------+-----+
| name|       dept|salary|  min|
+-----+-----------+------+-----+
| Lisa|      Sales| 10000|10000|
```

```
| Alex|      Sales| 30000|10000|
| Evan|      Sales| 32000|10000|
|Helen|  Marketing| 29000|29000|
| Jane|  Marketing| 29000|29000|
| Jeff|  Marketing| 35000|29000|
| Fred|Engineering| 21000|21000|
|  Tom|Engineering| 23000|21000|
|Chloe|Engineering| 23000|21000|
| Paul|Engineering| 29000|21000|
+-----+-----------+------+-----+
SELECT name, salary,
LAG(salary) OVER (PARTITION BY dept ORDER BY salary) AS lag,
LEAD(salary, 1, 0) OVER (PARTITION BY dept ORDER BY salary) AS lead
FROM employees;
+-----+-----------+------+-----+-----+
| name|       dept|salary|  lag| lead|
+-----+-----------+------+-----+-----+
| Lisa|      Sales| 10000|NULL |30000|
| Alex|      Sales| 30000|10000|32000|
| Evan|      Sales| 32000|30000|    0|
| Fred|Engineering| 21000| NULL|23000|
|Chloe|Engineering| 23000|21000|23000|
|  Tom|Engineering| 23000|23000|29000|
| Paul|Engineering| 29000|23000|    0|
|Helen|  Marketing| 29000| NULL|29000|
| Jane|  Marketing| 29000|29000|35000|
| Jeff|  Marketing| 35000|29000|    0|
+-----+-----------+------+-----+-----+
SELECT id, v,
LEAD(v, 0) IGNORE NULLS OVER w lead,
LAG(v, 0) IGNORE NULLS OVER w lag,
NTH_VALUE(v, 2) IGNORE NULLS OVER w nth_value,
FIRST_VALUE(v) IGNORE NULLS OVER w first_value,
LAST_VALUE(v) IGNORE NULLS OVER w last_value
FROM test_ignore_null
WINDOW w AS (ORDER BY id)
ORDER BY id;
+--+----+----+----+---------+-----------+----------+
|id|   v|lead| lag|nth_value|first_value|last_value|
+--+----+----+----+---------+-----------+----------+
| 0|NULL|NULL|NULL|     NULL|       NULL|      NULL|
| 1|   x|   x|   x|     NULL|          x|         x|
| 2|NULL|NULL|NULL|     NULL|          x|         x|
| 3|NULL|NULL|NULL|     NULL|          x|         x|
```

```
| 4|   y|   y|   y|        y|          x|          y|
| 5|NULL|NULL|NULL|        y|          x|          y|
| 6|   z|   z|   z|        y|          x|          z|
| 7|   v|   v|   v|        y|          x|          v|
| 8|NULL|NULL|NULL|        y|          x|          v|
+--+----+----+----+---------+-----------+-----------+
```

Conversion functions

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

Function	Description
bigint(expr)	Casts the value `expr` to the target data type `bigint`.
binary(expr)	Casts the value `expr` to the target data type `binary`.
boolean(expr)	Casts the value `expr` to the target data type `boolean`.
cast(expr AS type)	Casts the value `expr` to the target data type `type`.
date(expr)	Casts the value `expr` to the target data type `date`.
decimal(expr)	Casts the value `expr` to the target data type `decimal`.
double(expr)	Casts the value `expr` to the target data type `double`.
float(expr)	Casts the value `expr` to the target data type `float`.
int(expr)	Casts the value `expr` to the target data type `int`.
smallint(expr)	Casts the value `expr` to the target data type `smallint`.
string(expr)	Casts the value `expr` to the target data type `string`.
timestamp(expr)	Casts the value `expr` to the target data type `timestamp`.

Function	Description
tinyint(expr)	Casts the value `expr` to the target data type `tinyint`.

Examples

```
-- cast
SELECT cast(field as int);
+---------------+
|CAST(field AS INT)|
+---------------+
|             10|
+---------------+
```

Predicate functions

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called "Supported SQL commands".

Function	Description
! expr	Logical not.
expr1 < expr2	Returns true if `expr1` is less than `expr2`.
expr1 <= expr2	Returns true if `expr1` is less than or equal to `expr2`.
expr1 <=> expr2	Returns same result as the EQUAL(=) operator for non-null operands, but returns true if both are null, false if one of the them is null.
expr1 = expr2	Returns true if `expr1` equals `expr2`, or false otherwise.
expr1 == expr2	Returns true if `expr1` equals `expr2`, or false otherwise.
expr1 > expr2	Returns true if `expr1` is greater than `expr2`.

Function	Description
expr1 >= expr2	Returns true if `expr1` is greater than or equal to `expr2`.
expr1 and expr2	Logical AND.
str ilike pattern[ESCAPE escape]	Returns true if str matches `pattern` with `escape` case-insensitively, null if any arguments are null, false otherwise.
expr1 in(expr2, expr3, ...)	Returns true if `expr` equals to any valN.
isnan(expr)	Returns true if `expr` is NaN, or false otherwise.
isnotnull(expr)	Returns true if `expr` is not null, or false otherwise.
isnull(expr)	Returns true if `expr` is null, or false otherwise.
str like pattern[ESCAPE escape]	Returns true if str matches `pattern` with `escape`, null if any arguments are null, false otherwise.
not expr	Logical not.
expr1 or expr2	Logical OR.
regexp(str, regexp)	Returns true if `str` matches `regexp`, or false otherwise.
regexp_like(str, regexp)	Returns true if `str` matches `regexp`, or false otherwise.
rlike(str, regexp)	Returns true if `str` matches `regexp`, or false otherwise.

Examples

```
-- !
SELECT ! true;
+----------+
|(NOT true)|
+----------+
|     false|
+----------+
SELECT ! false;
+-----------+
```

```
|(NOT false)|
+-----------+
|       true|
+-----------+
SELECT ! NULL;
+----------+
|(NOT NULL)|
+----------+
|      NULL|
+----------+
-- <
SELECT to_date('2009-07-30 04:17:52') < to_date('2009-07-30 04:17:52');
+-------------------------------------------------------------+
|(to_date(2009-07-30 04:17:52) < to_date(2009-07-30 04:17:52))|
+-------------------------------------------------------------+
|                                                        false|
+-------------------------------------------------------------+
SELECT to_date('2009-07-30 04:17:52') < to_date('2009-08-01 04:17:52');
+-------------------------------------------------------------+
|(to_date(2009-07-30 04:17:52) < to_date(2009-08-01 04:17:52))|
+-------------------------------------------------------------+
|                                                         true|
+-------------------------------------------------------------+
SELECT 1 < NULL;
+----------+
|(1 < NULL)|
+----------+
|      NULL|
+----------+
-- <=
SELECT 2 <= 2;
+--------+
|(2 <= 2)|
+--------+
|    true|
+--------+
SELECT 1.0 <= '1';
+----------+
|(1.0 <= 1)|
+----------+
|      true|
+----------+
SELECT to_date('2009-07-30 04:17:52') <= to_date('2009-07-30 04:17:52');
+-------------------------------------------------------------+
```

```
|(to_date(2009-07-30 04:17:52) <= to_date(2009-07-30 04:17:52))|
+--------------------------------------------------------------+
|                                                          true|
+--------------------------------------------------------------+
SELECT to_date('2009-07-30 04:17:52') <= to_date('2009-08-01 04:17:52');
+--------------------------------------------------------------+
|(to_date(2009-07-30 04:17:52) <= to_date(2009-08-01 04:17:52))|
+--------------------------------------------------------------+
|                                                          true|
+--------------------------------------------------------------+
SELECT 1 <= NULL;
+-----------+
|(1 <= NULL)|
+-----------+
|       NULL|
+-----------+
-- <=>
SELECT 2 <=> 2;
+---------+
|(2 <=> 2)|
+---------+
|     true|
+---------+
SELECT 1 <=> '1';
+---------+
|(1 <=> 1)|
+---------+
|     true|
+---------+
SELECT true <=> NULL;
+---------------+
|(true <=> NULL)|
+---------------+
|          false|
+---------------+
SELECT NULL <=> NULL;
+---------------+
|(NULL <=> NULL)|
+---------------+
|           true|
+---------------+
-- =
SELECT 2 = 2;
+-------+
```

```
|(2 = 2)|
+-------+
|   true|
+-------+
SELECT 1 = '1';
+-------+
|(1 = 1)|
+-------+
|   true|
+-------+
SELECT true = NULL;
+-------------+
|(true = NULL)|
+-------------+
|         NULL|
+-------------+
SELECT NULL = NULL;
+-------------+
|(NULL = NULL)|
+-------------+
|         NULL|
+-------------+
-- ==
SELECT 2 == 2;
+-------+
|(2 = 2)|
+-------+
|   true|
+-------+
SELECT 1 == '1';
+-------+
|(1 = 1)|
+-------+
|   true|
+-------+
SELECT true == NULL;
+-------------+
|(true = NULL)|
+-------------+
|         NULL|
+-------------+
SELECT NULL == NULL;
+-------------+
|(NULL = NULL)|
```

```
+-------------+
|         NULL|
+-------------+
-- >
SELECT 2 > 1;
+-------+
|(2 > 1)|
+-------+
|   true|
+-------+
SELECT 2 > 1.1;
+-------+
|(2 > 1)|
+-------+
|   true|
+-------+
SELECT to_date('2009-07-30 04:17:52') > to_date('2009-07-30 04:17:52');
+-------------------------------------------------------------+
|(to_date(2009-07-30 04:17:52) > to_date(2009-07-30 04:17:52))|
+-------------------------------------------------------------+
|                                                        false|
+-------------------------------------------------------------+
SELECT to_date('2009-07-30 04:17:52') > to_date('2009-08-01 04:17:52');
+-------------------------------------------------------------+
|(to_date(2009-07-30 04:17:52) > to_date(2009-08-01 04:17:52))|
+-------------------------------------------------------------+
|                                                        false|
+-------------------------------------------------------------+
SELECT 1 > NULL;
+----------+
|(1 > NULL)|
+----------+
|      NULL|
+----------+
-- >=
SELECT 2 >= 1;
+--------+
|(2 >= 1)|
+--------+
|    true|
+--------+
SELECT 2.0 >= '2.1';
+------------+
|(2.0 >= 2.1)|
```

```
+------------+
|       false|
+------------+
SELECT to_date('2009-07-30 04:17:52') >= to_date('2009-07-30 04:17:52');
+--------------------------------------------------------------+
|(to_date(2009-07-30 04:17:52) >= to_date(2009-07-30 04:17:52))|
+--------------------------------------------------------------+
|                                                          true|
+--------------------------------------------------------------+
SELECT to_date('2009-07-30 04:17:52') >= to_date('2009-08-01 04:17:52');
+--------------------------------------------------------------+
|(to_date(2009-07-30 04:17:52) >= to_date(2009-08-01 04:17:52))|
+--------------------------------------------------------------+
|                                                         false|
+--------------------------------------------------------------+
SELECT 1 >= NULL;
+-----------+
|(1 >= NULL)|
+-----------+
|       NULL|
+-----------+
-- and
SELECT true and true;
+---------------+
|(true AND true)|
+---------------+
|           true|
+---------------+
SELECT true and false;
+----------------+
|(true AND false)|
+----------------+
|           false|
+----------------+
SELECT true and NULL;
+---------------+
|(true AND NULL)|
+---------------+
|           NULL|
+---------------+
SELECT false and NULL;
+----------------+
|(false AND NULL)|
+----------------+
```

```
|           false|
+----------------+
-- ilike
SELECT ilike('Wagon', '_Agon');
+-------------------+
|ilike(Wagon, _Agon)|
+-------------------+
|               true|
+-------------------+
SELECT '%SystemDrive%\Users\John' ilike '\%SystemDrive\%\\users%';
+-----------------------------------------------------------+
|ilike(%SystemDrive%\Users\John, \%SystemDrive\%\\users%)|
+-----------------------------------------------------------+
|                                                       true|
+-----------------------------------------------------------+
SELECT '%SystemDrive%\\USERS\\John' ilike '\%SystemDrive\%\\\\Users%';
+-----------------------------------------------------------+
|ilike(%SystemDrive%\USERS\John, \%SystemDrive\%\\Users%)|
+-----------------------------------------------------------+
|                                                       true|
+-----------------------------------------------------------+
SELECT '%SystemDrive%/Users/John' ilike '/%SYSTEMDrive/%//Users%' ESCAPE '/';
+-----------------------------------------------------------+
|ilike(%SystemDrive%/Users/John, /%SYSTEMDrive/%//Users%)|
+-----------------------------------------------------------+
|                                                       true|
+-----------------------------------------------------------+
-- in
SELECT 1 in(1, 2, 3);
+----------------+
|(1 IN (1, 2, 3))|
+----------------+
|            true|
+----------------+
SELECT 1 in(2, 3, 4);
+----------------+
|(1 IN (2, 3, 4))|
+----------------+
|           false|
+----------------+
SELECT named_struct('a', 1, 'b', 2) in(named_struct('a', 1, 'b', 1), named_struct('a',
 1, 'b', 3));
+-------------------------------------------------------------------------------------+
|(named_struct(a, 1, b, 2) IN (named_struct(a, 1, b, 1), named_struct(a, 1, b, 3)))|
```

```
+-----------------------------------------------------------------------------------------+
|                                                                                    false|
+-----------------------------------------------------------------------------------------+
SELECT named_struct('a', 1, 'b', 2) in(named_struct('a', 1, 'b', 2), named_struct('a',
 1, 'b', 3));
+-----------------------------------------------------------------------------------------+
|(named_struct(a, 1, b, 2) IN (named_struct(a, 1, b, 2), named_struct(a, 1, b, 3)))|
+-----------------------------------------------------------------------------------------+
|                                                                                     true|
+-----------------------------------------------------------------------------------------+
-- isnan
SELECT isnan(cast('NaN' as double));
+--------------------------+
|isnan(CAST(NaN AS DOUBLE))|
+--------------------------+
|                      true|
+--------------------------+
-- isnotnull
SELECT isnotnull(1);
+---------------+
|(1 IS NOT NULL)|
+---------------+
|           true|
+---------------+
-- isnull
SELECT isnull(1);
+-----------+
|(1 IS NULL)|
+-----------+
|      false|
+-----------+
-- like
SELECT like('Wagon', '_Agon');
+----------------+
|Wagon LIKE _Agon|
+----------------+
|            true|
+----------------+
-- not
SELECT not true;
+----------+
|(NOT true)|
+----------+
|     false|
```

```
+----------+
SELECT not false;
+-----------+
|(NOT false)|
+-----------+
|       true|
+-----------+
SELECT not NULL;
+----------+
|(NOT NULL)|
+----------+
|      NULL|
+----------+
-- or
SELECT true or false;
+---------------+
|(true OR false)|
+---------------+
|           true|
+---------------+
SELECT false or false;
+----------------+
|(false OR false)|
+----------------+
|           false|
+----------------+
SELECT true or NULL;
+--------------+
|(true OR NULL)|
+--------------+
|          true|
+--------------+
SELECT false or NULL;
+---------------+
|(false OR NULL)|
+---------------+
|           NULL|
+---------------+
```

Map functions

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called "Supported SQL commands".

Function	Description
element_at(array, index)	Returns element of array at given (1-based) index.
element_at(map, key)	Returns value for given key. The function returns NULL if the key is not contained in the map.
map(key0, value0, key1, value1, ...)	Creates a map with the given key/value pairs.
map_concat(map, ...)	Returns the union of all the given maps
map_contains_key(map, key)	Returns true if the map contains the key.
map_entries(map)	Returns an unordered array of all entries in the given map.
map_from_arrays(keys, values)	Creates a map with a pair of the given key/value arrays. All elements in keys should not be null
map_from_entries(arrayOfEntries)	Returns a map created from the given array of entries.
map_keys(map)	Returns an unordered array containing the keys of the map.
map_values(map)	Returns an unordered array containing the values of the map.

Function	Description
str_to_map(text[, pairDelim[, keyValueDelim]])	Creates a map after splitting the text into key/value pairs using delimiters. Default delimiters are ',' for `pairDelim` and ':' for `keyValueDelim`. Both `pairDelim` and `keyValueDelim` are treated as regular expressions.
try_element_at(array, index)	Returns element of array at given (1-based) index. If Index is 0, the system will throw an error. If index < 0, accesses elements from the last to the first. The function always returns NULL if the index exceeds the length of the array.
try_element_at(map, key)	Returns value for given key. The function always returns NULL if the key is not contained in the map.

Examples

```
-- element_at
SELECT element_at(array(1, 2, 3), 2);
+----------------------------+
|element_at(array(1, 2, 3), 2)|
+----------------------------+
|                           2|
+----------------------------+
SELECT element_at(map(1, 'a', 2, 'b'), 2);
+----------------------------+
|element_at(map(1, a, 2, b), 2)|
+----------------------------+
|                           b|
+----------------------------+
-- map
SELECT map(1.0, '2', 3.0, '4');
+--------------------+
| map(1.0, 2, 3.0, 4)|
```

```
+--------------------+
|{1.0 -> 2, 3.0 -> 4}|
+--------------------+
-- map_concat
SELECT map_concat(map(1, 'a', 2, 'b'), map(3, 'c'));
+--------------------------------------+
|map_concat(map(1, a, 2, b), map(3, c))|
+--------------------------------------+
|                  {1 -> a, 2 -> b, ...|
+--------------------------------------+
-- map_contains_key
SELECT map_contains_key(map(1, 'a', 2, 'b'), 1);
+-----------------------------------+
|map_contains_key(map(1, a, 2, b), 1)|
+-----------------------------------+
|                               true|
+-----------------------------------+
SELECT map_contains_key(map(1, 'a', 2, 'b'), 3);
+-----------------------------------+
|map_contains_key(map(1, a, 2, b), 3)|
+-----------------------------------+
|                              false|
+-----------------------------------+
-- map_entries
SELECT map_entries(map(1, 'a', 2, 'b'));
+----------------------------+
|map_entries(map(1, a, 2, b))|
+----------------------------+
|            [{1, a}, {2, b}]|
+----------------------------+
-- map_from_arrays
SELECT map_from_arrays(array(1.0, 3.0), array('2', '4'));
+-------------------------------------------+
|map_from_arrays(array(1.0, 3.0), array(2, 4))|
+-------------------------------------------+
|                       {1.0 -> 2, 3.0 -> 4}|
+-------------------------------------------+
-- map_from_entries
SELECT map_from_entries(array(struct(1, 'a'), struct(2, 'b')));
+---------------------------------------------------+
|map_from_entries(array(struct(1, a), struct(2, b)))|
+---------------------------------------------------+
|                                {1 -> a, 2 -> b}|
+---------------------------------------------------+
```

```
-- map_keys
SELECT map_keys(map(1, 'a', 2, 'b'));
+------------------------+
|map_keys(map(1, a, 2, b))|
+------------------------+
|                  [1, 2]|
+------------------------+
-- map_values
SELECT map_values(map(1, 'a', 2, 'b'));
+--------------------------+
|map_values(map(1, a, 2, b))|
+--------------------------+
|                    [a, b]|
+--------------------------+
-- str_to_map
SELECT str_to_map('a:1,b:2,c:3', ',', ':');
+----------------------------+
|str_to_map(a:1,b:2,c:3, ,, :)|
+----------------------------+
|         {a -> 1, b -> 2, ...|
+----------------------------+
SELECT str_to_map('a');
+------------------+
|str_to_map(a, ,, :)|
+------------------+
|        {a -> NULL}|
+------------------+
-- try_element_at
SELECT try_element_at(array(1, 2, 3), 2);
+---------------------------------+
|try_element_at(array(1, 2, 3), 2)|
+---------------------------------+
|                                2|
+---------------------------------+
SELECT try_element_at(map(1, 'a', 2, 'b'), 2);
+---------------------------------+
|try_element_at(map(1, a, 2, b), 2)|
+---------------------------------+
|                                 b|
+---------------------------------+
```

Mathematical functions

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

Function	Description
expr1 % expr2	Returns the remainder after `expr1`/`expr2`.
expr1 * expr2	Returns `expr1`*`expr2`.
expr1 + expr2	Returns `expr1`+`expr2`.
expr1 - expr2	Returns `expr1`-`expr2`.
expr1 / expr2	Returns `expr1`/`expr2`. It always performs floating point division.
abs(expr)	Returns the absolute value of the numeric or interval value.
acos(expr)	Returns the inverse cosine (a.k.a. arc cosine) of `expr`, as if computed by `java.lang.Math.acos`.
acosh(expr)	Returns inverse hyperbolic cosine of `expr`.
asin(expr)	Returns the inverse sine (a.k.a. arc sine) the arc sin of `expr`, as if computed by `java.lang.Math.asin`.
asinh(expr)	Returns inverse hyperbolic sine of `expr`.

Function	Description
atan(expr)	Returns the inverse tangent (a.k.a. arc tangent) of `expr`, as if computed by `java.lang.Math.atan`
atan2(exprY, exprX)	Returns the angle in radians between the positive x-axis of a plane and the point given by the coordinates (`exprX`, `exprY`), as if computed by `java.lang.Math.atan2`.
atanh(expr)	Returns inverse hyperbolic tangent of `expr`.
bin(expr)	Returns the string representation of the long value `expr` represented in binary.
bround(expr, d)	Returns `expr` rounded to `d` decimal places using HALF_EVEN rounding mode.
cbrt(expr)	Returns the cube root of `expr`.
ceil(expr[, scale])	Returns the smallest number after rounding up that is not smaller than `expr`. An optional `scale` parameter can be specified to control the rounding behavior.
ceiling(expr[, scale])	Returns the smallest number after rounding up that is not smaller than `expr`. An optional `scale` parameter can be specified to control the rounding behavior.
conv(num, from_base, to_base)	Convert `num` from `from_base` to `to_base`.
cos(expr)	Returns the cosine of `expr`, as if computed by `java.lang.Math.cos`.
cosh(expr)	Returns the hyperbolic cosine of `expr`, as if computed by `java.lang.Math.cosh`.

Function	Description
cot(expr)	Returns the cotangent of `expr`, as if computed by `1/java.lang.Math.tan`.
csc(expr)	Returns the cosecant of `expr`, as if computed by `1/java.lang.Math.sin`.
degrees(expr)	Converts radians to degrees.
expr1 div expr2	Divide `expr1` by `expr2`. It returns NULL if an operand is NULL or `expr2` is 0. The result is casted to long.
e()	Returns Euler's number, e.
exp(expr)	Returns e to the power of `expr`.
expm1(expr) - Returns exp(`expr`)	1
factorial(expr)	Returns the factorial of `expr`. `expr` is [0..20]. Otherwise, null.
floor(expr[, scale])	Returns the largest number after rounding down that is not greater than `expr`. An optional `scale` parameter can be specified to control the rounding behavior.
greatest(expr, ...)	Returns the greatest value of all parameters, skipping null values.
hex(expr)	Converts `expr` to hexadecimal.
hypot(expr1, expr2)	Returns sqrt(`expr1`**2 + `expr2`**2).
least(expr, ...)	Returns the least value of all parameters, skipping null values.
ln(expr)	Returns the natural logarithm (base e) of `expr`.

Function	Description
log(base, expr)	Returns the logarithm of `expr` with `base`.
log10(expr)	Returns the logarithm of `expr` with base 10.
log1p(expr)	Returns log(1 + `expr`).
log2(expr)	Returns the logarithm of `expr` with base 2.
expr1 mod expr2	Returns the remainder after `expr1`/`expr2`.
negative(expr)	Returns the negated value of `expr`.
pi()	Returns pi.
pmod(expr1, expr2)	Returns the positive value of `expr1` mod `expr2`.
positive(expr)	Returns the value of `expr`.
pow(expr1, expr2)	Raises `expr1` to the power of `expr2`.
power(expr1, expr2)	Raises `expr1` to the power of `expr2`.
radians(expr)	Converts degrees to radians.
rand([seed])	Returns a random value with independent and identically distributed (i.i.d.) uniformly distributed values in [0, 1).
randn([seed])	Returns a random value with independent and identically distributed (i.i.d.) values drawn from the standard normal distribution.
random([seed])	Returns a random value with independent and identically distributed (i.i.d.) uniformly distributed values in [0, 1).

Function	Description
rint(expr)	Returns the double value that is closest in value to the argument and is equal to a mathematical integer.
round(expr, d)	Returns `expr` rounded to `d` decimal places using HALF_UP rounding mode.
sec(expr)	Returns the secant of `expr`, as if computed by `1/java.lang.Math.cos`.
shiftleft(base, expr)	Bitwise left shift.
sign(expr)	Returns -1.0, 0.0 or 1.0 as `expr` is negative, 0 or positive.
signum(expr)	Returns -1.0, 0.0 or 1.0 as `expr` is negative, 0 or positive.
sin(expr)	Returns the sine of `expr`, as if computed by `java.lang.Math.sin`.
sinh(expr)	Returns hyperbolic sine of `expr`, as if computed by `java.lang.Math.sinh`.
sqrt(expr)	Returns the square root of `expr`.
tan(expr)	Returns the tangent of `expr`, as if computed by `java.lang.Math.tan`.
tanh(expr)	Returns the hyperbolic tangent of `expr`, as if computed by `java.lang.Math.tanh`.
try_add(expr1, expr2)	Returns the sum of `expr1`and `expr2` and the result is null on overflow. The acceptabl e input types are the same with the `+` operator.

Function	Description
try_divide(dividend, divisor)	Returns `dividend`/`divisor`. It always performs floating point division. Its result is always null if `expr2` is 0. `dividend` must be a numeric or an interval. `divisor` must be a numeric.
try_multiply(expr1, expr2)	Returns `expr1`*`expr2` and the result is null on overflow. The acceptable input types are the same with the `*` operator.
try_subtract(expr1, expr2)	Returns `expr1`-`expr2` and the result is null on overflow. The acceptable input types are the same with the `-` operator.
unhex(expr)	Converts hexadecimal `expr` to binary.
width_bucket(value, min_value, max_value, num_bucket)	Returns the bucket number to which `value` would be assigned in an equiwidth histogram with `num_bucket` buckets, in the range `min_value` to `max_value`."

Examples

```
-- %
SELECT 2 % 1.8;
+---------+
|(2 % 1.8)|
+---------+
|      0.2|
+---------+
SELECT MOD(2, 1.8);
+-----------+
|mod(2, 1.8)|
+-----------+
|        0.2|
+-----------+
-- *
SELECT 2 * 3;
```

```
+-------+
|(2 * 3)|
+-------+
|      6|
+-------+
-- +
SELECT 1 + 2;
+-------+
|(1 + 2)|
+-------+
|      3|
+-------+
-- -
SELECT 2 - 1;
+-------+
|(2 - 1)|
+-------+
|      1|
+-------+
-- /
SELECT 3 / 2;
+-------+
|(3 / 2)|
+-------+
|    1.5|
+-------+
SELECT 2L / 2L;
+-------+
|(2 / 2)|
+-------+
|    1.0|
+-------+
-- abs
SELECT abs(-1);
+-------+
|abs(-1)|
+-------+
|      1|
+-------+
SELECT abs(INTERVAL -'1-1' YEAR TO MONTH);
+----------------------------------+
|abs(INTERVAL '-1-1' YEAR TO MONTH)|
+----------------------------------+
|              INTERVAL '1-1' YE...|
```

```
+----------------------------------+
-- acos
SELECT acos(1);
+-------+
|ACOS(1)|
+-------+
|    0.0|
+-------+
SELECT acos(2);
+-------+
|ACOS(2)|
+-------+
|    NaN|
+-------+
-- acosh
SELECT acosh(1);
+--------+
|ACOSH(1)|
+--------+
|     0.0|
+--------+
SELECT acosh(0);
+--------+
|ACOSH(0)|
+--------+
|     NaN|
+--------+
-- asin
SELECT asin(0);
+-------+
|ASIN(0)|
+-------+
|    0.0|
+-------+
SELECT asin(2);
+-------+
|ASIN(2)|
+-------+
|    NaN|
+-------+
-- asinh
SELECT asinh(0);
+--------+
|ASINH(0)|
```

```
+--------+
|     0.0|
+--------+
-- atan
SELECT atan(0);
+-------+
|ATAN(0)|
+-------+
|    0.0|
+-------+
-- atan2
SELECT atan2(0, 0);
+-----------+
|ATAN2(0, 0)|
+-----------+
|        0.0|
+-----------+
-- atanh
SELECT atanh(0);
+--------+
|ATANH(0)|
+--------+
|     0.0|
+--------+
SELECT atanh(2);
+--------+
|ATANH(2)|
+--------+
|     NaN|
+--------+
-- bin
SELECT bin(13);
+-------+
|bin(13)|
+-------+
|   1101|
+-------+
SELECT bin(-13);
+--------------------+
|            bin(-13)|
+--------------------+
|1111111111111111...|
+--------------------+
SELECT bin(13.3);
```

```
+---------+
|bin(13.3)|
+---------+
|     1101|
+---------+
-- bround
SELECT bround(2.5, 0);
+--------------+
|bround(2.5, 0)|
+--------------+
|             2|
+--------------+
SELECT bround(25, -1);
+--------------+
|bround(25, -1)|
+--------------+
|            20|
+--------------+
-- cbrt
SELECT cbrt(27.0);
+----------+
|CBRT(27.0)|
+----------+
|       3.0|
+----------+
-- ceil
SELECT ceil(-0.1);
+----------+
|CEIL(-0.1)|
+----------+
|         0|
+----------+
SELECT ceil(5);
+-------+
|CEIL(5)|
+-------+
|      5|
+-------+
SELECT ceil(3.1411, 3);
+---------------+
|ceil(3.1411, 3)|
+---------------+
|          3.142|
+---------------+
```

```
SELECT ceil(3.1411, -3);
+----------------+
|ceil(3.1411, -3)|
+----------------+
|            1000|
+----------------+
-- ceiling
SELECT ceiling(-0.1);
+-------------+
|ceiling(-0.1)|
+-------------+
|            0|
+-------------+
SELECT ceiling(5);
+----------+
|ceiling(5)|
+----------+
|         5|
+----------+
SELECT ceiling(3.1411, 3);
+------------------+
|ceiling(3.1411, 3)|
+------------------+
|             3.142|
+------------------+
SELECT ceiling(3.1411, -3);
+-------------------+
|ceiling(3.1411, -3)|
+-------------------+
|               1000|
+-------------------+
-- conv
SELECT conv('100', 2, 10);
+----------------+
|conv(100, 2, 10)|
+----------------+
|               4|
+----------------+
SELECT conv(-10, 16, -10);
+------------------+
|conv(-10, 16, -10)|
+------------------+
|               -16|
+------------------+
```

```
-- cos
SELECT cos(0);
+------+
|COS(0)|
+------+
|   1.0|
+------+
-- cosh
SELECT cosh(0);
+-------+
|COSH(0)|
+-------+
|    1.0|
+-------+
-- cot
SELECT cot(1);
+------------------+
|            COT(1)|
+------------------+
|0.6420926159343306|
+------------------+
-- csc
SELECT csc(1);
+------------------+
|            CSC(1)|
+------------------+
|1.1883951057781212|
+------------------+
-- degrees
SELECT degrees(3.141592653589793);
+--------------------------+
|DEGREES(3.141592653589793)|
+--------------------------+
|                     180.0|
+--------------------------+
-- div
SELECT 3 div 2;
+---------+
|(3 div 2)|
+---------+
|        1|
+---------+
SELECT INTERVAL '1-1' YEAR TO MONTH div INTERVAL '-1' MONTH;
+-----------------------------------------------------+
```

```
|(INTERVAL '1-1' YEAR TO MONTH div INTERVAL '-1' MONTH)|
+------------------------------------------------------+
|                                                   -13|
+------------------------------------------------------+
-- e
SELECT e();
+-----------------+
|             E()|
+-----------------+
|2.718281828459045|
+-----------------+
-- exp
SELECT exp(0);
+------+
|EXP(0)|
+------+
|   1.0|
+------+
-- expm1
SELECT expm1(0);
+--------+
|EXPM1(0)|
+--------+
|     0.0|
+--------+
-- factorial
SELECT factorial(5);
+------------+
|factorial(5)|
+------------+
|         120|
+------------+
-- floor
SELECT floor(-0.1);
+-----------+
|FLOOR(-0.1)|
+-----------+
|         -1|
+-----------+
SELECT floor(5);
+--------+
|FLOOR(5)|
+--------+
|       5|
```

```
+--------+
SELECT floor(3.1411, 3);
+----------------+
|floor(3.1411, 3)|
+----------------+
|           3.141|
+----------------+
SELECT floor(3.1411, -3);
+-----------------+
|floor(3.1411, -3)|
+-----------------+
|                0|
+-----------------+
-- greatest
SELECT greatest(10, 9, 2, 4, 3);
+------------------------+
|greatest(10, 9, 2, 4, 3)|
+------------------------+
|                      10|
+------------------------+
-- hex
SELECT hex(17);
+-------+
|hex(17)|
+-------+
|     11|
+-------+
SELECT hex('SQL');
+------------------+
|    hex(SQL)|
+------------------+
|53514C|
+------------------+
-- hypot
SELECT hypot(3, 4);
+-----------+
|HYPOT(3, 4)|
+-----------+
|        5.0|
+-----------+
-- least
SELECT least(10, 9, 2, 4, 3);
+---------------------+
|least(10, 9, 2, 4, 3)|
```

```
+---------------------+
|                    2|
+---------------------+
-- ln
SELECT ln(1);
+-----+
|ln(1)|
+-----+
|  0.0|
+-----+
-- log
SELECT log(10, 100);
+------------+
|LOG(10, 100)|
+------------+
|         2.0|
+------------+
-- log10
SELECT log10(10);
+---------+
|LOG10(10)|
+---------+
|      1.0|
+---------+
-- log1p
SELECT log1p(0);
+--------+
|LOG1P(0)|
+--------+
|     0.0|
+--------+
-- log2
SELECT log2(2);
+-------+
|LOG2(2)|
+-------+
|    1.0|
+-------+
-- mod
SELECT 2 % 1.8;
+---------+
|(2 % 1.8)|
+---------+
|      0.2|
```

```
+---------+
SELECT MOD(2, 1.8);
+-----------+
|mod(2, 1.8)|
+-----------+
|        0.2|
+-----------+
-- negative
SELECT negative(1);
+-----------+
|negative(1)|
+-----------+
|         -1|
+-----------+
-- pi
SELECT pi();
+-----------------+
|             PI()|
+-----------------+
|3.141592653589793|
+-----------------+
-- pmod
SELECT pmod(10, 3);
+-----------+
|pmod(10, 3)|
+-----------+
|          1|
+-----------+
SELECT pmod(-10, 3);
+------------+
|pmod(-10, 3)|
+------------+
|           2|
+------------+
-- positive
SELECT positive(1);
+-----+
|(+ 1)|
+-----+
|    1|
+-----+
-- pow
SELECT pow(2, 3);
+---------+
```

```
|pow(2, 3)|
+---------+
|      8.0|
+---------+
-- power
SELECT power(2, 3);
+-----------+
|POWER(2, 3)|
+-----------+
|        8.0|
+-----------+
-- radians
SELECT radians(180);
+-----------------+
|     RADIANS(180)|
+-----------------+
|3.141592653589793|
+-----------------+
-- rand
SELECT rand();
+------------------+
|            rand()|
+------------------+
|0.7211420708112387|
+------------------+
SELECT rand(0);
+------------------+
|           rand(0)|
+------------------+
|0.7604953758285915|
+------------------+
SELECT rand(null);
+------------------+
|        rand(NULL)|
+------------------+
|0.7604953758285915|
+------------------+
-- randn
SELECT randn();
+-------------------+
|            randn()|
+-------------------+
|-0.8175603217732732|
+-------------------+
```

```
SELECT randn(0);
+------------------+
|          randn(0)|
+------------------+
|1.6034991609278433|
+------------------+
SELECT randn(null);
+------------------+
|       randn(NULL)|
+------------------+
|1.6034991609278433|
+------------------+
-- random
SELECT random();
+-----------------+
|           rand()|
+-----------------+
|0.394205008255365|
+-----------------+
SELECT random(0);
+------------------+
|           rand(0)|
+------------------+
|0.7604953758285915|
+------------------+
SELECT random(null);
+------------------+
|        rand(NULL)|
+------------------+
|0.7604953758285915|
+------------------+
-- rint
SELECT rint(12.3456);
+-------------+
|rint(12.3456)|
+-------------+
|         12.0|
+-------------+
-- round
SELECT round(2.5, 0);
+-------------+
|round(2.5, 0)|
+-------------+
|            3|
```

```
+-------------+
-- sec
SELECT sec(0);
+------+
|SEC(0)|
+------+
|   1.0|
+------+
-- shiftleft
SELECT shiftleft(2, 1);
+---------------+
|shiftleft(2, 1)|
+---------------+
|              4|
+---------------+
-- sign
SELECT sign(40);
+--------+
|sign(40)|
+--------+
|     1.0|
+--------+
SELECT sign(INTERVAL -'100' YEAR);
+--------------------------+
|sign(INTERVAL '-100' YEAR)|
+--------------------------+
|                      -1.0|
+--------------------------+
-- signum
SELECT signum(40);
+----------+
|SIGNUM(40)|
+----------+
|       1.0|
+----------+
SELECT signum(INTERVAL -'100' YEAR);
+----------------------------+
|SIGNUM(INTERVAL '-100' YEAR)|
+----------------------------+
|                        -1.0|
+----------------------------+
-- sin
SELECT sin(0);
+------+
```

```
|SIN(0)|
+------+
|   0.0|
+------+
-- sinh
SELECT sinh(0);
+-------+
|SINH(0)|
+-------+
|    0.0|
+-------+
-- sqrt
SELECT sqrt(4);
+-------+
|SQRT(4)|
+-------+
|    2.0|
+-------+
-- tan
SELECT tan(0);
+------+
|TAN(0)|
+------+
|   0.0|
+------+
-- tanh
SELECT tanh(0);
+-------+
|TANH(0)|
+-------+
|    0.0|
+-------+
-- try_add
SELECT try_add(1, 2);
+-------------+
|try_add(1, 2)|
+-------------+
|            3|
+-------------+
SELECT try_add(2147483647, 1);
+----------------------+
|try_add(2147483647, 1)|
+----------------------+
|                  NULL|
```

```
+----------------------+
SELECT try_add(date'2021-01-01', 1);
+-----------------------------+
|try_add(DATE '2021-01-01', 1)|
+-----------------------------+
|                   2021-01-02|
+-----------------------------+
SELECT try_add(date'2021-01-01', interval 1 year);
+-------------------------------------------+
|try_add(DATE '2021-01-01', INTERVAL '1' YEAR)|
+-------------------------------------------+
|                                 2022-01-01|
+-------------------------------------------+
SELECT try_add(timestamp'2021-01-01 00:00:00', interval 1 day);
+-----------------------------------------------------------+
|try_add(TIMESTAMP '2021-01-01 00:00:00', INTERVAL '1' DAY)|
+-----------------------------------------------------------+
|                                    2021-01-02 00:00:00|
+-----------------------------------------------------------+
SELECT try_add(interval 1 year, interval 2 year);
+-------------------------------------------+
|try_add(INTERVAL '1' YEAR, INTERVAL '2' YEAR)|
+-------------------------------------------+
|                           INTERVAL '3' YEAR|
+-------------------------------------------+
-- try_divide
SELECT try_divide(3, 2);
+----------------+
|try_divide(3, 2)|
+----------------+
|             1.5|
+----------------+
SELECT try_divide(2L, 2L);
+----------------+
|try_divide(2, 2)|
+----------------+
|             1.0|
+----------------+
SELECT try_divide(1, 0);
+----------------+
|try_divide(1, 0)|
+----------------+
|            NULL|
+----------------+
```

```
SELECT try_divide(interval 2 month, 2);
+--------------------------------+
|try_divide(INTERVAL '2' MONTH, 2)|
+--------------------------------+
|             INTERVAL '0-1' YE...|
+--------------------------------+
SELECT try_divide(interval 2 month, 0);
+--------------------------------+
|try_divide(INTERVAL '2' MONTH, 0)|
+--------------------------------+
|                            NULL|
+--------------------------------+
-- try_multiply
SELECT try_multiply(2, 3);
+------------------+
|try_multiply(2, 3)|
+------------------+
|                 6|
+------------------+
SELECT try_multiply(-2147483648, 10);
+-----------------------------+
|try_multiply(-2147483648, 10)|
+-----------------------------+
|                         NULL|
+-----------------------------+
SELECT try_multiply(interval 2 year, 3);
+---------------------------------+
|try_multiply(INTERVAL '2' YEAR, 3)|
+---------------------------------+
|              INTERVAL '6-0' YE...|
+---------------------------------+
-- try_subtract
SELECT try_subtract(2, 1);
+------------------+
|try_subtract(2, 1)|
+------------------+
|                 1|
+------------------+
SELECT try_subtract(-2147483648, 1);
+-----------------------------+
|try_subtract(-2147483648, 1)|
+-----------------------------+
|                         NULL|
+-----------------------------+
```

```
SELECT try_subtract(date'2021-01-02', 1);
+---------------------------------+
|try_subtract(DATE '2021-01-02', 1)|
+---------------------------------+
|                       2021-01-01|
+---------------------------------+
SELECT try_subtract(date'2021-01-01', interval 1 year);
+-------------------------------------------------+
|try_subtract(DATE '2021-01-01', INTERVAL '1' YEAR)|
+-------------------------------------------------+
|                                       2020-01-01|
+-------------------------------------------------+
SELECT try_subtract(timestamp'2021-01-02 00:00:00', interval 1 day);
+-----------------------------------------------------------------+
|try_subtract(TIMESTAMP '2021-01-02 00:00:00', INTERVAL '1' DAY)|
+-----------------------------------------------------------------+
|                                             2021-01-01 00:00:00|
+-----------------------------------------------------------------+
SELECT try_subtract(interval 2 year, interval 1 year);
+-------------------------------------------------+
|try_subtract(INTERVAL '2' YEAR, INTERVAL '1' YEAR)|
+-------------------------------------------------+
|                                 INTERVAL '1' YEAR|
+-------------------------------------------------+
-- unhex
SELECT decode(unhex('53514C'), 'UTF-8');
+---------------------------------------+
|decode(unhex(53514C), UTF-8)|
+---------------------------------------+
|                                  SQL|
+---------------------------------------+
-- width_bucket
SELECT width_bucket(5.3, 0.2, 10.6, 5);
+-------------------------------+
|width_bucket(5.3, 0.2, 10.6, 5)|
+-------------------------------+
|                              3|
+-------------------------------+
SELECT width_bucket(-2.1, 1.3, 3.4, 3);
+-------------------------------+
|width_bucket(-2.1, 1.3, 3.4, 3)|
+-------------------------------+
|                              0|
+-------------------------------+
```

```
SELECT width_bucket(8.1, 0.0, 5.7, 4);
+------------------------------+
|width_bucket(8.1, 0.0, 5.7, 4)|
+------------------------------+
|                             5|
+------------------------------+
SELECT width_bucket(-0.9, 5.2, 0.5, 2);
+-------------------------------+
|width_bucket(-0.9, 5.2, 0.5, 2)|
+-------------------------------+
|                              3|
+-------------------------------+
SELECT width_bucket(INTERVAL '0' YEAR, INTERVAL '0' YEAR, INTERVAL '10' YEAR, 10);
+-------------------------------------------------------------------------------+
|width_bucket(INTERVAL '0' YEAR, INTERVAL '0' YEAR, INTERVAL '10' YEAR, 10)|
+-------------------------------------------------------------------------------+
|                                                                              1|
+-------------------------------------------------------------------------------+
SELECT width_bucket(INTERVAL '1' YEAR, INTERVAL '0' YEAR, INTERVAL '10' YEAR, 10);
+-------------------------------------------------------------------------------+
|width_bucket(INTERVAL '1' YEAR, INTERVAL '0' YEAR, INTERVAL '10' YEAR, 10)|
+-------------------------------------------------------------------------------+
|                                                                              2|
+-------------------------------------------------------------------------------+
SELECT width_bucket(INTERVAL '0' DAY, INTERVAL '0' DAY, INTERVAL '10' DAY, 10);
+-----------------------------------------------------------------------+
|width_bucket(INTERVAL '0' DAY, INTERVAL '0' DAY, INTERVAL '10' DAY, 10)|
+-----------------------------------------------------------------------+
|                                                                      1|
+-----------------------------------------------------------------------+
SELECT width_bucket(INTERVAL '1' DAY, INTERVAL '0' DAY, INTERVAL '10' DAY, 10);
+-----------------------------------------------------------------------+
|width_bucket(INTERVAL '1' DAY, INTERVAL '0' DAY, INTERVAL '10' DAY, 10)|
+-----------------------------------------------------------------------+
|                                                                      2|
+-----------------------------------------------------------------------+
```

Generator functions

> **Note**
>
> To see which AWS data source integrations support these SQL functions, see the section called "Supported SQL commands".

Function	Description
explode(expr)	Separates the elements of array `expr` into multiple rows, or the elements of map `expr` into multiple rows and columns. Unless specified otherwise, uses the default column name `col` for elements of the array or `key` and `value` for the elements of the map.
explode_outer(expr)	Separates the elements of array `expr` into multiple rows, or the elements of map `expr` into multiple rows and columns. Unless specified otherwise, uses the default column name `col` for elements of the array or `key` and `value` for the elements of the map.
inline(expr)	Explodes an array of structs into a table. Uses column names col1, col2, etc. by default unless specified otherwise.
inline_outer(expr)	Explodes an array of structs into a table. Uses column names col1, col2, etc. by default unless specified otherwise.
posexplode(expr)	Separates the elements of array `expr` into multiple rows with positions, or the elements of map `expr` into multiple rows and columns with positions. Unless specified otherwise, uses the column name `pos` for position, `col` for elements of the array or `key` and `value` for elements of the map.
posexplode_outer(expr)	Separates the elements of array `expr` into multiple rows with positions, or the elements of map `expr` into multiple rows and columns with positions. Unless specified otherwise, uses the column name `pos` for position, `col` for elements of the array or `key` and `value` for elements of the map.

Function	Description
stack(n, expr1, ..., exprk)	Separates `expr1`, ..., `exprk` into `n` rows. Uses column names col0, col1, etc. by default unless specified otherwise.

Examples

```
-- explode
SELECT explode(array(10, 20));
+---+
|col|
+---+
| 10|
| 20|
+---+

SELECT explode(collection => array(10, 20));
+---+
|col|
+---+
| 10|
| 20|
+---+

SELECT * FROM explode(collection => array(10, 20));
+---+
|col|
+---+
| 10|
| 20|
+---+

-- explode_outer
SELECT explode_outer(array(10, 20));
+---+
|col|
+---+
| 10|
| 20|
+---+

SELECT explode_outer(collection => array(10, 20));
```

```
+---+
|col|
+---+
| 10|
| 20|
+---+

SELECT * FROM explode_outer(collection => array(10, 20));
+---+
|col|
+---+
| 10|
| 20|
+---+

-- inline
SELECT inline(array(struct(1, 'a'), struct(2, 'b')));
+----+----+
|col1|col2|
+----+----+
|   1|   a|
|   2|   b|
+----+----+

-- inline_outer
SELECT inline_outer(array(struct(1, 'a'), struct(2, 'b')));
+----+----+
|col1|col2|
+----+----+
|   1|   a|
|   2|   b|
+----+----+

-- posexplode
SELECT posexplode(array(10,20));
+---+---+
|pos|col|
+---+---+
|  0| 10|
|  1| 20|
+---+---+

SELECT * FROM posexplode(array(10,20));
+---+---+
```

```
|pos|col|
+---+---+
|  0| 10|
|  1| 20|
+---+---+

-- posexplode_outer
SELECT posexplode_outer(array(10,20));
+---+---+
|pos|col|
+---+---+
|  0| 10|
|  1| 20|
+---+---+

SELECT * FROM posexplode_outer(array(10,20));
+---+---+
|pos|col|
+---+---+
|  0| 10|
|  1| 20|
+---+---+

-- stack
SELECT stack(2, 1, 2, 3);
+----+----+
|col0|col1|
+----+----+
|   1|   2|
|   3|NULL|
+----+----+
```

SELECT clause

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

OpenSearch SQL supports a SELECT statement used for retrieving result sets from one or more tables. The following section describes the overall query syntax and the different constructs of a query.

Syntax

```
select_statement
[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select_statement, ... ]
[ ORDER BY
    { expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ]
    [ , ... ]
    }
]
[ SORT BY
    { expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ]
    [ , ... ]
    }
]
[ WINDOW { named_window [ , WINDOW named_window, ... ] } ]
[ LIMIT { ALL | expression } ]
```

While `select_statement` is defined as:

```
SELECT [ ALL | DISTINCT ] { [ [ named_expression ] [ , ... ] ] }
FROM { from_item [ , ... ] }
[ PIVOT clause ]
[ UNPIVOT clause ]
[ LATERAL VIEW clause ] [ ... ]
[ WHERE boolean_expression ]
[ GROUP BY expression [ , ... ] ]
[ HAVING boolean_expression ]
```

Parameters

- **ALL**

  Selects all matching rows from the relation and is enabled by default.

- **DISTINCT**

  Selects all matching rows from the relation after removing duplicates in results.

- **named_expression**

An expression with an assigned name. In general, it denotes a column expression.

Syntax: `expression [[AS] alias]`

- **from_item**

  Table relation

  Join relation

  Pivot relation

  Unpivot relation

  Table-value function

  Inline table

  `[ LATERAL ] ( Subquery )`

- **PIVOT**

  The `PIVOT` clause is used for data perspective. You can get the aggregated values based on specific column value.

- **UNPIVOT**

  The `UNPIVOT` clause transforms columns into rows. It is the reverse of `PIVOT`, except for aggregation of values.

- **LATERAL VIEW**

  The `LATERAL VIEW` clause is used in conjunction with generator functions such as `EXPLODE`, which will generate a virtual table containing one or more rows.

  `LATERAL VIEW` will apply the rows to each original output row.

- **WHERE**

  Filters the result of the `FROM` clause based on the supplied predicates.

- **GROUP BY**

  Specifies the expressions that are used to group the rows.

This is used in conjunction with aggregate functions (`MIN`, `MAX`, `COUNT`, `SUM`, `AVG`, and so on) to group rows based on the grouping expressions and aggregate values in each group.

When a `FILTER` clause is attached to an aggregate function, only the matching rows are passed to that function.

- **HAVING**

  Specifies the predicates by which the rows produced by `GROUP BY` are filtered.

  The `HAVING` clause is used to filter rows after the grouping is performed.

  If `HAVING` is specified without `GROUP BY`, it indicates a `GROUP BY` without grouping expressions (global aggregate).

- **ORDER BY**

  Specifies an ordering of the rows of the complete result set of the query.

  The output rows are ordered across the partitions.

  This parameter is mutually exclusive with `SORT BY` and `DISTRIBUTE BY` and can not be specified together.

- **SORT BY**

  Specifies an ordering by which the rows are ordered within each partition.

  This parameter is mutually exclusive with `ORDER BY` and can not be specified together.

- **LIMIT**

  Specifies the maximum number of rows that can be returned by a statement or subquery.

  This clause is mostly used in the conjunction with `ORDER BY` to produce a deterministic result.

- **boolean_expression**

  Specifies any expression that evaluates to a result type boolean.

  Two or more expressions may be combined together using the logical operators (`AND`, `OR`).

- **expression**

Specifies a combination of one or more values, operators, and SQL functions that evaluates to a value.

- **named_window**

  Specifies aliases for one or more source window specifications.

  The source window specifications can be referenced in the widow definitions in the query.

WHERE clause

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

The `WHERE` clause is used to limit the results of the `FROM` clause of a query or a subquery based on the specified condition.

Syntax

```
WHERE boolean_expression
```

Parameters

- **boolean_expression**

  Specifies any expression that evaluates to a result type boolean.

  Two or more expressions may be combined together using the logical operators (`AND`, `OR`).

Examples

```
CREATE TABLE person (id INT, name STRING, age INT);
INSERT INTO person VALUES
(100, 'John', 30),
(200, 'Mary', NULL),
(300, 'Mike', 80),
(400, 'Dan',  50);
```

```
-- Comparison operator in `WHERE` clause.
SELECT * FROM person WHERE id > 200 ORDER BY id;
+---+----+---+
| id|name|age|
+---+----+---+
|300|Mike| 80|
|400| Dan| 50|
+---+----+---+

-- Comparison and logical operators in `WHERE` clause.
SELECT * FROM person WHERE id = 200 OR id = 300 ORDER BY id;
+---+----+----+
| id|name| age|
+---+----+----+
|200|Mary|null|
|300|Mike|  80|
+---+----+----+

-- IS NULL expression in `WHERE` clause.
SELECT * FROM person WHERE id > 300 OR age IS NULL ORDER BY id;
+---+----+----+
| id|name| age|
+---+----+----+
|200|Mary|null|
|400| Dan|  50|
+---+----+----+

-- Function expression in `WHERE` clause.
SELECT * FROM person WHERE length(name) > 3 ORDER BY id;
+---+----+----+
| id|name| age|
+---+----+----+
|100|John|  30|
|200|Mary|null|
|300|Mike|  80|
+---+----+----+

-- `BETWEEN` expression in `WHERE` clause.
SELECT * FROM person WHERE id BETWEEN 200 AND 300 ORDER BY id;
+---+----+----+
| id|name| age|
+---+----+----+
|200|Mary|null|
```

```
|300|Mike|  80|
+---+----+----+

-- Scalar Subquery in `WHERE` clause.
SELECT * FROM person WHERE age > (SELECT avg(age) FROM person);
+---+----+---+
| id|name|age|
+---+----+---+
|300|Mike| 80|
+---+----+---+

-- Correlated Subquery in `WHERE` clause.
SELECT id FROM person
WHERE exists (SELECT id FROM person where id = 200);
+---+----+----+
|id |name|age |
+---+----+----+
|200|Mary|null|
+---+----+----+
```

GROUP BY clause

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called "Supported SQL commands".

The `GROUP BY` clause is used to group the rows based on a set of specified grouping expressions and compute aggregations on the group of rows based on one or more specified aggregate functions.

The system also does multiple aggregations for the same input record set via `GROUPING SETS`, `CUBE`, `ROLLUP` clauses. The grouping expressions and advanced aggregations can be mixed in the `GROUP BY` clause and nested in a `GROUPING SETS` clause. See more details in the `Mixed/Nested Grouping Analytics` section.

When a `FILTER` clause is attached to an aggregate function, only the matching rows are passed to that function.

Syntax

```
GROUP BY group_expression [ , group_expression [ , ... ] ] [ WITH { ROLLUP | CUBE } ]
GROUP BY { group_expression | { ROLLUP | CUBE | GROUPING SETS } (grouping_set
 [ , ...]) } [ , ... ]
```

While aggregate functions are defined as:

```
aggregate_name ( [ DISTINCT ] expression [ , ... ] ) [ FILTER ( WHERE
 boolean_expression ) ]
```

Parameters

- **group_expression**

  Specifies the criteria based on which the rows are grouped together. The grouping of rows is performed based on result values of the grouping expressions.

  A grouping expression may be a column name like `GROUP BY a`, a column position like `GROUP BY 0`, or an expression like `GROUP BY a + b`.
- **grouping_set**

  A grouping set is specified by zero or more comma-separated expressions in parentheses. When the grouping set has only one element, parentheses can be omitted.

  For example, `GROUPING SETS ((a), (b))` is the same as `GROUPING SETS (a, b)`.

  Syntax: `{ ( [ expression [ , ... ] ] ) | expression }`
- **GROUPING SETS**

  Groups the rows for each grouping set specified after `GROUPING SETS`.

  For example, `GROUP BY GROUPING SETS ((warehouse), (product))` is semantically equivalent to union of results of `GROUP BY warehouse` and `GROUP BY product`. This clause is a shorthand for a UNION ALL where each leg of the `UNION ALL` operator performs aggregation of each grouping set specified in the `GROUPING SETS` clause.

  Similarly, `GROUP BY GROUPING SETS ((warehouse, product), (product), ())` is semantically equivalent to the union of results of `GROUP BY warehouse, product`, `GROUP BY product` and global aggregate.
- **ROLLUP**

Specifies multiple levels of aggregations in a single statement. This clause is used to compute aggregations based on multiple grouping sets. ROLLUP is a shorthand for GROUPING SETS.

For example, GROUP BY warehouse, product WITH ROLLUP or GROUP BY ROLLUP(warehouse, product) is equivalent to GROUP BY GROUPING SETS((warehouse, product), (warehouse), ()).

GROUP BY ROLLUP(warehouse, product, (warehouse, location)) is equivalent to GROUP BY GROUPING SETS((warehouse, product, location), (warehouse, product), (warehouse), ()).

The N elements of a ROLLUP specification results in N+1 GROUPING SETS.

- **CUBE**

  CUBE clause is used to perform aggregations based on combination of grouping columns specified in the GROUP BY clause. CUBE is a shorthand for GROUPING SETS.

  For example, GROUP BY warehouse, product WITH CUBE or GROUP BY CUBE(warehouse, product) is equivalent to GROUP BY GROUPING SETS((warehouse, product), (warehouse), (product), ()).

  GROUP BY CUBE(warehouse, product, (warehouse, location)) is equivalent to GROUP BY GROUPING SETS((warehouse, product, location), (warehouse, product), (warehouse, location), (product, warehouse, location), (warehouse), (product), (warehouse, product), ()). The N elements of a CUBE specification results in 2^N GROUPING SETS.

- **Mixed/Nested Grouping Analytics**

  A GROUP BY clause can include multiple group_expressions and multiple CUBE|ROLLUP|GROUPING SETS. GROUPING SETS can also have nested CUBE|ROLLUP|GROUPING SETS clauses, such as GROUPING SETS(ROLLUP(warehouse, location), CUBE(warehouse, location)), GROUPING SETS(warehouse, GROUPING SETS(location, GROUPING SETS(ROLLUP(warehouse, location), CUBE(warehouse, location)))).

  CUBE|ROLLUP is just a syntax sugar for GROUPING SETS. Refer to the sections above for how to translate CUBE|ROLLUP to GROUPING SETS. group_expression can be treated as a single-group GROUPING SETS under this context.

For multiple `GROUPING SETS` in the `GROUP BY` clause, we generate a single `GROUPING SETS` by doing a cross-product of the original `GROUPING SETS`. For nested `GROUPING SETS` in the `GROUPING SETS` clause, we simply take its grouping sets and strip it.

For example, `GROUP BY warehouse, GROUPING SETS((product), ()), GROUPING SETS((location, size), (location), (size), ()) and GROUP BY warehouse, ROLLUP(product), CUBE(location, size)` is equivalent to `GROUP BY GROUPING SETS( (warehouse, product, location, size), (warehouse, product, location), (warehouse, product, size), (warehouse, product), (warehouse, location, size), (warehouse, location), (warehouse, size), (warehouse))`.

`GROUP BY GROUPING SETS(GROUPING SETS(warehouse), GROUPING SETS((warehouse, product)))` is equivalent to `GROUP BY GROUPING SETS((warehouse), (warehouse, product))`.

- **aggregate_name**

  Specifies an aggregate function name (`MIN`, `MAX`, `COUNT`, `SUM`, `AVG`, and so on).

- **DISTINCT**

  Removes duplicates in input rows before they are passed to aggregate functions.

- **FILTER**

  Filters the input rows for which the `boolean_expression` in the `WHERE` clause evaluates to true are passed to the aggregate function; other rows are discarded.

Examples

```
CREATE TABLE dealer (id INT, city STRING, car_model STRING, quantity INT);
INSERT INTO dealer VALUES
(100, 'Fremont', 'Honda Civic', 10),
(100, 'Fremont', 'Honda Accord', 15),
(100, 'Fremont', 'Honda CRV', 7),
(200, 'Dublin', 'Honda Civic', 20),
(200, 'Dublin', 'Honda Accord', 10),
(200, 'Dublin', 'Honda CRV', 3),
(300, 'San Jose', 'Honda Civic', 5),
(300, 'San Jose', 'Honda Accord', 8);
```

```
-- Sum of quantity per dealership. Group by `id`.
SELECT id, sum(quantity) FROM dealer GROUP BY id ORDER BY id;
+---+-------------+
| id|sum(quantity)|
+---+-------------+
|100|           32|
|200|           33|
|300|           13|
+---+-------------+

-- Use column position in GROUP by clause.
SELECT id, sum(quantity) FROM dealer GROUP BY 1 ORDER BY 1;
+---+-------------+
| id|sum(quantity)|
+---+-------------+
|100|           32|
|200|           33|
|300|           13|
+---+-------------+

-- Multiple aggregations.
-- 1. Sum of quantity per dealership.
-- 2. Max quantity per dealership.
SELECT id, sum(quantity) AS sum, max(quantity) AS max FROM dealer GROUP BY id ORDER BY
 id;
+---+---+---+
| id|sum|max|
+---+---+---+
|100| 32| 15|
|200| 33| 20|
|300| 13|  8|
+---+---+---+

-- Count the number of distinct dealer cities per car_model.
SELECT car_model, count(DISTINCT city) AS count FROM dealer GROUP BY car_model;
+------------+-----+
|   car_model|count|
+------------+-----+
| Honda Civic|    3|
|   Honda CRV|    2|
|Honda Accord|    3|
+------------+-----+

-- Sum of only 'Honda Civic' and 'Honda CRV' quantities per dealership.
```

```
SELECT id, sum(quantity) FILTER (
WHERE car_model IN ('Honda Civic', 'Honda CRV')
) AS `sum(quantity)` FROM dealer
GROUP BY id ORDER BY id;
+---+-------------+
| id|sum(quantity)|
+---+-------------+
|100|           17|
|200|           23|
|300|            5|
+---+-------------+

-- Aggregations using multiple sets of grouping columns in a single statement.
-- Following performs aggregations based on four sets of grouping columns.
-- 1. city, car_model
-- 2. city
-- 3. car_model
-- 4. Empty grouping set. Returns quantities for all city and car models.
SELECT city, car_model, sum(quantity) AS sum FROM dealer
GROUP BY GROUPING SETS ((city, car_model), (city), (car_model), ())
ORDER BY city;
+---------+------------+---+
|     city|   car_model|sum|
+---------+------------+---+
|     null|        null| 78|
|     null| HondaAccord| 33|
|     null|    HondaCRV| 10|
|     null|  HondaCivic| 35|
|   Dublin|        null| 33|
|   Dublin| HondaAccord| 10|
|   Dublin|    HondaCRV|  3|
|   Dublin|  HondaCivic| 20|
|  Fremont|        null| 32|
|  Fremont| HondaAccord| 15|
|  Fremont|    HondaCRV|  7|
|  Fremont|  HondaCivic| 10|
| San Jose|        null| 13|
| San Jose| HondaAccord|  8|
| San Jose|  HondaCivic|  5|
+---------+------------+---+

-- Group by processing with `ROLLUP` clause.
-- Equivalent GROUP BY GROUPING SETS ((city, car_model), (city), ())
SELECT city, car_model, sum(quantity) AS sum FROM dealer
```

```
GROUP BY city, car_model WITH ROLLUP
ORDER BY city, car_model;
+---------+------------+---+
|     city|   car_model|sum|
+---------+------------+---+
|     null|        null| 78|
|   Dublin|        null| 33|
|   Dublin| HondaAccord| 10|
|   Dublin|    HondaCRV|  3|
|   Dublin|  HondaCivic| 20|
|  Fremont|        null| 32|
|  Fremont| HondaAccord| 15|
|  Fremont|    HondaCRV|  7|
|  Fremont|  HondaCivic| 10|
| San Jose|        null| 13|
| San Jose| HondaAccord|  8|
| San Jose|  HondaCivic|  5|
+---------+------------+---+

-- Group by processing with `CUBE` clause.
-- Equivalent GROUP BY GROUPING SETS ((city, car_model), (city), (car_model), ())
SELECT city, car_model, sum(quantity) AS sum FROM dealer
GROUP BY city, car_model WITH CUBE
ORDER BY city, car_model;
+---------+------------+---+
|     city|   car_model|sum|
+---------+------------+---+
|     null|        null| 78|
|     null| HondaAccord| 33|
|     null|    HondaCRV| 10|
|     null|  HondaCivic| 35|
|   Dublin|        null| 33|
|   Dublin| HondaAccord| 10|
|   Dublin|    HondaCRV|  3|
|   Dublin|  HondaCivic| 20|
|  Fremont|        null| 32|
|  Fremont| HondaAccord| 15|
|  Fremont|    HondaCRV|  7|
|  Fremont|  HondaCivic| 10|
| San Jose|        null| 13|
| San Jose| HondaAccord|  8|
| San Jose|  HondaCivic|  5|
+---------+------------+---+
```

```
--Prepare data for ignore nulls example
CREATE TABLE person (id INT, name STRING, age INT);
INSERT INTO person VALUES
(100, 'Mary', NULL),
(200, 'John', 30),
(300, 'Mike', 80),
(400, 'Dan', 50);

--Select the first row in column age
SELECT FIRST(age) FROM person;
+--------------------+
| first(age, false)  |
+--------------------+
| NULL               |
+--------------------+

--Get the first row in column `age` ignore nulls,last row in column `id` and sum of
 column `id`.
SELECT FIRST(age IGNORE NULLS), LAST(id), SUM(id) FROM person;
+-------------------+------------------+----------+
| first(age, true)  | last(id, false)  | sum(id)  |
+-------------------+------------------+----------+
| 30                | 400              | 1000     |
+-------------------+------------------+----------+
```

HAVING clause

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

The `HAVING` clause is used to filter the results produced by `GROUP BY` based on the specified condition. It is often used in conjunction with a `GROUP BY` clause.

Syntax

```
HAVING boolean_expression
```

Parameters

- **boolean_expression**

  Specifies any expression that evaluates to a result type boolean. Two or more expressions may be combined together using the logical operators (AND, OR).

  Note The expressions specified in the HAVING clause can only refer to:

  1. Constants
  2. Expressions that appear in GROUP BY
  3. Aggregate functions

Examples

```
CREATE TABLE dealer (id INT, city STRING, car_model STRING, quantity INT);
INSERT INTO dealer VALUES
(100, 'Fremont', 'Honda Civic', 10),
(100, 'Fremont', 'Honda Accord', 15),
(100, 'Fremont', 'Honda CRV', 7),
(200, 'Dublin', 'Honda Civic', 20),
(200, 'Dublin', 'Honda Accord', 10),
(200, 'Dublin', 'Honda CRV', 3),
(300, 'San Jose', 'Honda Civic', 5),
(300, 'San Jose', 'Honda Accord', 8);

-- `HAVING` clause referring to column in `GROUP BY`.
SELECT city, sum(quantity) AS sum FROM dealer GROUP BY city HAVING city = 'Fremont';
+-------+---+
|   city|sum|
+-------+---+
|Fremont| 32|
+-------+---+

-- `HAVING` clause referring to aggregate function.
SELECT city, sum(quantity) AS sum FROM dealer GROUP BY city HAVING sum(quantity) > 15;
+-------+---+
|   city|sum|
+-------+---+
| Dublin| 33|
|Fremont| 32|
+-------+---+

-- `HAVING` clause referring to aggregate function by its alias.
```

```
SELECT city, sum(quantity) AS sum FROM dealer GROUP BY city HAVING sum > 15;
+-------+---+
|   city|sum|
+-------+---+
| Dublin| 33|
|Fremont| 32|
+-------+---+

-- `HAVING` clause referring to a different aggregate function than what is present in
-- `SELECT` list.
SELECT city, sum(quantity) AS sum FROM dealer GROUP BY city HAVING max(quantity) > 15;
+------+---+
|  city|sum|
+------+---+
|Dublin| 33|
+------+---+

-- `HAVING` clause referring to constant expression.
SELECT city, sum(quantity) AS sum FROM dealer GROUP BY city HAVING 1 > 0 ORDER BY city;
+--------+---+
|    city|sum|
+--------+---+
|  Dublin| 33|
| Fremont| 32|
|San Jose| 13|
+--------+---+

-- `HAVING` clause without a `GROUP BY` clause.
SELECT sum(quantity) AS sum FROM dealer HAVING sum(quantity) > 10;
+---+
|sum|
+---+
| 78|
+---+
```

ORDER BY clause

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

The ORDER BY clause is used to return the result rows in a sorted manner in the user specified order. Unlike the SORT BY clause, this clause guarantees a total order in the output.

Syntax

```
ORDER BY { expression [ sort_direction | nulls_sort_order ] [ , ... ] }
```

Parameters

- **ORDER BY**

  Specifies a comma-separated list of expressions along with optional parameters sort_direction and nulls_sort_order which are used to sort the rows.
- **sort_direction**

  Optionally specifies whether to sort the rows in ascending or descending order.

  The valid values for the sort direction are ASC for ascending and DESC for descending.

  If sort direction is not explicitly specified, then by default rows are sorted ascending.

  Syntax: [ASC | DESC]
- **nulls_sort_order**

  Optionally specifies whether NULL values are returned before/after non-NULL values.

  If null_sort_order is not specified, then NULLs sort first if sort order is ASC and NULLS sort last if sort order is DESC.

  1. If NULLS FIRST is specified, then NULL values are returned first regardless of the sort order.

  2. If NULLS LAST is specified, then NULL values are returned last regardless of the sort order.

  Syntax: [NULLS { FIRST | LAST }]

Examples

```
CREATE TABLE person (id INT, name STRING, age INT);
INSERT INTO person VALUES
(100, 'John', 30),
(200, 'Mary', NULL),
```

```
(300, 'Mike', 80),
(400, 'Jerry', NULL),
(500, 'Dan',  50);

-- Sort rows by age. By default rows are sorted in ascending manner with NULL FIRST.
SELECT name, age FROM person ORDER BY age;
+-----+----+
| name| age|
+-----+----+
|Jerry|null|
| Mary|null|
| John|  30|
|  Dan|  50|
| Mike|  80|
+-----+----+

-- Sort rows in ascending manner keeping null values to be last.
SELECT name, age FROM person ORDER BY age NULLS LAST;
+-----+----+
| name| age|
+-----+----+
| John|  30|
|  Dan|  50|
| Mike|  80|
| Mary|null|
|Jerry|null|
+-----+----+

-- Sort rows by age in descending manner, which defaults to NULL LAST.
SELECT name, age FROM person ORDER BY age DESC;
+-----+----+
| name| age|
+-----+----+
| Mike|  80|
|  Dan|  50|
| John|  30|
|Jerry|null|
| Mary|null|
+-----+----+

-- Sort rows in ascending manner keeping null values to be first.
SELECT name, age FROM person ORDER BY age DESC NULLS FIRST;
+-----+----+
| name| age|
```

```
+-----+----+
|Jerry|null|
| Mary|null|
| Mike|  80|
|  Dan|  50|
| John|  30|
+-----+----+

-- Sort rows based on more than one column with each column having different
-- sort direction.
SELECT * FROM person ORDER BY name ASC, age DESC;
+---+-----+----+
| id| name| age|
+---+-----+----+
|500|  Dan|  50|
|400|Jerry|null|
|100| John|  30|
|200| Mary|null|
|300| Mike|  80|
+---+-----+----+
```

JOIN clause

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called "Supported SQL commands".

A SQL join is used to combine rows from two relations based on join criteria. The following section describes the overall join syntax and the different types of joins along with examples.

Syntax

```
relation INNER JOIN relation [ join_criteria ]
```

Parameters

- **relation**

  Specifies the relation to be joined.
- **join_type**

Specifies the join type.

Syntax: `INNER | CROSS | LEFT OUTER`

- **join_criteria**

  Specifies how the rows from one relation will be combined with the rows of another relation.

  Syntax: `ON boolean_expression | USING ( column_name [ , ... ] )`

- **boolean_expression**

  Specifies an expression with a return type of boolean.

Join types

- **Inner Join**

  The inner join needs to be explicitly specified. It selects rows that have matching values in both relations.

  Syntax: `relation INNER JOIN relation [ join_criteria ]`

- **Left Join**

  A left join returns all values from the left relation and the matched values from the right relation, or appends NULL if there is no match. It is also referred to as a left outer join.

  Syntax: `relation LEFT OUTER JOIN relation [ join_criteria ]`

- **Cross Join**

  A cross join returns the Cartesian product of two relations.

  Syntax: `relation CROSS JOIN relation [ join_criteria ]`

Examples

```
-- Use employee and department tables to demonstrate different type of joins.
SELECT * FROM employee;
+---+-----+------+
| id| name|deptno|
+---+-----+------+
```

```
|105|Chloe|     5|
|103| Paul|     3|
|101| John|     1|
|102| Lisa|     2|
|104| Evan|     4|
|106|  Amy|     6|
+---+-----+------+
SELECT * FROM department;
+------+-----------+
|deptno|   deptname|
+------+-----------+
|     3|Engineering|
|     2|      Sales|
|     1|  Marketing|
+------+-----------+

-- Use employee and department tables to demonstrate inner join.
SELECT id, name, employee.deptno, deptname
FROM employee INNER JOIN department ON employee.deptno = department.deptno;
+---+-----+------+-----------|
| id| name|deptno|   deptname|
+---+-----+------+-----------|
|103| Paul|     3|Engineering|
|101| John|     1|  Marketing|
|102| Lisa|     2|      Sales|
+---+-----+------+-----------|

-- Use employee and department tables to demonstrate left join.
SELECT id, name, employee.deptno, deptname
FROM employee LEFT JOIN department ON employee.deptno = department.deptno;
+---+-----+------+-----------|
| id| name|deptno|   deptname|
+---+-----+------+-----------|
|105|Chloe|     5|       NULL|
|103| Paul|     3|Engineering|
|101| John|     1|  Marketing|
|102| Lisa|     2|      Sales|
|104| Evan|     4|       NULL|
|106|  Amy|     6|       NULL|
+---+-----+------+-----------|

-- Use employee and department tables to demonstrate cross join.
SELECT id, name, employee.deptno, deptname FROM employee CROSS JOIN department;
+---+-----+------+-----------|
```

```
| id| name|deptno|   deptname|
+---+-----+------+-----------+
|105|Chloe|     5|Engineering|
|105|Chloe|     5|  Marketing|
|105|Chloe|     5|      Sales|
|103| Paul|     3|Engineering|
|103| Paul|     3|  Marketing|
|103| Paul|     3|      Sales|
|101| John|     1|Engineering|
|101| John|     1|  Marketing|
|101| John|     1|      Sales|
|102| Lisa|     2|Engineering|
|102| Lisa|     2|  Marketing|
|102| Lisa|     2|      Sales|
|104| Evan|     4|Engineering|
|104| Evan|     4|  Marketing|
|104| Evan|     4|      Sales|
|106|  Amy|     4|Engineering|
|106|  Amy|     4|  Marketing|
|106|  Amy|     4|      Sales|
+---+-----+------+-----------+
```

LIMIT clause

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called "Supported SQL commands".

The `LIMIT` clause is used to constrain the number of rows returned by the `SELECT` statement. In general, this clause is used in conjunction with `ORDER BY` to ensure that the results are deterministic.

Syntax

```
LIMIT { ALL | integer_expression }
```

Parameters

- **ALL**

If specified, the query returns all the rows. In other words, no limit is applied if this option is specified.

- **integer_expression**

  Specifies a foldable expression that returns an integer.

Examples

```
CREATE TABLE person (name STRING, age INT);
INSERT INTO person VALUES
('Jane Doe', 25),
('Pat C', 18),
('Nikki W', 16),
('John D', 25),
('Juan L', 18),
('Jorge S', 16);

-- Select the first two rows.
SELECT name, age FROM person ORDER BY name LIMIT 2;
+-------+---+
|   name|age|
+-------+---+
|  Pat C| 18|
|Jorge S| 16|
+------+---+

-- Specifying ALL option on LIMIT returns all the rows.
SELECT name, age FROM person ORDER BY name LIMIT ALL;
+--------+---+
|    name|age|
+--------+---+
|   Pat C| 18|
| Jorge S| 16|
|  Juan L| 18|
|  John D| 25|
| Nikki W| 16|
|Jane Doe| 25|
+--------+---+

-- A function expression as an input to LIMIT.
SELECT name, age FROM person ORDER BY name LIMIT length('OPENSEARCH');
```

```
+-------+---+
|   name|age|
+-------+---+
|  Pat C| 18|
|Jorge S| 16|
| Juan L| 18|
| John D| 25|
|Nikki W| 16|
+-------+---+
```

CASE clause

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

The CASE clause uses a rule to return a specific result based on the specified condition, similar to if/else statements in other programming languages.

Syntax

```
CASE [ expression ] { WHEN boolean_expression THEN then_expression } [ ... ]
[ ELSE else_expression ]
END
```

Parameters

- **boolean_expression**

  Specifies any expression that evaluates to a result type boolean.

  Two or more expressions may be combined together using the logical operators (AND, OR).

- **then_expression**

  Specifies the then expression based on the boolean_expression condition.

  then_expression and else_expression should all be same type or coercible to a common type.

- **else_expression**

  Specifies the default expression.

  `then_expression` and `else_expression` should all be same type or coercible to a common type.

Examples

```
CREATE TABLE person (id INT, name STRING, age INT);
INSERT INTO person VALUES
(100, 'John', 30),
(200, 'Mary', NULL),
(300, 'Mike', 80),
(400, 'Dan', 50);
SELECT id, CASE WHEN id > 200 THEN 'bigger' ELSE 'small' END FROM person;
+------+-------------------------------------------------+
|  id  | CASE WHEN (id > 200) THEN bigger ELSE small END  |
+------+-------------------------------------------------+
| 100  | small                                           |
| 200  | small                                           |
| 300  | bigger                                          |
| 400  | bigger                                          |
+------+-------------------------------------------------+
SELECT id, CASE id WHEN 100 then 'bigger' WHEN  id > 300 THEN '300' ELSE 'small' END
 FROM person;
+------
+-------------------------------------------------------------------------------------------------------
+
|  id  | CASE WHEN (id = 100) THEN bigger WHEN (id = CAST((id > 300) AS INT)) THEN 300
 ELSE small END  |
+------
+-------------------------------------------------------------------------------------------------------
+
| 100  | bigger
                |
| 200  | small
                |
| 300  | small
                |
| 400  | small
                |
```

```
+------
+-------------------------------------------------------------------------------------------------------------------------------
+
```

Common table expression

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called "Supported SQL commands".

A common table expression (CTE) defines a temporary result set that a user can reference possibly multiple times within the scope of a SQL statement. A CTE is used mainly in a SELECT statement.

Syntax

```
WITH common_table_expression [ , ... ]
```

While `common_table_expression` is defined as:

```
Syntexpression_name [ ( column_name [ , ... ] ) ] [ AS ] ( query )
```

Parameters

- **expression_name**

  Specifies a name for the common table expression.
- **query**

  A SELECT statement.

Examples

```
-- CTE with multiple column aliases
WITH t(x, y) AS (SELECT 1, 2)
SELECT * FROM t WHERE x = 1 AND y = 2;
+---+---+
|  x|  y|
+---+---+
```

```
|  1|  2|
+---+---+

-- CTE in CTE definition
WITH t AS (
WITH t2 AS (SELECT 1)
SELECT * FROM t2
)
SELECT * FROM t;
+---+
|  1|
+---+
|  1|
+---+

-- CTE in subquery
SELECT max(c) FROM (
WITH t(c) AS (SELECT 1)
SELECT * FROM t
);
+------+
|max(c)|
+------+
|     1|
+------+

-- CTE in subquery expression
SELECT (
WITH t AS (SELECT 1)
SELECT * FROM t
);
+----------------+
|scalarsubquery()|
+----------------+
|               1|
+----------------+

-- CTE in CREATE VIEW statement
CREATE VIEW v AS
WITH t(a, b, c, d) AS (SELECT 1, 2, 3, 4)
SELECT * FROM t;
SELECT * FROM v;
+---+---+---+---+
|  a|  b|  c|  d|
```

```
+---+---+---+---+
|  1|  2|  3|  4|
+---+---+---+---+
```

EXPLAIN

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

The `EXPLAIN` statement is used to provide logical/physical plans for an input statement. By default, this clause provides information about a physical plan only.

Syntax

```
EXPLAIN [ EXTENDED | CODEGEN | COST | FORMATTED ] statement
```

Parameters

- **EXTENDED**

  Generates parsed logical plan, analyzed logical plan, optimized logical plan and physical plan.

  Parsed Logical plan is a unresolved plan that extracted from the query.

  Analyzed logical plans transforms which translates `unresolvedAttribute` and `unresolvedRelation` into fully typed objects.

  The optimized logical plan transforms through a set of optimization rules, resulting in the physical plan.

- **CODEGEN**

  Generates code for the statement, if any and a physical plan.

- **COST**

  If plan node statistics are available, generates a logical plan and the statistics.

- **FORMATTED**

Generates two sections: a physical plan outline and node details.

- **statement**

  Specifies a SQL statement to be explained.

Examples

```
-- Default Output
EXPLAIN select k, sum(v) from values (1, 2), (1, 3) t(k, v) group by k;
+----------------------------------------------------+
|                                                plan|
+----------------------------------------------------+
| == Physical Plan ==
*(2) HashAggregate(keys=[k#33], functions=[sum(cast(v#34 as bigint))])
+- Exchange hashpartitioning(k#33, 200), true, [id=#59]
+- *(1) HashAggregate(keys=[k#33], functions=[partial_sum(cast(v#34 as bigint))])
+- *(1) LocalTableScan [k#33, v#34]
|
+----------------------------------------------------

-- Using Extended
EXPLAIN EXTENDED select k, sum(v) from values (1, 2), (1, 3) t(k, v) group by k;
+----------------------------------------------------+
|                                                plan|
+----------------------------------------------------+
| == Parsed Logical Plan ==
'Aggregate ['k], ['k, unresolvedalias('sum('v), None)]
 +- 'SubqueryAlias `t`
+- 'UnresolvedInlineTable [k, v], [List(1, 2), List(1, 3)]

 == Analyzed Logical Plan ==
 k: int, sum(v): bigint
 Aggregate [k#47], [k#47, sum(cast(v#48 as bigint)) AS sum(v)#50L]
 +- SubqueryAlias `t`
    +- LocalRelation [k#47, v#48]

 == Optimized Logical Plan ==
 Aggregate [k#47], [k#47, sum(cast(v#48 as bigint)) AS sum(v)#50L]
 +- LocalRelation [k#47, v#48]

 == Physical Plan ==
```

```
 *(2) HashAggregate(keys=[k#47], functions=[sum(cast(v#48 as bigint))], output=[k#47,
 sum(v)#50L])
+- Exchange hashpartitioning(k#47, 200), true, [id=#79]
   +- *(1) HashAggregate(keys=[k#47], functions=[partial_sum(cast(v#48 as bigint))],
 output=[k#47, sum#52L])
    +- *(1) LocalTableScan [k#47, v#48]
|
+-----------------------------------------------------+

-- Using Formatted
EXPLAIN FORMATTED select k, sum(v) from values (1, 2), (1, 3) t(k, v) group by k;
+-----------------------------------------------------+
|                                                 plan|
+-----------------------------------------------------+
| == Physical Plan ==
 * HashAggregate (4)
 +- Exchange (3)
    +- * HashAggregate (2)
       +- * LocalTableScan (1)


 (1) LocalTableScan [codegen id : 1]
 Output: [k#19, v#20]

 (2) HashAggregate [codegen id : 1]
 Input: [k#19, v#20]

 (3) Exchange
 Input: [k#19, sum#24L]

 (4) HashAggregate [codegen id : 2]
 Input: [k#19, sum#24L]
|
+-----------------------------------------------------+
```

LATERAL SUBQUERY clause

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

LATERAL SUBQUERY is a subquery that is preceded by the keyword LATERAL. It provides a way to reference columns in the preceding FROM clause. Without the LATERAL keyword, subqueries can only refer to columns in the outer query, but not in the FROM clause. LATERAL SUBQUERY makes the complicated queries simpler and more efficient.

Syntax

```
[ LATERAL ] primary_relation [ join_relation ]
```

Parameters

- **primary_relation**

  Specifies the primary relation. It can be one of the following:

  1. Table relation
  2. Aliased query

     Syntax: `( query ) [ [ AS ] alias ]`
  3. Aliased relation

     Syntax: `( relation ) [ [ AS ] alias ]`

Examples

```
CREATE TABLE t1 (c1 INT, c2 INT);
INSERT INTO t1 VALUES (0, 1), (1, 2);
CREATE TABLE t2 (c1 INT, c2 INT);
INSERT INTO t2 VALUES (0, 2), (0, 3);
SELECT * FROM t1,
LATERAL (SELECT * FROM t2 WHERE t1.c1 = t2.c1);
+--------+-------+--------+-------+
|  t1.c1 | t1.c2 |  t2.c1 | t2.c2 |
+-------+--------+--------+-------+
|    0   |   1   |    0   |   3   |
|    0   |   1   |    0   |   2   |
+-------+--------+--------+-------+
SELECT a, b, c FROM t1,
LATERAL (SELECT c1 + c2 AS a),
LATERAL (SELECT c1 - c2 AS b),
LATERAL (SELECT a * b AS c);
```

```
+--------+-------+--------+
|    a   |   b   |    c   |
+-------+--------+--------+
|    3   |  -1   |   -3   |
|    1   |  -1   |   -1   |
+-------+--------+--------+
```

LATERAL VIEW clause

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

The `LATERAL VIEW` clause is used in conjunction with generator functions such as `EXPLODE`, which will generate a virtual table containing one or more rows. `LATERAL VIEW` will apply the rows to each original output row.

Syntax

```
LATERAL VIEW [ OUTER ] generator_function ( expression [ , ... ] ) [ table_alias ] AS
 column_alias [ , ... ]
```

Parameters

- **OUTER**

  If `OUTER` specified, returns null if an input array/map is empty or null.
- **generator_function**

  Specifies a generator function (`EXPLODE`, `INLINE`, and so on.).
- **table_alias**

  The alias for `generator_function`, which is optional.
- **column_alias**

  Lists the column aliases of `generator_function`, which may be used in output rows.

  You can have multiple aliases if `generator_function` has multiple output columns.

Examples

```
CREATE TABLE person (id INT, name STRING, age INT, class INT, address STRING);
INSERT INTO person VALUES
(100, 'John', 30, 1, 'Street 1'),
(200, 'Mary', NULL, 1, 'Street 2'),
(300, 'Mike', 80, 3, 'Street 3'),
(400, 'Dan', 50, 4, 'Street 4');
SELECT * FROM person
LATERAL VIEW EXPLODE(ARRAY(30, 60)) tableName AS c_age
LATERAL VIEW EXPLODE(ARRAY(40, 80)) AS d_age;
+------+-------+-------+--------+-----------+--------+--------+
|  id  | name  |  age  | class  |  address  | c_age  | d_age  |
+------+-------+-------+--------+-----------+--------+--------+
| 100  | John  | 30    | 1      | Street 1  | 30     | 40     |
| 100  | John  | 30    | 1      | Street 1  | 30     | 80     |
| 100  | John  | 30    | 1      | Street 1  | 60     | 40     |
| 100  | John  | 30    | 1      | Street 1  | 60     | 80     |
| 200  | Mary  | NULL  | 1      | Street 2  | 30     | 40     |
| 200  | Mary  | NULL  | 1      | Street 2  | 30     | 80     |
| 200  | Mary  | NULL  | 1      | Street 2  | 60     | 40     |
| 200  | Mary  | NULL  | 1      | Street 2  | 60     | 80     |
| 300  | Mike  | 80    | 3      | Street 3  | 30     | 40     |
| 300  | Mike  | 80    | 3      | Street 3  | 30     | 80     |
| 300  | Mike  | 80    | 3      | Street 3  | 60     | 40     |
| 300  | Mike  | 80    | 3      | Street 3  | 60     | 80     |
| 400  | Dan   | 50    | 4      | Street 4  | 30     | 40     |
| 400  | Dan   | 50    | 4      | Street 4  | 30     | 80     |
| 400  | Dan   | 50    | 4      | Street 4  | 60     | 40     |
| 400  | Dan   | 50    | 4      | Street 4  | 60     | 80     |
+------+-------+-------+--------+-----------+--------+--------+
SELECT c_age, COUNT(1) FROM person
LATERAL VIEW EXPLODE(ARRAY(30, 60)) AS c_age
LATERAL VIEW EXPLODE(ARRAY(40, 80)) AS d_age
GROUP BY c_age;
+--------+-----------+
| c_age  | count(1)  |
+--------+-----------+
| 60     | 8         |
| 30     | 8         |
+--------+-----------+
SELECT * FROM person
LATERAL VIEW EXPLODE(ARRAY()) tableName AS c_age;
+-----+-------+------+--------+----------+--------+
```

```
| id  | name  | age  | class  | address  | c_age  |
+-----+-------+------+--------+----------+--------+
+-----+-------+------+--------+----------+--------+
SELECT * FROM person
LATERAL VIEW OUTER EXPLODE(ARRAY()) tableName AS c_age;
+------+-------+-------+--------+-----------+--------+
|  id  | name  |  age  | class  |  address  | c_age  |
+------+-------+-------+--------+-----------+--------+
| 100  | John  | 30    | 1      | Street 1  | NULL   |
| 200  | Mary  | NULL  | 1      | Street 2  | NULL   |
| 300  | Mike  | 80    | 3      | Street 3  | NULL   |
| 400  | Dan   | 50    | 4      | Street 4  | NULL   |
+------+-------+-------+--------+-----------+--------+
```

LIKE predicate

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

A `LIKE` predicate is used to search for a specific pattern. This predicate also supports multiple patterns with quantifiers include `ANY`, `SOME`, and `ALL`.

Syntax

```
[ NOT ] { LIKE search_pattern [ ESCAPE esc_char ] | [ RLIKE | REGEXP ] regex_pattern }
[ NOT ] { LIKE quantifiers ( search_pattern [ , ... ]) }
```

Parameters

- **search_pattern**

  Specifies a string pattern to be searched by the LIKE clause. It can contain special pattern-matching characters:

  - `%` matches zero or more characters.
  - `_` matches exactly one character.
- **esc_char**

Specifies the escape character. The default escape character is \.

- **regex_pattern**

  Specifies a regular expression search pattern to be searched by the RLIKE or REGEXP clause.

- **quantifiers**

  Specifies the predicate quantifiers include ANY, SOME and ALL.

  ANY or SOME means if one of the patterns matches the input, then return true.

  ALL means if all the patterns matches the input, then return true.

Examples

```
CREATE TABLE person (id INT, name STRING, age INT);
INSERT INTO person VALUES
(100, 'John', 30),
(200, 'Mary', NULL),
(300, 'Mike', 80),
(400, 'Dan',  50),
(500, 'Evan_w', 16);
SELECT * FROM person WHERE name LIKE 'M%';
+---+----+----+
| id|name| age|
+---+----+----+
|300|Mike|  80|
|200|Mary|null|
+---+----+----+
SELECT * FROM person WHERE name LIKE 'M_ry';
+---+----+----+
| id|name| age|
+---+----+----+
|200|Mary|null|
+---+----+----+
SELECT * FROM person WHERE name NOT LIKE 'M_ry';
+---+------+---+
| id|  name|age|
+---+------+---+
|500|Evan_W| 16|
|300|  Mike| 80|
|100|  John| 30|
```

```
|400|   Dan| 50|
+---+------+---+
SELECT * FROM person WHERE name RLIKE 'M+';
+---+----+----+
| id|name| age|
+---+----+----+
|300|Mike|  80|
|200|Mary|null|
+---+----+----+
SELECT * FROM person WHERE name REGEXP 'M+';
+---+----+----+
| id|name| age|
+---+----+----+
|300|Mike|  80|
|200|Mary|null|
+---+----+----+
SELECT * FROM person WHERE name LIKE '%\_%';
+---+------+---+
| id|  name|age|
+---+------+---+
|500|Evan_W| 16|
+---+------+---+
SELECT * FROM person WHERE name LIKE '%$_%' ESCAPE '$';
+---+------+---+
| id|  name|age|
+---+------+---+
|500|Evan_W| 16|
+---+------+---+
SELECT * FROM person WHERE name LIKE ALL ('%an%', '%an');
+---+----+----+
| id|name| age|
+---+----+----+
|400| Dan|  50|
+---+----+----+
SELECT * FROM person WHERE name LIKE ANY ('%an%', '%an');
+---+------+---+
| id|  name|age|
+---+------+---+
|400|   Dan| 50|
|500|Evan_W| 16|
+---+------+---+
SELECT * FROM person WHERE name LIKE SOME ('%an%', '%an');
+---+------+---+
| id|  name|age|
```

```
+---+------+---+
|400|   Dan| 50|
|500|Evan_W| 16|
+---+------+---+
SELECT * FROM person WHERE name NOT LIKE ALL ('%an%', '%an');
+---+----+----+
| id|name| age|
+---+----+----+
|100|John|  30|
|200|Mary|null|
|300|Mike|  80|
+---+----+----+
SELECT * FROM person WHERE name NOT LIKE ANY ('%an%', '%an');
+---+------+----+
| id|  name| age|
+---+------+----+
|100|  John|  30|
|200|  Mary|null|
|300|  Mike|  80|
|500|Evan_W|  16|
+---+------+----+
SELECT * FROM person WHERE name NOT LIKE SOME ('%an%', '%an');
+---+------+----+
| id|  name| age|
+---+------+----+
|100|  John|  30|
|200|  Mary|null|
|300|  Mike|  80|
|500|Evan_W|  16|
+---+------+----+
```

OFFSET

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

The `OFFSET` clause is used to specify the number of rows to skip before beginning to return rows returned by the `SELECT` statement. In general, this clause is used in conjunction with `ORDER BY` to ensure that the results are deterministic.

Syntax

```
OFFSET integer_expression
```

Parameters

- **integer_expression**

  Specifies a foldable expression that returns an integer.

Examples

```
CREATE TABLE person (name STRING, age INT);
INSERT INTO person VALUES
('Jane Doe', 25),
('Pat C', 18),
('Nikki W', 16),
('Juan L', 25),
('John D', 18),
('Jorge S', 16);

-- Skip the first two rows.
SELECT name, age FROM person ORDER BY name OFFSET 2;
+-------+---+
|   name|age|
+-------+---+
| John D| 18|
| Juan L| 25|
|Nikki W| 16|
|Jane Doe| 25|
+-------+---+

-- Skip the first two rows and returns the next three rows.
SELECT name, age FROM person ORDER BY name LIMIT 3 OFFSET 2;
+-------+---+
|   name|age|
+-------+---+
| John D| 18|
| Juan L| 25|
|Nikki W| 16|
+-------+---+
```

```
-- A function expression as an input to OFFSET.
SELECT name, age FROM person ORDER BY name OFFSET length('WAGON');
+-------+---+
|   name|age|
+-------+---+
|Jane Doe| 25|
+-------+---+
```

PIVOT clause

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called "Supported SQL commands".

The `PIVOT` clause is used for data perspective. We can get the aggregated values based on specific column values, which will be turned to multiple columns used in `SELECT` clause. The `PIVOT` clause can be specified after the table name or subquery.

Syntax

```
PIVOT ( { aggregate_expression [ AS aggregate_expression_alias ] } [ , ... ] FOR
 column_list IN ( expression_list ) )
```

Parameters

- **aggregate_expression**

  Specifies an aggregate expression (`SUM(a)`, `COUNT(DISTINCT b)`, and so on.).

- **aggregate_expression_alias**

  Specifies an alias for the aggregate expression.

- **column_list**

  Contains columns in the `FROM` clause, which specifies the columns you want to replace with new columns. You can use brackets to surround the columns, such as `(c1, c2)`.

- **expression_list**

Specifies new columns, which are used to match values in `column_list` as the aggregating condition. You can also add aliases for them.

Examples

```
CREATE TABLE person (id INT, name STRING, age INT, class INT, address STRING);
INSERT INTO person VALUES
(100, 'John', 30, 1, 'Street 1'),
(200, 'Mary', NULL, 1, 'Street 2'),
(300, 'Mike', 80, 3, 'Street 3'),
(400, 'Dan', 50, 4, 'Street 4');
SELECT * FROM person
PIVOT (
SUM(age) AS a, AVG(class) AS c
FOR name IN ('John' AS john, 'Mike' AS mike)
);
+------+-----------+---------+---------+---------+---------+
|  id  |  address  | john_a  | john_c  | mike_a  | mike_c  |
+------+-----------+---------+---------+---------+---------+
| 200  | Street 2  | NULL    | NULL    | NULL    | NULL    |
| 100  | Street 1  | 30      | 1.0     | NULL    | NULL    |
| 300  | Street 3  | NULL    | NULL    | 80      | 3.0     |
| 400  | Street 4  | NULL    | NULL    | NULL    | NULL    |
+------+-----------+---------+---------+---------+---------+
SELECT * FROM person
PIVOT (
SUM(age) AS a, AVG(class) AS c
FOR (name, age) IN (('John', 30) AS c1, ('Mike', 40) AS c2)
);
+------+-----------+-------+-------+-------+-------+
|  id  |  address  | c1_a  | c1_c  | c2_a  | c2_c  |
+------+-----------+-------+-------+-------+-------+
| 200  | Street 2  | NULL  | NULL  | NULL  | NULL  |
| 100  | Street 1  | 30    | 1.0   | NULL  | NULL  |
| 300  | Street 3  | NULL  | NULL  | NULL  | NULL  |
| 400  | Street 4  | NULL  | NULL  | NULL  | NULL  |
+------+-----------+-------+-------+-------+-------+
```

Set operators

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

Set operators are used to combine two input relations into a single one. OpenSearch SQL supports three types of set operators:

- `EXCEPT` or `MINUS`
- `INTERSECT`
- `UNION`

Input relations must have the same number of columns and compatible data types for the respective columns.

EXCEPT

`EXCEPT` and `EXCEPT ALL` return the rows that are found in one relation but not the other. `EXCEPT` (alternatively, `EXCEPT DISTINCT`) takes only distinct rows while `EXCEPT ALL` does not remove duplicates from the result rows. Note that `MINUS` is an alias for `EXCEPT`.

Syntax

```
[ ( ] relation [ ) ] EXCEPT | MINUS [ ALL | DISTINCT ] [ ( ] relation [ ) ]
```

Examples

```
-- Use table1 and table2 tables to demonstrate set operators in this page.
SELECT * FROM table1;
+---+
|  c|
+---+
|  3|
|  1|
|  2|
|  2|
|  3|
```

```
|  4|
+---+
SELECT * FROM table2;
+---+
|  c|
+---+
|  5|
|  1|
|  2|
|  2|
+---+
SELECT c FROM table1 EXCEPT SELECT c FROM table2;
+---+
|  c|
+---+
|  3|
|  4|
+---+
SELECT c FROM table1 MINUS SELECT c FROM table2;
+---+
|  c|
+---+
|  3|
|  4|
+---+
SELECT c FROM table1 EXCEPT ALL (SELECT c FROM table2);
+---+
|  c|
+---+
|  3|
|  3|
|  4|
+---+
SELECT c FROM table1 MINUS ALL (SELECT c FROM table2);
+---+
|  c|
+---+
|  3|
|  3|
|  4|
+---+
```

INTERSECT

INTERSECT and INTERSECT ALL return the rows that are found in both relations. INTERSECT (alternatively, INTERSECT DISTINCT) takes only distinct rows while INTERSECT ALL does not remove duplicates from the result rows.

Syntax

```
[ ( ] relation [ ) ] INTERSECT [ ALL | DISTINCT ] [ ( ] relation [ ) ]
```

Examples

```
(SELECT c FROM table1) INTERSECT (SELECT c FROM table2);
+---+
|  c|
+---+
|  1|
|  2|
+---+
(SELECT c FROM table1) INTERSECT DISTINCT (SELECT c FROM table2);
+---+
|  c|
+---+
|  1|
|  2|
+---+
(SELECT c FROM table1) INTERSECT ALL (SELECT c FROM table2);
+---+
|  c|
+---+
|  1|
|  2|
|  2|
+---+
```

UNION

UNION and UNION ALL return the rows that are found in either relation. UNION (alternatively, UNION DISTINCT) takes only distinct rows while UNION ALL does not remove duplicates from the result rows.

Syntax

```
[ ( ] relation [ ) ] UNION [ ALL | DISTINCT ] [ ( ] relation [ ) ]
```

Examples

```
(SELECT c FROM table1) UNION (SELECT c FROM table2);
+---+
|  c|
+---+
|  1|
|  3|
|  5|
|  4|
|  2|
+---+
(SELECT c FROM table1) UNION DISTINCT (SELECT c FROM table2);
+---+
|  c|
+---+
|  1|
|  3|
|  5|
|  4|
|  2|
+---+
SELECT c FROM table1 UNION ALL (SELECT c FROM table2);
+---+
|  c|
+---+
|  3|
|  1|
|  2|
|  2|
|  3|
|  4|
|  5|
|  1|
|  2|
|  2|
+---+
```

SORT BY clause

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called "Supported SQL commands".

The SORT BY clause is used to return the result rows sorted within each partition in the user specified order. When there is more than one partition SORT BY may return result that is partially ordered. This is different than ORDER BY clause which guarantees a total order of the output.

Syntax

```
SORT BY { expression [ sort_direction | nulls_sort_order ] [ , ... ] }
```

Parameters

- **SORT BY**

  Specifies a comma-separated list of expressions along with optional parameters sort_direction and nulls_sort_order which are used to sort the rows within each partition.

- **sort_direction**

  Optionally specifies whether to sort the rows in ascending or descending order.

  The valid values for the sort direction are ASC for ascending and DESC for descending.

  If sort direction is not explicitly specified, then by default rows are sorted ascending.

  Syntax: [ASC | DESC]

- **nulls_sort_order**

  Optionally specifies whether NULL values are returned before/after non-NULL values.

  If null_sort_order is not specified, then NULLs sort first if the sort order is ASC and NULLS sort last if the sort order is DESC.

  1. If NULLS FIRST is specified, then NULL values are returned first regardless of the sort order.

  2. If NULLS LAST is specified, then NULL values are returned last regardless of the sort order.

Syntax: `[ NULLS { FIRST | LAST } ]`

Examples

```
CREATE TABLE person (zip_code INT, name STRING, age INT);
INSERT INTO person VALUES
(94588, 'Shirley Rodriguez', 50),
(94588, 'Juan Li', 18),
(94588, 'Anil K', 27),
(94588, 'John D', NULL),
(94511, 'David K', 42),
(94511, 'Aryan B.', 18),
(94511, 'Lalit B.', NULL);
-- Sort rows by `name` within each partition in ascending manner
SELECT name, age, zip_code FROM person SORT BY name;
+------------------+----+--------+
|              name| age|zip_code|
+------------------+----+--------+
|            Anil K|  27|   94588|
|           Juan Li|  18|   94588|
|            John D|null|   94588|
| Shirley Rodriguez|  50|   94588|
|          Aryan B.|  18|   94511|
|           David K|  42|   94511|
|          Lalit B.|null|   94511|
+------------------+----+--------+
-- Sort rows within each partition using column position.
SELECT name, age, zip_code FROM person SORT BY 1;
+----------------+----+----------+
|              name| age|zip_code|
+------------------+----+--------+
|            Anil K|  27|   94588|
|           Juan Li|  18|   94588|
|            John D|null|   94588|
| Shirley Rodriguez|  50|   94588|
|          Aryan B.|  18|   94511|
|           David K|  42|   94511|
|          Lalit B.|null|   94511|
+------------------+----+--------+

-- Sort rows within partition in ascending manner keeping null values to be last.
SELECT age, name, zip_code FROM person SORT BY age NULLS LAST;
```

```
+----+------------------+--------+
| age|              name|zip_code|
+----+------------------+--------+
|  18|           Juan Li|   94588|
|  27|            Anil K|   94588|
|  50| Shirley Rodriguez|   94588|
|null|            John D|   94588|
|  18|          Aryan B.|   94511|
|  42|           David K|   94511|
|null|          Lalit B.|   94511|
+----+------------------+--------+

-- Sort rows by age within each partition in descending manner, which defaults to NULL
 LAST.
SELECT age, name, zip_code FROM person SORT BY age DESC;
+----+------------------+--------+
| age|              name|zip_code|
+----+------------------+--------+
|  50|          Shirley Rodriguez|   94588|
|  27|            Anil K|   94588|
|  18|           Juan Li|   94588|
|null|            John D|   94588|
|  42|           David K|   94511|
|  18|          Aryan B.|   94511|
|null|          Lalit B.|   94511|
+----+------------------+--------+

-- Sort rows by age within each partition in descending manner keeping null values to
 be first.
SELECT age, name, zip_code FROM person SORT BY age DESC NULLS FIRST;
+----+------------------+--------+
| age|              name|zip_code|
+----+------------------+--------+
|null|            John D|   94588|
|  50| Shirley Rodriguez|   94588|
|  27|            Anil K|   94588|
|  18|           Juan Li|   94588|
|null|          Lalit B.|   94511|
|  42|           David K|   94511|
|  18|          Aryan B.|   94511|
+----+------------------+--------+

-- Sort rows within each partition based on more than one column with each column
 having
```

```
-- different sort direction.
SELECT name, age, zip_code FROM person
SORT BY name ASC, age DESC;
+------------------+----+--------+
|              name| age|zip_code|
+------------------+----+--------+
|            Anil K|  27|   94588|
|           Juan Li|  18|   94588|
|            John D|null|   94588|
| Shirley Rodriguez|  50|   94588|
|          Aryan B.|  18|   94511|
|           David K|  42|   94511|
|          Lalit B.|null|   94511|
+------------------+----+--------+
```

UNPIVOT

> **Note**
>
> To see which AWS data source integrations support this SQL command, see the section called “Supported SQL commands”.

The `UNPIVOT` clause transforms multiple columns into multiple rows used in `SELECT` clause. The `UNPIVOT` clause can be specified after the table name or subquery.

Syntax

```
UNPIVOT [ { INCLUDE | EXCLUDE } NULLS ] (
    { single_value_column_unpivot | multi_value_column_unpivot }
) [[AS] alias]

single_value_column_unpivot:
    values_column
    FOR name_column
    IN (unpivot_column [[AS] alias] [, ...])

multi_value_column_unpivot:
    (values_column [, ...])
    FOR name_column
    IN ((unpivot_column [, ...]) [[AS] alias] [, ...])
```

Parameters

- **unpivot_column**

  Contains columns in the FROM clause, which specifies the columns we want to unpivot.

- **name_column**

  The name for the column that holds the names of the unpivoted columns.

- **values_column**

  The name for the column that holds the values of the unpivoted columns.

Examples

```
CREATE TABLE sales_quarterly (year INT, q1 INT, q2 INT, q3 INT, q4 INT);
INSERT INTO sales_quarterly VALUES
(2020, null, 1000, 2000, 2500),
(2021, 2250, 3200, 4200, 5900),
(2022, 4200, 3100, null, null);
-- column names are used as unpivot columns
SELECT * FROM sales_quarterly
UNPIVOT (
sales FOR quarter IN (q1, q2, q3, q4)
);
+------+---------+-------+
| year | quarter | sales |
+------+---------+-------+
| 2020 | q2      | 1000  |
| 2020 | q3      | 2000  |
| 2020 | q4      | 2500  |
| 2021 | q1      | 2250  |
| 2021 | q2      | 3200  |
| 2021 | q3      | 4200  |
| 2021 | q4      | 5900  |
| 2022 | q1      | 4200  |
| 2022 | q2      | 3100  |
+------+---------+-------+
-- NULL values are excluded by default, they can be included
-- unpivot columns can be alias
-- unpivot result can be referenced via its alias
SELECT up.* FROM sales_quarterly
UNPIVOT INCLUDE NULLS (
```

```
sales FOR quarter IN (q1 AS Q1, q2 AS Q2, q3 AS Q3, q4 AS Q4)
) AS up;
+------+---------+-------+
| year | quarter | sales |
+------+---------+-------+
| 2020 | Q1      | NULL  |
| 2020 | Q2      | 1000  |
| 2020 | Q3      | 2000  |
| 2020 | Q4      | 2500  |
| 2021 | Q1      | 2250  |
| 2021 | Q2      | 3200  |
| 2021 | Q3      | 4200  |
| 2021 | Q4      | 5900  |
| 2022 | Q1      | 4200  |
| 2022 | Q2      | 3100  |
| 2022 | Q3      | NULL  |
| 2022 | Q4      | NULL  |
+------+---------+-------+
-- multiple value columns can be unpivoted per row
SELECT * FROM sales_quarterly
UNPIVOT EXCLUDE NULLS (
(first_quarter, second_quarter)
FOR half_of_the_year IN (
(q1, q2) AS H1,
(q3, q4) AS H2
)
);
+------+------------------+---------------+----------------+
|  id  | half_of_the_year | first_quarter | second_quarter |
+------+------------------+---------------+----------------+
| 2020 | H1               | NULL          | 1000           |
| 2020 | H2               | 2000          | 2500           |
| 2021 | H1               | 2250          | 3200           |
| 2021 | H2               | 4200          | 5900           |
| 2022 | H1               | 4200          | 3100           |
+------+------------------+---------------+----------------+
```

Supported PPL commands

The following reference tables show which PPL commands are supported in OpenSearch Discover for querying data in CloudWatch Logs, Amazon S3, or Security Lake, and which PPL commands are supported in CloudWatch Logs Insights. The PPL syntax supported in CloudWatch Logs Insights

and that supported in OpenSearch Discover for querying CloudWatch Logs are the same, and referenced as CloudWatch Logs in the following tables.

> **Note**
>
> When analyzing data outside of OpenSearch Service, commands may execute differently than they do on OpenSearch indexes.

Topics

Commands

PPL command	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called "fields"	Displays a set of fields that needs projection.	Supported	Supported	Supported	`fields field1, field2`
the section called "where"	Filters the data based on the conditions that you specify.	Supported	Supported	Supported	`where field1="success" | where field2 != "i-023fe0a90929d8822" | fields field3, col4, col5, col6 | head 1000`

PPL command	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called “stats”	Performs aggregations and calculations.	Supported	Supported	Supported	`stats count(), count(`field1`), min(`field1`), max(`field1`), avg(`field1`) by field2 | head 1000`
the section called “parse”	Extracts a regular expression (regex) pattern from a string and displays the extracted pattern. The extracted pattern can be further used to create new fields or filter data.	Supported	Supported	Supported	`parse `field1` ".*/(?<field2>[^/]+$)" | where field2 = "requestId" | fields field2, `field2` | head 1000`

PPL command	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called “patterns”	Extracts log patterns from a text field and appends the results to the search result. Grouping logs by their patterns makes it easier to aggregate stats from large volumes of log data for analysis and troublesh ooting.	Not supported	Supported	Supported	`patterns new_field='no_numbers' pattern='[0-9]' message | fields message, no_numbers`

PPL command	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called “sort”	Sort the displayed results by a field name. Use **sort *-FieldName*** to sort in descending order.	Supported	Supported	Supported	`stats count(), count(`field1`), min(`field1`) as field1Alias, max(`field1`), avg(`field1`) by field2 | sort -field1Alias | head 1000`

PPL command	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called “eval”	Modifies or processes the value of a field and stores it in a different field. This is useful to mathemati cally modify a column, apply string functions to a column, or apply date functions to a column.	Supported	Supported	Supported	`eval field2 = `field1` * 2 | fields field1, field2 | head 20`
the section called “rename”	Renames one or more fields in the search result.	Supported	Supported	Supported	`rename field2 as field1 | fields field1`
the section called “head”	Limits the displayed query results to the frst N rows.	Supported	Supported	Supported	`fields `@message` | head 20`

PPL command	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called “grok”	Parses a text field with a grok pattern based on regular expression, and appends the results to the search result.	Supported	Supported	Supported	`grok email '.+@%{HOSTNAME:host}' | fields email`
the section called “top”	Finds the most frequent values for a field.	Supported	Supported	Supported	`top 2 Field1 by Field2`
the section called “dedup”	Removes duplicate entries based on the fields that you specify.	Supported	Supported	Supported	`dedup field1 | fields field1, field2, field3`
the section called “join”	Joins two datasets together.	Not supported	Supported	Supported	`source=customer | join ON c_custkey = o_custkey orders | head 10`

PPL command	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called “lookup”	Enriches your search data by adding or replacing data from a lookup index (dimension table). You can extend fields of an index with values from a dimension table, append or replace values when lookup condition is matched	Not supported	Supported	Supported	`where orderType = 'Cancelled' | lookup account_list, mkt_id AS mkt_code replace amount, account_name as name | stats count(mkt_code), avg(amount) by name`

PPL command	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called “subquery”	Performs complex, nested queries within your Piped Processing Language (PPL) statements.	Not supported	Supported	Supported	`where id in [ subquery source=users | where user in [ subquery source=actions | where action="login" | fields user ] | fields uid ]`
the section called “rare”	Finds the least frequent values of all fields in the field list.	Supported	Supported	Supported	`rare Field1 by Field2`
the section called “trendline”	Calculates the moving averages of fields.	Supported	Supported	Supported	`trendline sma(2, field1) as field1Alias`

PPL command	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called “eventstats”	Enriches your event data with calculated summary statistics. It analyzes specified fields within your events, computes various statistical measures, and then appends these results to each original event as new fields.	Supported (except `count()`)	Supported	Supported	`eventstats sum(field1) by field2`
the section called “flatten”	Flattens a field, The field must be of this type: `struct<?,?> or array<struct<?,?>>`	Not supported	Supported	Supported	`source=table | flatten field1`

PPL command	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called “fieldsummary”	Calculates basic statistics for each field (count, distinct count, min, max, avg, stddev, and mean).	Supported (one field per query)	Supported	Supported	`where field1 != 200 | fieldsummary includefields=field1 nulls=true`
the section called “fillnull”	Fills null fields with the value that you provide. It can be used in one or more fields.	Not supported	Supported	Supported	`fields field1 | eval field2=field1 | fillnull value=0 field1`
the section called “Expand”	Breaks down a field containing multiple values into separate rows, creating a new row for each value in the specified field.	Not supported	Supported	Supported	`expand employee | stats max(salary) as max by state, company`

PPL command	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called “describe”	Gets detailed information about the structure and metadata of tables, schemas, and catalogs	Not supported	Supported	Supported	`describe schema.table`

Functions

PPL function	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called “String” (CONCAT, CONCAT_WS , LENGTH, LOWER, LTRIM, POSITION, REVERSE, RIGHT, RTRIM, SUBSTRING , TRIM, UPPER)	Built-in functions in PPL that can manipulate and transform string and text data within PPL queries. For example, converting case, combining strings,	Supported	Supported	Supported	`eval col1Len = LENGTH(col1) | fields col1Len`

PPL function	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
	extracting parts, and cleaning text.				

PPL function	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called "Date and time" (DAY, DAYOFMONTH , DAY_OF_MONTH ,DAYOFWEEK , DAY_OF_WEEK , DAYOFYEAR , DAY_OF_YEAR , DAYNAME, FROM_UNIXTIME , HOUR, HOUR_OF_DAY , LAST_DAY, LOCALTIMESTAMP , LOCALTIME , MAKE_DATE , MINUTE, MINUTE_OF_HOUR , MONTH, MONTHNAME , MONTH_OF_YEAR , NOW, QUARTER, SECOND, SECOND_OF_MINUTE , SUBDATE, SYSDATE, TIMESTAMP , UNIX_TIMESTAMP , WEEK, WEEKDAY, WEEK_OF_Y	Built-in functions for handling and transforming date and timestamp data in PPL queries. For example, **date_add**, **date_format**, **datediff**, and **current_date**.	Supported	Supported	Supported	`eval newDate = ADDDATE(DATE('2020-08-26'), 1) | fields newDate`

PPL function	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
EAR , DATE_ADD, DATE_SUB, TIMESTAMP ADD , TIMESTAMP DIFF , UTC_TIMESTAMP , CURRENT_TIMEZONE)					
the section called "Condition" (EXISTS, IF, IFNULL, ISNOTNULL , ISNULL, NULLIF)	Built-in functions that perform calculations on multiple rows to produce a single summarized value. For example, **sum**, **count**, **avg**, **max**, and **min**.	Supported	Supported	Supported	`eval field2 = isnull(col1) | fields field2, col1, field3`

PPL function	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called "Math" (ABS, ACOS, ASIN, ATAN, ATAN2, CEIL, CEILING, CONV, COS, COT, CRC32, DEGREES, E, EXP, FLOOR, LN, LOG, LOG2, LOG10, MOD, PI. POW, POWER, RADIANS, RAND, ROUND, SIGN, SIN, SQRT, CBRT)	Built-in functions for performing mathematical calculations and transformations in PPL queries. For example: **abs** (absolute value), **round** (rounds numbers), **sqrt** (square root), **pow** (power calculation), and **ceil** (rounds up to nearest integer).	Supported	Supported	Supported	`eval field2 = ACOS(col1) | fields col1`

PPL function	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called "Expressions" (Arithmetic operators (+, -, *), Predicate operators (>. <, IN))	Built-in functions for expressions, particularly value expressions, return a scalar value. Expressions have different types and forms.	Supported	Supported	Supported	`where age > (25 + 5) | fields age`
the section called "IP address" (CIDRMATCH)	Built-in functions for handling IP addresses such as CIDR.	Not supported	Supported	Supported	`where cidrmatch (ip, '********** ***/24') | fields ip`

PPL function	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called "JSON" (ARRAY_LENGTH , ARRAY_LENGTH , JSON, JSON_ARRAY , JSON_EXTRACT , JSON_KEYS , JSON_OBJECT , JSON_VALID , TO_JSON_STRING)	Built-in functions for handling JSON including arrays, extracting, and validation.	Not supported	Supported	Supported	`eval `json_extract('{"a":"b"}', '$.a')` = json_extract('{"a":"b"}', '$a')`
the section called "Lambda" (EXISTS, FILTER, REDUCE, TRANSFORM)	Built-in functions for handling JSON including arrays, extracting, and validation.	Not supported	Supported	Supported	`eval array = json_array(1, -1, 2), result = filter(array, x -> x > 0) | fields result`

PPL function	Description	CloudWatch Logs	Amazon S3	Security Lake	Example command
the section called "Cryptographic" (MD5, SHA1, SHA2)	Built-in functions that allow you to generate unique fingerprints of data, which can be used for verification, comparison, or as part of more complex security protocols.	Supported	Supported	Supported	`eval` `MD5('hello')` = MD5('hello') | fields `MD5('hello')`

Additional information for CloudWatch Logs Insights users using OpenSearch PPL

Although CloudWatch Logs Insights supports most OpenSearch PPL commands and functions, some commands and functions aren't currently supported. For example, it doesn't currently support JOIN, Lookup, or sub-queries in PPL. For a complete list of supported query commands and functions, see the Amazon CloudWatch Logs columns in the above tables.

Sample queries and quotas

The following applies to both CloudWatch Logs Insights users and OpenSearch users querying CloudWatch data.

For information about the limits that apply when querying CloudWatch Logs from OpenSearch Service, see CloudWatch Logs quotas in the Amazon CloudWatch Logs User Guide. Limits involve the number of CloudWatch Log groups you can query, the maximum concurrent queries that you can execute, the maximum query execution time, and the maximum number of rows returned in results. The limits are the same regardless of which language you use for querying CloudWatch Logs (namely, OpenSearch PPL, SQL, and Logs Insights QL).

PPL commands

Topics

- top command
- trendline command
- where command
- field summary
- expand command
- PPL functions

comment

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

PPL supports both line comments and block comments. The system doesn't evaluate comment text.

Line comments

Line comments begin with two slashes // and end with a new line.

Example:

```
os> source=accounts | top gender // finds most common gender of all the accounts
fetched rows / total rows = 2/2
+----------+
| gender   |
|----------|
| M        |
| F        |
+----------+
```

Block Comments

Block comments begin with a slash followed by an asterisk *, and end with an asterisk followed by a slash */.

Example:

```
os> source=accounts | dedup 2 gender /* dedup the document with gender field keep 2
 duplication */ | fields account_number, gender
fetched rows / total rows = 3/3
+------------------+----------+
| account_number   | gender   |
|------------------+----------|
| 1                | M        |
| 6                | M        |
| 13               | F        |
+------------------+----------+
```

correlation command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

You can correlate different data sources according to common dimensions and timeframes.

This correlation is crucial when you're dealing with large amounts of data from various verticals that share the same time periods but aren't formally synchronized.

By correlating these different data sources based on timeframes and similar dimensions, you can enrich your data and uncover valuable insights.

Example

The observability domain has three distinct data sources:

- Logs
- Metrics
- Traces

These data sources might share common dimensions. To transition from one data source to another, you need to correlate them correctly. Using semantic naming conventions, you can identify shared elements across logs, traces, and metrics.

Example:

```
{
  "@timestamp": "2018-07-02T22:23:00.186Z",
  "aws": {
    "elb": {
      "backend": {
        "http": {
          "response": {
            "status_code": 500
          }
        },
        "ip": "********",
        "port": "80"
      },
      ...
     "target_port": [
        "10.0.0.1:80"
      ],
      "target_status_code": [
        "500"
      ],
      "traceId": "Root=1-58337262-36d228ad5d99923122bbe354",
      "type": "http"
    }
  },
  "cloud": {
    "provider": "aws"
  },
  "http": {
    "request": {
    ...
  },
  "communication": {
    "source": {
      "address": "**************",
      "ip": "**************",
      "port": 2817
    }
  },
  "traceId": "Root=1-58337262-36d228ad5d99923122bbe354"
}
```


This example shows an AWS ELB log arriving from a service residing on AWS. It shows a backend HTTP response with a status code of 500, indicating an error. This could trigger an alert or be part

of your regular monitoring process. Your next step is to gather relevant data around this event for a thorough investigation.

While you might be tempted to query all data related to the timeframe, this approach can be overwhelming. You could end up with too much information, spending more time filtering out irrelevant data than identifying the root cause.

Instead, you can use a more targeted approach by correlating data from different sources. You can use these dimensions for correlation:

- **IP** - `"ip": "10.0.0.1" | "ip": "***************"`
- **Port** - `"port": 2817 | "target_port": "10.0.0.1:80"`

Assuming you have access to additional traces and metrics indices, and you're familiar with your schema structure, you can create a more precise correlation query.

Here's an example of a trace index document containing HTTP information you might want to correlate:

```
{
  "traceId": "c1d985bd02e1dbb85b444011f19a1ecc",
  "spanId": "55a698828fe06a42",
  "traceState": [],
  "parentSpanId": "",
  "name": "mysql",
  "kind": "CLIENT",
  "@timestamp": "2021-11-13T20:20:39+00:00",
  "events": [
    {
      "@timestamp": "2021-03-25T17:21:03+00:00",
       ...
    }
  ],
  "links": [
    {
      "traceId": "c1d985bd02e1dbb85b444011f19a1ecc",
      "spanId": "55a698828fe06a42w2",
      },
      "droppedAttributesCount": 0
    }
  ],
  "resource": {
```

```
    "service@name": "database",
    "telemetry@sdk@name": "opentelemetry",
    "host@hostname": "ip-172-31-10-8.us-west-2.compute.internal"
  },
  "status": {
    ...
  },
  "attributes": {
    "http": {
      "user_agent": {
        "original": "Mozilla/5.0"
      },
      "network": {
         ...
        }
      },
      "request": {
         ...
        }
      },
      "response": {
        "status_code": "200",
        "body": {
          "size": 500
        }
      },
      "client": {
        "server": {
          "socket": {
            "address": "***********",
            "domain": "example.com",
            "port": 80
          },
          "address": "***********",
          "port": 80
        },
        "resend_count": 0,
        "url": {
          "full": "http://example.com"
        }
      },
      "server": {
        "route": "/index",
        "address": "***********",
```

```
        "port": 8080,
        "socket": {
         ...
        },
        "client": {
         ...
         }
        },
        "url": {
         ...
        }
      }
    }
  }
}
```

In this approach you can see the `traceId` and the http's client/server `ip` that can be correlated with the elb logs to better understand the system's behaviour and condition.

New correlation query command

Here is the new command that would allow this type of investigation:

```
source alb_logs, traces | where alb_logs.ip="10.0.0.1" AND
 alb_logs.cloud.provider="aws"|
correlate exact fields(traceId, ip) scope(@timestamp, 1D) mapping(alb_logs.ip =
 traces.attributes.http.server.address, alb_logs.traceId = traces.traceId )
```

Here's what each part of the command does:

1. `source alb_logs, traces` - This selects the data sources that you want to correlate.
2. `where ip="10.0.0.1" AND cloud.provider="aws"` - This narrows down the scope of your search.
3. `correlate exact fields(traceId, ip)` - This tells the system to correlate data based on exact matches of the following fields:
   - The `ip` field has an explicit filter condition, so it will be used in the correlation for all data sources.
   - The `traceId` field has no explicit filter, so it will match the same traceIds across all data sources.

The field names indicate the logical meaning of the function within the correlation command. The actual join condition relies on the mapping statement you provide.

The term `exact` means that the correlation statements will require all fields to match in order to fulfill the query statement.

The term `approximate` will attempt to match on a best case scenario and will not reject rows with partial matches.

Addressing different field mapping

In cases where the same logical field (such as `ip`) has different names across your data sources, you need to provide the explicit mapping of path fields. To address this, you can extend your correlation conditions to match different field names with similar logical meanings. Here's how you might do this:

```
alb_logs.ip = traces.attributes.http.server.address, alb_logs.traceId = traces.traceId
```

For each field participating in the correlation join, you should provide a relevant mapping statement that includes all tables to be joined by this correlation command.

Example

In this example, there are 2 sources: `alb_logs, traces`

There are 2 fields: `traceId, ip`

There are 2 mapping statements: `alb_logs.ip = traces.attributes.http.server.address, alb_logs.traceId = traces.traceId`

Scoping the correlation timeframes

To simplify the work done by the execution engine (driver), you can add the scope statement. This explicitly directs the join query on the time it should scope for this search.

`scope(@timestamp, 1D)` i

In this example, the search scope focuses on a daily basis, so correlations appearing on the same day are grouped together. This scoping mechanism simplifies and allows better control over results, enabling incremental search resolution based on your needs.

Supporting drivers

The new correlation command is actually a 'hidden' join command. Therefore, only the following PPL drivers support this command. In these drivers, the correlation command will be directly translated into the appropriate Catalyst Join logical plan.

Example

```
source alb_logs, traces, metrics | where ip="10.0.0.1" AND
cloud.provider="aws"| correlate exact on (ip, port) scope(@timestamp,
2018-07-02T22:23:00, 1 D)
```

Logical Plan:

```
'Project [*]
+- 'Join Inner, ('ip && 'port)
   :- 'Filter (('ip === "10.0.0.1" & 'cloud.provider === "aws") &
 inTimeScope('@timestamp, "2018-07-02T22:23:00", "1 D"))
      +- 'UnresolvedRelation [alb_logs]
   +- 'Join Inner, ('ip & 'port)
      :- 'Filter (('ip === "10.0.0.1" & 'cloud.provider === "aws") &
 inTimeScope('@timestamp, "2018-07-02T22:23:00", "1 D"))
         +- 'UnresolvedRelation [traces]
      +- 'Filter (('ip === "10.0.0.1" & 'cloud.provider === "aws") &
 inTimeScope('@timestamp, "2018-07-02T22:23:00", "1 D"))
         +- 'UnresolvedRelation [metrics]
```

The catalyst engine optimizes this query according to the most efficient join ordering.

dedup command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

Use the dedup command to remove identical documents from your search results based on specified fields.

Syntax

Use the following syntax:

```
dedup [int] <field-list> [keepempty=<bool>] [consecutive=<bool>]
```

`int`

- Optional.
- The dedup command retains multiple events for each combination when you specify <int>. The number for <int> must be greater than 0. If you don't specify a number, only the first occurring event is kept. All other duplicates are removed from the results.
- Default: 1

`keepempty`

- Optional.
- If true, keeps documents where any field in the field-list has a NULL value or is MISSING.
- Default: false

`consecutive`

- Optional.
- If true, removes only events with consecutive duplicate combinations of values.
- Default: false

`field-list`

- Mandatory.
- A comma-delimited list of fields. At least one field is required.

Example 1: Dedup by one field

This example shows how to dedup documents using the gender field.

PPL query:

```
os> source=accounts | dedup gender | fields account_number, gender;
fetched rows / total rows = 2/2
```

```
+------------------+----------+
| account_number   | gender   |
|------------------+----------|
| 1                | M        |
| 13               | F        |
+------------------+----------+
```

Example 2: Keep 2 duplicates documents

The example shows how to dedup documents with the gender field, keeping two duplicates.

PPL query:

```
os> source=accounts | dedup 2 gender | fields account_number, gender;
fetched rows / total rows = 3/3
+------------------+----------+
| account_number   | gender   |
|------------------+----------|
| 1                | M        |
| 6                | M        |
| 13               | F        |
+------------------+----------+
```

Example 3: Keep or ignore the empty field by default

The example shows how to dedup the document by keeping the null value field.

PPL query:

```
os> source=accounts | dedup email keepempty=true | fields account_number, email;
fetched rows / total rows = 4/4
+------------------+-----------------------+
| account_number   | email                 |
+------------------+-----------------------+
| 1                | john_doe@example.com  |
| 6                | jane_doe@example.com  |
| 13               | null                  |
| 18               | juan_li@example.com   |
+------------------+-----------------------+
```

The example shows how to dedup the document by ignoring the empty value field.

PPL query:

```
os> source=accounts | dedup email | fields account_number, email;
fetched rows / total rows = 3/3
+------------------+-----------------------+
| account_number   | email                 |
+------------------+-----------------------+
| 1                | john_doe@example.com  |
| 6                | jane_doe@example.com  |
| 18               | juan_li@example.com   |
+------------------+-----------------------+
```

Example 4: Dedup in consecutive documents

The example shows how to dedup in consecutive documents.

PPL query:

```
os> source=accounts | dedup gender consecutive=true | fields account_number, gender;
fetched rows / total rows = 3/3
+------------------+----------+
| account_number   | gender   |
+------------------+----------+
| 1                | M        |
| 13               | F        |
| 18               | M        |
+------------------+----------+
```

Additional examples

- `source = table | dedup a | fields a,b,c`
- `source = table | dedup a,b | fields a,b,c`
- `source = table | dedup a keepempty=true | fields a,b,c`
- `source = table | dedup a,b keepempty=true | fields a,b,c`
- `source = table | dedup 1 a | fields a,b,c`
- `source = table | dedup 1 a,b | fields a,b,c`
- `source = table | dedup 1 a keepempty=true | fields a,b,c`
- `source = table | dedup 1 a,b keepempty=true | fields a,b,c`
- `source = table | dedup 2 a | fields a,b,c`
- `source = table | dedup 2 a,b | fields a,b,c`

- `source = table | dedup 2 a keepempty=true | fields a,b,c`
- `source = table | dedup 2 a,b keepempty=true | fields a,b,c`
- `source = table | dedup 1 a consecutive=true| fields a,b,c` (consecutive deduplication is unsupported)

Limitation

- For `| dedup 2 a, b keepempty=false`

```
DataFrameDropColumns('_row_number_)
+- Filter ('_row_number_ <= 2) // allowed duplication = 2
   +- Window [row_number() windowspecdefinition('a, 'b, 'a ASC NULLS FIRST, 'b ASC
 NULLS FIRST, specifiedwindowframe(RowFrame, unboundedpreceding$(), currentrow$()))
 AS _row_number_], ['a, 'b], ['a ASC NULLS FIRST, 'b ASC NULLS FIRST]
      +- Filter (isnotnull('a) AND isnotnull('b)) // keepempty=false
         +- Project
            +- UnresolvedRelation
```

- For `| dedup 2 a, b keepempty=true`

```
Union
:- DataFrameDropColumns('_row_number_)
:  +- Filter ('_row_number_ <= 2)
:     +- Window [row_number() windowspecdefinition('a, 'b, 'a ASC NULLS FIRST, 'b ASC
 NULLS FIRST, specifiedwindowframe(RowFrame, unboundedpreceding$(), currentrow$()))
 AS _row_number_], ['a, 'b], ['a ASC NULLS FIRST, 'b ASC NULLS FIRST]
:        +- Filter (isnotnull('a) AND isnotnull('b))
:           +- Project
:              +- UnresolvedRelation
+- Filter (isnull('a) OR isnull('b))
   +- Project
      +- UnresolvedRelation
```

describe command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

Use the `describe` command to get detailed information about the structure and metadata of tables, schemas, and catalogs. Here are various examples and use cases of the `describe` command.

Describe

- `describe table` This command is equal to the `DESCRIBE EXTENDED table` SQL command
- `describe schema.table`
- `describe schema.\`table\``
- `describe catalog.schema.table`
- `describe catalog.schema.\`table\``
- `describe \`catalog\`.\`schema\`.\`table\``

eval command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called "Commands".

The `eval` command evaluates the expression and appends the result to the search result.

Syntax

Use the following syntax:

```
eval <field>=<expression> ["," <field>=<expression> ]...
```

- `field`: Mandatory. If the field name doesn't exist, a new field is added. If the field name already exists, it will be overridden.
- `expression`: Mandatory. Any expression supported by the system.

Example 1: Create the new field

This example shows how to create a new `doubleAge` field for each document. The new `doubleAge` is the evaluation result of age multiplied by 2.

PPL query:

```
os> source=accounts | eval doubleAge = age * 2 | fields age, doubleAge ;
fetched rows / total rows = 4/4
+-------+-------------+
| age   | doubleAge   |
|-------+-------------|
| 32    | 64          |
| 36    | 72          |
| 28    | 56          |
| 33    | 66          |
+-------+-------------+
```

Example 2: Override the existing field

This example shows how to override the existing age field with age plus 1.

PPL query:

```
os> source=accounts | eval age = age + 1 | fields age ;
fetched rows / total rows = 4/4
+-------+
| age   |
|-------|
| 33    |
| 37    |
| 29    |
| 34    |
+-------+
```

Example 3: Create the new field with field defined in eval

This example shows how to create a new ddAge field with a field defined in the eval command. The new field ddAge is the evaluation result of doubleAge multiplied by 2, where doubleAge is defined in the eval command.

PPL query:

```
os> source=accounts | eval doubleAge = age * 2, ddAge = doubleAge * 2 | fields age,
 doubleAge, ddAge ;
fetched rows / total rows = 4/4
+-------+-------------+---------+
```

```
| age   | doubleAge   | ddAge   |
|-------+-------------+---------|
| 32    | 64          | 128     |
| 36    | 72          | 144     |
| 28    | 56          | 112     |
| 33    | 66          | 132     |
+-------+-------------+---------+
```

Assumptions: a, b, c are existing fields in `table`

Additional examples

- `source = table | eval f = 1 | fields a,b,c,f`
- `source = table | eval f = 1` (output a,b,c,f fields)
- `source = table | eval n = now() | eval t = unix_timestamp(a) | fields n,t`
- `source = table | eval f = a | where f > 1 | sort f | fields a,b,c | head 5`
- `source = table | eval f = a * 2 | eval h = f * 2 | fields a,f,h`
- `source = table | eval f = a * 2, h = f * 2 | fields a,f,h`
- `source = table | eval f = a * 2, h = b | stats avg(f) by h`
- `source = table | eval f = ispresent(a)`
- `source = table | eval r = coalesce(a, b, c) | fields r`
- `source = table | eval e = isempty(a) | fields e`
- `source = table | eval e = isblank(a) | fields e`
- `source = table | eval f = case(a = 0, 'zero', a = 1, 'one', a = 2, 'two', a = 3, 'three', a = 4, 'four', a = 5, 'five', a = 6, 'six', a = 7, 'se7en', a = 8, 'eight', a = 9, 'nine')`
- `source = table | eval f = case(a = 0, 'zero', a = 1, 'one' else 'unknown')`
- `source = table | eval f = case(a = 0, 'zero', a = 1, 'one' else concat(a, ' is an incorrect binary digit'))`
- `source = table | eval f = a in ('foo', 'bar') | fields f`
- `source = table | eval f = a not in ('foo', 'bar') | fields f`

Eval with case example:

```
source = table | eval e = eval status_category =
case(a >= 200 AND a < 300, 'Success',
a >= 300 AND a < 400, 'Redirection',
a >= 400 AND a < 500, 'Client Error',
a >= 500, 'Server Error'
else 'Unknown')
```

Eval with another case example:

Assumptions: a, b, c are existing fields in `table`

Additional examples

- `source = table | eval f = 1 | fields a,b,c,f`
- `source = table | eval f = 1` (output a,b,c,f fields)
- `source = table | eval n = now() | eval t = unix_timestamp(a) | fields n,t`
- `source = table | eval f = a | where f > 1 | sort f | fields a,b,c | head 5`
- `source = table | eval f = a * 2 | eval h = f * 2 | fields a,f,h`
- `source = table | eval f = a * 2, h = f * 2 | fields a,f,h`
- `source = table | eval f = a * 2, h = b | stats avg(f) by h`
- `source = table | eval f = ispresent(a)`
- `source = table | eval r = coalesce(a, b, c) | fields r`
- `source = table | eval e = isempty(a) | fields e`
- `source = table | eval e = isblank(a) | fields e`
- `source = table | eval f = case(a = 0, 'zero', a = 1, 'one', a = 2, 'two', a = 3, 'three', a = 4, 'four', a = 5, 'five', a = 6, 'six', a = 7, 'se7en', a = 8, 'eight', a = 9, 'nine')`
- `source = table | eval f = case(a = 0, 'zero', a = 1, 'one' else 'unknown')`
- `source = table | eval f = case(a = 0, 'zero', a = 1, 'one' else concat(a, ' is an incorrect binary digit'))`
- `source = table | eval f = a in ('foo', 'bar') | fields f`
- `source = table | eval f = a not in ('foo', 'bar') | fields f`

Eval with case example:

```
source = table | eval e = eval status_category =
case(a >= 200 AND a < 300, 'Success',
a >= 300 AND a < 400, 'Redirection',
a >= 400 AND a < 500, 'Client Error',
a >= 500, 'Server Error'
else 'Unknown')
```

Eval with another case example:

```
source = table |  where ispresent(a) |
eval status_category =
 case(a >= 200 AND a < 300, 'Success',
  a >= 300 AND a < 400, 'Redirection',
  a >= 400 AND a < 500, 'Client Error',
  a >= 500, 'Server Error'
  else 'Incorrect HTTP status code'
 )
 | stats count() by status_category
```

Limitations

- Overriding existing fields is unsupported. Queries attempting to do so will throw exceptions with the message "Reference 'a' is ambiguous".

```
- `source = table | eval a = 10 | fields a,b,c`
- `source = table | eval a = a * 2 | stats avg(a)`
- `source = table | eval a = abs(a) | where a > 0`
- `source = table | eval a = signum(a) | where a < 0`
```

eventstats command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

Use the `eventstats` command to enrich your event data with calculated summary statistics. It operates by analyzing specified fields within your events, computing various statistical measures, and then appending these results as new fields to each original event.

Key aspects of eventstats

1. It performs calculations across the entire result set or within defined groups.
2. The original events remain intact, with new fields added to contain the statistical results.
3. The command is particularly useful for comparative analysis, identifying outliers, or providing additional context to individual events.

Difference between stats and eventstats

The `stats` and `eventstats` commands are both used for calculating statistics, but they have some key differences in how they operate and what they produce.

Output format

- `stats`: Produces a summary table with only the calculated statistics.
- `eventstats`: Adds the calculated statistics as new fields to the existing events, preserving the original data.

Event retention

- `stats`: Reduces the result set to only the statistical summary, discarding individual events.
- `eventstats`: Retains all original events and adds new fields with the calculated statistics.

Use cases

- `stats`: Best for creating summary reports or dashboards. Often used as a final command to summarize results.
- `eventstats`: Useful when you need to enrich events with statistical context for further analysis or filtering. Can be used mid-search to add statistics that can be used in subsequent commands.

Syntax

Use the following syntax:

```
eventstats <aggregation>... [by-clause]
```

aggregation

- Mandatory.
- An aggregation function.
- The argument of aggregation must be a field.

by-clause

- Optional.
- Syntax: `by [span-expression,] [field,]...`
- The by clause can include fields and expressions such as scalar functions and aggregation functions. You can also use the span clause to split a specific field into buckets of equal intervals. The eventstats command then performs aggregation based on these span buckets.
- Default: If you don't specify a by clause, the eventstats command aggregates over the entire result set.

span-expression

- Optional, at most one.
- Syntax: `span(field_expr, interval_expr)`
- The unit of the interval expression is the natural unit by default. However, for date and time type fields, you need to specify the unit in the interval expression when using date/time units.

  For example, to split the field `age` into buckets by 10 years, use `span(age, 10)`. For time-based fields, you can split a `timestamp` field into hourly intervals using `span(timestamp, 1h)`.

Available time units

Span interval units
millisecond (ms)
second (s)

Span interval units
minute (m, case sensitive)
hour (h)
day (d)
week (w)
month (M, case sensitive)
quarter (q)
year (y)

Aggregation functions

COUNT

COUNT returns a count of the number of expr in the rows retrieved by a SELECT statement.

For CloudWatch Logs use queries, COUNT is not supported.

Example:

```
os> source=accounts | eventstats count();
fetched rows / total rows = 4/4
+----------------+---------+-----------+----------+-----+--------+-------------------
+------------+--------------------------+--------+-------+---------+
| account_number | balance  | firstname | lastname | age | gender | address
 | employer   | email                     | city   | state | count() |
+----------------+---------+-----------+----------+-----+--------+-------------------
+------------+--------------------------+--------+-------+---------+
| 1              | 39225    | Jane      | Doe      | 32  | M      | *** Any Lane
 | AnyCorp    | janedoe@anycorp.com       | Brogan | IL    | 4       |
| 6              | 5686     | Mary      | Major    | 36  | M      | 671 Example Street
 | AnyCompany | marymajor@anycompany.com | Dante  | TN    | 4       |
| 13             | 32838    | Nikki     | Wolf     | 28  | F      | 789 Any Street
 | AnyOrg     |                           | Nogal  | VA    | 4       |
| 18             | 4180     | Juan      | Li       | 33  | M      | *** Example Court
 |            | juanli@exampleorg.com     | Orick  | MD    | 4       |
```

```
+----------------+----------+-----------+----------+-----+--------+--------------------
+------------+-------------------------+--------+-------+---------+
```

SUM

SUM(expr) returns the sum of expr.

Example:

```
os> source=accounts | eventstats sum(age) by gender;
fetched rows / total rows = 4/4
+----------------+----------+-----------+----------+-----+--------
+-----------------------+------------+--------------------------+--------+-------
+--------------------+
| account_number | balance  | firstname | lastname | age | gender | address
   | employer   | email                    | city   | state | sum(age) by gender |
+----------------+----------+-----------+----------+-----+--------
+-----------------------+------------+--------------------------+--------+-------
+--------------------+
| 1              | 39225    | Jane      | Doe      | 32  | M      | 880 Any Lane
   | AnyCorp    | janedoe@anycorp.com      | Brogan | IL    | 101               |
| 6              | 5686     | Mary      | Major    | 36  | M      | 671 Example Street
   | AnyCompany | marymajor@anycompany.com | Dante  | TN    | 101               |
| 13             | 32838    | Nikki     | Wolf     | 28  | F      | 789 Any Street
   | AnyOrg     |                          | Nogal  | VA    | 28                |
| 18             | 4180     | Juan      | Li       | 33  | M      | 467 Example Court
   |            | juanli@exampleorg.com    | Orick  | MD    | 101               |
+----------------+----------+-----------+----------+-----+--------
+-----------------------+------------+--------------------------+--------+-------
+--------------------+
```

AVG

AVG(expr) returns the average value of expr.

Example:

```
os> source=accounts | eventstats avg(age) by gender;
fetched rows / total rows = 4/4
+----------------+----------+-----------+----------+-----+--------
+-----------------------+------------+--------------------------+--------+-------
+--------------------+
```

```
| account_number | balance  | firstname | lastname | age | gender | address
   | employer    | email                    | city   | state | avg(age) by gender |
+----------------+----------+-----------+----------+-----+--------
+----------------------+------------+--------------------------+--------+-------
+--------------------+
| 1              | 39225    | Jane      | Doe      | 32  | M      | 880 Any Lane
   | AnyCorp     | janedoe@anycorp.com      | Brogan | IL    | 33.67              |
| 6              | 5686     | Mary      | Major    | 36  | M      | 671 Example Street
   | Any Company | marymajor@anycompany.com | Dante  | TN    | 33.67              |
| 13             | 32838    | Nikki     | Wolf     | 28  | F      | 789 Any Street
   | AnyOrg      |                          | Nogal  | VA    | 28.00              |
| 18             | 4180     | Juan      | Li       | 33  | M      | 467 Example Court
   |             | juanli@exampleorg.com    | Orick  | MD    | 33.67              |
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+--------------------+
```

MAX

`MAX(expr)` Returns the maximum value of expr.

Example

```
os> source=accounts | eventstats max(age);
fetched rows / total rows = 4/4
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+-----------+
| account_number | balance  | firstname | lastname | age | gender | address
   | employer    | email                    | city   | state | max(age)  |
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+-----------+
| 1              | 39225    | Jane      | Doe      | 32  | M      | 880 Any Lane
   | AnyCorp     | janedoe@anycorp.com      | Brogan | IL    | 36        |
| 6              | 5686     | Mary      | Major    | 36  | M      | 671 Example Street
   | Any Company | marymajor@anycompany.com | Dante  | TN    | 36        |
| 13             | 32838    | Nikki     | Wolf     | 28  | F      | 789 Any Street
   | AnyOrg      |                          | Nogal  | VA    | 36        |
| 18             | 4180     | Juan      | Li       | 33  | M      | *** Example Court
   |             | juanli@exampleorg.com    | Orick  | MD    | 36        |
```

```
+---------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+-----------+
```

MIN

`MIN(expr)` Returns the minimum value of expr.

Example

```
os> source=accounts | eventstats min(age);
fetched rows / total rows = 4/4
+---------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+-----------+
| account_number | balance  | firstname | lastname | age | gender | address
   | employer    | email                    | city   | state | min(age)  |
+---------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+-----------+
| 1              | 39225    | Jane      | Doe      | 32  | M      | 880 Any Lane
   | AnyCorp     | janedoe@anycorp.com      | Brogan | IL    | 28        |
| 6              | 5686     | Mary      | Major    | 36  | M      | 671 Example Street
   | Any Company | marymajor@anycompany.com | Dante  | TN    | 28        |
| 13             | 32838    | Nikki     | Wolf     | 28  | F      | *** Any Street
   | AnyOrg      |                          | Nogal  | VA    | 28        |
| 18             | 4180     | Juan      | Li       | 33  | M      | *** Example Court
   |             | juanli@exampleorg.com    | Orick  | MD    | 28        |
+---------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+-----------+
```

STDDEV_SAMP

`STDDEV_SAMP(expr)` Return the sample standard deviation of expr.

Example

```
os> source=accounts | eventstats stddev_samp(age);
fetched rows / total rows = 4/4
+---------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+------------------------+
```

```
| account_number | balance  | firstname | lastname | age | gender | address
   | employer    | email                    | city   | state | stddev_samp(age)       |
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+------------------------+
| 1              | 39225    | Jane      | Doe      | 32  | M      | *** Any Lane
   | AnyCorp     | janedoe@anycorp.com      | Brogan | IL    | 3.304037933599835      |
| 6              | 5686     | Mary      | Major    | 36  | M      | 671 Example Street
   | Any Company | marymajor@anycompany.com | Dante  | TN    | 3.304037933599835      |
| 13             | 32838    | Nikki     | Wolf     | 28  | F      | 789 Any Street
   | AnyOrg      |                          | Nogal  | VA    | 3.304037933599835      |
| 18             | 4180     | Juan      | Li       | 33  | M      | 467 Example Court
   |             | juanli@exampleorg.com    | Orick  | MD    | 3.304037933599835      |
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+------------------------+
```

STDDEV_POP

`STDDEV_POP(expr)` Return the population standard deviation of expr.

Example

```
os> source=accounts | eventstats stddev_pop(age);
fetched rows / total rows = 4/4
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+------------------------+
| account_number | balance  | firstname | lastname | age | gender | address
   | employer    | email                    | city   | state | stddev_pop(age)        |
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+------------------------+
| 1              | 39225    | Jane      | Doe      | 32  | M      | 880 Any Lane
   | AnyCorp     | janedoe@anycorp.com      | Brogan | IL    | 2.****************     |
| 6              | 5686     | Mary      | Major    | 36  | M      | *** Example Street
   | Any Company | marymajor@anycompany.com | Dante  | TN    | 2.****************     |
| 13             | 32838    | Nikki     | Wolf     | 28  | F      | *** Any Street
   | AnyOrg      |                          | Nogal  | VA    | 2.****************     |
| 18             | 4180     | Juan      | Li       | 33  | M      | *** Example Court
   |             | juanli@exampleorg.com    | Orick  | MD    | 2.****************     |
```

```
+---------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+------------------------+
```

PERCENTILE or PERCENTILE_APPROX

`PERCENTILE(expr, percent)` or `PERCENTILE_APPROX(expr, percent)` Return the approximate percentile value of expr at the specified percentage.

percent

- The number must be a constant between 0 and 100.

Example

```
os> source=accounts | eventstats percentile(age, 90) by gender;
fetched rows / total rows = 4/4
+---------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+--------------------------------+
| account_number | balance  | firstname | lastname | age | gender | address
    | employer    | email                    | city   | state | percentile(age, 90) by
 gender  |
+---------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+--------------------------------+
| 1              | 39225    | Jane      | Doe      | 32  | M      | *** Any Lane
   | AnyCorp     | janedoe@anycorp.com      | Brogan | IL    | 36
       |
| 6              | 5686     | Mary      | Major    | 36  | M      | 671 Example Street
   | Any Company | marymajor@anycompany.com | Dante  | TN    | 36
       |
| 13             | 32838    | Nikki     | Wolf     | 28  | F      | 789 Any Street
   | AnyOrg      |                          | Nogal  | VA    | 28
       |
| 18             | 4180     | Juan      | Li       | 33  | M      | *** Example Court
   |             | juanli@exampleorg.com    | Orick  | MD    | 36
       |
+---------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+--------------------------------+
```

Example 1: Calculate the average, sum and count of a field by group

The example show calculate the average age, sum age and count of events of all the accounts group by gender.

```
os> source=accounts | eventstats avg(age) as avg_age, sum(age) as sum_age, count() as
 count by gender;
fetched rows / total rows = 4/4
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+-----------+-----------+-------+
| account_number | balance  | firstname | lastname | age | gender | address
    | employer    | email                    | city   | state | avg_age   | sum_age   |
 count |
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+-----------+-----------+-------+
| 1              | 39225    | Jane      | Doe      | 32  | M      | *** Any Lane
    | AnyCorp     | janedoe@anycorp.com      | Brogan | IL    | 33.666667 | 101       |
 3     |
| 6              | 5686     | Mary      | Major    | 36  | M      | 671 Example Street
    | Any Company | marymajor@anycompany.com | Dante  | TN    | 33.666667 | 101       |
 3     |
| 13             | 32838    | Nikki     | Wolf     | 28  | F      | 789 Any Street
    | AnyOrg      |                          | Nogal  | VA    | 28.000000 | 28        |
 1     |
| 18             | 4180     | Juan      | Li       | 33  | M      | *** Example Court
    |             | juanli@exampleorg.com    | Orick  | MD    | 33.666667 | 101       |
 3     |
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+-----------+-----------+-------+
```

Example 2: Calculate the count by a span

The example gets the count of age by the interval of 10 years.

```
os> source=accounts | eventstats count(age) by span(age, 10) as age_span
fetched rows / total rows = 4/4
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+----------+
```

```
| account_number | balance  | firstname | lastname | age | gender | address
   | employer    | email                    | city   | state | age_span |
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+-----------+
| 1              | 39225    | Jane      | Doe      | 32  | M      | *** Any Lane
   | AnyCorp     | janedoe@anycorp.com      | Brogan | IL    | 3        |
| 6              | 5686     | Mary      | Major    | 36  | M      | 671 Example Street
   | Any Company | marymajor@anycompany.com | Dante  | TN    | 3        |
| 13             | 32838    | Nikki     | Wolf     | 28  | F      | 789 Any Street
   | AnyOrg      |                          | Nogal  | VA    | 1        |
| 18             | 4180     | Juan      | Li       | 33  | M      | *** Example Court
   |             | juanli@exampleorg.com    | Orick  | MD    | 3        |
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+-----------+
```

Example 3: Calculate the count by a gender and span

The example gets the count of age by the interval of 5 years and group by gender.

```
os> source=accounts | eventstats count() as cnt by span(age, 5) as age_span, gender
fetched rows / total rows = 4/4
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+-----+
| account_number | balance  | firstname | lastname | age | gender | address
   | employer    | email                    | city   | state | cnt |
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+-----+
| 1              | 39225    | Jane      | Doe      | 32  | M      | *** Any Lane
   | AnyCorp     | janedoe@anycorp.com      | Brogan | IL    | 2   |
| 6              | 5686     | Mary      | Majo     | 36  | M      | 671 Example Street
   | Any Company | hattiebond@anycompany.com | Dante  | TN    | 1   |
| 13             | 32838    | Nikki     | Wolf     | 28  | F      | *** Any Street
   | AnyOrg      |                          | Nogal  | VA    | 1   |
| 18             | 4180     | Juan      | Li       | 33  | M      | *** Example Court
   |             | juanli@exampleorg.com    | Orick  | MD    | 2   |
+----------------+----------+-----------+----------+-----+--------
+----------------------+-------------+--------------------------+--------+-------
+-----+
```

Usage

- `source = table | eventstats avg(a)`
- `source = table | where a < 50 | eventstats avg(c)`
- `source = table | eventstats max(c) by b`
- `source = table | eventstats count(c) by b | head 5`
- `source = table | eventstats distinct_count(c)`
- `source = table | eventstats stddev_samp(c)`
- `source = table | eventstats stddev_pop(c)`
- `source = table | eventstats percentile(c, 90)`
- `source = table | eventstats percentile_approx(c, 99)`

Aggregations with span

- `source = table | eventstats count(a) by span(a, 10) as a_span`
- `source = table | eventstats sum(age) by span(age, 5) as age_span | head 2`
- `source = table | eventstats avg(age) by span(age, 20) as age_span, country | sort - age_span | head 2`

Aggregations with time window span (tumble windowing function)

- `source = table | eventstats sum(productsAmount) by span(transactionDate, 1d) as age_date | sort age_date`
- `source = table | eventstats sum(productsAmount) by span(transactionDate, 1w) as age_date, productId`

Aggregations group by multiple levels

- `source = table | eventstats avg(age) as avg_state_age by country, state | eventstats avg(avg_state_age) as avg_country_age by country`
- `source = table | eventstats avg(age) as avg_city_age by country, state, city | eval new_avg_city_age = avg_city_age - 1 | eventstats`

```
avg(new_avg_city_age) as avg_state_age by country, state |
where avg_state_age > 18 | eventstats avg(avg_state_age) as
avg_adult_country_age by country
```

expand command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called "Commands".

Use the `expand` command to flatten a field of type:

- `Array<Any>`
- `Map<Any>`

Syntax

Use the following syntax:

```
expand <field> [As alias]
```

field

- The field to be expanded (exploded). Must be of a supported type.

alias

- Optional. The name to be used instead of the original field name.

Usage

The `expand` command produces a row for each element in the specified array or map field, where:

- Array elements become individual rows.
- Map key-value pairs are broken into separate rows, with each key-value represented as a row.

- When an alias is provided, the exploded values are represented under the alias instead of the original field name.
- This can be used in combination with other commands, such as `stats`, `eval`, and `parse` to manipulate or extract data post-expansion.

Examples

- `source = table | expand employee | stats max(salary) as max by state, company`
- `source = table | expand employee as worker | stats max(salary) as max by state, company`
- `source = table | expand employee as worker | eval bonus = salary * 3 | fields worker, bonus`
- `source = table | expand employee | parse description '(?<email>.+@.+)' | fields employee, email`
- `source = table | eval array=json_array(1, 2, 3) | expand array as uid | fields name, occupation, uid`
- `source = table | expand multi_valueA as multiA | expand multi_valueB as multiB`

explain command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

The `explain` command helps you understand query execution plans, enabling you to analyze and optimize your queries for better performance. This introduction provides a concise overview of the explain command's purpose and its importance in query optimization.

Comment

- `source=accounts | top gender // finds most common gender of all the accounts` (line comment)

- `source=accounts | dedup 2 gender /* dedup the document with gender field keep 2 duplication */ | fields account_number, gender` (block comment)

Describe

- `describe table` This command is equal to the `DESCRIBE EXTENDED table` SQL command
- `describe schema.table`
- `describe schema.\`table\``
- `describe catalog.schema.table`
- `describe catalog.schema.\`table\``
- `describe \`catalog\`.\`schema\`.\`table\``

Explain

- `explain simple | source = table | where a = 1 | fields a,b,c`
- `explain extended | source = table`
- `explain codegen | source = table | dedup a | fields a,b,c`
- `explain cost | source = table | sort a | fields a,b,c`
- `explain formatted | source = table | fields - a`
- `explain simple | describe table`

Fields

- `source = table`
- `source = table | fields a,b,c`
- `source = table | fields + a,b,c`
- `source = table | fields - b,c`
- `source = table | eval b1 = b | fields - b1,c`

Field summary

- `source = t | fieldsummary includefields=status_code nulls=false`

- `source = t | fieldsummary includefields= id, status_code, request_path nulls=true`
- `source = t | where status_code != 200 | fieldsummary includefields= status_code nulls=true`

Nested field

- `source = catalog.schema.table1, catalog.schema.table2 | fields A.nested1, B.nested1`
- `source = catalog.table | where struct_col2.field1.subfield > 'valueA' | sort int_col | fields int_col, struct_col.field1.subfield, struct_col2.field1.subfield`
- `source = catalog.schema.table | where struct_col2.field1.subfield > 'valueA' | sort int_col | fields int_col, struct_col.field1.subfield, struct_col2.field1.subfield`

Filters

- `source = table | where a = 1 | fields a,b,c`
- `source = table | where a >= 1 | fields a,b,c`
- `source = table | where a < 1 | fields a,b,c`
- `source = table | where b != 'test' | fields a,b,c`
- `source = table | where c = 'test' | fields a,b,c | head 3`
- `source = table | where ispresent(b)`
- `source = table | where isnull(coalesce(a, b)) | fields a,b,c | head 3`
- `source = table | where isempty(a)`
- `source = table | where isblank(a)`
- `source = table | where case(length(a) > 6, 'True' else 'False') = 'True'`
- `source = table | where a not in (1, 2, 3) | fields a,b,c`
- `source = table | where a between 1 and 4` - Note: This returns a >= 1 and a <= 4, i.e. [1, 4]
- `source = table | where b not between '2024-09-10' and '2025-09-10'` - Note: This returns b >= '**********' and b <= '2025-09-10'

- `source = table | where cidrmatch(ip, '***********/24')`
- `source = table | where cidrmatch(ipv6, '2003:db8::/32')`
- `source = table | trendline sma(2, temperature) as temp_trend`

IP related queries

- `source = table | where cidrmatch(ip, '**************')`
- `source = table | where isV6 = false and isValid = true and cidrmatch(ipAddress, '**************')`
- `source = table | where isV6 = true | eval inRange = case(cidrmatch(ipAddress, '2003:***::/32'), 'in' else 'out') | fields ip, inRange`

Complex filters

```
source = table | eval status_category =
case(a >= 200 AND a < 300, 'Success',
    a >= 300 AND a < 400, 'Redirection',
    a >= 400 AND a < 500, 'Client Error',
    a >= 500, 'Server Error'
else 'Incorrect HTTP status code')
| where case(a >= 200 AND a < 300, 'Success',
    a >= 300 AND a < 400, 'Redirection',
    a >= 400 AND a < 500, 'Client Error',
    a >= 500, 'Server Error'
else 'Incorrect HTTP status code'
) = 'Incorrect HTTP status code'
```

```
source = table
| eval factor = case(a > 15, a - 14, isnull(b), a - 7, a < 3, a + 1 else 1)
| where case(factor = 2, 'even', factor = 4, 'even', factor = 6, 'even', factor = 8,
 'even' else 'odd') = 'even'
| stats count() by factor
```

Filters with logical conditions

- `source = table | where c = 'test' AND a = 1 | fields a,b,c`

- `source = table | where c != 'test' OR a > 1 | fields a,b,c | head 1`
- `source = table | where c = 'test' NOT a > 1 | fields a,b,c`

Eval

Assumptions: a, b, c are existing fields in `table`

- `source = table | eval f = 1 | fields a,b,c,f`
- `source = table | eval f = 1` (output a,b,c,f fields)
- `source = table | eval n = now() | eval t = unix_timestamp(a) | fields n,t`
- `source = table | eval f = a | where f > 1 | sort f | fields a,b,c | head 5`
- `source = table | eval f = a * 2 | eval h = f * 2 | fields a,f,h`
- `source = table | eval f = a * 2, h = f * 2 | fields a,f,h`
- `source = table | eval f = a * 2, h = b | stats avg(f) by h`
- `source = table | eval f = ispresent(a)`
- `source = table | eval r = coalesce(a, b, c) | fields r`
- `source = table | eval e = isempty(a) | fields e`
- `source = table | eval e = isblank(a) | fields e`
- `source = table | eval f = case(a = 0, 'zero', a = 1, 'one', a = 2, 'two', a = 3, 'three', a = 4, 'four', a = 5, 'five', a = 6, 'six', a = 7, 'se7en', a = 8, 'eight', a = 9, 'nine')`
- `source = table | eval f = case(a = 0, 'zero', a = 1, 'one' else 'unknown')`
- `source = table | eval f = case(a = 0, 'zero', a = 1, 'one' else concat(a, ' is an incorrect binary digit'))`
- `source = table | eval digest = md5(fieldName) | fields digest`
- `source = table | eval digest = sha1(fieldName) | fields digest`
- `source = table | eval digest = sha2(fieldName,256) | fields digest`
- `source = table | eval digest = sha2(fieldName,512) | fields digest`

fillnull command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

Description

Use the `fillnull` command to replace null values with a specified value in one or more fields of your search results.

Syntax

Use the following syntax:

```
fillnull [with <null-replacement> in <nullable-field>["," <nullable-field>]] | [using
 <source-field> = <null-replacement> [","<source-field> = <null-replacement>]]
```

- null-replacement: Mandatory. The value used to replace null values.
- nullable-field: Mandatory. Field reference. The null values in this field will be replaced with the value specified in null-replacement.

Example 1: Fillnull one field

The example shows how to use fillnull on a single field:

```
os> source=logs | fields status_code | eval input=status_code | fillnull with 0 in
 status_code;
| input | status_code |
|-------|-------------|
| 403   | 403         |
| 403   | 403         |
| NULL  | 0           |
| NULL  | 0           |
| 200   | 200         |
| 404   | 404         |
| 500   | 500         |
| NULL  | 0           |
| 500   | 500         |
```

```
| 404   | 404         |
| 200   | 200         |
| 500   | 500         |
| NULL  | 0           |
| NULL  | 0           |
| 404   | 404         |
```

Example 2: Fillnull applied to multiple fields

The example shows fillnull applied to multiple fields.

```
os> source=logs | fields request_path, timestamp | eval
 input_request_path=request_path, input_timestamp = timestamp | fillnull with '???' in
 request_path, timestamp;
| input_request_path | input_timestamp       | request_path | timestamp              |
|--------------------------------------------------------------------------------------|
| /contact           | NULL                  | /contact     | ???                    |
| /home              | NULL                  | /home        | ???                    |
| /about             | 2023-10-01 10:30:00   | /about       | 2023-10-01 10:30:00    |
| /home              | 2023-10-01 10:15:00   | /home        | 2023-10-01 10:15:00    |
| NULL               | 2023-10-01 10:20:00   | ???          | 2023-10-01 10:20:00    |
| NULL               | 2023-10-01 11:05:00   | ???          | 2023-10-01 11:05:00    |
| /about             | NULL                  | /about       | ???                    |
| /home              | 2023-10-01 10:00:00   | /home        | 2023-10-01 10:00:00    |
| /contact           | NULL                  | /contact     | ???                    |
| NULL               | 2023-10-01 10:05:00   | ???          | 2023-10-01 10:05:00    |
| NULL               | 2023-10-01 10:50:00   | ???          | 2023-10-01 10:50:00    |
| /services          | NULL                  | /services    | ???                    |
| /home              | 2023-10-01 10:45:00   | /home        | 2023-10-01 10:45:00    |
| /services          | 2023-10-01 11:00:00   | /services    | 2023-10-01 11:00:00    |
| NULL               | 2023-10-01 10:35:00   | ???          | 2023-10-01 10:35:00    |
```

Example 3: Fillnull applied to multiple fields with various null replacement values.

The example show fillnull with various values used to replace nulls.

- `/error` in `request_path` field
- `1970-01-01 00:00:00` in `timestamp` field

```
os> source=logs | fields request_path, timestamp | eval
 input_request_path=request_path, input_timestamp = timestamp | fillnull using
 request_path = '/error', timestamp='1970-01-01 00:00:00';
```

```
| input_request_path | input_timestamp       | request_path | timestamp             |
|-------------------------------------------------------------------------------|
| /contact           | NULL                  | /contact     | 1970-01-01 00:00:00   |
| /home              | NULL                  | /home        | 1970-01-01 00:00:00   |
| /about             | 2023-10-01 10:30:00   | /about       | 2023-10-01 10:30:00   |
| /home              | 2023-10-01 10:15:00   | /home        | 2023-10-01 10:15:00   |
| NULL               | 2023-10-01 10:20:00   | /error       | 2023-10-01 10:20:00   |
| NULL               | 2023-10-01 11:05:00   | /error       | 2023-10-01 11:05:00   |
| /about             | NULL                  | /about       | 1970-01-01 00:00:00   |
| /home              | 2023-10-01 10:00:00   | /home        | 2023-10-01 10:00:00   |
| /contact           | NULL                  | /contact     | 1970-01-01 00:00:00   |
| NULL               | 2023-10-01 10:05:00   | /error       | 2023-10-01 10:05:00   |
| NULL               | 2023-10-01 10:50:00   | /error       | 2023-10-01 10:50:00   |
| /services          | NULL                  | /services    | 1970-01-01 00:00:00   |
| /home              | 2023-10-01 10:45:00   | /home        | 2023-10-01 10:45:00   |
| /services          | 2023-10-01 11:00:00   | /services    | 2023-10-01 11:00:00   |
| NULL               | 2023-10-01 10:35:00   | /error       | 2023-10-01 10:35:00   |
```

fields command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called "Commands".

Use the `fields` command to keep or remove fields from the search result.

Syntax

Use the following syntax:

```
field [+|-] <field-list>
```

- `index`: Optional.

  If the plus (+) is used, only the fields specified in the field list will be kept.

  If the minus (-) is used, all the fields specified in the field list will be removed.

  Default: +

- `field list`: Mandatory. A comma-delimited list of fields to keep or remove.

Example 1: Select specified fields from result

This example shows how to fetch `account_number`, `firstname`, and `lastname` fields from search results.

PPL query:

```
os> source=accounts | fields account_number, firstname, lastname;
fetched rows / total rows = 4/4
+------------------+-------------+------------+
| account_number   | firstname   | lastname   |
|------------------+-------------+------------|
| 1                | Jane        | Doe        |
| 6                | John        | Doe        |
| 13               | Jorge       | Souza      |
| 18               | Juan        | Li         |
+------------------+-------------+------------+
```

Example 2: Remove specified fields from result

This example shows how to remove the `account_number` field from search results.

PPL query:

```
os> source=accounts | fields account_number, firstname, lastname | fields -
 account_number ;
fetched rows / total rows = 4/4
+-------------+------------+
| firstname   | lastname   |
|-------------+------------|
| Jane        | Doe        |
| John        | Doe        |
| Jorge       | Souza      |
| Juan        | Li         |
+-------------+------------+
```

Additional examples

- `source = table`

- `source = table | fields a,b,c`
- `source = table | fields + a,b,c`
- `source = table | fields - b,c`
- `source = table | eval b1 = b | fields - b1,c`

Nested-fields example:

```
`source = catalog.schema.table1, catalog.schema.table2 | fields A.nested1, B.nested1`
`source = catalog.table | where struct_col2.field1.subfield > 'valueA' | sort int_col |
 fields  int_col, struct_col.field1.subfield, struct_col2.field1.subfield`
`source = catalog.schema.table | where struct_col2.field1.subfield > 'valueA' | sort
 int_col | fields  int_col, struct_col.field1.subfield, struct_col2.field1.subfield`
```

flatten command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

Use the flatten command to expand fields of the following types:

- `struct<?,?>`
- `array<struct<?,?>>`

Syntax

Use the following syntax:

```
flatten <field>
```

- *field*: The field to be flattened. The field must be of supported type.

Schema

col_name	data_type
_time	string
bridges	array<struct<length:bigint,name:string>>
city	string
coor	struct<alt:bigint,lat:double,long:double>
country	string

Data

_time	bridges	city	coor	country
2024-09-13T12:00:00	[{801, Tower Bridge}, {928, London Bridge}]	London	{35, 51.5074, -0.1278}	England
2024-09-13T12:00:00	[{232, Pont Neuf}, {160, Pont Alexandre III}]	Paris	{35, 48.8566, 2.3522}	France
2024-09-13T12:00:00	[{48, Rialto Bridge}, {11, Bridge of Sighs}]	Venice	{2, 45.4408, 12.3155}	Italy

_time	bridges	city	coor	country
2024-09-13T12:00:00	[{***, Charles Bridge}, {343, Legion Bridge}]	Prague	{200, 50.0755, 14.4378}	Czech Republic
2024-09-13T12:00:00	[{375, Chain Bridge}, {333, Liberty Bridge}]	Budapest	{96, 47.4979, 19.0402}	Hungary
1990-09-13T12:00:00	NULL	Warsaw	NULL	Poland

Example 1: flatten struct

This example shows how to flatten a struct field.

PPL query:

```
source=table | flatten coor
```

_time	bridges	city	country	alt	lat	long
2024-09-13T12:00:00	[{801, Tower Bridge}, {928, London Bridge}]	London	England	35	51.5074	-0.1278

_time	bridges	city	country	alt	lat	long
2024-09-13T12:00:00	[{232, Pont Neuf}, {160, Pont Alexandre III}]	Paris	France	35	48.8566	2.3522
2024-09-13T12:00:00	[{48, Rialto Bridge}, {11, Bridge of Sighs}]	Venice	Italy	2	45.4408	12.3155
2024-09-13T12:00:00	[{516, Charles Bridge}, {343, Legion Bridge}]	Prague	Czech Republic	200	50.0755	14.4378
2024-09-13T12:00:00	[{375, Chain Bridge}, {333, Liberty Bridge}]	Budapest	Hungary	96	47.4979	19.0402
1990-09-13T12:00:00	NULL	Warsaw	Poland	NULL	NULL	NULL

Example 2: flatten array

The example shows how to flatten an array of struct fields.

PPL query:

```
source=table | flatten bridges
```

_time	city	coor	country	length	name
2024-09-13T12:00:00	London	{35, 51.5074, -0.1278}	England	801	Tower Bridge
2024-09-13T12:00:00	London	{35, 51.5074, -0.1278}	England	928	London Bridge
2024-09-13T12:00:00	Paris	{35, 48.8566, 2.3522}	France	232	Pont Neuf
2024-09-13T12:00:00	Paris	{35, 48.8566, 2.3522}	France	160	Pont Alexandre III
2024-09-13T12:00:00	Venice	{2, 45.4408, 12.3155}	Italy	48	Rialto Bridge
2024-09-13T12:00:00	Venice	{2, 45.4408, 12.3155}	Italy	11	Bridge of Sighs
2024-09-13T12:00:00	Prague	{200, 50.0755, 14.4378}	Czech Republic	516	Charles Bridge
2024-09-13T12:00:00	Prague	{200, 50.0755, 14.4378}	Czech Republic	343	Legion Bridge
2024-09-13T12:00:00	Budapest	{96, 47.4979, 19.0402}	Hungary	375	Chain Bridge
2024-09-13T12:00:00	Budapest	{96, 47.4979, 19.0402}	Hungary	333	Liberty Bridge
1990-09-13T12:00:00	Warsaw	NULL	Poland	NULL	NULL

Example 3: flatten array and struct

This example shows how to flatten multiple fields.

PPL query:

```
source=table | flatten bridges | flatten coor
```

_time	city	country	length	name	alt	lat	long
2024-09-13T12:00:00	London	England	801	Tower Bridge	35	51.5074	-0.1278
2024-09-13T12:00:00	London	England	928	London Bridge	35	51.5074	-0.1278
2024-09-13T12:00:00	Paris	France	232	Pont Neuf	35	48.8566	2.3522
2024-09-13T12:00:00	Paris	France	160	Pont Alexandre III	35	48.8566	2.3522
2024-09-13T12:00:00	Venice	Italy	48	Rialto Bridge	2	45.4408	12.3155
2024-09-13T12:00:00	Venice	Italy	11	Bridge of Sighs	2	45.4408	12.3155
2024-09-13T12:00:00	Prague	Czech Republic	516	Charles Bridge	200	50.0755	14.4378

_time	city	country	length	name	alt	lat	long
2024-09-13T12:00:00	Prague	Czech Republic	343	Legion Bridge	200	50.0755	14.4378
2024-09-13T12:00:00	Budape	Hungary	375	Chain Bridge	96	47.4979	19.0402
2024-09-13T12:00:00	Budape	Hungary	333	Liberty Bridge	96	47.4979	19.0402
1990-09-13T12:00:00	Warsaw	Poland	NULL	NULL	NULL	NULL	NULL

grok command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called "Commands".

The `grok` command parses a text field with a grok pattern and appends the results to the search result.

Syntax

Use the following syntax:

```
grok <field> <pattern>
```

field

- Mandatory.
- The field must be a text field.

pattern

- Mandatory.
- The grok pattern used to extract new fields from the given text field.
- If a new field name already exists, it will replace the original field.

Grok pattern

The grok pattern is used to match the text field of each document to extract new fields.

Example 1: Create the new field

This example shows how to create a new field `host` for each document. `host` will be the host name after `@` in the `email` field. Parsing a null field will return an empty string.

```
os> source=accounts | grok email '.+@%{HOSTNAME:host}' | fields email, host ;
fetched rows / total rows = 4/4
+-------------------------+-------------+
| email                   | host        |
|-------------------------+-------------|
| jane_doe@example.com    | example.com |
| arnav_desai@example.net | example.net |
| null                    |             |
| juan_li@example.org     | example.org |
+-------------------------+-------------+
```

Example 2: Override the existing field

This example shows how to override the existing `address` field with the street number removed.

```
os> source=accounts | grok address '%{NUMBER} %{GREEDYDATA:address}' | fields address ;
fetched rows / total rows = 4/4
+------------------+
| address          |
|------------------|
| Example Lane     |
| Any Street       |
| Main Street      |
| Example Court    |
+------------------+
```

Example 3: Using grok to parse logs

This example shows how to use grok to parse raw logs.

```
os> source=apache | grok message '%{COMMONAPACHELOG}' | fields COMMONAPACHELOG,
 timestamp, response, bytes ;
fetched rows / total rows = 4/4
+-----------------------------------------------------------------------------------------------------
+----------------------------+------------+---------+
| COMMONAPACHELOG
                                          | timestamp                  | response   |
 bytes   |
|-----------------------------------------------------------------------------------------------------
+----------------------------+------------+---------|
| 177.95.8.74 - upton5450 [28/Sep/2022:10:15:57 -0700] "HEAD /e-business/mindshare
 HTTP/1.0" 404 19927                        | 28/Sep/2022:10:15:57 -0700 | 404        |
 19927   |
| 127.45.152.6 - pouros8756 [28/Sep/2022:10:15:57 -0700] "GET /architectures/
convergence/niches/mindshare HTTP/1.0" 100 28722 | 28/Sep/2022:10:15:57 -0700 | 100
    | 28722   |
| *************** - - [28/Sep/2022:10:15:57 -0700] "PATCH /strategize/out-of-the-box
 HTTP/1.0" 401 27439                       | 28/Sep/2022:10:15:57 -0700 | 401        |
 27439   |
| ************** - - [28/Sep/2022:10:15:57 -0700] "POST /users HTTP/1.1" 301 9481
                                         | 28/Sep/2022:10:15:57 -0700 | 301        | 9481
    |
+-----------------------------------------------------------------------------------------------------
+----------------------------+------------+---------+
```

Limitations

The grok command has the same limitations as the parse command.

head command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

Use the head command to return the first N number of specified results after an optional offset in search order.

Syntax

Use the following syntax:

```
head [<size>] [from <offset>]
```

<size>

- Optional integer.
- The number of results to return.
- Default: 10

<offset>

- Integer after optional `from`.
- The number of results to skip.
- Default: 0

Example 1: Get first 10 results

This example shows how to retrieve a maximum of 10 results from the accounts index.

PPL query:

```
os> source=accounts | fields firstname, age | head;
fetched rows / total rows = 4/4
+-------------+-------+
| firstname   | age   |
|-------------+-------|
| Jane        | 32    |
| John        | 36    |
| Jorge       | 28    |
| Juan        | 33    |
+-------------+-------+
```

Example 2: Get first N results

The example shows the first N results from the accounts index.

PPL query:

```
os> source=accounts | fields firstname, age | head 3;
fetched rows / total rows = 3/3
+-------------+-------+
| firstname   | age   |
|-------------+-------|
| Jane        | 32    |
| John        | 36    |
| Jorge       | 28    |
+-------------+-------+
```

Example 3: Get first N results after offset M

This example shows how to retrieve the first N results after skipping M results from the accounts index.

PPL query:

```
os> source=accounts | fields firstname, age | head 3 from 1;
fetched rows / total rows = 3/3
+-------------+-------+
| firstname   | age   |
|-------------+-------|
| John        | 36    |
| Jorge       | 28    |
| Juan        | 33    |
+-------------+-------+
```

join command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

The join command allows you to combine data from multiple sources based on common fields, enabling you to perform complex analyses and gain deeper insights from your distributed datasets

Schema

There are least two indices, `otel-v1-apm-span-*` (large) and `otel-v1-apm-service-map` (small).

Relevant fields from indices:

`otel-v1-apm-span-*`

- traceId - A unique identifier for a trace. All spans from the same trace share the same traceId.
- spanId - A unique identifier for a span within a trace, assigned when the span is created.
- parentSpanId - The spanId of this span's parent span. If this is a root span, then this field must be empty.
- durationInNanos - The difference in nanoseconds between startTime and endTime. (this is `latency` in UI)
- serviceName - The resource from which the span originates.
- traceGroup - The name of the trace's root span.

`otel-v1-apm-service-map`

- serviceName - The name of the service that emitted the span.
- destination.domain - The serviceName of the service being called by this client.
- destination.resource - The span name (API, operation, and so on) being called by this client.
- target.domain - The serviceName of the service being called by a client.
- target.resource - The span name (API, operation, and so on) being called by a client.
- traceGroupName - The top-level span name that started the request chain.

Requirement

Support **join** to calculate the following:

For each service, join span index on service map index to calculate metrics under different type of filters.

This sample query calculates latency when filtered by trace group `client_cancel_order` for the `order` service.

```
SELECT avg(durationInNanos)
FROM `otel-v1-apm-span-000001` t1
WHERE t1.serviceName = `order`
  AND ((t1.name in
          (SELECT target.resource
```

```
           FROM `otel-v1-apm-service-map`
           WHERE serviceName = `order`
             AND traceGroupName = `client_cancel_order`)
       AND t1.parentSpanId != NULL)
      OR (t1.parentSpanId = NULL
          AND t1.name = `client_cancel_order`))
  AND t1.traceId in
    (SELECT traceId
     FROM `otel-v1-apm-span-000001`
     WHERE serviceName = `order`)
```

Migrate to PPL

Syntax of the join command

```
SEARCH source=<left-table>
| <other piped command>
| [joinType] JOIN
    [leftAlias]
    ON joinCriteria
    <right-table>
| <other piped command>
```

Rewriting

```
SEARCH source=otel-v1-apm-span-000001
| WHERE serviceName = 'order'
| JOIN left=t1 right=t2
    ON t1.traceId = t2.traceId AND t2.serviceName = 'order'
    otel-v1-apm-span-000001 -- self inner join
| EVAL s_name = t1.name -- rename to avoid ambiguous
| EVAL s_parentSpanId = t1.parentSpanId -- RENAME command would be better when it is
 supported
| EVAL s_durationInNanos = t1.durationInNanos
| FIELDS s_name, s_parentSpanId, s_durationInNanos -- reduce colunms in join
| LEFT JOIN left=s1 right=t3
    ON s_name = t3.target.resource AND t3.serviceName = 'order' AND t3.traceGroupName =
 'client_cancel_order'
    otel-v1-apm-service-map
| WHERE (s_parentSpanId IS NOT NULL OR (s_parentSpanId IS NULL AND s_name =
 'client_cancel_order'))
| STATS avg(s_durationInNanos) -- no need to add alias if there is no ambiguous
```

joinType

- Syntax: `INNER | LEFT OUTER | CROSS`
- Optional
- The type of join to perform. The default is `INNER` if not specified.

leftAlias

- Syntax: `left = <leftAlias>`
- Optional
- The subquery alias to use with the left join side, to avoid ambiguous naming.

joinCriteria

- Syntax: `<expression>`
- Required
- The syntax starts with `ON`. It could be any comparison expression. Generally, the join criteria looks like `<leftAlias>.<leftField>=<rightAlias>.<rightField>`.

  For example: `l.id = r.id`. If the join criteria contains multiple conditions, you can specify `AND` and `OR` operator between each comparison expression. For example, `l.id = r.id AND l.email = r.email AND (r.age > 65 OR r.age < 18)`.

More examples

Migration from SQL query (TPC-H Q13):

```
SELECT c_count, COUNT(*) AS custdist
FROM
  ( SELECT c_custkey, COUNT(o_orderkey) c_count
    FROM customer LEFT OUTER JOIN orders ON c_custkey = o_custkey
        AND o_comment NOT LIKE '%unusual%packages%'
    GROUP BY c_custkey
  ) AS c_orders
GROUP BY c_count
ORDER BY custdist DESC, c_count DESC;
```

Rewritten by PPL join query:

```
SEARCH source=customer
| FIELDS c_custkey
| LEFT OUTER JOIN
    ON c_custkey = o_custkey AND o_comment NOT LIKE '%unusual%packages%'
    orders
| STATS count(o_orderkey) AS c_count BY c_custkey
| STATS count() AS custdist BY c_count
| SORT - custdist, - c_count
```

Limitation: sub searches are unsupported in join right side.

If sub searches are supported, you can rewrite the above PPL query as follows:

```
SEARCH source=customer
| FIELDS c_custkey
| LEFT OUTER JOIN
   ON c_custkey = o_custkey
   [
      SEARCH source=orders
      | WHERE o_comment NOT LIKE '%unusual%packages%'
      | FIELDS o_orderkey, o_custkey
   ]
| STATS count(o_orderkey) AS c_count BY c_custkey
| STATS count() AS custdist BY c_count
| SORT - custdist, - c_count
```

lookup command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

Use the `lookup` command to enrich your search data by adding or replacing data from a lookup index (dimension table). This command allows you to extend fields of an index with values from a dimension table. You can also use it to append or replace values when lookup conditions are met. The `lookup` command is more suitable than the `Join` command for enriching source data with a static dataset.

Syntax

Use the following syntax:

```
SEARCH source=<sourceIndex>
| <other piped command>
| LOOKUP <lookupIndex> (<lookupMappingField> [AS <sourceMappingField>])...
    [(REPLACE | APPEND) (<inputField> [AS <outputField>])...]
| <other piped command>
```

lookupIndex

- Required.
- The name of the lookup index (dimension table).

lookupMappingField

- Required.
- A mapping key in the lookup index, analogous to a join key from the right table. You can specify multiple fields, separated by commas.

sourceMappingField

- Optional.
- Default: <lookupMappingField>.
- A mapping key from the source query, analogous to a join key from the left side.

inputField

- Optional.
- Default: All fields of the lookup index where matched values are found.
- A field in the lookup index where matched values are applied to the result output. You can specify multiple fields, separated by commas.

outputField

- Optional.
- Default: `<inputField>`.

- A field in the output. You can specify multiple output fields. If you specify an existing field name from the source query, its values will be replaced or appended by matched values from inputField. If you specify a new field name, it will be added to the results.

REPLACE | APPEND

- Optional.
- Default: REPLACE
- Specifies how to handle matched values. If you specify REPLACE, matched values in <lookupIndex> field overwrite the values in result. If you specify APPEND, matched values in <lookupIndex> field only append to the missing values in result.

Usage

- LOOKUP <lookupIndex> id AS cid REPLACE mail AS email
- LOOKUP <lookupIndex> name REPLACE mail AS email
- LOOKUP <lookupIndex> id AS cid, name APPEND address, mail AS email
- LOOKUP <lookupIndex> id

Example

See the following examples.

```
SEARCH source=<sourceIndex>
| WHERE orderType = 'Cancelled'
| LOOKUP account_list, mkt_id AS mkt_code REPLACE amount, account_name AS name
| STATS count(mkt_code), avg(amount) BY name
```

```
SEARCH source=<sourceIndex>
| DEDUP market_id
| EVAL category=replace(category, "-", ".")
| EVAL category=ltrim(category, "dvp.")
| LOOKUP bounce_category category AS category APPEND classification
```

```
SEARCH source=<sourceIndex>
| LOOKUP bounce_category category
```

parse command

The `parse` command parses a text field with a regular expression and appends the result to the search result.

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called "Commands".

Syntax

Use the following syntax:

```
parse <field> <pattern>
```

`field`

- Mandatory.
- The field must be a text field.

`pattern`

- Mandatory string.
- This is the regular expression pattern used to extract new fields from the given text field.
- If a new field name already exists, it will replace the original field.

Regular expression

The regular expression pattern is used to match the whole text field of each document with Java regex engine. Each named capture group in the expression will become a new `STRING` field.

Example 1: Create a new field

The example shows how to create a new field `host` for each document. `host` will be the host name after `@` in the `email` field. Parsing a null field will return an empty string.

PPL query:

```
os> source=accounts | parse email '.+@(?<host>.+)' | fields email, host ;
fetched rows / total rows = 4/4
+----------------------+-------------+
| email                | host        |
|----------------------+-------------|
| jane_doe@example.com  | example.com |
| john_doe@example.net  | example.net |
| null                 |             |
| juan_li@example.org   | example.org |
+----------------------+-------------+
```

Example 2: Override an existing field

The example shows how to override the existing `address` field with the street number removed.

PPL query:

```
os> source=accounts | parse address '\d+ (?<address>.+)' | fields address ;
fetched rows / total rows = 4/4
+------------------+
| address          |
|------------------|
| Example Lane     |
| Example Street   |
| Example Avenue   |
| Example Court    |
+------------------+
```

Example 3: Filter and sort by casted parsed field

The example shows how to sort street numbers that are higher than 500 in the `address` field.

PPL query:

```
os> source=accounts | parse address '(?<streetNumber>\d+) (?<street>.+)' | where
 cast(streetNumber as int) > 500 | sort num(streetNumber) | fields streetNumber,
 street ;
fetched rows / total rows = 3/3
+----------------+----------------+
| streetNumber   | street         |
|----------------+----------------|
| ***            | Example Street |
```

```
| ***            | Example Avenue |
| 880            | Example Lane   |
+----------------+----------------+
```

Limitations

There are a few limitations with the parse command:

- Fields defined by parse cannot be parsed again.

  The following command will not work:

  ```
  source=accounts | parse address '\d+ (?<street>.+)' | parse street '\w+ (?<road>\w+)'
  ```

- Fields defined by parse cannot be overridden with other commands.

  `where` will not match any documents since `street` cannot be overridden:

  ```
  source=accounts | parse address '\d+ (?<street>.+)' | eval street='1' | where
   street='1' ;
  ```

- The text field used by parse cannot be overridden.

  `street` will not be successfully parsed since `address` is overridden:

  ```
  source=accounts | parse address '\d+ (?<street>.+)' | eval address='1' ;
  ```

- Fields defined by parse cannot be filtered or sorted after using them in the `stats` command.

  `where` in the following command will not work:

  ```
  source=accounts | parse email '.+@(?<host>.+)' | stats avg(age) by host | where
   host=pyrami.com ;
  ```

patterns command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

The `patterns` command extracts log patterns from a text field and appends the results to the search result. Grouping logs by their patterns makes it easier to aggregate stats from large volumes of log data for analysis and troubleshooting.

Syntax

Use the following syntax:

```
patterns [new_field=<new-field-name>] [pattern=<pattern>] <field>
```

new-field-name

- Optional string.
- This is the name of the new field for extracted patterns.
- The default is `patterns_field`.
- If the name already exists, it will replace the original field.

pattern

- Optional string.
- This it the regex pattern of characters that should be filtered out from the text field.
- If absent, the default pattern is alphanumeric characters (`[a-zA-Z\d]`).

field

- Mandatory.
- The field must be a text field.

Example 1: Create the new field

The example shows how to use extract punctuations in `email` for each document. Parsing a null field will return an empty string.

PPL query:

```
os> source=accounts | patterns email | fields email, patterns_field ;
fetched rows / total rows = 4/4
+----------------------+------------------+
```

```
| email                | patterns_field   |
|----------------------+------------------|
| jane_doe@example.com | @.               |
| john_doe@example.net | @.               |
| null                 |                  |
| juan_li@example.org  | @.               |
+----------------------+------------------+
```

Example 2: Extract log patterns

The example shows how to extract punctuations from a raw log field using the default patterns.

PPL query:

```
os> source=apache | patterns message | fields message, patterns_field ;
fetched rows / total rows = 4/4
+-------------------------------------------------------------------------------------------------------
+---------------------------------+
| message
                                       | patterns_field                 |
|-------------------------------------------------------------------------------------------------------
+---------------------------------|
| 177.95.8.74 - upton5450 [28/Sep/2022:10:15:57 -0700] "HEAD /e-business/mindshare
 HTTP/1.0" 404 19927                         | ... -  [//::: -] " /-/ /."      |
| ************ - pouros8756 [28/Sep/2022:10:15:57 -0700] "GET /architectures/
convergence/niches/mindshare HTTP/1.0" 100 28722 | ... -  [//::: -] " //// /."     |
| *************** - - [28/Sep/2022:10:15:57 -0700] "PATCH /strategize/out-of-the-box
 HTTP/1.0" 401 27439                        | ... - - [//::: -] " //--- /."   |
| ************** - - [28/Sep/2022:10:15:57 -0700] "POST /users HTTP/1.1" 301 9481
                                       | ... - - [//::: -] " / /."       |
+-------------------------------------------------------------------------------------------------------
+---------------------------------+
```

Example 3: Extract log patterns with custom regex pattern

The example shows how to extract punctuations from a raw log field using user defined patterns.

PPL query:

```
os> source=apache | patterns new_field='no_numbers' pattern='[0-9]' message | fields
 message, no_numbers ;
fetched rows / total rows = 4/4
```

```
+-----------------------------------------------------------------------------------------------------------
+-------------------------------------------------------------------------------
+
| message
                                       | no_numbers
                                       |
|-----------------------------------------------------------------------------------------------------------
+-------------------------------------------------------------------------------|
| 177.95.8.74 - upton5450 [28/Sep/2022:10:15:57 -0700] "HEAD /e-business/mindshare
 HTTP/1.0" 404 19927                        | ... - upton [/Sep/::: -] "HEAD /e-
business/mindshare HTTP/."                          |
| 127.45.152.6 - pouros8756 [28/Sep/2022:10:15:57 -0700] "GET /architectures/
convergence/niches/mindshare HTTP/1.0" 100 28722 | ... - pouros [/Sep/::: -] "GET /
architectures/convergence/niches/mindshare HTTP/."   |
| ********** - - [28/Sep/2022:10:15:57 -0700] "PATCH /strategize/out-of-the-box
 HTTP/1.0" 401 27439                       | ... - - [/Sep/::: -] "PATCH /strategize/
out-of-the-box HTTP/."                         |
| ********** - - [28/Sep/2022:10:15:57 -0700] "POST /users HTTP/1.1" 301 9481
                                       | ... - - [/Sep/::: -] "POST /users HTTP/."
                                       |
+-----------------------------------------------------------------------------------------------------------
+-------------------------------------------------------------------------------
+
```

Limitation

The patterns command has the same limitations as the parse command.

rare command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

Use the `rare` command to find the least common tuple of values of all fields in the field list.

> **Note**
>
> A maximum of 10 results is returned for each distinct tuple of values of the group-by fields.

Syntax

Use the following syntax:

```
rare [N] <field-list> [by-clause] rare_approx [N] <field-list> [by-clause]
```

field-list

- Mandatory.
- A comma-delimited list of field names.

by-clause

- Optional.
- One or more fields to group the results by.

N

- The number of results to return.
- Default: 10

rare_approx

- The approximate count of the rare (n) fields by using estimated cardinality by HyperLogLog++ algorithm.

Example 1: Find the least common values in a field

The example finds the least common gender of all the accounts.

PPL query:

```
os> source=accounts | rare gender;
os> source=accounts | rare_approx 10 gender;
os> source=accounts | rare_approx gender;
fetched rows / total rows = 2/2
+----------+
| gender   |
```

```
|----------|
| F        |
| M        |
+----------+
```

Example 2: Find the least common values organized by gender

The example finds the least common age of all the accounts group by gender.

PPL query:

```
os> source=accounts | rare 5 age by gender;
os> source=accounts | rare_approx 5 age by gender;
fetched rows / total rows = 4/4
+----------+-------+
| gender   | age   |
|----------+-------|
| F        | 28    |
| M        | 32    |
| M        | 33    |
| M        | 36    |
+----------+-------+
```

rename command

Use the `rename` command to change the names of one or more fields in the search result.

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

Syntax

Use the following syntax:

```
rename <source-field> AS <target-field>["," <source-field> AS <target-field>]...
```

source-field

- Mandatory.

- This is the name of the field you want to rename.

target-field

- Mandatory.
- This is the name you want to rename to.

Example 1: Rename one field

This example shows how to rename a single field.

PPL query:

```
os> source=accounts | rename account_number as an | fields an;
fetched rows / total rows = 4/4
+------+
| an   |
|------|
| 1    |
| 6    |
| 13   |
| 18   |
+------+
```

Example 2: Rename multiple fields

This example shows how to rename multiple fields.

PPL query:

```
os> source=accounts | rename account_number as an, employer as emp | fields an, emp;
fetched rows / total rows = 4/4
+------+---------+
| an   | emp     |
|------+---------|
| 1    | Pyrami  |
| 6    | Netagy  |
| 13   | Quility |
| 18   | null    |
+------+---------+
```

Limitations

- Overriding existing field is unsupported:

```
source=accounts | grok address '%{NUMBER} %{GREEDYDATA:address}' | fields address
```

search command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called "Commands".

Use the `search` command to retrieve documents from an index. The `search` command can only be used as the first command in a PPL query.

Syntax

Use the following syntax:

```
search source=[<remote-cluster>:]<index> [boolean-expression]
```

search

- Optional.
- Search keywords, which can be omitted.

index

- Mandatory.
- The search command must specify which index to query from.
- The index name can be prefixed by `<cluster name>:` for cross-cluster searches.

bool-expression

- Optional.

- Any expression that evaluates to a boolean value.

Example 1: Fetch all the data

The example show fetch all the document from accounts index.

PPL query:

```
os> source=accounts;
+------------------+-------------+----------------------+-----------+----------
+--------+----------------+---------+-------+-----------------------+------------+
| account_number   | firstname   | address              | balance   | gender   | city
 | employer       | state   | age   | email                 | lastname   |
|------------------+-------------+----------------------+-----------+----------
+--------+----------------+---------+-------+-----------------------+------------|
| 1                | Jorge       | *** Any Lane         | 39225     | M        | Brogan
 | ExampleCorp    | IL      | 32    | jane_doe@example.com  | Souza      |
| 6                | John        | *** Example Street   | 5686      | M        | Dante
 | AnyCorp        | TN      | 36    | john_doe@example.com  | Doe        |
| 13               | Jane        | *** Any Street       | *****     | F        | Nogal
 | ExampleCompany | VA      | 28    | null                  | Doe        |
| 18               | Juan        | *** Example Court    | 4180      | M        | Orick
 | null           | MD      | 33    | juan_li@example.org   | Li         |
+------------------+-------------+----------------------+-----------+----------
+--------+----------------+---------+-------+-----------------------+------------+
```

Example 2: Fetch data with condition

The example show fetch all the document from accounts index with .

PPL query:

```
os> SEARCH source=accounts account_number=1 or gender="F";
+------------------+-------------+--------------------+-----------+----------+--------
+----------------+---------+-------+-------------------------+------------+
| account_number   | firstname   | address            | balance   | gender   | city   |
 employer       | state   | age   | email                -  | lastname   |
|------------------+-------------+--------------------+-----------+----------+--------
+----------------+---------+-------+-------------------------+------------|
| 1                | Jorge       | *** Any Lane       | *****     | M        | Brogan |
 ExampleCorp    | IL      | 32    | jorge_souza@example.com | Souza      |
```

```
| 13               | Jane        | *** Any Street     | *****     | F        | Nogal  |
 ExampleCompany | VA      | 28    | null                    | Doe        |
+------------------+-------------+--------------------+-----------+----------+--------
+-----------------+---------+-------+-----------------------+------------+
```

sort command

Use the `sort` command to sort search result by specified fields.

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called "Commands".

Syntax

Use the following syntax:

```
sort <[+|-] sort-field>...
```

[+|-]

- Optional.
- The plus [+] stands for ascending order with NULL/MISSING values first.
- The minus [-] stands for descending order with NULL/MISSING values last.
- Default: Ascending order with NULL/MISSING values first.

sort-field

- Mandatory.
- The field used for sorting.

Example 1: Sort by one field

The example shows how to sort the document with the age field in ascending order.

PPL query:

```
os> source=accounts | sort age | fields account_number, age;
fetched rows / total rows = 4/4
+------------------+-------+
| account_number   | age   |
|------------------+-------|
| 13               | 28    |
| 1                | 32    |
| 18               | 33    |
| 6                | 36    |
+------------------+-------+
```

Example 2: Sort by one field and return all the results

The example shows how to sort the document with the age field in ascending order.

PPL query:

```
os> source=accounts | sort age | fields account_number, age;
fetched rows / total rows = 4/4
+------------------+-------+
| account_number   | age   |
|------------------+-------|
| 13               | 28    |
| 1                | 32    |
| 18               | 33    |
| 6                | 36    |
+------------------+-------+
```

Example 3: Sort by one field in descending order

The example shows how to sort the document with the age field in descending order.

PPL query:

```
os> source=accounts | sort - age | fields account_number, age;
fetched rows / total rows = 4/4
+------------------+-------+
| account_number   | age   |
|------------------+-------|
| 6                | 36    |
| 18               | 33    |
| 1                | 32    |
```

```
| 13                | 28    |
+-------------------+-------+
```

Example 4: Sort by multiple fields

The example shows how to sort the document with the gender field in ascending order and the age field in descending order.

PPL query:

```
os> source=accounts | sort + gender, - age | fields account_number, gender, age;
fetched rows / total rows = 4/4
+------------------+----------+-------+
| account_number   | gender   | age   |
|------------------+----------+-------|
| 13               | F        | 28    |
| 6                | M        | 36    |
| 18               | M        | 33    |
| 1                | M        | 32    |
+------------------+----------+-------+
```

Example 5: Sort by field include null value

The example shows how to sort the employer field by the default option (ascending order and null first). The result shows that the null value is in the first row.

PPL query:

```
os> source=accounts | sort employer | fields employer;
fetched rows / total rows = 4/4
+------------+
| employer   |
|------------|
| null       |
| AnyCompany |
| AnyCorp    |
| AnyOrgty   |
+------------+
```

stats command

Use the `stats` command to calculate the aggregation from search result.

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

NULL/MISSING values handling

NULL/MISSING values handling

Function	NULL	MISSING
COUNT	Not counted	Not counted
SUM	Ignore	Ignore
AVG	Ignore	Ignore
MAX	Ignore	Ignore
MIN	Ignore	Ignore

Syntax

Use the following syntax:

```
stats <aggregation>... [by-clause]
```

aggregation

- Mandatory.
- An aggregation function applied to a field.

by-clause

- Optional.
- Syntax: by `[span-expression,] [field,]...`

- Specifies fields and expressions for grouping the aggregation results. The by-clause allows you to group your aggregation results using fields and expressions. You can use scalar functions, aggregation functions, and even span expressions to split specific fields into buckets of equal intervals.
- Default: If no `<by-clause>` is specified, the stats command returns a single row representing the aggregation over the entire result set.

span-expression

- Optional, at most one.
- Syntax: `span(field_expr, interval_expr)`
- The unit of the interval expression is the natural unit by default. If the field is a date and time type field, and the interval is in date/time units, you specify the unit in the interval expression.
- For example, splitting the `age` field into buckets by 10 years, it looks like `span(age, 10)`. To split a timestamp field into hourly intervals, use `span(timestamp, 1h)`.

Available time units

Span interval units
millisecond (ms)
second (s)
minute (m, case sensitive)
hour (h)
day (d)
week (w)
month (M, case sensitive)
quarter (q)
year (y)

Aggregation functions

COUNT

Returns a count of the number of expr in the rows retrieved by a SELECT statement.

Example:

```
os> source=accounts | stats count();
fetched rows / total rows = 1/1
+-----------+
| count()   |
|-----------|
| 4         |
+-----------+
```

SUM

Use `SUM(expr)` to return the sum of expr.

Example

```
os> source=accounts | stats sum(age) by gender;
fetched rows / total rows = 2/2
+------------+----------+
| sum(age)   | gender   |
|------------+----------|
| 28         | F        |
| 101        | M        |
+------------+----------+
```

AVG

Use `AVG(expr)` to return the average value of expr.

Example

```
os> source=accounts | stats avg(age) by gender;
fetched rows / total rows = 2/2
+--------------------+----------+
| avg(age)           | gender   |
```

```
|--------------------+----------|
| 28.0               | F        |
| 33.666666666666664 | M        |
+--------------------+----------+
```

MAX

Use `MAX(expr)` to return the maximum value of expr.

Example

```
os> source=accounts | stats max(age);
fetched rows / total rows = 1/1
+------------+
| max(age)   |
|------------|
| 36         |
+------------+
```

MIN

Use `MIN(expr)` to return the minimum value of expr.

Example

```
os> source=accounts | stats min(age);
fetched rows / total rows = 1/1
+------------+
| min(age)   |
|------------|
| 28         |
+------------+
```

STDDEV_SAMP

Use `STDDEV_SAMP(expr)` to return the sample standard deviation of expr.

Example:

```
os> source=accounts | stats stddev_samp(age);
fetched rows / total rows = 1/1
+--------------------+
```

```
| stddev_samp(age)   |
|--------------------|
| 3.304037933599835  |
+--------------------+
```

STDDEV_POP

Use `STDDEV_POP(expr)` to return the population standard deviation of expr.

Example:

```
os> source=accounts | stats stddev_pop(age);
fetched rows / total rows = 1/1
+--------------------+
| stddev_pop(age)    |
|--------------------|
| 2.**************** |
+--------------------+
```

TAKE

Use `TAKE(field [, size])` to return the original values of a field. It does not guarantee on the order of values.

field

- Mandatory.
- The field must be a text field.

size

- Optional integer.
- The number of values should be returned.
- Default is 10.

Example

```
os> source=accounts | stats take(firstname);
```

```
fetched rows / total rows = 1/1
+----------------------------+
| take(firstname)            |
|----------------------------|
| [Jane, Mary, Nikki, Juan   |
+----------------------------+
```

PERCENTILE or PERCENTILE_APPROX

Use `PERCENTILE(expr, percent)` or `PERCENTILE_APPROX(expr, percent)` to return the approximate percentile value of expr at the specified percentage.

percent

- The number must be a constant between 0 and 100.

Example

```
os> source=accounts | stats percentile(age, 90) by gender;
fetched rows / total rows = 2/2
+-----------------------+----------+
| percentile(age, 90)   | gender   |
|-----------------------+----------|
| 28                    | F        |
| 36                    | M        |
+-----------------------+----------+
```

Example 1: Calculate the count of events

The example shows how to calculate the count of events in the accounts.

```
os> source=accounts | stats count();
fetched rows / total rows = 1/1
+-----------+
| count()   |
|-----------|
| 4         |
+-----------+
```

Example 2: Calculate the average of a field

The example shows how to calculate the average age for all accounts.

```
os> source=accounts | stats avg(age);
fetched rows / total rows = 1/1
+------------+
| avg(age)   |
|------------|
| 32.25      |
+------------+
```

Example 3: Calculate the average of a field by group

The example shows how to calculate the average age for all accounts, grouped by gender.

```
os> source=accounts | stats avg(age) by gender;
fetched rows / total rows = 2/2
+--------------------+----------+
| avg(age)           | gender   |
|--------------------+----------|
| 28.0               | F        |
| 33.666666666666664 | M        |
+--------------------+----------+
```

Example 4: Calculate the average, sum, and count of a field by group

The example shows how to calculate the average age, sum age, and count of events for all the accounts, grouped by gender.

```
os> source=accounts | stats avg(age), sum(age), count() by gender;
fetched rows / total rows = 2/2
+--------------------+------------+-----------+----------+
| avg(age)           | sum(age)   | count()   | gender   |
|--------------------+------------+-----------+----------|
| 28.0               | 28         | 1         | F        |
| 33.666666666666664 | 101        | 3         | M        |
+--------------------+------------+-----------+----------+
```

Example 5: Calculate the maximum of a field

The example calculates the maximum age for all accounts.

```
os> source=accounts | stats max(age);
```

```
fetched rows / total rows = 1/1
+------------+
| max(age)   |
|------------|
| 36         |
+------------+
```

Example 6: Calculate the maximum and minimum of a field by group

The example calculates the maximum and minimum age values for all accounts, grouped by gender.

```
os> source=accounts | stats max(age), min(age) by gender;
fetched rows / total rows = 2/2
+------------+------------+----------+
| max(age)   | min(age)   | gender   |
|------------+------------+----------|
| 28         | 28         | F        |
| 36         | 32         | M        |
+------------+------------+----------+
```

Example 7: Calculate the distinct count of a field

To get the count of distinct values of a field, you can use the `DISTINCT_COUNT` (or `DC`) function instead of `COUNT`. The example calculates both the count and the distinct count of gender field of all the accounts.

```
os> source=accounts | stats count(gender), distinct_count(gender);
fetched rows / total rows = 1/1
+-----------------+--------------------------+
| count(gender)   | distinct_count(gender)   |
|-----------------+--------------------------|
| 4               | 2                        |
+-----------------+--------------------------+
```

Example 8: Calculate the count by a span

The example gets the count of age by the interval of 10 years.

```
os> source=accounts | stats count(age) by span(age, 10) as age_span
fetched rows / total rows = 2/2
```

```
+--------------+------------+
| count(age)   | age_span   |
|--------------+------------|
| 1            | 20         |
| 3            | 30         |
+--------------+------------+
```

Example 9: Calculate the count by a gender and span

This example counts records grouped by gender and age spans of 5 years.

```
os> source=accounts | stats count() as cnt by span(age, 5) as age_span, gender
fetched rows / total rows = 3/3
+-------+------------+----------+
| cnt   | age_span   | gender   |
|-------+------------+----------|
| 1     | 25         | F        |
| 2     | 30         | M        |
| 1     | 35         | M        |
+-------+------------+----------+
```

The span expression always appears as the first grouping key, regardless of the order specified in the command.

```
os> source=accounts | stats count() as cnt by gender, span(age, 5) as age_span
fetched rows / total rows = 3/3
+-------+------------+----------+
| cnt   | age_span   | gender   |
|-------+------------+----------|
| 1     | 25         | F        |
| 2     | 30         | M        |
| 1     | 35         | M        |
+-------+------------+----------+
```

Example 10: Calculate the count and get email list by a gender and span

The example gets the count of age by the interval of 10 years and group by gender, additionally for each row get a list of at most 5 emails.

```
os> source=accounts | stats count() as cnt, take(email, 5) by span(age, 5) as age_span,
 gender
fetched rows / total rows = 3/3
```

```
+-------+-----------------------------------------------------+------------+----------+
| cnt   | take(email, 5)                                      | age_span   | gender   |
|-------+-----------------------------------------------------+------------+----------|
| 1     | []                                                  | 25         | F        |
| 2     | [janedoe@anycompany.com,juanli@examplecompany.org] | 30         | M        |
| 1     | [marymajor@examplecorp.com]                         | 35         | M        |
+-------+-----------------------------------------------------+------------+----------+
```

Example 11: Calculate the percentile of a field

The example shows how to calculate the percentile 90th age of all the accounts.

```
os> source=accounts | stats percentile(age, 90);
fetched rows / total rows = 1/1
+-----------------------+
| percentile(age, 90)   |
|-----------------------|
| 36                    |
+-----------------------+
```

Example 12: Calculate the percentile of a field by group

The example shows how to calculate the percentile 90th age of all the accounts group by gender.

```
os> source=accounts | stats percentile(age, 90) by gender;
fetched rows / total rows = 2/2
+-----------------------+----------+
| percentile(age, 90)   | gender   |
|-----------------------+----------|
| 28                    | F        |
| 36                    | M        |
+-----------------------+----------+
```

Example 13: Calculate the percentile by a gender and span

The example gets the percentile 90th age by the interval of 10 years and group by gender.

```
os> source=accounts | stats percentile(age, 90) as p90 by span(age, 10) as age_span,
 gender
fetched rows / total rows = 2/2
+-------+------------+----------+
```

```
| p90   | age_span   | gender   |
|-------+------------+----------|
| 28    | 20         | F        |
| 36    | 30         | M        |
+-------+------------+----------+
```

```
- `source = table | stats avg(a) `
- `source = table | where a < 50 | stats avg(c) `
- `source = table | stats max(c) by b`
- `source = table | stats count(c) by b | head 5`
- `source = table | stats distinct_count(c)`
- `source = table | stats stddev_samp(c)`
- `source = table | stats stddev_pop(c)`
- `source = table | stats percentile(c, 90)`
- `source = table | stats percentile_approx(c, 99)`
```

Aggregations with span

```
- `source = table  | stats count(a) by span(a, 10) as a_span`
- `source = table  | stats sum(age) by span(age, 5) as age_span | head 2`
- `source = table  | stats avg(age) by span(age, 20) as age_span, country  | sort -
 age_span |  head 2`
```

Aggregations with timewindow span (tumble windowing function)

```
- `source = table | stats sum(productsAmount) by span(transactionDate, 1d) as age_date
 | sort age_date`
- `source = table | stats sum(productsAmount) by span(transactionDate, 1w) as age_date,
 productId`
```

Aggregations group by multiple levels

```
- `source = table | stats avg(age) as avg_state_age by country, state | stats
 avg(avg_state_age) as avg_country_age by country`
- `source = table | stats avg(age) as avg_city_age by country, state, city | eval
 new_avg_city_age = avg_city_age - 1 | stats avg(new_avg_city_age) as avg_state_age
 by country, state | where avg_state_age > 18 | stats avg(avg_state_age) as
 avg_adult_country_age by country`
```

subquery command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

Use the `subquery` command to perform complex, nested queries within your Piped Processing Language (PPL) statements.

```
source=logs | where field in [ subquery source=events | where condition | fields
 field ]
```

In this example, the primary search (`source=logs`) is filtered by results from the subquery (`source=events`).

The subquery command supports multiple levels of nesting for complex data analysis.

Nested Subquery Example

```
source=logs | where id in [ subquery source=users | where user in [ subquery
 source=actions | where action="login" | fields user] | fields uid ]
```

InSubquery Usage

- `source = outer | where a in [ source = inner | fields b ]`
- `source = outer | where (a) in [ source = inner | fields b ]`
- `source = outer | where (a,b,c) in [ source = inner | fields d,e,f ]`
- `source = outer | where a not in [ source = inner | fields b ]`
- `source = outer | where (a) not in [ source = inner | fields b ]`
- `source = outer | where (a,b,c) not in [ source = inner | fields d,e,f ]`
- `source = outer a in [ source = inner | fields b ]` (search filtering with subquery)
- `source = outer a not in [ source = inner | fields b ]` (search filtering with subquery)
- `source = outer | where a in [ source = inner1 | where b not in [ source = inner2 | fields c ] | fields b ]` (nested)

- `source = table1 | inner join left = l right = r on l.a = r.a AND r.a in [ source = inner | fields d ] | fields l.a, r.a, b, c` (as join filter)

SQL Migration Examples with IN-Subquery PPL

TPC-H Q4 (in-subquery with aggregation)

```
select
  o_orderpriority,
  count(*) as order_count
from
  orders
where
  o_orderdate >= date '1993-07-01'
  and o_orderdate < date '1993-07-01' + interval '3' month
  and o_orderkey in (
    select
      l_orderkey
    from
      lineitem
    where l_commitdate < l_receiptdate
  )
group by
  o_orderpriority
order by
  o_orderpriority
```

Rewritten by PPL InSubquery query:

```
source = orders
| where o_orderdate >= "1993-07-01" and o_orderdate < "1993-10-01" and o_orderkey IN
  [ source = lineitem
    | where l_commitdate < l_receiptdate
    | fields l_orderkey
  ]
| stats count(1) as order_count by o_orderpriority
| sort o_orderpriority
| fields o_orderpriority, order_count
```

TPC-H Q20 (nested in-subquery)

```
select
```

```
  s_name,
  s_address
from
  supplier,
  nation
where
  s_suppkey in (
    select
      ps_suppkey
    from
      partsupp
    where
      ps_partkey in (
        select
          p_partkey
        from
          part
        where
          p_name like 'forest%'
      )
  )
  and s_nationkey = n_nationkey
  and n_name = 'CANADA'
order by
  s_name
```

Rewritten by PPL InSubquery query:

```
source = supplier
| where s_suppkey IN [
    source = partsupp
    | where ps_partkey IN [
        source = part
        | where like(p_name, "forest%")
        | fields p_partkey
      ]
    | fields ps_suppkey
  ]
| inner join left=l right=r on s_nationkey = n_nationkey and n_name = 'CANADA'
  nation
| sort s_name
```

ExistsSubquery usage

Assumptions: a, b are fields of table outer, c, d are fields of table inner, e, f are fields of table inner2.

- `source = outer | where exists [ source = inner | where a = c ]`
- `source = outer | where not exists [ source = inner | where a = c ]`
- `source = outer | where exists [ source = inner | where a = c and b = d ]`
- `source = outer | where not exists [ source = inner | where a = c and b = d ]`
- `source = outer exists [ source = inner | where a = c ]` (search filtering with subquery)
- `source = outer not exists [ source = inner | where a = c ]` (search filtering with subquery)
- `source = table as t1 exists [ source = table as t2 | where t1.a = t2.a ]` (table alias is useful in exists subquery)
- `source = outer | where exists [ source = inner1 | where a = c and exists [ source = inner2 | where c = e ] ]` (nested)
- `source = outer | where exists [ source = inner1 | where a = c | where exists [ source = inner2 | where c = e ] ]` (nested)
- `source = outer | where exists [ source = inner | where c > 10 ]` (uncorrelated exists)
- `source = outer | where not exists [ source = inner | where c > 10 ]` (uncorrelated exists)
- `source = outer | where exists [ source = inner ] | eval l = "nonEmpty" | fields l` (special uncorrelated exists)

ScalarSubquery usage

Assumptions: a, b are fields of table outer, c, d are fields of table inner, e, f are fields of table nested

Uncorrelated scalar subquery

In Select:

- `source = outer | eval m = [ source = inner | stats max(c) ] | fields m, a`

- `source = outer | eval m = [ source = inner | stats max(c) ] + b | fields m, a`

In Where:

- `source = outer | where a > [ source = inner | stats min(c) ] | fields a`

In Search filter:

- `source = outer a > [ source = inner | stats min(c) ] | fields a`

Correlated scalar subquery

In Select:

- `source = outer | eval m = [ source = inner | where outer.b = inner.d | stats max(c) ] | fields m, a`
- `source = outer | eval m = [ source = inner | where b = d | stats max(c) ] | fields m, a`
- `source = outer | eval m = [ source = inner | where outer.b > inner.d | stats max(c) ] | fields m, a`

In Where:

- `source = outer | where a = [ source = inner | where outer.b = inner.d | stats max(c) ]`
- `source = outer | where a = [ source = inner | where b = d | stats max(c) ]`
- `source = outer | where [ source = inner | where outer.b = inner.d OR inner.d = 1 | stats count() ] > 0 | fields a`

In Search filter:

- `source = outer a = [ source = inner | where b = d | stats max(c) ]`
- `source = outer [ source = inner | where outer.b = inner.d OR inner.d = 1 | stats count() ] > 0 | fields a`

Nested scalar subquery

- `source = outer | where a = [ source = inner | stats max(c) | sort c ] OR b = [ source = inner | where c = 1 | stats min(d) | sort d ]`
- `source = outer | where a = [ source = inner | where c = [ source = nested | stats max(e) by f | sort f ] | stats max(d) by c | sort c | head 1 ]`

(Relation) Subquery

`InSubquery`, `ExistsSubquery` and `ScalarSubquery` are all subquery expressions. But `RelationSubquery` is not a subquery expression, it is a subquery plan which is common used in Join or From clause.

- `source = table1 | join left = l right = r [ source = table2 | where d > 10 | head 5 ]` (subquery in join right side)
- `source = [ source = table1 | join left = l right = r [ source = table2 | where d > 10 | head 5 ] | stats count(a) by b ] as outer | head 1`

Additional Context

`InSubquery`, `ExistsSubquery`, and `ScalarSubquery` are subquery expressions commonly used in `where` clauses and search filters.

Where command:

```
| where <boolean expression> | ...
```

Search filter:

```
search source=* <boolean expression> | ...
```

A subquery expression could be used in a boolean expression:

```
| where orders.order_id in [ source=returns | where return_reason="damaged" | field
 order_id ]
```

The `orders.order_id in [ source=... ]` is a `<boolean expression>`.

In general, we name this kind of subquery clause the `InSubquery` expression. It is a `<boolean expression>`.

Subquery with different join types

Example using a `ScalarSubquery`:

```
source=employees
| join source=sales on employees.employee_id = sales.employee_id
| where sales.sale_amount > [ source=targets | where target_met="true" | fields
 target_value ]
```

Unlike InSubquery, ExistsSubquery, and ScalarSubquery, a RelationSubquery is not a subquery expression. Instead, it's a subquery plan.

```
SEARCH source=customer
| FIELDS c_custkey
| LEFT OUTER JOIN left = c, right = o ON c.c_custkey = o.o_custkey
   [
      SEARCH source=orders
      | WHERE o_comment NOT LIKE '%unusual%packages%'
      | FIELDS o_orderkey, o_custkey
   ]
| STATS ...
```

top command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

Use the `top` command to find the most common tuple of values of all fields in the field list.

Syntax

Use the following syntax:

```
top [N] <field-list> [by-clause] top_approx [N] <field-list> [by-clause]
```

N

- The number of results to return.
- Default: 10

field-list

- Mandatory.
- A comma-delimited list of field names.

by-clause

- Optional.
- One or more fields to group the results by.

top_approx

- An approximate count of the (n) top fields by using the estimated cardinality by HyperLogLog++ algorithm.

Example 1: Find the most common values in a field

The example finds the most common gender for all accounts.

PPL query:

```
os> source=accounts | top gender;
os> source=accounts | top_approx gender;
fetched rows / total rows = 2/2
+----------+
| gender   |
|----------|
| M        |
| F        |
+----------+
```

Example 2: Find the most common values in a field (limited to 1)

The example finds the single most common gender for all accounts.

PPL query:

```
os> source=accounts | top_approx 1 gender;
fetched rows / total rows = 1/1
+----------+
| gender   |
|----------|
| M        |
+----------+
```

Example 3: Find the most common values, grouped by gender

The example finds the most common age for all accounts, grouped by gender.

PPL query:

```
os> source=accounts | top 1 age by gender;
os> source=accounts | top_approx 1 age by gender;
fetched rows / total rows = 2/2
+----------+-------+
| gender   | age   |
|----------+-------|
| F        | 28    |
| M        | 32    |
+----------+-------+
```

trendline command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

Use the `trendline` command to calculate moving averages of fields.

Syntax

Use the following syntax

```
TRENDLINE [sort <[+|-] sort-field>] SMA(number-of-datapoints, field) [AS alias]
 [SMA(number-of-datapoints, field) [AS alias]]...
```

[+|-]

- Optional.
- The plus [+] stands for ascending order with NULL/MISSING values first.
- The minus [-] stands for descending order with NULL/MISSING values last.
- Default: Ascending order with NULL/MISSING values first.

sort-field

- Mandatory when sorting is used.
- The field used for sorting.

number-of-datapoints

- Mandatory.
- The number of datapoints that calculate the moving average.
- Must be greater than zero.

field

- Mandatory.
- The name of the field the moving average should be calculated for.

alias

- Optional.
- The name of the resulting column containing the moving average.

Only the Simple Moving Average (SMA) type is supported. It is calculated like this:

```
f[i]: The value of field 'f' in the i-th data-point
n: The number of data-points in the moving window (period)
t: The current time index

SMA(t) = (1/n) * Σ(f[i]), where i = t-n+1 to t
```

Example 1: Calculate simple moving average for a timeseries of temperatures

The example calculates the simple moving average over temperatures using two datapoints.

PPL query:

```
os> source=t | trendline sma(2, temperature) as temp_trend;
fetched rows / total rows = 5/5
+-----------+---------+--------------------+----------+
|temperature|device-id|           timestamp|temp_trend|
+-----------+---------+--------------------+----------+
|         12|     1492|2023-04-06 17:07:...|      NULL|
|         12|     1492|2023-04-06 17:07:...|      12.0|
|         13|      256|2023-04-06 17:07:...|      12.5|
|         14|      257|2023-04-06 17:07:...|      13.5|
|         15|      258|2023-04-06 17:07:...|      14.5|
+-----------+---------+--------------------+----------+
```

Example 2: Calculate simple moving averages for a timeseries of temperatures with sorting

The example calculates two simple moving average over temperatures using two and three datapoints sorted descending by device-id.

PPL query:

```
os> source=t | trendline sort - device-id sma(2, temperature) as temp_trend_2 sma(3,
 temperature) as temp_trend_3;
fetched rows / total rows = 5/5
+-----------+---------+--------------------+------------+------------------+
|temperature|device-id|           timestamp|temp_trend_2|      temp_trend_3|
+-----------+---------+--------------------+------------+------------------+
|         15|      258|2023-04-06 17:07:...|        NULL|              NULL|
|         14|      257|2023-04-06 17:07:...|        14.5|              NULL|
|         13|      256|2023-04-06 17:07:...|        13.5|              14.0|
|         12|     1492|2023-04-06 17:07:...|        12.5|              13.0|
|         12|     1492|2023-04-06 17:07:...|        12.0|12.333333333333334|
+-----------+---------+--------------------+------------+------------------+
```

where command

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called “Commands”.

The `where` command uses a bool-expression to filter the search result. It only returns the result when bool-expression evaluates to true.

Syntax

Use the following syntax:

```
where <boolean-expression>
```

bool-expression

- Optional.
- Any expression which could be evaluated to a boolean value.

Example 1: Filter result set with condition

The example shows how to fetch documents from the accounts index that meet specific conditions.

PPL query:

```
os> source=accounts | where account_number=1 or gender="F" | fields account_number,
 gender;
fetched rows / total rows = 2/2
+------------------+----------+
| account_number   | gender   |
|------------------+----------|
| 1                | M        |
| 13               | F        |
+------------------+----------+
```

Additional examples

Filters with logical conditions

- `source = table | where c = 'test' AND a = 1 | fields a,b,c`
- `source = table | where c != 'test' OR a > 1 | fields a,b,c | head 1`
- `source = table | where c = 'test' NOT a > 1 | fields a,b,c`
- `source = table | where a = 1 | fields a,b,c`
- `source = table | where a >= 1 | fields a,b,c`
- `source = table | where a < 1 | fields a,b,c`
- `source = table | where b != 'test' | fields a,b,c`
- `source = table | where c = 'test' | fields a,b,c | head 3`
- `source = table | where ispresent(b)`
- `source = table | where isnull(coalesce(a, b)) | fields a,b,c | head 3`
- `source = table | where isempty(a)`
- `source = table | where isblank(a)`
- `source = table | where case(length(a) > 6, 'True' else 'False') = 'True'`
- `source = table | where a between 1 and 4` - Note: This returns a >= 1 and a <= 4, i.e. [1, 4]
- `source = table | where b not between '2024-09-10' and '2025-09-10'` - Note: This returns b >= '**********' and b <= '2025-09-10'
- `source = table | where cidrmatch(ip, '***********/24')`
- `source = table | where cidrmatch(ipv6, '2003:db8::/32')`

```
source = table | eval status_category =
    case(a >= 200 AND a < 300, 'Success',
    a >= 300 AND a < 400, 'Redirection',
    a >= 400 AND a < 500, 'Client Error',
    a >= 500, 'Server Error'
    else 'Incorrect HTTP status code')
    | where case(a >= 200 AND a < 300, 'Success',
    a >= 300 AND a < 400, 'Redirection',
    a >= 400 AND a < 500, 'Client Error',
    a >= 500, 'Server Error'
    else 'Incorrect HTTP status code'
    ) = 'Incorrect HTTP status code'
```

```
source = table
    | eval factor = case(a > 15, a - 14, isnull(b), a - 7, a < 3, a + 1 else 1)
    | where case(factor = 2, 'even', factor = 4, 'even', factor = 6, 'even', factor =
 8, 'even' else 'odd') = 'even'
    |  stats count() by factor
```

field summary

> **Note**
>
> To see which AWS data source integrations support this PPL command, see the section called "Commands".

Use the `fieldsummary` command to calculate basic statistics for each field (count, distinct count, min, max, avg, stddev, mean) and determine the data type of each field. This command can be used with any preceding pipe and will take them into account.

Syntax

Use the following syntax. For CloudWatch Logs use cases, only one field in a query is supported.

```
... | fieldsummary <field-list> (nulls=true/false)
```

includefields

- List of all the columns to be collected with statistics into a unified result set.

Nulls

- Optional.
- If set to true, include null values in the aggregation calculations (replace null with zero for numeric values).

Example 1

PPL query:

```
os> source = t | where status_code != 200 | fieldsummary includefields= status_code
 nulls=true
+------------------+-------------+------------+------------+------------+------------
+------------+------------+----------------|
| Fields           | COUNT       | COUNT_DISTINCT    |  MIN  |  MAX   |  AVG   |  MEAN
  |        STDDEV       | NUlls | TYPEOF |
|------------------+-------------+------------+------------+------------+------------
+------------+------------+----------------|
| "status_code"    |      2      |         2         | 301   |   403  |  352.0 |  352.0
  |  72.12489168102785  |  0    | "int"  |
+------------------+-------------+------------+------------+------------+------------
+------------+------------+----------------|
```

Example 2

PPL query:

```
os> source = t | fieldsummary includefields= id, status_code, request_path nulls=true
+------------------+-------------+------------+------------+------------+------------
+------------+------------+----------------|
| Fields           | COUNT       | COUNT_DISTINCT    |  MIN  |  MAX   |  AVG   |  MEAN
  |        STDDEV       | NUlls | TYPEOF |
|------------------+-------------+------------+------------+------------+------------
+------------+------------+----------------|
|       "id"       |      6      |         6         | 1     |   6    |  3.5   |   3.5
 |  1.8708286933869707  |  0    | "int"  |
+------------------+-------------+------------+------------+------------+------------
+------------+------------+----------------|
| "status_code"    |      4      |         3         | 200   |   403  |  184.0 |  184.0
  |  161.16699413961905 |  2    | "int"  |
+------------------+-------------+------------+------------+------------+------------
+------------+------------+----------------|
| "request_path"   |      2      |         2         | /about| /home  |  0.0    |  0.0
    |      0            |  2    |"string"|
+------------------+-------------+------------+------------+------------+------------
+------------+------------+----------------|
```

expand command

> **Note**
>
> To see which AWS data source integrations support this PPL function, see the section called "Functions".

Use the expand command to flatten a field of type Array<Any> or Map<Any>, producing individual rows for each element or key-value pair.

Syntax

Use the following syntax:

```
expand <field> [As alias]
```

field

- The field to be expanded (exploded).
- The field must be of a supported type.

alias

- Optional.
- The name to be used instead of the original field name.

Usage guidelines

The expand command produces a row for each element in the specified array or map field, where:

- Array elements become individual rows.
- Map key-value pairs are broken into separate rows, with each key-value represented as a row.
- When an alias is provided, the exploded values are represented under the alias instead of the original field name.

You can use this command in combination with other commands, such as stats, eval, and parse, to manipulate or extract data post-expansion.

Examples

- `source = table | expand employee | stats max(salary) as max by state, company`
- `source = table | expand employee as worker | stats max(salary) as max by state, company`
- `source = table | expand employee as worker | eval bonus = salary * 3 | fields worker, bonus`
- `source = table | expand employee | parse description '(?<email>.+@.+)' | fields employee, email`
- `source = table | eval array=json_array(1, 2, 3) | expand array as uid | fields name, occupation, uid`
- `source = table | expand multi_valueA as multiA | expand multi_valueB as multiB`

You can use the expand command in combination with other commands such as eval, stats, and more. Using multiple expand commands will create a Cartesian product of all the internal elements within each composite array or map.

Effective SQL push-down query

The expand command is translated into an equivalent SQL operation using LATERAL VIEW explode, allowing for efficient exploding of arrays or maps at the SQL query level.

```
SELECT customer exploded_productId
FROM table
LATERAL VIEW explode(productId) AS exploded_productId
```

The explode command offers the following functionality:

- It is a column operation that returns a new column.
- It creates a new row for every element in the exploded column.
- Internal nulls are ignored as part of the exploded field (no row is created/exploded for null).

PPL functions

Topics

- PPL condition functions
- PPL cryptographic hash functions
- PPL date and time functions
- PPL expressions
- PPL IP address functions
- PPL JSON functions
- PPL Lambda functions
- PPL mathematical functions
- PPL string functions
- PPL type conversion functions

PPL condition functions

> **Note**
>
> To see which AWS data source integrations support this PPL function, see the section called "Functions".

ISNULL

Description: `isnull(field)` returns true if the field is null.

Argument type:

- All supported data types.

Return type:

- BOOLEAN

Example:

```
os> source=accounts | eval result = isnull(employer) | fields result, employer,
 firstname
fetched rows / total rows = 4/4
```

```
+----------+-------------+-------------+
| result   | employer    | firstname   |
|----------+-------------+-------------|
| False    | AnyCompany  | Mary        |
| False    | ExampleCorp | Jane        |
| False    | ExampleOrg  | Nikki       |
| True     | null        | Juan        |
+----------+-------------+-------------+
```

ISNOTNULL

Description: `isnotnull(field)` returns true if the field is not null.

Argument type:

- All supported data types.

Return type:

- BOOLEAN

Example:

```
os> source=accounts | where not isnotnull(employer) | fields account_number, employer
fetched rows / total rows = 1/1
+------------------+------------+
| account_number   | employer   |
|------------------+------------|
| 18               | null       |
+------------------+------------+
```

EXISTS

Example:

```
os> source=accounts | where exists(email) | fields account_number, email
fetched rows / total rows = 1/1
```

IFNULL

Description: `ifnull(field1, field2)` returns `field2` if `field1` is null.

Argument type:

- All supported data types.
- If the two parameters have different types, the function will fail the semantic check.

Return type:

- Any

Example:

```
os> source=accounts | eval result = ifnull(employer, 'default') | fields result,
 employer, firstname
fetched rows / total rows = 4/4
+------------+------------+-------------+
| result     | employer   | firstname   |
|------------+------------+-------------|
| AnyCompany | AnyCompany | Mary        |
| ExampleCorp| ExampleCorp| Jane        |
| ExampleOrg | ExampleOrg | Nikki       |
| default    | null       | Juan        |
+------------+------------+-------------+
```

NULLIF

Description: `nullif(field1, field2)` return null if two parameters are same, otherwise return field1.

Argument type:

- All supported data types.
- If the two parameters have different types, the function will fail the semantic check.

Return type:

- Any

Example:

```
os> source=accounts | eval result = nullif(employer, 'AnyCompany') | fields result,
 employer, firstname
fetched rows / total rows = 4/4
+----------------+----------------+-------------+
| result         | employer       | firstname   |
|----------------+----------------+-------------|
| null           | AnyCompany     | Mary        |
| ExampleCorp    | ExampleCorp    | Jane        |
| ExampleOrg     | ExampleOrg     | Nikki       |
| null           | null           | Juan        |
+----------------+----------------+-------------+
```

IF

Description: `if(condition, expr1, expr2)` returns `expr1` if the condition is true, otherwise it returns `expr2`.

Argument type:

- All supported data types.
- If the two parameters have different types, the function will fail the semantic check.

Return type:

- Any

Example:

```
os> source=accounts | eval result = if(true, firstname, lastname) | fields result,
 firstname, lastname
fetched rows / total rows = 4/4
+----------+-------------+----------+
| result   | firstname | lastname   |
|----------+-------------+----------|
| Jane     | Jane      | Doe        |
| Mary     | Mary      | Major      |
| Pat      | Pat       | Candella   |
| Dale     | Jorge     | Souza      |
+----------+-----------+------------+
```

```
os> source=accounts | eval result = if(false, firstname, lastname) | fields result,
 firstname, lastname
fetched rows / total rows = 4/4
+----------+-------------+------------+
| result   | firstname   | lastname   |
|----------+-------------+------------|
| Doe      | Jane        | Doe        |
| Major    | Mary        | Major      |
| Candella | Pat         | Candella   |
| Souza    | Jorge       | Souza      |
+----------+-------------+------------+

os> source=accounts | eval is_vip = if(age > 30 AND isnotnull(employer), true, false) |
 fields is_vip, firstname, lastname
fetched rows / total rows = 4/4
+----------+-------------+------------+
| is_vip   | firstname   | lastname   |
|----------+-------------+------------|
| True     | Jane        | Doe        |
| True     | Mary        | Major      |
| False    | Pat         | Candella   |
| False    | Jorge       | Souza      |
+----------+-------------+------------+
```

PPL cryptographic hash functions

> **Note**
>
> To see which AWS data source integrations support this PPL function, see the section called "Functions".

MD5

MD5 calculates the MD5 digest and returns the value as a 32 character hex string.

Usage: `md5('hello')`

Argument type:

- STRING

Return type:

- STRING

Example:

```
os> source=people | eval `MD5('hello')` = MD5('hello') | fields `MD5('hello')`
fetched rows / total rows = 1/1
+----------------------------------+
| MD5('hello')                     |
|----------------------------------|
| <32 character hex string>        |
+----------------------------------+
```

SHA1

SHA1 returns the hex string result of SHA-1.

Usage: `sha1('hello')`

Argument type:

- STRING

Return type:

- STRING

Example:

```
os> source=people | eval `SHA1('hello')` = SHA1('hello') | fields `SHA1('hello')`
fetched rows / total rows = 1/1
+------------------------------------------+
| SHA1('hello')                            |
|------------------------------------------|
| <40-character SHA-1 hash result>         |
+------------------------------------------+
```

SHA2

SHA2 returns the hex string result of SHA-2 family of hash functions (SHA-224, SHA-256, SHA-384, and SHA-512). The numBits indicates the desired bit length of the result, which must have a value of 224, 256, 384, 512

Usage:

- `sha2('hello',256)`
- `sha2('hello',512)`

Argument type:

- STRING, INTEGER

Return type:

- STRING

Example:

```
os> source=people | eval `SHA2('hello',256)` = SHA2('hello',256) | fields
 `SHA2('hello',256)`
fetched rows / total rows = 1/1
+------------------------------------------------------------------+
| SHA2('hello',256)                                                |
|------------------------------------------------------------------|
| <64-character SHA-256 hash result>                               |
+------------------------------------------------------------------+

os> source=people | eval `SHA2('hello',512)` = SHA2('hello',512) | fields
 `SHA2('hello',512)`
fetched rows / total rows = 1/1
+------------------------------------------------------------------+
| SHA2('hello',512)                                                |
                                          |
|------------------------------------------------------------------|
| <128-character SHA-512 hash result>                              |
+------------------------------------------------------------------+
```

PPL date and time functions

> **Note**
>
> To see which AWS data source integrations support this PPL function, see the section called "Functions".

DAY

Usage: DAY(date) extracts the day of the month for a date, in the range 1 to 31.

Argument type: STRING/DATE/TIMESTAMP

Return type: INTEGER

Synonyms: DAYOFMONTH, DAY_OF_MONTH

Example:

```
os> source=people | eval `DAY(DATE('2020-08-26'))` = DAY(DATE('2020-08-26')) | fields
 `DAY(DATE('2020-08-26'))`
fetched rows / total rows = 1/1
+---------------------------+
| DAY(DATE('2020-08-26'))   |
|---------------------------|
| 26                        |
+---------------------------+
```

DAYOFMONTH

Usage: DAYOFMONTH(date) extracts the day of the month for a date, in the range 1 to 31.

Argument type: STRING/DATE/TIMESTAMP

Return type: INTEGER

Synonyms: DAY, DAY_OF_MONTH

Example:

```
os> source=people | eval `DAYOFMONTH(DATE('2020-08-26'))` =
 DAYOFMONTH(DATE('2020-08-26')) | fields `DAYOFMONTH(DATE('2020-08-26'))`
```

```
fetched rows / total rows = 1/1
+----------------------------------+
| DAYOFMONTH(DATE('2020-08-26'))   |
|----------------------------------|
| 26                               |
+----------------------------------+
```

DAY_OF_MONTH

Usage: DAY_OF_MONTH(DATE) extracts the day of the month for a date, in the range 1 to 31.

Argument type: STRING/DATE/TIMESTAMP

Return type: INTEGER

Synonyms: DAY, DAYOFMONTH

Example:

```
os> source=people | eval `DAY_OF_MONTH(DATE('2020-08-26'))` =
 DAY_OF_MONTH(DATE('2020-08-26')) | fields `DAY_OF_MONTH(DATE('2020-08-26'))`
fetched rows / total rows = 1/1
+-----------------------------------+
| DAY_OF_MONTH(DATE('2020-08-26'))  |
|-----------------------------------|
| 26                                |
+-----------------------------------+
```

DAYOFWEEK

Usage: DAYOFWEEK(DATE) returns the weekday index for a date (1 = Sunday, 2 = Monday, ..., 7 = Saturday).

Argument type: STRING/DATE/TIMESTAMP

Return type: INTEGER

Synonyms: DAY_OF_WEEK

Example:

```
os> source=people | eval `DAYOFWEEK(DATE('2020-08-26'))` =
 DAYOFWEEK(DATE('2020-08-26')) | fields `DAYOFWEEK(DATE('2020-08-26'))`
```

```
fetched rows / total rows = 1/1
+---------------------------------+
| DAYOFWEEK(DATE('2020-08-26'))   |
|---------------------------------|
| 4                               |
+---------------------------------+
```

DAY_OF_WEEK

Usage: DAY_OF_WEEK(DATE) returns the weekday index for a date (1 = Sunday, 2 = Monday, ..., 7 = Saturday).

Argument type: STRING/DATE/TIMESTAMP

Return type: INTEGER

Synonyms: DAYOFWEEK

Example:

```
os> source=people | eval `DAY_OF_WEEK(DATE('2020-08-26'))` =
 DAY_OF_WEEK(DATE('2020-08-26')) | fields `DAY_OF_WEEK(DATE('2020-08-26'))`
fetched rows / total rows = 1/1
+----------------------------------+
| DAY_OF_WEEK(DATE('2020-08-26'))  |
|----------------------------------|
| 4                                |
+----------------------------------+
```

DAYOFYEAR

Usage: DAYOFYEAR(DATE) returns the day of the year for a date, in the range 1 to 366.

Argument type: STRING/DATE/TIMESTAMP

Return type: INTEGER

Synonyms: DAY_OF_YEAR

Example:

```
os> source=people | eval `DAYOFYEAR(DATE('2020-08-26'))` =
 DAYOFYEAR(DATE('2020-08-26')) | fields `DAYOFYEAR(DATE('2020-08-26'))`
```

```
fetched rows / total rows = 1/1
+--------------------------------+
| DAYOFYEAR(DATE('2020-08-26'))   |
|--------------------------------|
| 239                            |
+--------------------------------+
```

DAY_OF_YEAR

Usage: DAY_OF_YEAR(DATE) returns the day of the year for a date, in the range 1 to 366.

Argument type: STRING/DATE/TIMESTAMP

Return type: INTEGER

Synonyms: DAYOFYEAR

Example:

```
os> source=people | eval `DAY_OF_YEAR(DATE('2020-08-26'))` =
 DAY_OF_YEAR(DATE('2020-08-26')) | fields `DAY_OF_YEAR(DATE('2020-08-26'))`
fetched rows / total rows = 1/1
+----------------------------------+
| DAY_OF_YEAR(DATE('2020-08-26'))   |
|----------------------------------|
| 239                              |
+----------------------------------+
```

DAYNAME

Usage: DAYNAME(DATE) returns the name of the weekday for a date, including Monday, Tuesday, Wednesday, Thursday, Friday, Saturday and Sunday.

Argument type: STRING/DATE/TIMESTAMP

Return type: STRING

Example:

```
os> source=people | eval `DAYNAME(DATE('2020-08-26'))` = DAYNAME(DATE('2020-08-26')) |
 fields `DAYNAME(DATE('2020-08-26'))`
fetched rows / total rows = 1/1
+------------------------------+
```

```
| DAYNAME(DATE('2020-08-26'))   |
|-------------------------------|
| Wednesday                     |
+-------------------------------+
```

FROM_UNIXTIME

Usage: FROM_UNIXTIME returns a representation of the argument given as a timestamp or character string value. This function performs a reverse conversion of the UNIX_TIMESTAMP function.

If you provide a second argument, FROM_UNIXTIME uses it to format the result similar to the DATE_FORMAT function.

If the timestamp is outside of the range 1970-01-01 00:00:00 to 3001-01-18 23:59:59.999999 (0 to 32536771199.999999 epoch time), the function returns NULL.

Argument type: DOUBLE, STRING

Return type map:

DOUBLE -> TIMESTAMP

DOUBLE, STRING -> STRING

Examples:

```
os> source=people | eval `FROM_UNIXTIME(1220249547)` = FROM_UNIXTIME(1220249547) |
 fields `FROM_UNIXTIME(1220249547)`
fetched rows / total rows = 1/1
+-----------------------------+
| FROM_UNIXTIME(1220249547)   |
|-----------------------------|
| 2008-09-01 06:12:27         |
+-----------------------------+

os> source=people | eval `FROM_UNIXTIME(1220249547, 'HH:mm:ss')` =
 FROM_UNIXTIME(1220249547, 'HH:mm:ss') | fields `FROM_UNIXTIME(1220249547, 'HH:mm:ss')`
fetched rows / total rows = 1/1
+---------------------------------------+
| FROM_UNIXTIME(1220249547, 'HH:mm:ss')   |
|---------------------------------------|
| 06:12:27                              |
```

```
+----------------------------------------+
```

HOUR

Usage: HOUR(TIME) extracts the hour value for time.

Unlike a standard time of day, the time value in this function can have a range larger than 23. As a result, the return value of HOUR(TIME) can be greater than 23.

Argument type: STRING/TIME/TIMESTAMP

Return type: INTEGER

Synonyms: HOUR_OF_DAY

Example:

```
os> source=people | eval `HOUR(TIME('01:02:03'))` = HOUR(TIME('01:02:03')) | fields
 `HOUR(TIME('01:02:03'))`
fetched rows / total rows = 1/1
+--------------------------+
| HOUR(TIME('01:02:03'))   |
|--------------------------|
| 1                        |
+--------------------------+
```

HOUR_OF_DAY

Usage: HOUR_OF_DAY(TIME) extracts the hour value from the given time.

Unlike a standard time of day, the time value in this function can have a range larger than 23. As a result, the return value of HOUR_OF_DAY(TIME) can be greater than 23.

Argument type: STRING/TIME/TIMESTAMP

Return type: INTEGER

Synonyms: HOUR

Example:

```
os> source=people | eval `HOUR_OF_DAY(TIME('01:02:03'))` =
 HOUR_OF_DAY(TIME('01:02:03')) | fields `HOUR_OF_DAY(TIME('01:02:03'))`
fetched rows / total rows = 1/1
```

```
+---------------------------------+
| HOUR_OF_DAY(TIME('01:02:03'))   |
|---------------------------------|
| 1                               |
+---------------------------------+
```

LAST_DAY

Usage: LAST_DAY returns the last day of the month as a DATE value for the given date argument.

Argument type: DATE/STRING/TIMESTAMP/TIME

Return type: DATE

Example:

```
os> source=people | eval `last_day('2023-02-06')` = last_day('2023-02-06') | fields
 `last_day('2023-02-06')`
fetched rows / total rows = 1/1
+-------------------------+
| last_day('2023-02-06')  |
|-------------------------|
| 2023-02-28              |
+-------------------------+
```

LOCALTIMESTAMP

Usage: LOCALTIMESTAMP() is a synonyms for NOW().

Example:

```
> source=people | eval `LOCALTIMESTAMP()` = LOCALTIMESTAMP() | fields
 `LOCALTIMESTAMP()`
fetched rows / total rows = 1/1
+---------------------+
| LOCALTIMESTAMP()    |
|---------------------|
| 2022-08-02 15:54:19 |
+---------------------+
```

LOCALTIME

Usage: LOCALTIME() is a synonym for NOW().

Example:

```
> source=people | eval `LOCALTIME()` = LOCALTIME() | fields `LOCALTIME()`
fetched rows / total rows = 1/1
+---------------------+
| LOCALTIME()         |
|---------------------|
| 2022-08-02 15:54:19 |
+---------------------+
```

MAKE_DATE

Usage: MAKE_DATE returns a date value based on the given year, month, and day values. All arguments are rounded to integers.

Specifications: 1. MAKE_DATE(INTEGER, INTEGER, INTEGER) -> DATE

Argument type: INTEGER, INTEGER, INTEGER

Return type: DATE

Example:

```
os> source=people | eval `MAKE_DATE(1945, 5, 9)` = MAKEDATE(1945, 5, 9) | fields
 `MAKEDATE(1945, 5, 9)`
fetched rows / total rows = 1/1
+------------------------+
| MAKEDATE(1945, 5, 9)   |
|------------------------|
| 1945-05-09             |
+------------------------+
```

MINUTE

Usage: MINUTE(TIME) returns the minute component of the given time, as an integer in the range 0 to 59.

Argument type: STRING/TIME/TIMESTAMP

Return type: INTEGER

Synonyms: MINUTE_OF_HOUR

Example:

```
os> source=people | eval `MINUTE(TIME('01:02:03'))` =  MINUTE(TIME('01:02:03')) |
 fields `MINUTE(TIME('01:02:03'))`
fetched rows / total rows = 1/1
+----------------------------+
| MINUTE(TIME('01:02:03'))   |
|----------------------------|
| 2                          |
+----------------------------+
```

MINUTE_OF_HOUR

Usage: `MINUTE_OF_HOUR(TIME)` returns the minute component of the given time, as an integer in the range 0 to 59.

Argument type: STRING/TIME/TIMESTAMP

Return type: INTEGER

Synonyms: `MINUTE`

Example:

```
os> source=people | eval `MINUTE_OF_HOUR(TIME('01:02:03'))` =
 MINUTE_OF_HOUR(TIME('01:02:03')) | fields `MINUTE_OF_HOUR(TIME('01:02:03'))`
fetched rows / total rows = 1/1
+-----------------------------------+
| MINUTE_OF_HOUR(TIME('01:02:03'))   |
|-----------------------------------|
| 2                                 |
+-----------------------------------+
```

MONTH

Usage: `MONTH(DATE)` returns the month of the given date as an integer, in the range 1 to 12 (where 1 represents January and 12 represents December).

Argument type: STRING/DATE/TIMESTAMP

Return type: INTEGER

Synonyms: `MONTH_OF_YEAR`

Example:

```
os> source=people | eval `MONTH(DATE('2020-08-26'))` =  MONTH(DATE('2020-08-26')) |
 fields `MONTH(DATE('2020-08-26'))`
fetched rows / total rows = 1/1
+-----------------------------+
| MONTH(DATE('2020-08-26'))   |
|-----------------------------|
| 8                           |
+-----------------------------+
```

MONTHNAME

Usage: `MONTHNAME(DATE)` returns the month of the given date as an integer, in the range 1 to 12 (where 1 represents January and 12 represents December).

Argument type: STRING/DATE/TIMESTAMP

Return type: INTEGER

Synonyms: `MONTH_OF_YEAR`

Example:

```
os> source=people | eval `MONTHNAME(DATE('2020-08-26'))` =
 MONTHNAME(DATE('2020-08-26')) | fields `MONTHNAME(DATE('2020-08-26'))`
fetched rows / total rows = 1/1
+---------------------------------+
| MONTHNAME(DATE('2020-08-26'))   |
|---------------------------------|
| August                          |
+---------------------------------+
```

MONTH_OF_YEAR

Usage: `MONTH_OF_YEAR(DATE)`returns the month of the given date as an integer, in the range 1 to 12 (where 1 represents January and 12 represents December).

Argument type: STRING/DATE/TIMESTAMP

Return type: INTEGER

Synonyms: `MONTH`

Example:

```
os> source=people | eval `MONTH_OF_YEAR(DATE('2020-08-26'))` =
 MONTH_OF_YEAR(DATE('2020-08-26')) | fields `MONTH_OF_YEAR(DATE('2020-08-26'))`
fetched rows / total rows = 1/1
+------------------------------------+
| MONTH_OF_YEAR(DATE('2020-08-26'))  |
|------------------------------------|
| 8                                  |
+------------------------------------+
```

NOW

Usage: NOW returns the current date and time as a TIMESTAMP value in the 'YYYY-MM-DD hh:mm:ss' format. The value is expressed in the cluster time zone.

> **Note**
>
> NOW( ) returns a constant time that indicates when the statement began to execute. This differs from SYSDATE( ), which returns the exact time of execution.

Return type: TIMESTAMP

Specification: NOW() -> TIMESTAMP

Example:

```
os> source=people | eval `value_1` = NOW(), `value_2` = NOW() | fields `value_1`,
 `value_2`
fetched rows / total rows = 1/1
+---------------------+---------------------+
| value_1             | value_2             |
|---------------------+---------------------|
| 2022-08-02 15:39:05 | 2022-08-02 15:39:05 |
+---------------------+---------------------+
```

QUARTER

Usage: QUARTER(DATE) returns the quarter of the year for the given date as an integer, in the range 1 to 4.

Argument type: STRING/DATE/TIMESTAMP

Return type: INTEGER

Example:

```
os> source=people | eval `QUARTER(DATE('2020-08-26'))` = QUARTER(DATE('2020-08-26')) |
 fields `QUARTER(DATE('2020-08-26'))`
fetched rows / total rows = 1/1
+------------------------------+
| QUARTER(DATE('2020-08-26'))  |
|------------------------------|
| 3                            |
+------------------------------+
```

SECOND

Usage: `SECOND(TIME)` returns the second component of the given time as an integer, in the range 0 to 59.

Argument type: STRING/TIME/TIMESTAMP

Return type: INTEGER

Synonyms: `SECOND_OF_MINUTE`

Example:

```
os> source=people | eval `SECOND(TIME('01:02:03'))` = SECOND(TIME('01:02:03')) | fields
 `SECOND(TIME('01:02:03'))`
fetched rows / total rows = 1/1
+---------------------------+
| SECOND(TIME('01:02:03'))  |
|---------------------------|
| 3                         |
+---------------------------+
```

SECOND_OF_MINUTE

Usage: `SECOND_OF_MINUTE(TIME)` returns the second component of the given time as an integer, in the range 0 to 59.

Argument type: STRING/TIME/TIMESTAMP

Return type: INTEGER

Synonyms: SECOND

Example:

```
os> source=people | eval `SECOND_OF_MINUTE(TIME('01:02:03'))` =
 SECOND_OF_MINUTE(TIME('01:02:03')) | fields `SECOND_OF_MINUTE(TIME('01:02:03'))`
fetched rows / total rows = 1/1
+-------------------------------------+
| SECOND_OF_MINUTE(TIME('01:02:03'))  |
|-------------------------------------|
| 3                                   |
+-------------------------------------+
```

SUBDATE

Usage: SUBDATE(DATE, DAYS) subtracts the second argument (such as DATE or DAYS) from the given date.

Argument type: DATE/TIMESTAMP, LONG

Return type map: (DATE, LONG) -> DATE

Antonyms: ADDDATE

Example:

```
os> source=people | eval `'2008-01-02' - 31d` = SUBDATE(DATE('2008-01-02'), 31),
 `'2020-08-26' - 1` = SUBDATE(DATE('2020-08-26'), 1), `ts '2020-08-26 01:01:01' -
 1` = SUBDATE(TIMESTAMP('2020-08-26 01:01:01'), 1) | fields `'2008-01-02' - 31d`,
 `'2020-08-26' - 1`, `ts '2020-08-26 01:01:01' - 1`
fetched rows / total rows = 1/1
+---------------------+--------------------+--------------------------------+
| '2008-01-02' - 31d  | '2020-08-26' - 1   | ts '2020-08-26 01:01:01' - 1   |
|---------------------+--------------------+--------------------------------|
| 2007-12-02 00:00:00 | 2020-08-25         | 2020-08-25 01:01:01            |
+---------------------+--------------------+--------------------------------+
```

SYSDATE

Usage: SYSDATE() returns the current date and time as a TIMESTAMP value in the 'YYYY-MM-DD hh:mm:ss.nnnnnn' format.

SYSDATE( )returns the exact time at which it executes. This differs from NOW(), which returns a constant time indicating when the statement began to execute.

Optional argument type: INTEGER (0 to 6) - Specifies the number of digits for fractional seconds in the return value.

Return type: TIMESTAMP

Example:

```
os> source=people | eval `SYSDATE()` = SYSDATE() | fields `SYSDATE()`
fetched rows / total rows = 1/1
+----------------------------+
| SYSDATE()                  |
|----------------------------|
| 2022-08-02 15:39:05.123456 |
+----------------------------+
```

`TIMESTAMP`

Usage: `TIMESTAMP(EXPR)` constructs a timestamp type with the input string `expr` as an timestamp.

With a single argument, `TIMESTAMP(expr)` constructs a timestamp from the input. If `expr` is a string, it's interpreted as a timestamp. For non-string arguments, the function casts `expr` to a timestamp using the UTC timezone. When `expr` is a `TIME` value, the function applies today's date before casting.

When used with two arguments, `TIMESTAMP(expr1, expr2)` adds the time expression (`expr2`) to the date or timestamp expression (`expr1`) and returns the result as a timestamp value.

Argument type: STRING/DATE/TIME/TIMESTAMP

Return type map:

(STRING/DATE/TIME/TIMESTAMP) -> TIMESTAMP

(STRING/DATE/TIME/TIMESTAMP, STRING/DATE/TIME/TIMESTAMP) -> TIMESTAMP

Example:

```
os> source=people | eval `TIMESTAMP('2020-08-26 13:49:00')` = TIMESTAMP('2020-08-26
 13:49:00'), `TIMESTAMP('2020-08-26 13:49:00', TIME('12:15:42'))` =
```

```
 TIMESTAMP('2020-08-26 13:49:00', TIME('12:15:42')) | fields `TIMESTAMP('2020-08-26
 13:49:00')`, `TIMESTAMP('2020-08-26 13:49:00', TIME('12:15:42'))`
fetched rows / total rows = 1/1
+-----------------------------------
+-----------------------------------------------------+
| TIMESTAMP('2020-08-26 13:49:00')   | TIMESTAMP('2020-08-26 13:49:00',
 TIME('12:15:42'))   |
|-----------------------------------
+-----------------------------------------------------|
| 2020-08-26 13:49:00                | 2020-08-27 02:04:42
     |
+-----------------------------------
+-----------------------------------------------------+
```

UNIX_TIMESTAMP

Usage: UNIX_TIMESTAMP converts a given date argument to Unix time (seconds since the Epoch, which began at the start of 1970). If no argument is provided, it returns the current Unix time.

The date argument can be a DATE, a TIMESTAMP string, or a number in one of these formats: YYMMDD, YYMMDDhhmmss, YYYYMMDD, or YYYYMMDDhhmmss. If the argument includes a time component, it may optionally include fractional seconds.

If the argument is in an invalid format or falls outside the range of 1970-01-01 00:00:00 to 3001-01-18 23:59:59.999999 (0 to 32536771199.999999 in epoch time), the function returns NULL.

The function accepts DATE, TIMESTAMP, or DOUBLE as argument types, or no argument. It always returns a DOUBLE value representing the Unix timestamp.

For the reverse conversion, you can use the FROM_UNIXTIME function.

Argument type: <NONE>/DOUBLE/DATE/TIMESTAMP

Return type: DOUBLE

Example:

```
os> source=people | eval `UNIX_TIMESTAMP(double)` = UNIX_TIMESTAMP(20771122143845),
 `UNIX_TIMESTAMP(timestamp)` = UNIX_TIMESTAMP(TIMESTAMP('1996-11-15 17:05:42')) |
 fields `UNIX_TIMESTAMP(double)`, `UNIX_TIMESTAMP(timestamp)`
fetched rows / total rows = 1/1
+-------------------------+----------------------------+
```

```
| UNIX_TIMESTAMP(double)   | UNIX_TIMESTAMP(timestamp)   |
|--------------------------+-----------------------------|
| 3404817525.0             | 848077542.0                 |
+--------------------------+-----------------------------+
```

WEEK

Usage: WEEK(DATE) returns the week number for a given date.

Argument type: DATE/TIMESTAMP/STRING

Return type: INTEGER

Synonyms: WEEK_OF_YEAR

Example:

```
os> source=people | eval `WEEK(DATE('2008-02-20'))` = WEEK(DATE('2008-02-20')) | fields
 `WEEK(DATE('2008-02-20'))`
fetched rows / total rows = 1/1
+----------------------------+
| WEEK(DATE('2008-02-20'))   |
|----------------------------|
| 8                          |
+----------------------------+
```

WEEKDAY

Usage: WEEKDAY(DATE) returns the weekday index for date (0 = Monday, 1 = Tuesday, ..., 6 = Sunday).

It is similar to the dayofweek function, but returns different indexes for each day.

Argument type: STRING/DATE/TIME/TIMESTAMP

Return type: INTEGER

Example:

```
os> source=people | eval `weekday(DATE('2020-08-26'))` = weekday(DATE('2020-08-26'))
 | eval `weekday(DATE('2020-08-27'))` = weekday(DATE('2020-08-27')) | fields
 `weekday(DATE('2020-08-26'))`, `weekday(DATE('2020-08-27'))`
fetched rows / total rows = 1/1
```

```
+------------------------------+------------------------------+
| weekday(DATE('2020-08-26'))   | weekday(DATE('2020-08-27'))   |
|------------------------------+------------------------------|
| 2                            | 3                            |
+------------------------------+------------------------------+
```

WEEK_OF_YEAR

Usage: WEEK_OF_YEAR(DATE) returns the week number for the given date.

Argument type: DATE/TIMESTAMP/STRING

Return type: INTEGER

Synonyms: WEEK

Example:

```
os> source=people | eval `WEEK_OF_YEAR(DATE('2008-02-20'))` = WEEK(DATE('2008-02-20'))|
 fields `WEEK_OF_YEAR(DATE('2008-02-20'))`
fetched rows / total rows = 1/1
+-----------------------------------+
| WEEK_OF_YEAR(DATE('2008-02-20'))   |
|-----------------------------------|
| 8                                 |
+-----------------------------------+
```

YEAR

Usage: YEAR(DATE) returns the year for date, in the range 1000 to 9999, or 0 for the "zero" date.

Argument type: STRING/DATE/TIMESTAMP

Return type: INTEGER

Example:

```
os> source=people | eval `YEAR(DATE('2020-08-26'))` = YEAR(DATE('2020-08-26')) | fields
 `YEAR(DATE('2020-08-26'))`
fetched rows / total rows = 1/1
+---------------------------+
| YEAR(DATE('2020-08-26'))   |
|---------------------------|
| 2020                      |
```

```
+---------------------------+
```

DATE_ADD

Usage: `DATE_ADD(date, INTERVAL expr unit)` adds the specified interval to the given date.

Argument type: DATE, INTERVAL

Return type: DATE

Antonyms: DATE_SUB

Example:

```
os> source=people | eval `'2020-08-26' + 1d` = DATE_ADD(DATE('2020-08-26'), INTERVAL 1
 DAY) | fields `'2020-08-26' + 1d`
fetched rows / total rows = 1/1
+---------------------+
| '2020-08-26' + 1d   |
|---------------------|
| 2020-08-27          |
+---------------------+
```

DATE_SUB

Usage: `DATE_SUB(date, INTERVAL expr unit)` subtracts the interval expr from date.

Argument type: DATE, INTERVAL

Return type: DATE

Antonyms: DATE_ADD

Example:

```
os> source=people | eval `'2008-01-02' - 31d` = DATE_SUB(DATE('2008-01-02'), INTERVAL
 31 DAY) | fields `'2008-01-02' - 31d`
fetched rows / total rows = 1/1
+---------------------+
| '2008-01-02' - 31d  |
|---------------------|
| 2007-12-02          |
+---------------------+
```

`TIMESTAMPADD`

Usage: Returns a `TIMESTAMP` value after adding a specified time interval to a given date.

Arguments:

- interval: INTERVAL (SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR)
- integer: INTEGER
- date: DATE, TIMESTAMP, or STRING

If you provide a `STRING` as the date argument, format it as a valid `TIMESTAMP`. The function automatically converts a `DATE` argument to a `TIMESTAMP`.

Examples:

```
os> source=people | eval `TIMESTAMPADD(DAY, 17, '2000-01-01 00:00:00')` =
 TIMESTAMPADD(DAY, 17, '2000-01-01 00:00:00') | eval `TIMESTAMPADD(QUARTER, -1,
 '2000-01-01 00:00:00')` = TIMESTAMPADD(QUARTER, -1, '2000-01-01 00:00:00') | fields
 `TIMESTAMPADD(DAY, 17, '2000-01-01 00:00:00')`, `TIMESTAMPADD(QUARTER, -1, '2000-01-01
 00:00:00')`
fetched rows / total rows = 1/1
+-----------------------------------------------
+-------------------------------------------------+
| TIMESTAMPADD(DAY, 17, '2000-01-01 00:00:00') | TIMESTAMPADD(QUARTER, -1, '2000-01-01
 00:00:00') |
|-----------------------------------------------
+-------------------------------------------------|
| 2000-01-18 00:00:00                          | 1999-10-01 00:00:00
           |
+-----------------------------------------------
+-------------------------------------------------+
```

`TIMESTAMPDIFF`

Usage: `TIMESTAMPDIFF(interval, start, end)` returns the difference between the start and end date/times in specified interval units.

Arguments:

- interval: INTERVAL (SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR)
- start: DATE, TIMESTAMP, or STRING
- end: DATE, TIMESTAMP, or STRING

The function automatically converts arguments to TIMESTAMP when appropriate. Format STRING arguments as valid TIMESTAMPs.

Examples:

```
os> source=people | eval `TIMESTAMPDIFF(YEAR, '1997-01-01 00:00:00', '2001-03-06
 00:00:00')` = TIMESTAMPDIFF(YEAR, '1997-01-01 00:00:00', '2001-03-06 00:00:00') |
 eval `TIMESTAMPDIFF(SECOND, timestamp('1997-01-01 00:00:23'), timestamp('1997-01-01
 00:00:00'))` = TIMESTAMPDIFF(SECOND, timestamp('1997-01-01 00:00:23'),
 timestamp('1997-01-01 00:00:00')) | fields `TIMESTAMPDIFF(YEAR, '1997-01-01 00:00:00',
 '2001-03-06 00:00:00')`, `TIMESTAMPDIFF(SECOND, timestamp('1997-01-01 00:00:23'),
 timestamp('1997-01-01 00:00:00'))`
fetched rows / total rows = 1/1
+-------------------------------------------------------------------
+-------------------------------------------------------------------------------------------
+
| TIMESTAMPDIFF(YEAR, '1997-01-01 00:00:00', '2001-03-06 00:00:00') |
 TIMESTAMPDIFF(SECOND, timestamp('1997-01-01 00:00:23'), timestamp('1997-01-01
 00:00:00')) |
|-------------------------------------------------------------------
+-------------------------------------------------------------------------------------------|
| 4                                                                 | -23
                                                                        |
+-------------------------------------------------------------------
+-------------------------------------------------------------------------------------------
+
```

UTC_TIMESTAMP

Usage: UTC_TIMESTAMP returns the current UTC timestamp as a value in 'YYYY-MM-DD hh:mm:ss'.

Return type: TIMESTAMP

Specification: UTC_TIMESTAMP() -> TIMESTAMP

Example:

```
> source=people | eval `UTC_TIMESTAMP()` = UTC_TIMESTAMP() | fields `UTC_TIMESTAMP()`
```

```
fetched rows / total rows = 1/1
+---------------------+
| UTC_TIMESTAMP()     |
|---------------------|
| 2022-10-03 17:54:28 |
+---------------------+
```

CURRENT_TIMEZONE

Usage: CURRENT_TIMEZONE returns the current local timezone.

Return type: STRING

Example:

```
> source=people | eval `CURRENT_TIMEZONE()` = CURRENT_TIMEZONE() | fields
 `CURRENT_TIMEZONE()`
fetched rows / total rows = 1/1
+------------------------+
| CURRENT_TIMEZONE()     |
|------------------------|
| America/Chicago        |
+------------------------+
```

PPL expressions

> **Note**
>
> To see which AWS data source integrations support this PPL function, see the section called "Functions".

Expressions, particularly value expressions, return a scalar value. Expressions have different types and forms. For example, there are literal values as atom expressions and arithmetic, predicate and function expressions built on top of them. You can use expressions in different clauses, such as using arithmetic expressions in `Filter` and `Stats` commands.

Operators

An arithmetic expression is an expression formed by numeric literals and binary arithmetic operators as follows:

1. +: Add.
2. -: Subtract.
3. *: Multiply.
4. /: Divide (For integers, the result is an integer with the fractional part discarded)
5. %: Modulo (Use with integers only; the result is the remainder of the division)

Precedence

Use parentheses to control the precedence of arithmetic operators. Otherwise, operators of higher precedence are performed first.

Type conversion

Implicit type conversion is performed when looking up operator signatures. For example, an integer + a real number matches signature `+(double,double)` which results in a real number. This rule also applies to function calls.

Example for different type of arithmetic expressions:

```
os> source=accounts | where age > (25 + 5) | fields age ;
fetched rows / total rows = 3/3
+-------+
| age   |
|-------|
| 32    |
| 36    |
| 33    |
+-------+
```

Predicate operators

A predicate operator is an expression that evaluates to be true. The `MISSING` and `NULL` value comparison follow these rules:

- A `MISSING` value only equals a `MISSING` value and is less than other values.
- A `NULL` value equals a `NULL` value, is larger than a `MISSING` value, but is less than all other values.

Operators

Predicate operators

Name	Description
>	Greater than operator
>=	Greater than or equal operator
<	Less than operator
!=	Not equal operator
<=	Less than or equal operator
=	Equal operator
LIKE	Simple pattern matching
IN	NULL value test
AND	AND operator
OR	OR operator
XOR	XOR operator
NOT	NOT NULL value test

You can compare datetimes. When comparing different datetime types (for example DATE and TIME), both convert to DATETIME. The following rules apply to conversion:

- TIME applies to today's date.
- DATE is interpreted at midnight.

Basic predicate operator

Example for comparison operators:

```
os> source=accounts | where age > 33 | fields age ;
fetched rows / total rows = 1/1
+-------+
```

```
| age   |
|-------|
| 36    |
+-------+
```

IN

Example of the `IN` operator test field in value lists:

```
os> source=accounts | where age in (32, 33) | fields age ;
fetched rows / total rows = 2/2
+-------+
| age   |
|-------|
| 32    |
| 33    |
+-------+
```

OR

Example of the `OR` operator:

```
os> source=accounts | where age = 32 OR age = 33 | fields age ;
fetched rows / total rows = 2/2
+-------+
| age   |
|-------|
| 32    |
| 33    |
+-------+
```

NOT

Example of the `NOT` operator:

```
os> source=accounts | where age not in (32, 33) | fields age ;
fetched rows / total rows = 2/2
+-------+
| age   |
|-------|
| 36    |
| 28    |
```

```
+-------+
```

PPL IP address functions

> **Note**
>
> To see which AWS data source integrations support this PPL function, see the section called "Functions".

`CIDRMATCH`

Usage: `CIDRMATCH(ip, cidr)` checks if the specified IP address is within the given cidr range.

Argument type:

- STRING, STRING
- Return type: BOOLEAN

Example:

```
os> source=ips | where cidrmatch(ip, '***********/24') | fields ip
fetched rows / total rows = 1/1
+--------------+
| ip           |
|--------------|
| ***********  |
+--------------+

os> source=ipsv6 | where cidrmatch(ip, '2003:db8::/32') | fields ip
fetched rows / total rows = 1/1
+----------------------------------------+
| ip                                     |
|----------------------------------------|
| 2003:0db8:****:****:****:****:****:0000 |
+----------------------------------------+
```

> **Note**
>
> - `ip` can be an IPv4 or an IPv6 address.

- `cidr` can be an IPv4 or an IPv6 block.
- `ip` and `cidr` must be either both IPv4 or both IPv6.
- `ip` and `cidr` must both be valid and non-empty/non-null.

PPL JSON functions

> **Note**
>
> To see which AWS data source integrations support this PPL function, see the section called "Functions".

`JSON`

Usage: `json(value)` evaluates whether a string can be parsed as JSON format. The function returns the original string if it's valid JSON, or null if it's invalid.

Argument type: STRING

Return type: STRING/NULL. A STRING expression of a valid JSON object format.

Examples:

```
os> source=people | eval `valid_json()` = json('[1,2,3,{"f1":1,"f2":[5,6]},4]') |
 fields valid_json
fetched rows / total rows = 1/1
+---------------------------------+
| valid_json                      |
+---------------------------------+
| [1,2,3,{"f1":1,"f2":[5,6]},4]   |
+---------------------------------+

os> source=people | eval `invalid_json()` = json('{"invalid": "json"') | fields
 invalid_json
fetched rows / total rows = 1/1
+----------------+
| invalid_json   |
+----------------+
| null           |
```

```
+----------------+
```

JSON_OBJECT

Usage: `json_object(<key>, <value>[, <key>, <value>]...)` returns a JSON object from members of key-value pairs.

Argument type:

- A <key> must be STRING.
- A <value> can be any data types.

Return type: JSON_OBJECT. A StructType expression of a valid JSON object.

Examples:

```
os> source=people | eval result = json_object('key', 123.45) | fields result
fetched rows / total rows = 1/1
+------------------+
| result           |
+------------------+
| {"key":123.45}   |
+------------------+

os> source=people | eval result = json_object('outer', json_object('inner', 123.45)) |
 fields result
fetched rows / total rows = 1/1
+-----------------------------+
| result                      |
+-----------------------------+
| {"outer":{"inner":123.45}}  |
+-----------------------------+
```

JSON_ARRAY

Usage: `json_array(<value>...)` creates a JSON ARRAY using a list of values.

Argument type: A `<value>` can be any kind of value such as string, number, or boolean.

Return type: ARRAY. An array of any supported data type for a valid JSON array.

Examples:

```
os> source=people | eval `json_array` = json_array(1, 2, 0, -1, 1.1, -0.11)
fetched rows / total rows = 1/1
+------------------------------+
| json_array                   |
+------------------------------+
| [1.0,2.0,0.0,-1.0,1.1,-0.11] |
+------------------------------+

os> source=people | eval `json_array_object` = json_object("array", json_array(1, 2, 0,
 -1, 1.1, -0.11))
fetched rows / total rows = 1/1
+---------------------------------------+
| json_array_object                     |
+---------------------------------------+
| {"array":[1.0,2.0,0.0,-1.0,1.1,-0.11]} |
+---------------------------------------+
```

TO_JSON_STRING

Usage: `to_json_string(jsonObject)` returns a JSON string with a given json object value.

Argument type: JSON_OBJECT

Return type: STRING

Examples:

```
os> source=people | eval `json_string` = to_json_string(json_array(1, 2, 0, -1, 1.1,
 -0.11)) | fields json_string
fetched rows / total rows = 1/1
+-------------------------------+
| json_string                   |
+-------------------------------+
| [1.0,2.0,0.0,-1.0,1.1,-0.11]  |
+-------------------------------+

os> source=people | eval `json_string` = to_json_string(json_object('key', 123.45)) |
 fields json_string
fetched rows / total rows = 1/1
+-----------------+
| json_string     |
+-----------------+
| {'key', 123.45} |
```

```
+-----------------+
```

ARRAY_LENGTH

Usage: `array_length(jsonArray)` returns the number of elements in the outermost array.

Argument type: ARRAY. An ARRAY or JSON_ARRAY object.

Return type: INTEGER

Example:

```
os> source=people | eval `json_array` = json_array_length(json_array(1,2,3,4)),
 `empty_array` = json_array_length(json_array())
fetched rows / total rows = 1/1
+--------------+---------------+
| json_array   | empty_array   |
+--------------+---------------+
| 4            | 0             |
+--------------+---------------+
```

JSON_EXTRACT

Usage: `json_extract(jsonStr, path)` extracts a JSON object from a JSON string based on the specified JSON path. The function returns null if the input JSON string is invalid.

Argument type: STRING, STRING

Return type: STRING

- A STRING expression of a valid JSON object format.
- NULL is returned in case of an invalid JSON.

Examples:

```
os> source=people | eval `json_extract('{"a":"b"}', '$.a')` = json_extract('{"a":"b"}',
 '$a')
fetched rows / total rows = 1/1
+---------------------------------+
| json_extract('{"a":"b"}', 'a')   |
```

```
+---------------------------------+
| b                               |
+---------------------------------+

os> source=people | eval `json_extract('{"a":[{"b":1},{"b":2}]}', '$.a[1].b')` =
 json_extract('{"a":[{"b":1},{"b":2}]}', '$.a[1].b')
fetched rows / total rows = 1/1
+-------------------------------------------------------------+
| json_extract('{"a":[{"b":1.0},{"b":2.0}]}', '$.a[1].b')   |
+-------------------------------------------------------------+
| 2.0                                                         |
+-------------------------------------------------------------+

os> source=people | eval `json_extract('{"a":[{"b":1},{"b":2}]}', '$.a[*].b')` =
 json_extract('{"a":[{"b":1},{"b":2}]}', '$.a[*].b')
fetched rows / total rows = 1/1
+-------------------------------------------------------------+
| json_extract('{"a":[{"b":1.0},{"b":2.0}]}', '$.a[*].b')   |
+-------------------------------------------------------------+
| [1.0,2.0]                                                   |
+-------------------------------------------------------------+

os> source=people | eval `invalid_json` = json_extract('{"invalid": "json"')
fetched rows / total rows = 1/1
+----------------+
| invalid_json   |
+----------------+
| null           |
+----------------+
```

JSON_KEYS

Usage: `json_keys(jsonStr)` returns all the keys of the outermost JSON object as an array.

Argument type: STRING. A STRING expression of a valid JSON object format.

Return type: ARRAY[STRING]. The function returns `NULL` for any other valid JSON string, an empty string, or an invalid JSON.

Examples:

```
os> source=people | eval `keys` = json_keys('{"f1":"abc","f2":{"f3":"a","f4":"b"}}')
fetched rows / total rows = 1/1
```

```
+------------+
| keus       |
+------------+
| [f1, f2]   |
+------------+

os> source=people | eval `keys` = json_keys('[1,2,3,{"f1":1,"f2":[5,6]},4]')
fetched rows / total rows = 1/1
+--------+
| keys   |
+--------+
| null   |
+--------+
```

JSON_VALID

Usage: `json_valid(jsonStr)` evaluates whether a JSON string uses valid JSON syntax and returns TRUE or FALSE.

Argument type: STRING

Return type: BOOLEAN

Examples:

```
os> source=people | eval `valid_json` = json_valid('[1,2,3,4]'), `invalid_json` =
 json_valid('{"invalid": "json"') | feilds `valid_json`, `invalid_json`
fetched rows / total rows = 1/1
+--------------+----------------+
| valid_json   | invalid_json   |
+--------------+----------------+
| True         | False          |
+--------------+----------------+

os> source=accounts | where json_valid('[1,2,3,4]') and isnull(email) | fields
 account_number, email
fetched rows / total rows = 1/1
+------------------+---------+
| account_number   | email   |
|------------------+---------|
| 13               | null    |
+------------------+---------+
```

PPL Lambda functions

> **Note**
>
> To see which AWS data source integrations support this PPL function, see the section called "Functions".

`EXISTS`

Usage: `exists(array, lambda)` evaluates whether a Lambda predicate holds for one or more elements in the array.

Argument type: ARRAY, LAMBDA

Return type: BOOLEAN. Returns TRUE if at least one element in the array satisfies the Lambda predicate, otherwise FALSE.

Examples:

```
 os> source=people | eval array = json_array(1, -1, 2), result = exists(array, x -> x >
 0) | fields result
fetched rows / total rows = 1/1
+-----------+
| result    |
+-----------+
| true      |
+-----------+

os> source=people | eval array = json_array(-1, -3, -2), result = exists(array, x -> x
 > 0) | fields result
fetched rows / total rows = 1/1
+-----------+
| result    |
+-----------+
| false     |
+-----------+
```

`FILTER`

Usage: `filter(array, lambda)` filters the input array using the given Lambda function.

Argument type: ARRAY, LAMBDA

Return type: ARRAY. An ARRAY that contains all elements in the input array that satisfy the lambda predicate.

Examples:

```
 os> source=people | eval array = json_array(1, -1, 2), result = filter(array, x -> x >
 0) | fields result
fetched rows / total rows = 1/1
+-----------+
| result    |
+-----------+
| [1, 2]    |
+-----------+

os> source=people | eval array = json_array(-1, -3, -2), result = filter(array, x -> x
 > 0) | fields result
fetched rows / total rows = 1/1
+-----------+
| result    |
+-----------+
| []        |
+-----------+
```

`TRANSFORM`

Usage: `transform(array, lambda)` transforms elements in an array using the Lambda transform function. The second argument implies the index of the element if using binary Lambda function. This is similar to a `map` in functional programming.

Argument type: ARRAY, LAMBDA

Return type: ARRAY. An ARRAY that contains the result of applying the lambda transform function to each element in the input array.

Examples:

```
os> source=people | eval array = json_array(1, 2, 3), result = transform(array, x -> x
 + 1) | fields result
fetched rows / total rows = 1/1
+--------------+
```

```
| result       |
+--------------+
| [2, 3, 4]    |
+--------------+

os> source=people | eval array = json_array(1, 2, 3), result = transform(array, (x, i)
 -> x + i) | fields result
fetched rows / total rows = 1/1
+--------------+
| result       |
+--------------+
| [1, 3, 5]    |
+--------------+
```

REDUCE

Usage: `reduce(array, start, merge_lambda, finish_lambda)` reduces an array to a single value by applying lambda functions. The function applies the merge_lambda to the start value and all array elements, then applies the `finish_lambda` to the result.

Argument type: ARRAY, ANY, LAMBDA, LAMBDA

Return type: ANY. The final result of applying the Lambda functions to the start value and the input array.

Examples:

```
 os> source=people | eval array = json_array(1, 2, 3), result = reduce(array, 0, (acc,
 x) -> acc + x) | fields result
fetched rows / total rows = 1/1
+-----------+
| result    |
+-----------+
| 6         |
+-----------+

os> source=people | eval array = json_array(1, 2, 3), result = reduce(array, 10, (acc,
 x) -> acc + x) | fields result
fetched rows / total rows = 1/1
+-----------+
| result    |
+-----------+
| 16        |
```

```
+-----------+

os> source=people | eval array = json_array(1, 2, 3), result = reduce(array, 0, (acc,
 x) -> acc + x, acc -> acc * 10) | fields result
fetched rows / total rows = 1/1
+-----------+
| result    |
+-----------+
| 60        |
+-----------+
```

PPL mathematical functions

> **Note**
>
> To see which AWS data source integrations support this PPL function, see the section called "Functions".

ABS

Usage: `ABS(x)` calculates the absolute value of x.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: INTEGER/LONG/FLOAT/DOUBLE

Example:

```
os> source=people | eval `ABS(-1)` = ABS(-1) | fields `ABS(-1)`
fetched rows / total rows = 1/1
+-----------+
| ABS(-1)   |
|-----------|
| 1         |
+-----------+
```

ACOS

Usage: `ACOS(x)` calculates the arc cosine of x. It returns `NULL` if x is not in the range -1 to 1.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Example:

```
os> source=people | eval `ACOS(0)` = ACOS(0) | fields `ACOS(0)`
fetched rows / total rows = 1/1
+--------------------+
| ACOS(0)            |
|--------------------|
| 1.5707963267948966 |
+--------------------+
```

ASIN

Usage: `asin(x)` calculates the arc sine of x. It returns `NULL` if x is not in the range -1 to 1.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Example:

```
os> source=people | eval `ASIN(0)` = ASIN(0) | fields `ASIN(0)`
fetched rows / total rows = 1/1
+-----------+
| ASIN(0)   |
|-----------|
| 0.0       |
+-----------+
```

ATAN

Usage: `ATAN(x)` calculates the arc tangent of x. `atan(y, x)` calculates the arc tangent of y / x, except that the signs of both arguments determine the quadrant of the result.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Example:

```
os> source=people | eval `ATAN(2)` = ATAN(2), `ATAN(2, 3)` = ATAN(2, 3) | fields
 `ATAN(2)`, `ATAN(2, 3)`
```

```
fetched rows / total rows = 1/1
+--------------------+--------------------+
| ATAN(2)            | ATAN(2, 3)         |
|--------------------+--------------------|
| 1.1071487177940904 | 0.5880026035475675 |
+--------------------+--------------------+
```

ATAN2

Usage: `ATAN2(y, x)` calculates the arc tangent of y / x, except that the signs of both arguments determine the quadrant of the result.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Example:

```
os> source=people | eval `ATAN2(2, 3)` = ATAN2(2, 3) | fields `ATAN2(2, 3)`
fetched rows / total rows = 1/1
+--------------------+
| ATAN2(2, 3)        |
|--------------------|
| 0.5880026035475675 |
+--------------------+
```

CBRT

Usage: CBRT calculates the cube root of a number.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE:

INTEGER/LONG/FLOAT/DOUBLE -> DOUBLE

Example:

```
opensearchsql> source=location | eval `CBRT(8)` = CBRT(8), `CBRT(9.261)` = CBRT(9.261),
 `CBRT(-27)` = CBRT(-27) | fields `CBRT(8)`, `CBRT(9.261)`, `CBRT(-27)`;
fetched rows / total rows = 2/2
+-----------+---------------+-------------+
```

```
| CBRT(8)   | CBRT(9.261)   | CBRT(-27)   |
|-----------+---------------+-------------|
| 2.0       | 2.1           | -3.0        |
| 2.0       | 2.1           | -3.0        |
+-----------+---------------+-------------+
```

CEIL

Usage: An alias for the `CEILING` function. `CEILING(T)` takes the ceiling of value T.

Limitation: `CEILING` only works as expected when IEEE 754 double type displays a decimal when stored.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: LONG

Example:

```
os> source=people | eval `CEILING(0)` = CEILING(0), `CEILING(50.00005)` =
 CEILING(50.00005), `CEILING(-50.00005)` = CEILING(-50.00005) | fields `CEILING(0)`,
 `CEILING(50.00005)`, `CEILING(-50.00005)`
fetched rows / total rows = 1/1
+--------------+---------------------+----------------------+
| CEILING(0)   | CEILING(50.00005)   | CEILING(-50.00005)   |
|--------------+---------------------+----------------------|
| 0            | 51                  | -50                  |
+--------------+---------------------+----------------------+

os> source=people | eval `CEILING(3147483647.12345)` = CEILING(3147483647.12345),
 `CEILING(113147483647.12345)` = CEILING(113147483647.12345),
 `CEILING(3147483647.00001)` = CEILING(3147483647.00001) | fields
 `CEILING(3147483647.12345)`, `CEILING(113147483647.12345)`,
 `CEILING(3147483647.00001)`
fetched rows / total rows = 1/1
+----------------------------+------------------------------
+----------------------------+
| CEILING(3147483647.12345)   | CEILING(113147483647.12345)   |
 CEILING(3147483647.00001)   |
|----------------------------+------------------------------
+----------------------------|
| 3147483648                  | 113147483648                 | 3147483648
     |
```

```
+----------------------------+-------------------------------
+----------------------------+
```

CONV

Usage: `CONV(x, a, b)` converts the number x from a base to b base.

Argument type: x: STRING, a: INTEGER, b: INTEGER

Return type: STRING

Example:

```
os> source=people | eval `CONV('12', 10, 16)` = CONV('12', 10, 16), `CONV('2C', 16,
 10)` = CONV('2C', 16, 10), `CONV(12, 10, 2)` = CONV(12, 10, 2), `CONV(1111, 2, 10)` =
 CONV(1111, 2, 10) | fields `CONV('12', 10, 16)`, `CONV('2C', 16, 10)`, `CONV(12, 10,
 2)`, `CONV(1111, 2, 10)`
fetched rows / total rows = 1/1
+----------------------+----------------------+-------------------
+----------------------+
| CONV('12', 10, 16)   | CONV('2C', 16, 10)   | CONV(12, 10, 2)   | CONV(1111, 2, 10)
 |
|----------------------+----------------------+-------------------
+----------------------|
| c                    | 44                   | 1100              | 15
 |
+----------------------+----------------------+-------------------
+----------------------+
```

COS

Usage: `COS(x)` calculates the cosine of x, where x is given in radians.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Example:

```
os> source=people | eval `COS(0)` = COS(0) | fields `COS(0)`
fetched rows / total rows = 1/1
+----------+
| COS(0)   |
```

```
|----------|
| 1.0      |
+----------+
```

COT

Usage: COT(x) calculates the cotangent of x. It returns out-of-range error if x is equal to 0.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Example:

```
os> source=people | eval `COT(1)` = COT(1) | fields `COT(1)`
fetched rows / total rows = 1/1
+--------------------+
| COT(1)             |
|--------------------|
| 0.6420926159343306 |
+--------------------+
```

CRC32

Usage: CRC32 calculates a cyclic redundancy check value and returns a 32-bit unsigned value.

Argument type: STRING

Return type: LONG

Example:

```
os> source=people | eval `CRC32('MySQL')` = CRC32('MySQL') | fields `CRC32('MySQL')`
fetched rows / total rows = 1/1
+------------------+
| CRC32('MySQL')   |
|------------------|
| 3259397556       |
+------------------+
```

DEGREES

Usage: DEGREES(x) converts x from radians to degrees.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Example:

```
os> source=people | eval `DEGREES(1.57)` = DEGREES(1.57) | fields `DEGREES(1.57)`
fetched rows / total rows  = 1/1
+-------------------+
| DEGREES(1.57)     |
|-------------------|
| 89.95437383553924 |
+-------------------+
```

E

Usage: E() returns the Euler's number.

Return type: DOUBLE

Example:

```
os> source=people | eval `E()` = E() | fields `E()`
fetched rows / total rows = 1/1
+-------------------+
| E()               |
|-------------------|
| 2.718281828459045 |
+-------------------+
```

EXP

Usage: EXP(x) returns e raised to the power of x.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Example:

```
os> source=people | eval `EXP(2)` = EXP(2) | fields `EXP(2)`
fetched rows / total rows = 1/1
```

```
+------------------+
| EXP(2)           |
|------------------|
| 7.38905609893065 |
+------------------+
```

FLOOR

Usage: FLOOR(T) takes the floor of value T.

Limitation: FLOOR only works as expected when IEEE 754 double type displays a decimal when stored.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: LONG

Example:

```
os> source=people | eval `FLOOR(0)` = FLOOR(0), `FLOOR(50.00005)` = FLOOR(50.00005),
 `FLOOR(-50.00005)` = FLOOR(-50.00005) | fields `FLOOR(0)`, `FLOOR(50.00005)`,
 `FLOOR(-50.00005)`
fetched rows / total rows = 1/1
+------------+------------------+--------------------+
| FLOOR(0)   | FLOOR(50.00005)   | FLOOR(-50.00005)   |
|------------+------------------+--------------------|
| 0          | 50                | -51                |
+------------+------------------+--------------------+

os> source=people | eval `FLOOR(3147483647.12345)` = FLOOR(3147483647.12345),
 `FLOOR(113147483647.12345)` = FLOOR(113147483647.12345), `FLOOR(3147483647.00001)`
 = FLOOR(3147483647.00001) | fields `FLOOR(3147483647.12345)`,
 `FLOOR(113147483647.12345)`, `FLOOR(3147483647.00001)`
fetched rows / total rows = 1/1
+--------------------------+----------------------------+--------------------------+
| FLOOR(3147483647.12345)   | FLOOR(113147483647.12345)   | FLOOR(3147483647.00001)   |
|--------------------------+----------------------------+--------------------------|
| 3147483647                | 113147483647                | 3147483647                |
+--------------------------+----------------------------+--------------------------+

os> source=people | eval `FLOOR(282474973688888.022)` = FLOOR(282474973688888.022),
 `FLOOR(9223372036854775807.022)` = FLOOR(9223372036854775807.022),
 `FLOOR(9223372036854775807.0000001)` = FLOOR(9223372036854775807.0000001)
```

```
 | fields `FLOOR(282474973688888.022)`, `FLOOR(9223372036854775807.022)`,
 `FLOOR(9223372036854775807.0000001)`
fetched rows / total rows = 1/1
+-----------------------------+-----------------------------------
+-------------------------------------+
| FLOOR(282474973688888.022)   | FLOOR(9223372036854775807.022)   |
 FLOOR(9223372036854775807.0000001)   |
|-----------------------------+-----------------------------------
+-------------------------------------|
| 282474973688888              | 9223372036854775807              | 9223372036854775807
                  |
+-----------------------------+-----------------------------------
+-------------------------------------+
```

LN

Usage: `LN(x)` returns the natural logarithm of x.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Example:

```
os> source=people | eval `LN(2)` = LN(2) | fields `LN(2)`
fetched rows / total rows = 1/1
+--------------------+
| LN(2)              |
|--------------------|
| 0.6931471805599453 |
+--------------------+
```

LOG

Usage: `LOG(x)` returns the natural logarithm of x that is the base e logarithm of the x. log(B, x) is equivalent to log(x)/log(B).

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Example:

```
os> source=people | eval `LOG(2)` = LOG(2), `LOG(2, 8)` = LOG(2, 8) | fields `LOG(2)`,
 `LOG(2, 8)`
fetched rows / total rows = 1/1
+--------------------+-------------+
| LOG(2)             | LOG(2, 8)   |
|--------------------+-------------|
| 0.6931471805599453 | 3.0         |
+--------------------+-------------+
```

LOG2

Usage: `LOG2(x)` is equivalent to `log(x)/log(2)`.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Example:

```
os> source=people | eval `LOG2(8)` = LOG2(8) | fields `LOG2(8)`
fetched rows / total rows = 1/1
+-----------+
| LOG2(8)   |
|-----------|
| 3.0       |
+-----------+
```

LOG10

Usage: `LOG10(x)` is equivalent to `log(x)/log(10)`.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Example:

```
os> source=people | eval `LOG10(100)` = LOG10(100) | fields `LOG10(100)`
fetched rows / total rows = 1/1
+--------------+
| LOG10(100)   |
|--------------|
| 2.0          |
```

```
+--------------+
```

MOD

Usage: `MOD(n, m)` calculates the remainder of the number n divided by m.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: Wider type between types of n and m if m is nonzero value. If m equals to 0, then returns NULL.

Example:

```
os> source=people | eval `MOD(3, 2)` = MOD(3, 2), `MOD(3.1, 2)` = MOD(3.1, 2) | fields
 `MOD(3, 2)`, `MOD(3.1, 2)`
fetched rows / total rows = 1/1
+-------------+---------------+
| MOD(3, 2)   | MOD(3.1, 2)   |
|-------------+---------------|
| 1           | 1.1           |
+-------------+---------------+
```

PI

Usage: `PI()` returns the constant pi.

Return type: DOUBLE

Example:

```
os> source=people | eval `PI()` = PI() | fields `PI()`
fetched rows / total rows = 1/1
+-------------------+
| PI()              |
|-------------------|
| 3.141592653589793 |
+-------------------+
```

POW

Usage: `POW(x, y)` calculates the value of x raised to the power of y. Bad inputs return a `NULL` result.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Synonyms: `POWER(_, _)`

Example:

```
os> source=people | eval `POW(3, 2)` = POW(3, 2), `POW(-3, 2)` = POW(-3, 2), `POW(3,
 -2)` = POW(3, -2) | fields `POW(3, 2)`, `POW(-3, 2)`, `POW(3, -2)`
fetched rows / total rows = 1/1
+-------------+--------------+--------------------+
| POW(3, 2)   | POW(-3, 2)   | POW(3, -2)         |
|-------------+--------------+--------------------|
| 9.0         | 9.0          | 0.1111111111111111 |
+-------------+--------------+--------------------+
```

POWER

Usage: `POWER(x, y)` calculates the value of x raised to the power of y. Bad inputs return a `NULL` result.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Synonyms: `POW(_, _)`

Example:

```
os> source=people | eval `POWER(3, 2)` = POWER(3, 2), `POWER(-3, 2)` = POWER(-3, 2),
 `POWER(3, -2)` = POWER(3, -2) | fields `POWER(3, 2)`, `POWER(-3, 2)`, `POWER(3, -2)`
fetched rows / total rows = 1/1
+---------------+----------------+--------------------+
| POWER(3, 2)   | POWER(-3, 2)   | POWER(3, -2)       |
|---------------+----------------+--------------------|
| 9.0           | 9.0            | 0.1111111111111111 |
+---------------+----------------+--------------------+
```

RADIANS

Usage: `RADIANS(x)` converts x from degrees to radians.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Example:

```
os> source=people | eval `RADIANS(90)` = RADIANS(90) | fields `RADIANS(90)`
fetched rows / total rows  = 1/1
+--------------------+
| RADIANS(90)        |
|--------------------|
| 1.5707963267948966 |
+--------------------+
```

RAND

Usage: RAND()/RAND(N) returns a random floating-point value in the range 0 <= value < 1.0. If you specify integer N, the function initializes the seed before execution. One implication of this behavior is that with an identical argument N, `rand(N)` returns the same value each time, producing a repeatable sequence of column values.

Argument type: INTEGER

Return type: FLOAT

Example:

```
os> source=people | eval `RAND(3)` = RAND(3) | fields `RAND(3)`
fetched rows / total rows = 1/1
+------------+
| RAND(3)    |
|------------|
| 0.73105735 |
+------------+
```

ROUND

Usage: ROUND(x, d) rounds the argument x to d decimal places. If you don't specify d, it defaults to 0.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type map:

- (INTEGER/LONG [,INTEGER]) -> LONG
- (FLOAT/DOUBLE [,INTEGER]) -> LONG

Example:

```
os> source=people | eval `ROUND(12.34)` = ROUND(12.34), `ROUND(12.34, 1)` =
 ROUND(12.34, 1), `ROUND(12.34, -1)` = ROUND(12.34, -1), `ROUND(12, 1)` = ROUND(12, 1)
 | fields `ROUND(12.34)`, `ROUND(12.34, 1)`, `ROUND(12.34, -1)`, `ROUND(12, 1)`
fetched rows / total rows = 1/1
+----------------+-------------------+--------------------+----------------+
| ROUND(12.34)   | ROUND(12.34, 1)   | ROUND(12.34, -1)   | ROUND(12, 1)   |
|----------------+-------------------+--------------------+----------------|
| 12.0           | 12.3              | 10.0               | 12             |
+----------------+-------------------+--------------------+----------------+
```

SIGN

Usage: `SIGN` returns the sign of the argument as -1, 0, or 1, depending on whether the number is negative, zero, or positive.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: INTEGER

Example:

```
os> source=people | eval `SIGN(1)` = SIGN(1), `SIGN(0)` = SIGN(0), `SIGN(-1.1)` =
 SIGN(-1.1) | fields `SIGN(1)`, `SIGN(0)`, `SIGN(-1.1)`
fetched rows / total rows = 1/1
+-----------+-----------+--------------+
| SIGN(1)   | SIGN(0)   | SIGN(-1.1)   |
|-----------+-----------+--------------|
| 1         | 0         | -1           |
+-----------+-----------+--------------+
```

SIN

Usage: `sin(x)` calculates the sine of x, where x is given in radians.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type: DOUBLE

Example:

```
os> source=people | eval `SIN(0)` = SIN(0) | fields `SIN(0)`
fetched rows / total rows = 1/1
+----------+
| SIN(0)   |
|----------|
| 0.0      |
+----------+
```

SQRT

Usage: SQRT calculates the square root of a non-negative number.

Argument type: INTEGER/LONG/FLOAT/DOUBLE

Return type map:

- (Non-negative) INTEGER/LONG/FLOAT/DOUBLE -> DOUBLE
- (Negative) INTEGER/LONG/FLOAT/DOUBLE -> NULL

Example:

```
os> source=people | eval `SQRT(4)` = SQRT(4), `SQRT(4.41)` = SQRT(4.41) | fields
 `SQRT(4)`, `SQRT(4.41)`
fetched rows / total rows = 1/1
+-----------+-------------+
| SQRT(4)   | SQRT(4.41)   |
|-----------+-------------|
| 2.0       | 2.1          |
+-----------+-------------+
```

PPL string functions

> **Note**
>
> To see which AWS data source integrations support this PPL function, see the section called "Functions".

CONCAT

Usage: `CONCAT(str1, str2, ...., str_9)` adds up to 9 strings together.

Argument type:

- STRING, STRING, ..., STRING
- Return type: STRING

Example:

```
os> source=people | eval `CONCAT('hello', 'world')` = CONCAT('hello', 'world'),
 `CONCAT('hello ', 'whole ', 'world', '!')` = CONCAT('hello ', 'whole ', 'world', '!')
 | fields `CONCAT('hello', 'world')`, `CONCAT('hello ', 'whole ', 'world', '!')`
fetched rows / total rows = 1/1
+----------------------------+----------------------------------------------+
| CONCAT('hello', 'world')   | CONCAT('hello ', 'whole ', 'world', '!')   |
|----------------------------+----------------------------------------------|
| helloworld                 | hello whole world!                           |
+----------------------------+----------------------------------------------+
```

CONCAT_WS

Usage: `CONCAT_WS(sep, str1, str2)` concatenates two or more strings using a specified separator between them.

Argument type:

- STRING, STRING, ..., STRING
- Return type: STRING

Example:

```
os> source=people | eval `CONCAT_WS(',', 'hello', 'world')` = CONCAT_WS(',', 'hello',
 'world') | fields `CONCAT_WS(',', 'hello', 'world')`
fetched rows / total rows = 1/1
+-----------------------------------+
| CONCAT_WS(',', 'hello', 'world')   |
|-----------------------------------|
| hello,world                       |
```

```
+----------------------------------+
```

LENGTH

Usage: `length(str)` returns the length of the input string measured in bytes.

Argument type:

- STRING
- Return type: INTEGER

Example:

```
os> source=people | eval `LENGTH('helloworld')` = LENGTH('helloworld') | fields
 `LENGTH('helloworld')`
fetched rows / total rows = 1/1
+------------------------+
| LENGTH('helloworld')   |
|------------------------|
| 10                     |
+------------------------+
```

LOWER

Usage: `lower(string)` converts the input string to lowercase.

Argument type:

- STRING
- Return type: STRING

Example:

```
os> source=people | eval `LOWER('helloworld')` = LOWER('helloworld'),
 `LOWER('HELLOWORLD')` = LOWER('HELLOWORLD') | fields `LOWER('helloworld')`,
 `LOWER('HELLOWORLD')`
fetched rows / total rows = 1/1
+----------------------+-----------------------+
| LOWER('helloworld')   | LOWER('HELLOWORLD')   |
|----------------------+-----------------------|
| helloworld            | helloworld            |
```

```
+----------------------+----------------------+
```

LTRIM

Usage: `ltrim(str)` removes leading space characters from the input string.

Argument type:

- STRING
- Return type: STRING

Example:

```
os> source=people | eval `LTRIM('   hello')` = LTRIM('   hello'), `LTRIM('hello   ')` =
 LTRIM('hello   ') | fields `LTRIM('   hello')`, `LTRIM('hello   ')`
fetched rows / total rows = 1/1
+---------------------+---------------------+
| LTRIM('   hello')   | LTRIM('hello   ')   |
|---------------------+---------------------|
| hello               | hello               |
+---------------------+---------------------+
```

POSITION

Usage: `POSITION(substr IN str)` returns the position of the first occurrence of substring in string. It returns 0 if the substring is not in the string. It returns NULL if any argument is NULL.

Argument type:

- STRING, STRING
- Return type INTEGER

Example:

```
os> source=people | eval `POSITION('world' IN 'helloworld')` = POSITION('world'
 IN 'helloworld'), `POSITION('invalid' IN 'helloworld')`= POSITION('invalid' IN
 'helloworld')  | fields `POSITION('world' IN 'helloworld')`, `POSITION('invalid' IN
 'helloworld')`
fetched rows / total rows = 1/1
+-----------------------------------+---------------------------------------+
```

```
| POSITION('world' IN 'helloworld')   | POSITION('invalid' IN 'helloworld')   |
|-------------------------------------+---------------------------------------|
| 6                                   | 0                                     |
+-------------------------------------+---------------------------------------+
```

REVERSE

Usage: `REVERSE(str)` returns the reversed string of the input string.

Argument type:

- STRING
- Return type: STRING

Example:

```
os> source=people | eval `REVERSE('abcde')` = REVERSE('abcde') | fields
 `REVERSE('abcde')`
fetched rows / total rows = 1/1
+--------------------+
| REVERSE('abcde')   |
|--------------------|
| edcba              |
+--------------------+
```

RIGHT

Usage: `right(str, len)` returns the rightmost characters from the input string. It returns 0 if the substring is not in the string. It returns NULL if any argument is NULL.

Argument type:

- STRING, INTEGER
- Return type: STRING

Example:

```
os> source=people | eval `RIGHT('helloworld', 5)` = RIGHT('helloworld', 5),
 `RIGHT('HELLOWORLD', 0)` = RIGHT('HELLOWORLD', 0) | fields `RIGHT('helloworld', 5)`,
 `RIGHT('HELLOWORLD', 0)`
fetched rows / total rows = 1/1
```

```
+--------------------------+--------------------------+
| RIGHT('helloworld', 5)   | RIGHT('HELLOWORLD', 0)   |
|--------------------------+--------------------------|
| world                    |                          |
+--------------------------+--------------------------+
```

RTRIM

Usage: `rtrim(str)` trims trailing space characters from the input string.

Argument type:

- STRING
- Return type: **STRING**

Example:

```
os> source=people | eval `RTRIM('   hello')` = RTRIM('   hello'), `RTRIM('hello   ')` =
 RTRIM('hello   ') | fields `RTRIM('   hello')`, `RTRIM('hello   ')`
fetched rows / total rows = 1/1
+---------------------+---------------------+
| RTRIM('   hello')   | RTRIM('hello   ')   |
|---------------------+---------------------|
|    hello            | hello               |
+---------------------+---------------------+
```

SUBSTRING

Usage: `substring(str, start)` or `substring(str, start, length)` returns a substring of the input string. With no length specified, it returns the entire string from the start position.

Argument type:

- STRING, INTEGER, INTEGER
- Return type: STRING

Example:

```
os> source=people | eval `SUBSTRING('helloworld', 5)` = SUBSTRING('helloworld',
 5), `SUBSTRING('helloworld', 5, 3)` = SUBSTRING('helloworld', 5, 3) | fields
 `SUBSTRING('helloworld', 5)`, `SUBSTRING('helloworld', 5, 3)`
```

```
fetched rows / total rows = 1/1
+------------------------------+----------------------------------+
| SUBSTRING('helloworld', 5)   | SUBSTRING('helloworld', 5, 3)    |
|------------------------------+----------------------------------|
| oworld                       | owo                              |
+------------------------------+----------------------------------+
```

TRIM

Usage: `trim(string)` removes leading and trailing whitespace from the input string.

Argument type:

- STRING
- Return type: **STRING**

Example:

```
os> source=people | eval `TRIM('   hello')` = TRIM('   hello'), `TRIM('hello   ')` =
 TRIM('hello   ') | fields `TRIM('   hello')`, `TRIM('hello   ')`
fetched rows / total rows = 1/1
+--------------------+--------------------+
| TRIM('   hello')   | TRIM('hello   ')   |
|--------------------+--------------------|
| hello              | hello              |
+--------------------+--------------------+
```

UPPER

Usage: `upper(string)` converts the input string to uppercase.

Argument type:

- STRING
- Return type: STRING

Example:

```
os> source=people | eval `UPPER('helloworld')` = UPPER('helloworld'),
 `UPPER('HELLOWORLD')` = UPPER('HELLOWORLD') | fields `UPPER('helloworld')`,
 `UPPER('HELLOWORLD')`
```

```
fetched rows / total rows = 1/1
+----------------------+----------------------+
| UPPER('helloworld')   | UPPER('HELLOWORLD')   |
|----------------------+----------------------|
| HELLOWORLD            | HELLOWORLD            |
+----------------------+----------------------+
```

PPL type conversion functions

> **Note**
>
> To see which AWS data source integrations support this PPL function, see the section called "Functions".

`TRIM`

Usage: `cast(expr as dateType)` casts the `expr` to the `dataType` and returns the value of the `dataType`.

The following conversion rules apply:

Type conversion rules

Src/Target	STRING	NUMBER	BOOLEAN	TIMESTAMP	DATE	TIME
STRING		Note1	Note1	TIMESTAMP()	DATE()	TIME()
NUMBER	Note1		v!=0	N/A	N/A	N/A
BOOLEAN	Note1	v?1:0		N/A	N/A	N/A
TIMESTAMP	Note1	N/A	N/A		DATE()	TIME()
DATE	Note1	N/A	N/A	N/A		N/A
TIME	Note1	N/A	N/A	N/A	N/A	

Cast to string example:

```
os> source=people | eval `cbool` = CAST(true as string), `cint` = CAST(1 as string),
 `cdate` = CAST(CAST('2012-08-07' as date) as string) | fields `cbool`, `cint`, `cdate`
fetched rows / total rows = 1/1
+---------+--------+------------+
| cbool   | cint   | cdate      |
|---------+--------+------------|
| true    | 1      | 2012-08-07 |
+---------+--------+------------+
```

Cast to number example:

```
os> source=people | eval `cbool` = CAST(true as int), `cstring` = CAST('1' as int) |
 fields `cbool`, `cstring`
fetched rows / total rows = 1/1
+---------+-----------+
| cbool   | cstring   |
|---------+-----------|
| 1       | 1         |
+---------+-----------+
```

Cast to date example:

```
os> source=people | eval `cdate` = CAST('2012-08-07' as date), `ctime` =
 CAST('01:01:01' as time), `ctimestamp` = CAST('2012-08-07 01:01:01' as timestamp) |
 fields `cdate`, `ctime`, `ctimestamp`
fetched rows / total rows = 1/1
+------------+----------+---------------------+
| cdate      | ctime    | ctimestamp          |
|------------+----------+---------------------|
| 2012-08-07 | 01:01:01 | 2012-08-07 01:01:01 |
+------------+----------+---------------------+
```

Chained cast example:

```
os> source=people | eval `cbool` = CAST(CAST(true as string) as boolean) | fields
 `cbool`
fetched rows / total rows = 1/1
+---------+
| cbool   |
|---------|
| True    |
```

```
+----------+
```

Monitoring Amazon OpenSearch Service domains

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon OpenSearch Service and your other AWS solutions. AWS provides the following tools to monitor your OpenSearch Service resources, report issues, and take automatic actions when appropriate:

Amazon CloudWatch

Amazon CloudWatch monitors your OpenSearch Service resources in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a metric reaches a certain threshold. For more information, see the Amazon CloudWatch User Guide.

Amazon CloudWatch Logs

Amazon CloudWatch Logs lets you monitor, store, and access your OpenSearch log files. CloudWatch Logs monitors the information in log files and can notify you when certain thresholds are met. For more information, see the Amazon CloudWatch Logs User Guide.

Amazon EventBridge

Amazon EventBridge delivers a near real-time stream of system events that describe changes in your OpenSearch Service domains. You can create rules that watch for certain events, and trigger automated actions in other AWS services when these events occur. For more information, see the Amazon EventBridge User Guide.

AWS CloudTrail

AWS CloudTrail captures configuration API calls made to OpenSearch Service as events. It can deliver these events to an Amazon S3 bucket that you specify. Using this information, you can identify which users and accounts made requests, the source IP address from which the requests were made, and when the requests occurred. For more information, see the AWS CloudTrail User Guide.

Topics

- Monitoring OpenSearch Service events with Amazon EventBridge
- Monitoring Amazon OpenSearch Service API calls with AWS CloudTrail

Monitoring OpenSearch cluster metrics with Amazon CloudWatch

Amazon OpenSearch Service publishes data from your domains to Amazon CloudWatch. CloudWatch lets you retrieve statistics about those data points as an ordered set of time-series data, known as *metrics*. OpenSearch Service sends most metrics to CloudWatch in 60-second intervals. If you use General Purpose or Magnetic EBS volumes, the EBS volume metrics update only every five minutes. All cumulative metrics (e.g. `ThreadpoolWriteRejected`, `ThreadpoolSearchRejected`) are in-memory and will lose state. Metrics will reset during a node drop, node bounce, node replacement, and blue/green deployment. For more information about Amazon CloudWatch, see the Amazon CloudWatch User Guide.

The OpenSearch Service console displays a series of charts based on the raw data from CloudWatch. Depending on your needs, you might prefer to view cluster data in CloudWatch instead of the graphs in the console. The service archives metrics for two weeks before discarding them. The metrics are provided at no extra charge, but CloudWatch still charges for creating dashboards and alarms. For more information, see Amazon CloudWatch pricing.

OpenSearch Service publishes the following metrics to CloudWatch:

- the section called “Cluster metrics”
- the section called “Dedicated master node metrics”
- the section called “EBS volume metrics”
- the section called “Instance metrics”
- the section called “UltraWarm metrics”
- the section called “Dedicated coordinator node metrics”
- the section called “Cold storage metrics”
- the section called “Alerting metrics”
- the section called “Anomaly detection metrics”
- the section called “Asynchronous search metrics”
- the section called “SQL metrics”

- the section called “k-NN metrics”
- the section called “Cross-cluster search metrics”
- the section called “Cross-cluster replication metrics”
- the section called “Learning to Rank metrics”
- the section called “Piped Processing Language metrics”

Viewing metrics in CloudWatch

CloudWatch metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace.

To view metrics using the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, find **Metrics** and choose **All metrics**. Select the **ES/OpenSearchService** namespace.
3. Choose a dimension to view the corresponding metrics. Metrics for individual nodes are in the `ClientId, DomainName, NodeId` dimension. Cluster metrics are in the `Per-Domain, Per-Client Metrics` dimension. Some node metrics are aggregated at the cluster level and thus included in both dimensions. Shard metrics are in the `ClientId, DomainName, NodeId, ShardRole` dimension.

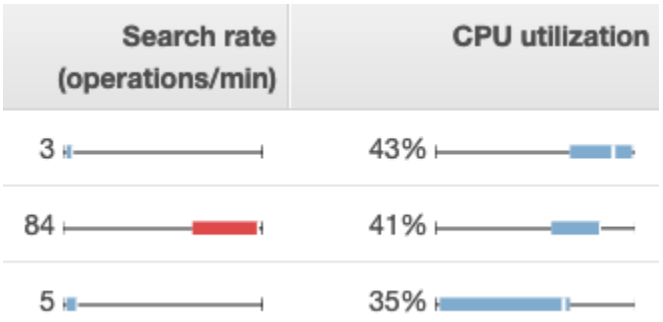
To view a list of metrics using the AWS CLI

Run the following command:

```
aws cloudwatch list-metrics --namespace "AWS/ES"
```

Interpreting health charts in OpenSearch Service

To view metrics in OpenSearch Service, use the **Cluster health** and **Instance health** tabs. The **Instance health** tab uses box charts to provide at-a-glance visibility into the health of each OpenSearch node:



- Each colored box shows the range of values for the node over the specified time period.
- Blue boxes represent values that are consistent with other nodes. Red boxes represent outliers.
- The white line within each box shows the node's current value.
- The “whiskers” on either side of each box show the minimum and maximum values for all nodes over the time period.

If you make configuration changes to your domain, the list of individual instances in the **Cluster health** and **Instance health** tabs often double in size for a brief period before returning to the correct number. For an explanation of this behavior, see the section called “Configuration changes”.

Cluster metrics

Amazon OpenSearch Service provides the following metrics for clusters.

Metric	Description
`ClusterStatus.green`	A value of 1 indicates that all index shards are allocated to nodes in the cluster. Relevant statistics: Maximum
`ClusterStatus.yellow`	A value of 1 indicates that the primary shards for all indexes are allocated to nodes in the cluster, but replica shards for at least one index are not. For more information, see the section called “Yellow cluster status”. Relevant statistics: Maximum

Metric	Description
ClusterStatus.red	A value of 1 indicates that the primary and replica shards for at least one index are not allocated to nodes in the cluster. For more information, see the section called “Red cluster status”. Relevant statistics: Maximum
Shards.active	The total number of active primary and replica shards. Relevant statistics: Maximum, Sum
Shards.unassigned	The number of shards that are not allocated to nodes in the cluster. Relevant statistics: Maximum, Sum
Shards.delayedUnassigned	The number of shards whose node allocation has been delayed by the timeout settings. Relevant statistics: Maximum, Sum
Shards.activePrimary	The number of active primary shards. Relevant statistics: Maximum, Sum
Shards.initializing	The number of shards that are under initialization. Relevant statistics: Sum
Shards.relocating	The number of shards that are under relocation. Relevant statistics: Sum
Nodes	The number of nodes in the OpenSearch Service cluster, including dedicated master nodes and UltraWarm nodes. For more information, see the section called “Configuration changes”. Relevant statistics: Maximum

Metric	Description
SearchableDocuments	The total number of searchable documents across all data nodes in the cluster. Relevant statistics: Minimum, Maximum, Average
DeletedDocuments	The total number of documents marked for deletion across all data nodes in the cluster. These documents no longer appear in search results, but OpenSearch only removes deleted documents from disk during segment merges. This metric increases after delete requests and decreases after segment merges. Relevant statistics: Minimum, Maximum, Average
CPUUtilization	The percentage of CPU usage for data nodes in the cluster. Maximum shows the node with the highest CPU usage. Average represents all nodes in the cluster. This metric is also available for individual nodes. Relevant statistics: Maximum, Average

Metric	Description
FreeStorageSpace	The free space for data nodes in the cluster. Sum shows total free space for the cluster, but you must leave the period at one minute to get an accurate value. Minimum and Maximum show the nodes with the least and most free space, respectively. This metric is also available for individual nodes. OpenSearch Service throws a ClusterBlockException when this metric reaches 0. To recover, you must either delete indexes, add larger instances, or add EBS-based storage to existing instances. To learn more, see the section called "Lack of available storage space". The OpenSearch Service console displays this value in GiB. The Amazon CloudWatch console displays it in MiB. **Note** FreeStorageSpace will always be lower than the values that the OpenSearch _cluster/stats and _cat/allocation APIs provide. OpenSearch Service reserves a percentage of the storage space on each instance for internal operations. For more information, see Calculating storage requirements. Relevant statistics: Minimum, Maximum, Average, Sum
ClusterUsedSpace	The total used space for the cluster. You must leave the period at one minute to get an accurate value. The OpenSearch Service console displays this value in GiB. The Amazon CloudWatch console displays it in MiB. Relevant statistics: Minimum, Maximum

Metric	Description
ClusterIndexWritesBlocked	Indicates whether your cluster is accepting or blocking incoming write requests. A value of 0 means that the cluster is accepting requests. A value of 1 means that it is blocking requests. Some common factors include the following: FreeStorageSpace is too low or JVMMemoryPressure is too high. To alleviate this issue, consider adding more disk space or scaling your cluster. Relevant statistics: Maximum
JVMMemoryPressure	The maximum percentage of the Java heap used for all data nodes in the cluster. OpenSearch Service uses half of an instance's RAM for the Java heap, up to a heap size of 32 GiB. You can scale instances vertically up to 64 GiB of RAM, at which point you can scale horizontally by adding instances. See the section called "Recommended CloudWatch alarms". Relevant statistics: Maximum **Note** The logic for this metric changed in service software R20220323. For more information, see the release notes.
OldGenJVMMemoryPressure	The maximum percentage of the Java heap used for the "old generation" on all data nodes in the cluster. This metric is also available at the node level. Relevant statistics: Maximum
AutomatedSnapshotFailure	The number of failed automated snapshots for the cluster. A value of 1 indicates that no automated snapshot was taken for the domain in the previous 36 hours. Relevant statistics: Minimum, Maximum

Metric	Description
CPUCreditBalance	The remaining CPU credits available for data nodes in the cluster. A CPU credit provides the performance of a full CPU core for one minute. For more information, see CPU credits in the *Amazon EC2 Developer Guide*. This metric is available only for the T2 instance types. Relevant statistics: Minimum
OpenSearchDashboardsHealthyNodes	A health check for OpenSearch Dashboards. If the minimum, maximum, and average are all equal to 1, Dashboards is behaving normally. If you have 10 nodes with a maximum of 1, minimum of 0, and average of 0.7, this means 7 nodes (70%) are healthy and 3 nodes (30%) are unhealthy. Relevant statistics: Minimum, Maximum, Average
OpensearchDashboardsReportingFailedRequestSysErrCount	The number of requests to generate OpenSearch Dashboards reports that failed due to server problems or feature limitations. Relevant statistics: Sum
OpensearchDashboardsReportingFailedRequestUserErrCount	The number of requests to generate OpenSearch Dashboards reports that failed due to client issues. Relevant statistics: Sum
OpensearchDashboardsReportingRequestCount	The total number of requests to generate OpenSearch Dashboards reports. Relevant statistics: Sum
OpensearchDashboardsReportingSuccessCount	The number of successful requests to generate OpenSearch Dashboards reports. Relevant statistics: Sum

Metric	Description
KMSKeyError	A value of 1 indicates that the AWS KMS key used to encrypt data at rest has been disabled. To restore the domain to normal operations, re-enable the key. The console displays this metric only for domains that encrypt data at rest. Relevant statistics: Minimum, Maximum
KMSKeyInaccessible	A value of 1 indicates that the AWS KMS key used to encrypt data at rest has been deleted or revoked its grants to OpenSearch Service. You can't recover domains that are in this state. If you have a manual snapshot, though, you can use it to migrate the domain's data to a new domain. The console displays this metric only for domains that encrypt data at rest. Relevant statistics: Minimum, Maximum
InvalidHostHeaderRequests	The number of HTTP requests made to the OpenSearch cluster that included an invalid (or missing) host header. Valid requests include the domain hostname as the host header value. OpenSearch Service rejects invalid requests for public access domains that don't have a restrictive access policy. We recommend applying a restrictive access policy to all domains. If you see large values for this metric, confirm that your OpenSearch clients include the domain hostname (and not, for example, its IP address) in their requests. Relevant statistics: Sum
OpenSearchRequests (previously ElasticsearchRequests)	The number of requests made to the OpenSearch cluster. Relevant statistics: Sum

Metric	Description
`2xx, 3xx, 4xx, 5xx`	The number of requests to the domain that resulted in the given HTTP response code (2*xx*, 3*xx*, 4*xx*, 5*xx*). Relevant statistics: Sum
`ThroughputThrottle`	Indicates whether or not disks have been throttled. Throttlin g occurs when the combined throughput of `ReadThrou ghputMicroBursting` and `WriteThroughputMic roBursting` is higher than maximum throughput, `MaxProvis ionedThroughput` . `MaxProvisionedThroughput` is the lower value of the instance throughput or the provisioned volume throughput. A value of 1 indicates that disks have been throttled. A value of 0 indicates normal behavior. For information on instance throughput, see Amazon EBS–optimized instances. For information on volume throughput, see Amazon EBS volume types. Relevant statistics: Minimum, Maximum
`IopsThrottle`	Indicates whether or not the number of input/output operations per second (IOPS) on the domain have been throttled. Throttling occurs when IOPS of the data node breach the maximum allowed limit of the EBS volume or the EC2 instance of the data node. For information on instance IOPS, see Amazon EBS–optimized instances. For information on volume IOPS, see Amazon EBS volume types. Relevant statistics: Minimum, Maximum
`HighSwapUsage`	A value of 1 indicates that swapping due to page faults has potentially caused spikes in underlying disk usage during a specific time period. Relevant statistics: Maximum

Dedicated master node metrics

Amazon OpenSearch Service provides the following metrics for dedicated master nodes.

Metric	Description
MasterCPUUtilization	The maximum percentage of CPU resources used by the dedicated master nodes. We recommend increasing the size of the instance type when this metric reaches 60 percent. Relevant statistics: Maximum
MasterFreeStorageSpace	This metric is not relevant and can be ignored. The service does not use master nodes as data nodes.
MasterJVMMemoryPressure	The maximum percentage of the Java heap used for all dedicated master nodes in the cluster. We recommend moving to a larger instance type when this metric reaches 85 percent. Relevant statistics: Maximum **Note** The logic for this metric changed in service software R20220323. For more information, see the release notes.
MasterOldGenJVMMemoryPressure	The maximum percentage of the Java heap used for the "old generation" per master node. Relevant statistics: Maximum
MasterCPUCreditBalance	The remaining CPU credits available for dedicated master nodes in the cluster. A CPU credit provides the performance of a full CPU core for one minute. For more information, see CPU credits in the *Amazon EC2 Developer Guide*. This metric is available only for the T2 instance types. Relevant statistics: Minimum

Metric	Description
MasterReachableFromNode	A health check for MasterNotDiscovered exceptions. A value of 1 indicates normal behavior. A value of 0 indicates that /_cluster/health/ is failing. Failures mean that the master node is unreachable from the source node. They're usually the result of a network connectivity issue or an AWS dependency problem. Relevant statistics: Maximum
MasterSysMemoryUtilization	The percentage of the master node's memory that is in use. Relevant statistics: Maximum

Dedicated coordinator node metrics

Amazon OpenSearch Service provides the following metrics for Dedicated coordinator nodes.

Metric	Description
CoordinatorCPUUtilization	The maximum percentage of CPU resources used by the dedicated coordinator nodes. We recommend increasing the size of the instance type when this metric reaches 80 percent. Relevant statistics: Maximum
CoordinatorJVMMemoryPressure	The maximum percentage of the Java heap used for all dedicated coordinator nodes in the cluster. We recommend moving to a larger instance type when this metric reaches 85 percent. Relevant statistics: Maximum
CoordinatorOldGenJVMMemoryPressure	The maximum percentage of the Java heap used for the "old generation" per master node. Relevant statistics: Maximum

Metric	Description
CoordinatorSysMemoryUtilization	The percentage of the coordinator node's memory that is in use. Relevant statistics: Maximum
CoordinatorFreeStorageSpace	This metric indicates that the service does not use coordinator nodes as data nodes.

EBS volume metrics

Amazon OpenSearch Service provides the following metrics for EBS volumes.

Metric	Description
ReadLatency	The latency, in seconds, for read operations on EBS volumes. This metric is also available for individual nodes. Relevant statistics: Minimum, Maximum, Average
WriteLatency	The latency, in seconds, for write operations on EBS volumes. This metric is also available for individual nodes. Relevant statistics: Minimum, Maximum, Average
ReadThroughput	The throughput, in bytes per second, for read operations on EBS volumes. This metric is also available for individual nodes. Relevant statistics: Minimum, Maximum, Average
ReadThroughputMicroBursting	The throughput, in bytes per second, for read operations on EBS volumes when micro-bursting is taken into consideration. This metric is also available for individual nodes. Micro-bursting occurs when an EBS volume bursts high IOPS or throughput for significantly shorter periods of time (less than one minute). Relevant statistics: Minimum, Maximum, Average

Metric	Description
`WriteThroughput`	The throughput, in bytes per second, for write operations on EBS volumes. This metric is also available for individual nodes. Relevant statistics: Minimum, Maximum, Average
`WriteThroughputMicroBursting`	The throughput, in bytes per second, for write operations on EBS volumes when micro-bursting is taken into consideration. This metric is also available for individual nodes. Micro-bursting occurs when an EBS volume bursts high IOPS or throughput for significantly shorter periods of time (less than one minute). Relevant statistics: Minimum, Maximum, Average
`DiskQueueDepth`	The number of pending input and output (I/O) requests for an EBS volume. Relevant statistics: Minimum, Maximum, Average
`ReadIOPS`	The number of input and output (I/O) operations per second for read operations on EBS volumes. This metric is also available for individual nodes. Relevant statistics: Minimum, Maximum, Average
`ReadIOPSMicroBursting`	The number of input and output (I/O) operations per second for read operations on EBS volumes when micro-bursting is taken into consideration. This metric is also available for individual nodes. Micro-bursting occurs when an EBS volume bursts high IOPS or throughput for significantly shorter periods of time (less than one minute). Relevant statistics: Minimum, Maximum, Average
`WriteIOPS`	The number of input and output (I/O) operations per second for write operations on EBS volumes. This metric is also available for individual nodes. Relevant statistics: Minimum, Maximum, Average

Metric	Description
`WriteIOPS MicroBursting`	The number of input and output (I/O) operations per second for write operations on EBS volumes when micro-bursting is taken into consideration. This metric is also available for individual nodes. Micro-bursting occurs when an EBS volume bursts high IOPS or throughput for significantly shorter periods of time (less than one minute). Relevant statistics: Minimum, Maximum, Average
`BurstBalance`	The percentage of input and output (I/O) credits remaining in the burst bucket for an EBS volume. A value of 100 means that the volume has accumulated the maximum number of credits. If this percentage falls below 70%, see the section called "Low EBS burst balance". The burst balance stays at 0 for domains with gp3 volumes types, and domains with gp2 volumes that have a volume size above 1000 GiB. Relevant statistics: Minimum, Maximum, Average
`VolumeStalledIOcheck`	The status of your EBS volumes to determine when they are impaired. The metrics is a binary value that returns a 0 (pass) or a 1 (fail) status based on whether the EBS volume can complete input and output operations. `VolumeStalledIOcheck` is also available for individual nodes. Relevant statistics: Minimum, Maximum, Average

Instance metrics

Amazon OpenSearch Service provides the following metrics for each instance in a domain. OpenSearch Service also aggregates these instance metrics to provide insight into overall cluster health. You can verify this behavior using the **Sample Count** statistic in the console. Note that each metric in the following table has relevant statistics for the node *and* the cluster.

> **Important**
>
> Different versions of Elasticsearch use different thread pools to process calls to the `_index` API. Elasticsearch 1.5 and 2.3 use the index thread pool. Elasticsearch 5.*x*, 6.0, and 6.2 use

the bulk thread pool. OpenSearch and Elasticsearch 6.3 and later use the write thread pool. Currently, the OpenSearch Service console doesn't include a graph for the bulk thread pool. Use `GET _cluster/settings?include_defaults=true` to check thread pool and queue sizes for your cluster.

Metric	Description
`FetchLatency`	The difference in total time, in milliseconds, taken by all shard fetch operations in a node between minute N and minute (N - 1). Relevant node statistics: Average Relevant cluster statistics: Average, Maximum
`FetchRate`	The total number of shard fetch operations per minute for all shards on a data node. Relevant node statistics: Average Relevant cluster statistics: Average, Maximum, Sum
`ScrollTotal`	The total number of shard scroll operations per minute for all shards on a data node. Relevant node statistics: Average, Maximum Relevant cluster statistics: Average, Maximum, Sum
`ScrollCurrent`	The number of shard scroll operations that are currently running. Relevant node statistics: Average, Maximum Relevant cluster statistics: Average, Maximum, Sum
`OpenContexts`	The number of open search contexts. Relevant node statistics: Average, Maximum Relevant cluster statistics: Average, Maximum, Sum

Metric	Description
ThreadCount	The total number of threads currently being utilized by the OpenSearch process. Relevant node statistics: Average, Maximum Relevant cluster statistics: Average, Maximum, Sum
ShardReactivateCount	The total number of times that all shards have been activated from an idle state. Relevant node statistics: Sum, Maximum Relevant cluster statistics: Sum, Maximum
ConcurrentSearchRate	The total number of search requests using concurrent segment search per minute for all shards on a data node. A single call to the _search API might return results from many different shards. If five of these shards are on one node, the node would report 5 for this metric, even though the client only made one request. Relevant node statistics: Average Relevant cluster statistics: Average, Maximum, Sum
ConcurrentSearchLatency	The difference in total time, in milliseconds, taken by all searches using concurrent segment search in a node between minute N and minute (N-1). Relevant node statistics: Average Relevant cluster statistics: Average, Maximum
IndexingLatency	The difference in total time, in milliseconds, taken by all indexing operations in a node between minute N and minute (N-1). Relevant node statistics: Average Relevant cluster statistics: Average, Maximum

Metric	Description
IndexingRate	The number of indexing operations per minute. A single call to the _bulk API that adds two documents and updates two counts as four operations, which might be spread across one or more nodes. If that index has one or more replicas and is on an OpenSearch domain without optimized instances, other nodes in the cluster also record a total of four indexing operations. For OpenSearch domains with optimized instances, other nodes with replicas do not record any operation. Document deletions do not count towards this metric. Relevant node statistics: Average Relevant cluster statistics: Average, Maximum, Sum
SearchLatency	The difference in total time, in milliseconds, taken by all searches in a node between minute N and minute (N-1). Relevant node statistics: Average Relevant cluster statistics: Average, Maximum
SearchRate	The total number of search requests per minute for all shards on a data node. A single call to the _search API might return results from many different shards. If five of these shards are on one node, the node would report 5 for this metric, even though the client only made one request. Relevant node statistics: Average Relevant cluster statistics: Average, Maximum, Sum
SegmentCount	The number of segments on a data node. The more segments you have, the longer each search takes. OpenSearch occasionally merges smaller segments into a larger one. Relevant node statistics: Maximum, Average Relevant cluster statistics: Sum, Maximum, Average

Metric	Description
SysMemoryUtilization	The percentage of the instance's memory that is in use. High values for this metric are normal and usually do not represent a problem with your cluster. For a better indicator of potential performance and stability issues, see the JVMMemoryPressure metric. Relevant node statistics: Minimum, Maximum, Average Relevant cluster statistics: Minimum, Maximum, Average
JVMGCYoungCollectionCount	The number of times that "young generation" garbage collection has run. A large, ever-growing number of runs is a normal part of cluster operations. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
JVMGCYoungCollectionTime	The amount of time, in milliseconds, that the cluster has spent performing "young generation" garbage collection. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
JVMGCOldCollectionCount	The number of times that "old generation" garbage collection has run. In a cluster with sufficient resources, this number should remain small and grow infrequently. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
JVMGCOldCollectionTime	The amount of time, in milliseconds, that the cluster has spent performing "old generation" garbage collection. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average

Metric	Description
OpenSearchDashboardsConcurrentConnections	The number of active concurrent connections to OpenSearch Dashboards. If this number is consistently high, consider scaling your cluster. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
OpenSearchDashboardsHealthyNode	A health check for the individual OpenSearch Dashboards node. A value of 1 indicates normal behavior. A value of 0 indicates that Dashboards is inaccessible. Relevant node statistics: Minimum Relevant cluster statistics: Minimum, Maximum, Average
OpenSearchDashboardsHeapTotal	The amount of heap memory allocated to OpenSearch Dashboards in MiB. Different EC2 instance types can impact the exact memory allocation. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
OpenSearchDashboardsHeapUsed	The absolute amount of heap memory used by OpenSearch Dashboards in MiB. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
OpenSearchDashboardsHeapUtilization	The maximum percentage of available heap memory used by OpenSearch Dashboards. If this value increases above 80%, consider scaling your cluster. Relevant node statistics: Maximum Relevant cluster statistics: Minimum, Maximum, Average

Metric	Description
OpenSearchDashboardsOS1MinuteLoad	The one-minute CPU load average for OpenSearch Dashboards. The CPU load should ideally stay below 1.00. While temporary spikes are fine, we recommend increasing the size of the instance type if this metric is consistently above 1.00. Relevant node statistics: Average Relevant cluster statistics: Average, Maximum
OpenSearchDashboardsRequestTotal	The total count of HTTP requests made to OpenSearch Dashboards. If your system is slow or you see high numbers of Dashboards requests, consider increasing the size of the instance type. Relevant node statistics: Sum Relevant cluster statistics: Sum
OpenSearchDashboardsResponseTimesMaxInMillis	The maximum amount of time, in milliseconds, that it takes for OpenSearch Dashboards to respond to a request. If requests consistently take a long time to return results, consider increasing the size of the instance type. Relevant node statistics: Maximum Relevant cluster statistics: Maximum, Average
SearchTaskCancelled	The number of coordinator node cancellations. Relevant node statistics: Sum Relevant cluster statistics: Sum
SearchShardTaskCancelled	The number of data node cancellations. Relevant node statistics: Sum Relevant cluster statistics: Sum,

Metric	Description
ThreadpoolForce_mergeQueue	The number of queued tasks in the force merge thread pool. If the queue size is consistently high, consider scaling your cluster. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
ThreadpoolForce_mergeRejected	The number of rejected tasks in the force merge thread pool. If this number continually grows, consider scaling your cluster. Relevant node statistics: Maximum Relevant cluster statistics: Sum
ThreadpoolForce_mergeThreads	The size of the force merge thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Average, Sum
ThreadpoolIndexQueue	The number of queued tasks in the index thread pool. If the queue size is consistently high, consider scaling your cluster. The maximum index queue size is 200. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
ThreadpoolIndexRejected	The number of rejected tasks in the index thread pool. If this number continually grows, consider scaling your cluster. Relevant node statistics: Maximum Relevant cluster statistics: Sum
ThreadpoolIndexThreads	The size of the index thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Average, Sum

Metric	Description
ThreadpoolSearchQueue	The number of queued tasks in the search thread pool. If the queue size is consistently high, consider scaling your cluster. The maximum search queue size is 1,000. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
ThreadpoolSearchRejected	The number of rejected tasks in the search thread pool. If this number continually grows, consider scaling your cluster. Relevant node statistics: Maximum Relevant cluster statistics: Sum
ThreadpoolSearchThreads	The size of the search thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Average, Sum
Threadpoolsql-workerQueue	The number of queued tasks in the SQL search thread pool. If the queue size is consistently high, consider scaling your cluster. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
Threadpoolsql-workerRejected	The number of rejected tasks in the SQL search thread pool. If this number continually grows, consider scaling your cluster. Relevant node statistics: Maximum Relevant cluster statistics: Sum
Threadpoolsql-workerThreads	The size of the SQL search thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Average, Sum

Metric	Description
ThreadpoolBulkQueue	The number of queued tasks in the bulk thread pool. If the queue size is consistently high, consider scaling your cluster. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
ThreadpoolBulkRejected	The number of rejected tasks in the bulk thread pool. If this number continually grows, consider scaling your cluster. Relevant node statistics: Maximum Relevant cluster statistics: Sum
ThreadpoolBulkThreads	The size of the bulk thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Average, Sum
ThreadpoolIndexSearcherQueue	The number of queued tasks in the index searcher thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
ThreadpoolIndexSearcherRejected	The number of rejected tasks in the index searcher thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Sum
ThreadpoolIndexSearcherThreads	The size of the index searcher thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Average, Sum

Metric	Description
`ThreadpoolWriteThreads`	The size of the write thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Average, Sum
`ThreadpoolWriteQueue`	The number of queued tasks in the write thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Average, Sum
`ThreadpoolWriteRejected`	The number of rejected tasks in the write thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Average, Sum **Note** Because the default write queue size was increased from 200 to 10000 in version 7.1, this metric is no longer the only indicator of rejections from OpenSearch Service. Use the `CoordinatingWriteRejected` , `PrimaryWriteRejected` , and `ReplicaWriteRejected` metrics to monitor rejections in versions 7.1 and later.
`CoordinatingWriteRejected`	The total number of rejections happened on the coordinating node due to indexing pressure since the last OpenSearch Service process startup. Relevant node statistics: Maximum Relevant cluster statistics: Average, Sum This metric is available in version 7.1 and above.

Metric	Description
PrimaryWriteRejected	The total number of rejections happened on the primary shards due to indexing pressure since the last OpenSearch Service process startup. Relevant node statistics: Maximum Relevant cluster statistics: Average, Sum This metric is available in version 7.1 and above.
ReplicaWriteRejected	The total number of rejections happened on the replica shards due to indexing pressure since the last OpenSearch Service process startup. Relevant node statistics: Maximum Relevant cluster statistics: Average, Sum This metric is available in version 7.1 and above.
WorkloadManagementEnabled	Indicates whether the workload management feature is enabled. A value of 1 means it is enabled, and a value of 0 means its monitor_only or disabled. Relevant node statistics: Maximum, Minimum Relevant cluster statistics: Average, Sum This metric is available in version 7.1 and above.
SoftQueryGroupCount	Number of query groups under soft mode in the domain. Relevant node statistics: Average, Maximum Relevant cluster statistics: Average, Maximum, Sum This metric is available in version 7.1 and above.

Metric	Description
EnforcedQueryGroupCount	Number of query groups under enforced mode in the domain. Relevant node statistics: Average, Maximum Relevant cluster statistics: Average, Maximum, Sum This metric is available in version 7.1 and above.

UltraWarm metrics

Amazon OpenSearch Service provides the following metrics for UltraWarm nodes.

Metric	Description
WarmCPUUtilization	The percentage of CPU usage for UltraWarm nodes in the cluster. Maximum shows the node with the highest CPU usage. Average represents all UltraWarm nodes in the cluster. This metric is also available for individual UltraWarm nodes. Relevant statistics: Maximum, Average
WarmFreeStorageSpace	The amount of free warm storage space in MiB. Because UltraWarm uses Amazon S3 rather than attached disks, Sum is the only relevant statistic. You must leave the period at one minute to get an accurate value. Relevant statistics: Sum
WarmSearchableDocuments	The total number of searchable documents across all warm indexes in the cluster. You must leave the period at one minute to get an accurate value. Relevant statistics: Sum
WarmSearchLatency	The difference in total time, in milliseconds, taken by all searches in an UltraWarm between minute N and minute (N-1).

Metric	Description
	Relevant node statistics: Average Relevant cluster statistics: Average, Maximum
`WarmSearchRate`	The total number of search requests per minute for all shards on an UltraWarm node. A single call to the `_search` API might return results from many different shards. If five of these shards are on one node, the node would report 5 for this metric, even though the client only made one request. Relevant node statistics: Average Relevant cluster statistics: Average, Maximum, Sum
`WarmStorageSpaceUtilization`	The total amount of warm storage space, in MiB, that the cluster is using. Relevant statistics: Maximum
`HotStorageSpaceUtilization`	The total amount of hot storage space that the cluster is using. Relevant statistics: Maximum
`WarmSysMemoryUtilization`	The percentage of the warm node's memory that is in use. Relevant statistics: Maximum
`HotToWarmMigrationQueueSize`	The number of indexes currently waiting to migrate from hot to warm storage. Relevant statistics: Maximum
`WarmToHotMigrationQueueSize`	The number of indexes currently waiting to migrate from warm to hot storage. Relevant statistics: Maximum

Metric	Description
HotToWarmMigrationFailureCount	The total number of failed hot to warm migrations. Relevant statistics: Sum
HotToWarmMigrationForceMergeLatency	The average latency of the force merge stage of the migration process. If this stage consistently takes too long, consider increasing `index.ultrawarm.migration.force_merge.max_num_segments` . Relevant statistics: Average
HotToWarmMigrationSnapshotLatency	The average latency of the snapshot stage of the migration process. If this stage consistently takes too long, ensure that your shards are appropriately sized and distributed throughout the cluster. Relevant statistics: Average
HotToWarmMigrationProcessingLatency	The average latency of successful hot to warm migrations, *not* including time spent in the queue. This value is the sum of the amount of time it takes to complete the force merge, snapshot, and shard relocation stages of the migration process. Relevant statistics: Average
HotToWarmMigrationSuccessCount	The total number of successful hot to warm migrations. Relevant statistics: Sum
HotToWarmMigrationSuccessLatency	The average latency of successful hot to warm migrations, including time spent in the queue. Relevant statistics: Average
WarmThreadpoolSearchThreads	The size of the UltraWarm search thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Average, Sum

Metric	Description
WarmThreadpoolSearchRejected	The number of rejected tasks in the UltraWarm search thread pool. If this number continually grows, consider adding more UltraWarm nodes. Relevant node statistics: Maximum Relevant cluster statistics: Sum
WarmThreadpoolSearchQueue	The number of queued tasks in the UltraWarm search thread pool. If the queue size is consistently high, consider adding more UltraWarm nodes. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
WarmJVMMemoryPressure	The maximum percentage of the Java heap used for the UltraWarm nodes. Relevant statistics: Maximum **Note** The logic for this metric changed in service software R20220323. For more information, see the release notes.
WarmOldGenJVMMemoryPressure	The maximum percentage of the Java heap used for the "old generation" per UltraWarm node. Relevant statistics: Maximum
WarmJVMGCYoungCollectionCount	The number of times that "young generation" garbage collection has run on UltraWarm nodes. A large, ever-growing number of runs is a normal part of cluster operations. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average

Metric	Description
`WarmJVMGCYoungCollectionTime`	The amount of time, in milliseconds, that the cluster has spent performing "young generation" garbage collection on UltraWarm nodes. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
`WarmJVMGCOldCollectionCount`	The number of times that "old generation" garbage collection has run on UltraWarm nodes. In a cluster with sufficient resources, this number should remain small and grow infrequently. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
`WarmConcurrentSearchRate`	The total number of search requests using concurrent segment search per minute for all shards on a UltraWarm node. A single call to the `_search` API might return results from many different shards. If five of these shards are on one node, the node would report 5 for this metric, even though the client only made one request. Relevant node statistics: Average Relevant cluster statistics: Sum, Maximum, Average
`WarmConcurrentSearchLatency`	The difference in total time, in milliseconds, taken by all searches using concurrent segment search in a UltraWarm node between minute N and minute (N-1). Relevant node statistics: Average Relevant cluster statistics: Maximum, Average

Metric	Description
WarmThreadpoolIndexSearcherQueue	The number of queued tasks in the UltraWarm index searcher thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Maximum, Average
WarmThreadpoolIndexSearcherRejected	The number of rejected tasks in the UltraWarm index searcher thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Sum
WarmThreadpoolIndexSearcherThreads	The size of the UltraWarm index searcher thread pool. Relevant node statistics: Maximum Relevant cluster statistics: Sum, Average

Cold storage metrics

Amazon OpenSearch Service provides the following metrics for cold storage.

Metric	Description
ColdStorageSpaceUtilization	The total amount of cold storage space, in MiB, that the cluster is using. Relevant statistics: Max
ColdToWarmMigrationFailureCount	The total number of failed cold to warm migrations. Relevant statistics: Sum
ColdToWarmMigrationLatency	The amount of time for successful cold to warm migrations to complete.

Metric	Description
	Relevant statistics: Average
ColdToWarmMigrationQueueSize	The number of indexes currently waiting to migrate from cold to warm storage. Relevant statistics: Maximum
ColdToWarmMigrationSuccessCount	The total number of successful cold to warm migrations. Relevant statistics: Sum
WarmToColdMigrationFailureCount	The total number of failed warm to cold migrations. Relevant statistics: Sum
WarmToColdMigrationLatency	The amount of time for successful warm to cold migrations to complete. Relevant statistics: Average
WarmToColdMigrationQueueSize	The number of indexes currently waiting to migrate from warm to cold storage. Relevant statistics: Maximum
WarmToColdMigrationSuccessCount	The total number of successful warm to cold migrations. Relevant statistics: Sum

OR1 metrics

Amazon OpenSearch Service provides the following metrics for OR1 instances.

Metric	Description
RemoteStorageUsedSpace	The total amount of Amazon S3 space, in MiB, that the cluster is using.

Metric	Description
	Relevant statistics: Sum
`RemoteStorageWriteRejected`	The total number of requests rejected on primary shards due to remote storage and replication pressure. This is calculated starting from the last OpenSearch Service process startup. Relevant statistics: Sum
`ReplicationLagMaxTime`	The amount of time, in milliseconds, that replica shards are behind the primary shards. Relevant statistics: Maximum

Alerting metrics

Amazon OpenSearch Service provides the following metrics for alerting.

Metric	Description
`AlertingDegraded`	A value of 1 means that either the alerting index is red or one or more nodes is not on schedule. A value of 0 indicates normal behavior. Relevant statistics: Maximum
`AlertingIndexExists`	A value of 1 means the `.opensearch-alerting-config` index exists. A value of 0 means it does not. Until you use the alerting feature for the first time, this value remains 0. Relevant statistics: Maximum
`AlertingIndexStatus.green`	The health of the index. A value of 1 means green. A value of 0 means that the index either doesn't exist or isn't green. Relevant statistics: Maximum
`AlertingIndexStatus.red`	The health of the index. A value of 1 means red. A value of 0 means that the index either doesn't exist or isn't red.

Metric	Description
	Relevant statistics: Maximum
`AlertingIndexStatus.yellow`	The health of the index. A value of 1 means yellow. A value of 0 means that the index either doesn't exist or isn't yellow. Relevant statistics: Maximum
`AlertingNodesNotOnSchedule`	A value of 1 means some jobs are not running on schedule. A value of 0 means that all alerting jobs are running on schedule (or that no alerting jobs exist). Check the OpenSearch Service console or make a `_nodes/stats` request to see if any nodes show high resource usage. Relevant statistics: Maximum
`AlertingNodesOnSchedule`	A value of 1 means that all alerting jobs are running on schedule (or that no alerting jobs exist). A value of 0 means some jobs are not running on schedule. Relevant statistics: Maximum
`AlertingScheduledJobEnabled`	A value of 1 means that the `opensearch.scheduled_jobs.enabled` cluster setting is true. A value of 0 means it is false, and scheduled jobs are disabled. Relevant statistics: Maximum

Anomaly detection metrics

Amazon OpenSearch Service provides the following metrics for anomaly detection.

Metric	Description
`ADPluginUnhealthy`	A value of 1 means that the anomaly detection plugin is not functioning properly, either because of a high number of failures or because one of the indexes that it uses is red. A value of 0 indicates the plugin is working as expected.

Metric	Description
	Relevant statistics: Maximum
ADExecuteRequestCount	The number of requests to detect anomalies. Relevant statistics: Sum
ADExecuteFailureCount	The number of failed requests to detect anomalies. Relevant statistics: Sum
ADHCExecuteFailureCount	The number of failed requests to detect anomalies for high cardinality detectors. Relevant statistics: Sum
ADHCExecuteRequestCount	The number of requests to detect anomalies for high cardinality detectors. Relevant statistics: Sum
ADAnomalyResultsIndexStatusIndexExists	A value of 1 means the index that the `.opensearch-anomaly-results` alias points to exists. Until you use anomaly detection for the first time, this value remains 0. Relevant statistics: Maximum
ADAnomalyResultsIndexStatus.red	A value of 1 means the index that the `.opensearch-anomaly-results` alias points to is red. A value of 0 means it is not. Until you use anomaly detection for the first time, this value remains 0. Relevant statistics: Maximum
ADAnomalyDetectorsIndexStatusIndexExists	A value of 1 means that the `.opensearch-anomaly-detectors` index exists. A value of 0 means it does not. Until you use anomaly detection for the first time, this value remains 0. Relevant statistics: Maximum

Metric	Description
ADAnomalyDetectorsIndexStatus.red	A value of 1 means that the .opensearch-anomaly-detectors index is red. A value of 0 means it is not. Until you use anomaly detection for the first time, this value remains 0. Relevant statistics: Maximum
ADModelsCheckpointIndexStatusIndexExists	A value of 1 means that the .opensearch-anomaly-checkpoints index exists. A value of 0 means it does not. Until you use anomaly detection for the first time, this value remains 0. Relevant statistics: Maximum
ADModelsCheckpointIndexStatus.red	A value of 1 means that the .opensearch-anomaly-checkpoints index is red. A value of 0 means it is not. Until you use anomaly detection for the first time, this value remains 0. Relevant statistics: Maximum

Asynchronous search metrics

Amazon OpenSearch Service provides the following metrics for asynchronous search.

Asynchronous search coordinator node statistics (per coordinator node)

Metric	Description
AsynchronousSearchSubmissionRate	The number of asynchronous searches submitted in the last minute.
AsynchronousSearchInitializedRate	The number of asynchronous searches initialized in the last minute.

Metric	Description
AsynchronousSearchRunningCurrent	The number of asynchronous searches currently running.
AsynchronousSearchCompletionRate	The number of asynchronous searches successfully completed in the last minute.
AsynchronousSearchFailureRate	The number of asynchronous searches that completed and failed in the last minute.
AsynchronousSearchPersistRate	The number of asynchronous searches that persisted in the last minute.
AsynchronousSearchPersistFailedRate	The number of asynchronous searches that failed to persist in the last minute.
AsynchronousSearchRejected	The total number of asynchronous searches rejected since the node up time.
AsynchronousSearchCancelled	The total number of asynchronous searches cancelled since the node up time.
AsynchronousSearchMaxRunningTime	The duration of longest running asynchronous search on a node in the last minute.

Asynchronous search cluster statistics

Metric	Description
AsynchronousSearchStoreHealth	The health of the store in the persisted index (RED/non-RED) in the last minute.
AsynchronousSearchStoreSize	The size of the system index across all shards in the last minute.
AsynchronousSearchStoredResponseCount	The numbers of stored responses in the system index in the last minute.

Auto-Tune metrics

Amazon OpenSearch Service provides the following metrics for Auto-Tune.

Metric	Description
AutoTuneChangesHistoryHeapSize	The change history in MiB for heap size tuning values.
AutoTuneChangesHistoryJVMYoungGenArgs	The change history for JVM YongGen arguments.
AutoTuneFailed	A boolean that indicates if the Auto-Tune change failed.
AutoTuneSucceeded	A boolean that indicates if the Auto-Tune change succeeded.
AutoTuneValue	The queue change history (count) and cache tunings change history (in MiB) for non-disruptive changes.

Multi-AZ with Standby metrics

Amazon OpenSearch Service provides the following metrics for Multi-AZ with Standby.

Node-level metrics for data nodes in active Availability Zones

Metric	Description
`CPUUtilization`	The percentage of CPU usage for data nodes in the cluster. Maximum shows the node with the highest CPU usage. Average represents all nodes in the cluster. This metric is also available for individual nodes.
`FreeStorageSpace`	The free space for data nodes in the cluster. `Sum` shows total free space for the cluster, but you must leave the period at one minute to get an accurate value. `Minimum` and `Maximum` show the nodes with the least and most free space, respectively. This metric is also available for individual nodes. OpenSearch Service throws a `ClusterBlockException` when this metric reaches `0`. To recover, you must either delete indexes, add larger instances, or add EBS-based storage to existing instances. To learn more, see the section called "Lack of available storage space". The OpenSearch Service console displays this value in GiB. The Amazon CloudWatch console displays it in MiB.
`JVMMemoryPressure`	The maximum percentage of the Java heap used for all data nodes in the cluster. OpenSearch Service uses half of an instance's RAM for the Java heap, up to a heap size of 32 GiB. You can scale instances vertically up to 64 GiB of RAM, at which point you can scale horizontally by adding instances. See the section called "Recommended CloudWatch alarms".
`SysMemoryUtilization`	The percentage of the instance's memory that is in use. High values for this metric are normal and usually do not represent a problem with your cluster. For a better indicator of potential performance and stability issues, see the `JVMMemoryPressure` metric.

Metric	Description
IndexingLatency	The difference in total time, in milliseconds, taken by all indexing operations in a node between minute N and minute (N-1).
IndexingRate	The number of indexing operations per minute.
SearchLatency	The difference in total time, in milliseconds, taken by all searches in a node between minute N and minute (N-1).
SearchRate	The total number of search requests per minute for all shards on a data node.
ThreadpoolSearchQueue	The number of queued tasks in the search thread pool. If the queue size is consistently high, consider scaling your cluster. The maximum search queue size is 1,000.
ThreadpoolWriteQueue	The number of queued tasks in the write thread pool.
ThreadpoolSearchRejected	The number of rejected tasks in the search thread pool. If this number continually grows, consider scaling your cluster.
ThreadpoolWriteRejected	The number of rejected tasks in the write thread pool.

Cluster-level metrics for clusters in active Availability Zones

Metric	Description
DataNodes	The total number of active and standby shards.
DataNodesShards.active	The total number of active primary and replica shards.

Metric	Description
DataNodes Shards.unassigned	The number of shards that are not allocated to nodes in the cluster.
DataNodes Shards.initializing	The number of shards that are under initialization.
DataNodes Shards.relocating	The number of shards that are under relocation.

Availability Zone rotation metrics

If `ActiveReads.`*`Availability-Zone`* `= 1`, then the zone is active. If `ActiveReads.`*`Availability-Zone`* `= 0`, then the zone is in standby.

Point in time metrics

Amazon OpenSearch Service provides the following metrics for point in time (PIT) searches.

PIT coordinator node statistics (per coordinator node)

Metric	Description
CurrentPointInTime	The number of active PIT search contexts in the node.
TotalPointInTime	The number of expired PIT search contexts since the node up time.
AvgPointInTimeAliveTime	The average keep alive of PIT search contexts since the node up time.
HasActivePointInTime	A value of 1 indicates that there are active PIT contexts on nodes since the node up time. A value of 0 means there are not.

Metric	Description
HasUsedPointInTime	A value of 1 indicates that there are expired PIT contexts on nodes since the node up time. A value of 0 means there are not.

SQL metrics

Amazon OpenSearch Service provides the following metrics for SQL support.

Metric	Description
SQLFailedRequestCountByCusErr	The number of requests to the _sql API that failed due to a client issue. For example, a request might return HTTP status code 400 due to an IndexNotFoundException . Relevant statistics: Sum
SQLFailedRequestCountBySysErr	The number of requests to the _sql API that failed due to a server problem or feature limitation. For example, a request might return HTTP status code 503 due to a VerificationException . Relevant statistics: Sum
SQLRequestCount	The number of requests to the _sql API. Relevant statistics: Sum
SQLDefaultCursorRequestCount	Similar to SQLRequestCount , but only counts pagination requests. Relevant statistics: Sum
SQLUnhealthy	A value of 1 indicates that, in response to certain requests, the SQL plugin is returning 5*xx* response codes or passing invalid query DSL to OpenSearch. Other requests should continue to succeed. A value of 0 indicates no recent failures. If you see a sustained value of 1, troubleshoot the requests your clients are making to the plugin. Relevant statistics: Maximum

k-NN metrics

Amazon OpenSearch Service includes the following metrics for the k-nearest neighbor (k-NN) plugin.

Metric	Description
KNNCacheCapacityReached	Per-node metric for whether cache capacity has been reached. This metric is only relevant to approximate k-NN search. Relevant statistics: Maximum
KNNCircuitBreakerTriggered	Per-cluster metric for whether the circuit breaker is triggered. If any nodes return a value of 1 for KNNCacheCapacityReached , this value will also return 1. This metric is only relevant to approximate k-NN search. Relevant statistics: Maximum
KNNEvictionCount	Per-node metric for the number of graphs that have been evicted from the cache due to memory constraints or idle time. Explicit evictions that occur because of index deletion are not counted. This metric is only relevant to approximate k-NN search. Relevant statistics: Sum
KNNGraphIndexErrors	Per-node metric for the number of requests to add the knn_vector field of a document to a graph that produced an error. Relevant statistics: Sum
KNNGraphIndexRequests	Per-node metric for the number of requests to add the knn_vector field of a document to a graph. Relevant statistics: Sum

Metric	Description
KNNGraphMemoryUsage	Per-node metric for the current cache size (total size of all graphs in memory) in kilobytes. This metric is only relevant to approximate k-NN search. Relevant statistics: Average
KNNGraphQueryErrors	Per-node metric for the number of graph queries that produced an error. Relevant statistics: Sum
KNNGraphQueryRequests	Per-node metric for the number of graph queries. Relevant statistics: Sum
KNNHitCount	Per-node metric for the number of cache hits. A cache hit occurs when a user queries a graph that is already loaded into memory. This metric is only relevant to approximate k-NN search. Relevant statistics: Sum
KNNLoadExceptionCount	Per-node metric for the number of times an exception occurred while trying to load a graph into the cache. This metric is only relevant to approximate k-NN search. Relevant statistics: Sum
KNNLoadSuccessCount	Per-node metric for the number of times the plugin successfu lly loaded a graph into the cache. This metric is only relevant to approximate k-NN search. Relevant statistics: Sum

Metric	Description
KNNMissCount	Per-node metric for the number of cache misses. A cache miss occurs when a user queries a graph that is not yet loaded into memory. This metric is only relevant to approximate k-NN search. Relevant statistics: Sum
KNNQueryRequests	Per-node metric for the number of query requests the k-NN plugin received. Relevant statistics: Sum
KNNScriptCompilationErrors	Per-node metric for the number of errors during script compilation. This statistic is only relevant to k-NN score script search. Relevant statistics: Sum
KNNScriptCompilations	Per-node metric for the number of times the k-NN script has been compiled. This value should usually be 1 or 0, but if the cache containing the compiled scripts is filled, the k-NN script might be recompiled. This statistic is only relevant to k-NN score script search. Relevant statistics: Sum
KNNScriptQueryErrors	Per-node metric for the number of errors during script queries. This statistic is only relevant to k-NN score script search. Relevant statistics: Sum
KNNScriptQueryRequests	Per-node metric for the total number of script queries. This statistic is only relevant to k-NN score script search. Relevant statistics: Sum

Metric	Description
KNNTotalLoadTime	The time in nanoseconds that k-NN has taken to load graphs into the cache. This metric is only relevant to approximate k-NN search. Relevant statistics: Sum

Cross-cluster search metrics

Amazon OpenSearch Service provides the following metrics for cross-cluster search.

Source domain metrics

Metric	Dimension	Description
CrossClusterOutboundConnections	ConnectionId	Number of connected nodes. If your response includes one or more skipped domains, use this metric to trace any unhealthy connections. If this number drops to 0, then the connection is unhealthy.
CrossClusterOutboundRequests	ConnectionId	Number of search requests sent to the destination domain. Use to check if the load of cross-cluster search requests are overwhelming your domain, correlate any spike in this metric with any JVM/CPU spike.

Destination domain metric

Metric	Dimension	Description
CrossClusterInboundRequests	ConnectionId	Number of incoming connection requests received from the source domain.

Add a CloudWatch alarm in the event that you lose a connection unexpectedly. For steps to create an alarm, see Create a CloudWatch Alarm Based on a Static Threshold.

Cross-cluster replication metrics

Amazon OpenSearch Service provides the following metrics for cross-cluster replication.

Metric	Description
ReplicationRate	The average rate of replication operations per second. This metric is similar to the IndexingRate metric.
LeaderCheckPoint	For a specific connection, the sum of leader checkpoint values across all replicating indexes. You can use this metric to measure replication latency.
FollowerCheckPoint	For a specific connection, the sum of follower checkpoint values across all replicating indexes. You can use this metric to measure replication latency.
ReplicationNumSyncingIndices	The number of indexes that have a replication status of SYNCING.
ReplicationNumBootstrappingIndices	The number of indexes that have a replication status of BOOTSTRAPPING .
ReplicationNumPausedIndices	The number of indexes that have a replication status of PAUSED.
ReplicationNumFailedIndices	The number of indexes that have a replication status of FAILED.
CrossClusterOutboundReplicationRequests	The number of replication transport requests on the follower domain. Transport requests are internal and occur each time a replication API operation is called. They also occur when the follower domain polls changes from the leader domain.

Metric	Description
CrossClusterInboundReplicationRequests	The number of replication transport requests on the leader domain. Transport requests are internal and occur each time a replication API operation is called.
AutoFollowNumSuccessStartReplication	The number of follower indexes that have been successfully created by a replication rule for a specific connection.
AutoFollowNumFailedStartReplication	The number of follower indexes that failed to be created by a replication rule when there was a matching pattern. This problem might arise due to a network issue on the remote cluster, or a security issue (i.e. the associated role doesn't have permission to start replication).
AutoFollowLeaderCallFailure	Whether there have been any failed queries from the follower index to the leader index to pull new data. A value of 1 means that there have been 1 or more failed calls in the last minute.

Learning to Rank metrics

Amazon OpenSearch Service provides the following metrics for Learning to Rank.

Metric	Description
LTRRequestTotalCount	Total count of ranking requests.
LTRRequestErrorCount	Total count of unsuccessful requests.
LTRStatus.red	Tracks if one of the indexes needed to run the plugin is red.
LTRMemoryUsage	Total memory used by the plugin.

Metric	Description
LTRFeatureMemoryUsageInBytes	The amount of memory, in bytes, used by Learning to Rank feature fields.
LTRFeaturesetMemoryUsageInBytes	The amount of memory, in bytes, used by all Learning to Rank feature sets.
LTRModelMemoryUsageInBytes	The amount of memory, in bytes, used by all Learning to Rank models.

Piped Processing Language metrics

Amazon OpenSearch Service provides the following metrics for Piped Processing Language.

Metric	Description
PPLFailedRequestCountByCusErr	The number of requests to the _ppl API that failed due to a client issue. For example, a request might return HTTP status code 400 due to an IndexNotFoundException .
PPLFailedRequestCountBySysErr	The number of requests to the _ppl API that failed due to a server problem or feature limitation. For example, a request might return HTTP status code 503 due to a VerificationException .
PPLRequestCount	The number of requests to the _ppl API.

Monitoring OpenSearch logs with Amazon CloudWatch Logs

Amazon OpenSearch Service exposes the following OpenSearch logs through Amazon CloudWatch Logs:

- Error logs

- Search request slow logs
- Shard slow logs
- Audit logs

Search shard slow logs, indexing shard slow logs, and error logs are useful for troubleshooting performance and stability issues. Audit logs track user activity for compliance purposes. All the logs are *disabled* by default. If enabled, standard CloudWatch pricing applies.

> **Note**
>
> Error logs are available only for OpenSearch and Elasticsearch versions 5.1 and later. Slow logs are available for all OpenSearch and Elasticsearch versions.

For its logs, OpenSearch uses Apache Log4j 2 and its built-in log levels (from least to most severe) of `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, and `FATAL`.

If you enable error logs, OpenSearch Service publishes log lines of `WARN`, `ERROR`, and `FATAL` to CloudWatch. OpenSearch Service also publishes several exceptions from the `DEBUG` level, including the following:

- `org.opensearch.index.mapper.MapperParsingException`
- `org.opensearch.index.query.QueryShardException`
- `org.opensearch.action.search.SearchPhaseExecutionException`
- `org.opensearch.common.util.concurrent.OpenSearchRejectedExecutionException`
- `java.lang.IllegalArgumentException`

Error logs can help with troubleshooting in many situations, including the following:

- Painless script compilation issues
- Invalid queries
- Indexing issues
- Snapshot failures
- Index State Management migration failures

> **Note**
>
> OpenSearch Service does not log all errors that occur.

Topics

Enabling log publishing (console)

The OpenSearch Service console is the simplest way to enable the publishing of logs to CloudWatch.

To enable log publishing to CloudWatch (console)

1. Go to https://aws.amazon.com, and then choose **Sign In to the Console**.
2. Under **Analytics**, choose **Amazon OpenSearch Service**.
3. Select the domain you want to update.
4. On the **Logs** tab, select a log type and choose **Enable**.
5. Create a new CloudWatch log group or choose an existing one.

   > **Note**
   >
   > If you plan to enable multiple logs, we recommend publishing each to its own log group. This separation makes the logs easier to scan.

6. Choose an access policy that contains the appropriate permissions, or create a policy using the JSON that the console provides:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "es.amazonaws.com"
      },
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogStream"
      ],
      "Resource": "cw_log_group_arn:*"
    }
  ]
}
```

We recommend that you add the `aws:SourceAccount` and `aws:SourceArn` condition keys to the policy to protect yourself against the confused deputy problem. The source account is the owner of the domain and the source ARN is the ARN of the domain. Your domain must be on service software R20211203 or later in order to add these condition keys.

For example, you could add the following condition block to the policy:

```
"Condition": {
    "StringEquals": {
        "aws:SourceAccount": "account-id"
    },
    "ArnLike": {
        "aws:SourceArn": "arn:aws:es:region:account-id:domain/domain-name"
    }
}
```

Important

CloudWatch Logs supports 10 resource policies per Region. If you plan to enable logs for several OpenSearch Service domains, you should create and reuse a broader policy that includes multiple log groups to avoid reaching this limit. For steps on updating your policy, see the section called “Enabling log publishing (AWS CLI)”.

7. Choose **Enable**.

   The status of your domain changes from **Active** to **Processing**. The status must return to **Active** before log publishing is enabled. This change typically takes 30 minutes, but can take longer depending on your domain configuration.

If you enabled one of the shard slow logs, see the section called "Setting shard slow log thresholds". If you enabled audit logs, see the section called "Step 2: Turn on audit logs in OpenSearch Dashboards". If you enabled only error logs, you don't need to perform any additional configuration steps.

Enabling log publishing (AWS CLI)

Before you can enable log publishing, you need a CloudWatch log group. If you don't already have one, you can create one using the following command:

```
aws logs create-log-group --log-group-name my-log-group
```

Enter the next command to find the log group's ARN, and then *make a note of it*:

```
aws logs describe-log-groups --log-group-name my-log-group
```

Now you can give OpenSearch Service permissions to write to the log group. You must provide the log group's ARN near the end of the command:

```
aws logs put-resource-policy \
  --policy-name my-policy \
  --policy-document '{ "Version": "2012-10-17", "Statement": [{ "Sid": "",
 "Effect": "Allow", "Principal": { "Service": "es.amazonaws.com"}, "Action":
[ "logs:PutLogEvents","logs:CreateLogStream"],"Resource": "cw_log_group_arn:*"}]}'
```

Important

CloudWatch Logs supports 10 resource policies per Region. If you plan to enable shard slow logs for several OpenSearch Service domains, you should create and reuse a broader policy that includes multiple log groups to avoid reaching this limit.

If you need to review this policy at a later time, use the `aws logs describe-resource-policies` command. To update the policy, issue the same `aws logs put-resource-policy` command with a new policy document.

Finally, you can use the `--log-publishing-options` option to enable publishing. The syntax for the option is the same for both the `create-domain` and `update-domain-config` commands.

Parameter	Valid Values
`--log-publishing-options`	`SEARCH_SLOW_LOGS={CloudWatchLogsLogGroupArn= cw_log_group_arn ,Enabled=true|false}`
	`INDEX_SLOW_LOGS={CloudWatchLogsLogGroupArn= cw_log_group_arn ,Enabled=true|false}`
	`ES_APPLICATION_LOGS={CloudWatchLogsLogGroupArn= cw_log_group_arn ,Enabled=true|false}`
	`AUDIT_LOGS={CloudWatchLogsLogGroupArn= cw_log_group_arn ,Enabled=true|false}`

> **Note**
>
> If you plan to enable multiple logs, we recommend publishing each to its own log group. This separation makes the logs easier to scan.

Example

The following example enables the publishing of search and indexing shard slow logs for the specified domain:

```
aws opensearch update-domain-config \
  --domain-name my-domain \
  --log-publishing-options
 "SEARCH_SLOW_LOGS={CloudWatchLogsLogGroupArn=arn:aws:logs:us-east-1:123456789012:log-group:my-log-
group,Enabled=true},INDEX_SLOW_LOGS={CloudWatchLogsLogGroupArn=arn:aws:logs:us-east-1:123456789012:log-group:my-other-log-group,Enabled=true}"
```

To disable publishing to CloudWatch, run the same command with `Enabled=false`.

If you enabled one of the shard slow logs, see the section called "Setting shard slow log thresholds". If you enabled audit logs, see the section called "Step 2: Turn on audit logs in OpenSearch Dashboards". If you enabled only error logs, you don't need to perform any additional configuration steps.

Enabling log publishing (AWS SDKs)

Before you can enable log publishing, you must first create a CloudWatch log group, get its ARN, and give OpenSearch Service permissions to write to it. The relevant operations are documented in the Amazon CloudWatch Logs API Reference:

- `CreateLogGroup`
- `DescribeLogGroup`
- `PutResourcePolicy`

You can access these operations using the AWS SDKs.

The AWS SDKs (except the Android and iOS SDKs) support all the operations that are defined in the Amazon OpenSearch Service API Reference, including the `--log-publishing-options` option for `CreateDomain` and `UpdateDomainConfig`.

If you enabled one of the shard slow logs, see the section called "Setting shard slow log thresholds". If you enabled only error logs, you don't need to perform any additional configuration steps.

Enabling log publishing (CloudFormation)

In this example, we use CloudFormation to create a log group called `opensearch-logs`, assign the appropriate permissions, and then create a domain with log publishing enabled for application logs, search shard slow logs, and indexing slow logs.

Before you can enable log publishing, you need to create a CloudWatch log group:

```
Resources:
  OpenSearchLogGroup:
    Type: AWS::Logs::LogGroup
```

```
    Properties:
      LogGroupName: opensearch-logs
Outputs:
  Arn:
    Value:
      'Fn::GetAtt':
        - OpenSearchLogGroup
        - Arn
```

The template outputs the ARN of the log group. In this case, the ARN is `arn:aws:logs:us-east-1:123456789012:log-group:opensearch-logs`.

Using the ARN, create a resource policy that gives OpenSearch Service permissions to write to the log group:

```
Resources:
 OpenSearchLogPolicy:
   Type: AWS::Logs::ResourcePolicy
   Properties:
     PolicyName: my-policy
     PolicyDocument: "{ \"Version\": \"2012-10-17\", \"Statement\": [{ \"Sid\": \"\",
 \"Effect\": \"Allow\", \"Principal\": { \"Service\": \"es.amazonaws.com\"}, \"Action
\":[ \"logs:PutLogEvents\",\"logs:CreateLogStream\"],\"Resource\": \"arn:aws:logs:us-
east-1:123456789012:log-group:opensearch-logs:*\"}]}"
```

Finally, create the following CloudFormation stack, which generates an OpenSearch Service domain with log publishing. The access policy permits the user for the AWS account to make all HTTP requests to the domain.

```
Resources:
  OpenSearchServiceDomain:
    Type: "AWS::OpenSearchService::Domain"
    Properties:
      DomainName: my-domain
      EngineVersion: "OpenSearch_1.0"
      ClusterConfig:
        InstanceCount: 2
        InstanceType: "r6g.xlarge.search"
        DedicatedMasterEnabled: true
        DedicatedMasterCount: 3
        DedicatedMasterType: "r6g.xlarge.search"
      EBSOptions:
```

```
        EBSEnabled: true
        VolumeSize: 10
        VolumeType: "gp2"
      AccessPolicies:
        Version: "2012-10-17"
        Statement:
            Effect: "Allow"
            Principal:
                AWS: "arn:aws:iam::123456789012:user/es-user"
            Action: "es:*"
            Resource: "arn:aws:es:us-east-1:123456789012:domain/my-domain/*"
      LogPublishingOptions:
        ES_APPLICATION_LOGS:
          CloudWatchLogsLogGroupArn: "arn:aws:logs:us-east-1:123456789012:log-
group:opensearch-logs"
          Enabled: true
        SEARCH_SLOW_LOGS:
          CloudWatchLogsLogGroupArn: "arn:aws:logs:us-east-1:123456789012:log-
group:opensearch-logs"
          Enabled: true
        INDEX_SLOW_LOGS:
          CloudWatchLogsLogGroupArn: "arn:aws:logs:us-east-1:123456789012:log-
group:opensearch-logs"
          Enabled: true
```

For detailed syntax information, see the log publishing options in the *AWS CloudFormation User Guide*.

Setting search request slow log thresholds

Search request slow logs are available for search on OpenSearch Service domains running on version 2.13 and later. Search request slow log thresholds are configured for total request took time. This is different from shard request slow logs, which are configured for individual shard took time.

You can specify search request slow logs with cluster settings. This differs from shard slow logs, which you enable with index settings. For example, you can specify the following settings through the OpenSearch REST API:

```
PUT domain-endpoint/_cluster/settings
{
  "transient": {
```

```
    "cluster.search.request.slowlog.threshold.warn": "5s",
    "cluster.search.request.slowlog.threshold.info": "2s"
  }
}
```

Setting shard slow log thresholds

OpenSearch disables shard slow logs by default. After you enable the *publishing* of shard slow logs to CloudWatch, you still must specify logging thresholds for each OpenSearch index. These thresholds define precisely what should be logged and at which log level.

For example, you can specify these settings through the OpenSearch REST API:

```
PUT domain-endpoint/index/_settings
{
  "index.search.slowlog.threshold.query.warn": "5s",
  "index.search.slowlog.threshold.query.info": "2s"
}
```

Testing slow logs

To test that both search request and shard slow logs are publishing successfully, consider starting with very low values to verify that logs appear in CloudWatch, and then increase the thresholds to more useful levels.

If the logs don't appear, check the following:

- Does the CloudWatch log group exist? Check the CloudWatch console.
- Does OpenSearch Service have permissions to write to the log group? Check the OpenSearch Service console.
- Is the OpenSearch Service domain configured to publish to the log group? Check the OpenSearch Service console, use the AWS CLI `describe-domain-config` option, or call `DescribeDomainConfig` using one of the SDKs.
- Are the OpenSearch logging thresholds low enough that your requests are exceeding them?

  To review your search request slow log thresholds for a domain, use the following command:

  ```
  GET domain-endpoint/_cluster/settings?flat_settings
  ```

To review your shard slow log thresholds for an index, use the following command:

```
GET domain-endpoint/index/_settings?pretty
```

If you want to disable slow logs for an index, return any thresholds that you changed to their default values of `-1`.

Disabling publishing to CloudWatch using the OpenSearch Service console or AWS CLI does *not* stop OpenSearch from generating logs; it only stops the *publishing* of those logs. Be sure to check your index settings if you no longer need the shard slow logs, and your domain settings if you no longer need the search request slow logs.

Viewing logs

Viewing the application and slow logs in CloudWatch is just like viewing any other CloudWatch log. For more information, see View Log Data in the *Amazon CloudWatch Logs User Guide*.

Here are some considerations for viewing the logs:

- OpenSearch Service publishes only the first 255,000 characters of each line to CloudWatch. Any remaining content is truncated. For audit logs, it's 10,000 characters per message.
- In CloudWatch, the log stream names have suffixes of `-index-slow-logs`, `-search-slow-logs`, `-application-logs`, and `-audit-logs` to help identify their contents.

Monitoring audit logs in Amazon OpenSearch Service

If your Amazon OpenSearch Service domain uses fine-grained access control, you can enable audit logs for your data. Audit logs are highly customizable and let you track user activity on your OpenSearch clusters, including authentication success and failures, requests to OpenSearch, index changes, and incoming search queries. The default configuration tracks a popular set of user actions, but we recommend tailoring the settings to your exact needs.

Just like OpenSearch application logs and slow logs, OpenSearch Service publishes audit logs to CloudWatch Logs. If enabled, standard CloudWatch pricing applies.

> **Note**
>
> To enable audit logs, your user role must be mapped to the `security_manager` role, which gives you access to the OpenSearch `plugins/_security` REST API. To learn more, see the section called "Modifying the master user".

Topics

Limitations

Audit logs have the following limitations:

- Audit logs don't include cross-cluster search requests that were rejected by the destination's domain access policy.
- The maximum size of each audit log message is 10,000 characters. The audit log message is truncated if it exceeds this limit.

Enabling audit logs

Enabling audit logs is a two-step process. First, you configure your domain to publish audit logs to CloudWatch Logs. Then, you enable audit logs in OpenSearch Dashboards and configure them to meet your needs.

Important

If you encounter an error while following these steps, see the section called "Can't enable audit logs" for troubleshooting information.

Step 1: Enable audit logs and configure an access policy

These steps describe how to enable audit logs using the console. You can also enable them using the AWS CLI, or the OpenSearch Service API.

To enable audit logs for an OpenSearch Service domain (console)

1. Choose the domain to open its configuration, then go to the **Logs** tab.
2. Select **Audit logs** and then **Enable**.
3. Create a CloudWatch log group, or choose an existing one.
4. Choose an access policy that contains the appropriate permissions, or create a policy using the JSON that the console provides:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "es.amazonaws.com"
      },
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogStream"
      ],
      "Resource": "cw_log_group_arn"
    }
  ]
}
```

We recommend that you add the `aws:SourceAccount` and `aws:SourceArn` condition keys to the policy to protect yourself against the confused deputy problem. The source account is

the owner of the domain and the source ARN is the ARN of the domain. Your domain must be on service software R20211203 or later in order to add these condition keys.

For example, you could add the following condition block to the policy:

```
"Condition": {
    "StringEquals": {
        "aws:SourceAccount": "account-id"
    },
    "ArnLike": {
        "aws:SourceArn": "arn:aws:es:region:account-id:domain/domain-name"
    }
}
```

5. Choose **Enable**.

Step 2: Turn on audit logs in OpenSearch Dashboards

After you enable audit logs in the OpenSearch Service console, you *must* also enable them in OpenSearch Dashboards and configure them to match your needs.

1. Open OpenSearch Dashboards and choose **Security** from the left side menu.
2. Choose **Audit logs**.
3. Choose **Enable audit logging**.

The Dashboards UI offers full control of audit log settings under **General settings** and **Compliance settings**. For a description of all configuration options, see Audit log settings.

Enable audit logging using the AWS CLI

The following AWS CLI command enables audit logs on an existing domain:

```
aws opensearch update-domain-config --domain-name my-domain --log-publishing-options
 "AUDIT_LOGS={CloudWatchLogsLogGroupArn=arn:aws:logs:us-east-1:123456789012:log-
group:my-log-group,Enabled=true}"
```

You can also enable audit logs when you create a domain. For detailed information, see the AWS CLI Command Reference.

Enable audit logging using the configuration API

The following request to the configuration API enables audit logs on an existing domain:

```
POST https://es.us-east-1.amazonaws.com/2021-01-01/opensearch/domain/my-domain/config
{
  "LogPublishingOptions": {
    "AUDIT_LOGS": {
      "CloudWatchLogsLogGroupArn":"arn:aws:logs:us-east-1:123456789012:log-
group1:sample-domain",
      "Enabled":true
    }
  }
}
```

For more information, see the Amazon OpenSearch Service API reference.

Audit log layers and categories

Cluster communication occurs over two separate *layers*: the REST layer and the transport layer.

- The REST layer covers communication with HTTP clients such as curl, Logstash, OpenSearch Dashboards, the Java high-level REST client, the Python Requests library—all HTTP requests that arrive at the cluster.
- The transport layer covers communication between nodes. For example, after a search request arrives at the cluster (over the REST layer), the coordinating node serving the request sends the query to other nodes, receives their responses, gathers the necessary documents, and collates them into the final response. Operations such as shard allocation and rebalancing also occur over the transport layer.

You can enable or disable audit logs for entire layers, as well as individual audit categories for a layer. The following table contains a summary of audit categories and the layers for which they are available.

Category	Description	Available for REST	Available for transport
FAILED_LOGIN	A request contained invalid credentials, and authentication failed.	Yes	Yes
MISSING_PRIVILEGES	A user did not have the privileges to make the request.	Yes	Yes
GRANTED_PRIVILEGES	A user had the privileges to make the request.	Yes	Yes
OPENSEARCH_SECURITY_INDEX_ATTEMPT	A request tried to modify the `.opendistro_security` index.	No	Yes
AUTHENTICATED	A request contained valid credentials, and authentication succeeded.	Yes	Yes
INDEX_EVENT	A request performed an administrative operation on an index, such as creating one, setting an alias, or performing a force merge. The full list of `indices:admin/` actions that this category includes are available in the OpenSearch documentation.	No	Yes

In addition to these standard categories, fine-grained access control offers several additional categories designed to meet data compliance requirements.

Category	Description
COMPLIANCE_DOC_READ	A request performed a read event on a document in an index.
COMPLIANCE_DOC_WRITE	A request performed a write event on a document in an index.
COMPLIANCE_INTERNAL_CONFIG_READ	A request performed a read event on the `.opendistro_security` index.
COMPLIANCE_INTERNAL_CONFIG_WRITE	A request performed a write event on the `.opendistro_security` index.

You can have any combination of categories and message attributes. For example, if you send a REST request to index a document, you might see the following lines in the audit logs:

- AUTHENTICATED on REST layer (authentication)
- GRANTED_PRIVILEGE on transport layer (authorization)
- COMPLIANCE_DOC_WRITE (document written to an index)

Audit log settings

Audit logs have numerous configuration options.

General settings

General settings let you enable or disable individual categories or entire layers. We highly recommend leaving GRANTED_PRIVILEGES and AUTHENTICATED as excluded categories. Otherwise, these categories are logged for every valid request to the cluster.

Name	Backend setting	Description
REST layer	enable_rest	Enable or disable events that occur on the REST layer.
REST disabled categories	disabled_rest_categories	Specify audit categories to ignore on the REST layer. Modifying these categories can dramatically increase the size of the audit logs.
Transport layer	enable_transport	Enable or disable events that happen on the transport layer.
Transport disabled categories	disabled_transport_categories	Specify audit categories which must be ignored on the transport layer. Modifying these categories can dramatically increase the size of the audit logs.

Attribute settings let you customize the amount of detail in each log line.

Name	Backend setting	Description
Bulk requests	resolve_bulk_requests	Enabling this setting generates a log for each document in a bulk request, which can dramatically increase the size of the audit logs.
Request body	log_request_body	Include the request body of the requests.
Resolve indices	resolve_indices	Resolve aliases to indices.

Use ignore settings to exclude a set of users or API paths:

Name	Backend setting	Description
Ignored users	ignore_users	Specify users that you want to exclude.

Name	Backend setting	Description
Ignored requests	ignore_requests	Specify request patterns that you want to exclude.

Compliance settings

Compliance settings let you tune for index, document, or field-level access.

Name	Backend setting	Description
Compliance logging	enable_compliance	Enable or disable compliance logging.

You can specify the following settings for read and write event logging.

Name	Backend setting	Description
Internal config logging	internal_config	Enable or disable logging of events on the `.opendistro_security` index.

You can specify the following settings for read events.

Name	Backend setting	Description
Read metadata	read_metadata_only	Include only metadata for read events. Do not include any document fields.
Ignored users	read_ignore_users	Do not include certain users for read events.

Name	Backend setting	Description
Watched fields	read_watched_fields	Specify the indices and fields to watch for read events. Adding watched fields generates one log per document access, which can dramatically increase the size of the audit logs. Watched fields support index patterns and field patterns: `{ "index-name-pattern": [ "field-name-pattern" ], "logs*": [ "message" ], "twitter": [ "id", "user*" ] }`

You can specify the following settings for write events.

Name	Backend setting	Description
Write metadata	write_metadata_only	Include only metadata for write events. Do not include any document fields.
Log diffs	write_log_diffs	If write_metadata_only is false, include only the differences between write events.
Ignored users	write_ignore_users	Do not include certain users for write events.
Watch indices	write_watched_indices	Specify the indices or index patters to watch for write events. Adding watched fields generates one log per

Name	Backend setting	Description
		document access, which can dramatically increase the size of the audit logs.

Audit log example

This section includes an example configuration, search request, and the resulting audit log for all read and write events of an index.

Step 1: Configure audit logs

After you enable the publishing of audit logs to a CloudWatch Logs group, navigate to the OpenSearch Dashboards audit logging page and choose **Enable audit logging**.

1. In **General Settings**, choose **Configure** and make sure that the **REST layer** is enabled.
2. In **Compliance Settings**, choose **Configure**.
3. Under **Write**, in **Watched Fields**, add `accounts` for all write events to this index.
4. Under **Read**, in **Watched Fields**, add `ssn` and `id-` fields of the `accounts` index:

```
{
  "accounts-": [
    "ssn",
    "id-"
  ]
}
```

Step 2: Perform read and write events

1. Navigate to OpenSearch Dashboards, choose **Dev Tools**, and index a sample document:

```
PUT accounts/_doc/0
{
  "ssn": "123",
  "id-": "456"
}
```

2. To test a read event, send the following request:

```
GET accounts/_search
{
  "query": {
    "match_all": {}
  }
}
```

Step 3: Observe the logs

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Log groups**.
3. Choose the log group that you specified while enabling audit logs. Within the log group, OpenSearch Service creates a log stream for each node in your domain.
4. In **Log streams**, choose **Search all**.
5. For the read and write events, see the corresponding logs. You can expect a delay of 5 seconds before the log appears.

   Sample write audit log

```
{
  "audit_compliance_operation": "CREATE",
  "audit_cluster_name": "824471164578:audit-test",
  "audit_node_name": "be217225a0b77c2bd76147d3ed3ff83c",
  "audit_category": "COMPLIANCE_DOC_WRITE",
  "audit_request_origin": "REST",
  "audit_compliance_doc_version": 1,
  "audit_node_id": "3xNJhm4XS_yTzEgDWcGRjA",
  "@timestamp": "2020-08-23T05:28:02.285+00:00",
  "audit_format_version": 4,
  "audit_request_remote_address": "3.236.145.227",
  "audit_trace_doc_id": "lxnJGXQBqZSlDB91r_uZ",
  "audit_request_effective_user": "admin",
  "audit_trace_shard_id": 8,
  "audit_trace_indices": [
    "accounts"
  ],
  "audit_trace_resolved_indices": [
    "accounts"
```

```
  ]
}
```

Sample read audit log

```
{
  "audit_cluster_name": "824471164578:audit-docs",
  "audit_node_name": "806f6050cb45437e2401b07534a1452f",
  "audit_category": "COMPLIANCE_DOC_READ",
  "audit_request_origin": "REST",
  "audit_node_id": "saSevm9ASte0-pjAtYi2UA",
  "@timestamp": "2020-08-31T17:57:05.015+00:00",
  "audit_format_version": 4,
  "audit_request_remote_address": "54.240.197.228",
  "audit_trace_doc_id": "config:7.7.0",
  "audit_request_effective_user": "admin",
  "audit_trace_shard_id": 0,
  "audit_trace_indices": [
    "accounts"
  ],
  "audit_trace_resolved_indices": [
    "accounts"
  ]
}
```

To include the request body, return to **Compliance settings** in OpenSearch Dashboards and disable **Write metadata**. To exclude events by a specific user, add the user to **Ignored Users**.

For a description of each audit log field, see Audit log field reference. For information on searching and analyzing your audit log data, see Analyzing Log Data with CloudWatch Logs Insights in the *Amazon CloudWatch Logs User Guide*.

Configuring audit logs using the REST API

We recommend using OpenSearch Dashboards to configure audit logs, but you can also use the fine-grained access control REST API. This section contains a sample request. Full documentation on the REST API is available in the OpenSearch documentation.

```
PUT _opendistro/_security/api/audit/config
{
  "enabled": true,
```

```
  "audit": {
    "enable_rest": true,
    "disabled_rest_categories": [
      "GRANTED_PRIVILEGES",
      "AUTHENTICATED"
    ],
    "enable_transport": true,
    "disabled_transport_categories": [
      "GRANTED_PRIVILEGES",
      "AUTHENTICATED"
    ],
    "resolve_bulk_requests": true,
    "log_request_body": true,
    "resolve_indices": true,
    "exclude_sensitive_headers": true,
    "ignore_users": [
      "kibanaserver"
    ],
    "ignore_requests": [
      "SearchRequest",
      "indices:data/read/*",
      "/_cluster/health"
    ]
  },
  "compliance": {
    "enabled": true,
    "internal_config": true,
    "external_config": false,
    "read_metadata_only": true,
    "read_watched_fields": {
      "read-index-1": [
        "field-1",
        "field-2"
      ],
      "read-index-2": [
        "field-3"
      ]
    },
    "read_ignore_users": [
      "read-ignore-1"
    ],
    "write_metadata_only": true,
    "write_log_diffs": false,
    "write_watched_indices": [
```

```
      "write-index-1",
      "write-index-2",
      "log-*",
      "*"
    ],
    "write_ignore_users": [
      "write-ignore-1"
    ]
  }
}
```

Monitoring OpenSearch Service events with Amazon EventBridge

Amazon OpenSearch Service integrates with Amazon EventBridge to notify you of certain events that affect your domains. Events from AWS services are delivered to EventBridge in near real time. The same events are also sent to Amazon CloudWatch Events, the predecessor of Amazon EventBridge. You can write simple rules to indicate which events are of interest to you, and what automated actions to take when an event matches a rule. The actions that can be automatically triggered include the following:

- Invoking an AWS Lambda function
- Invoking an Amazon EC2 Run Command
- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine
- Notifying an Amazon SNS topic or an Amazon SQS queue

For more information, see Get started with Amazon EventBridge in the *Amazon EventBridge User Guide*.

Topics

- Node retirement events
- Degraded node retirement events
- Domain error events
- Tutorial: Listening for Amazon OpenSearch Service EventBridge events
- Tutorial: Sending Amazon SNS alerts for available software updates

Service software update events

OpenSearch Service sends events to EventBridge when one of the following service software update events occur.

Service software update available

OpenSearch Service sends this event when a service software update is available.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Amazon OpenSearch Service Software Update Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Service Software Update",
    "status": "Available",
    "severity": "Informational",
    "description": "Service software update R20220928 available. Service Software
 Deployment Mechanism:
                   Blue/Green. For more information on deployment configuration,
 please
                   see: https://docs.aws.amazon.com/opensearch-service/latest/
developerguide/managedomains-configuration-changes.html"
  }
}
```

Service software update scheduled

OpenSearch Service sends this event when a service software update has been scheduled. For *optional* updates, you receive the notification on the scheduled date and you have the option to reschedule at any time. For *required* updates, you receive the notification three days before the scheduled date, and you have the option to reschedule it within the mandatory window.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Amazon OpenSearch Service Software Update Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Service Software Update",
    "status": "Scheduled",
    "severity": "High",
    "description": "A new service software update [R20200330-p1] has been scheduled at
 [21st May 2023 12:40 GMT].
                    Please see documentation for more information on scheduling
 software updates:
                    https://docs.aws.amazon.com/opensearch-service/latest/
developerguide/service-software.html."
  }
}
```

Service software update rescheduled

OpenSearch Service sends this event when an optional service software update has been rescheduled. For more information, see the section called “Optional versus required updates”.

Example

The following is an example event of this type:

```
{
```

```
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Amazon OpenSearch Service Software Update Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Service Software Update",
    "status": "Rescheduled",
    "severity": "High",
    "description": "The service software update [R20200330-p1], which was originally
 scheduled for
                    [21st May 2023 12:40 GMT], has been rescheduled to [23rd May 2023
 12:40 GMT].
                    Please see documentation for more information on scheduling
 software updates:
                    https://docs.aws.amazon.com/opensearch-service/latest/
developerguide/service-software.html."
  }
}
```

Service software update started

OpenSearch Service sends this event when a service software update has started.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Amazon OpenSearch Service Software Update Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Service Software Update",
    "status": "Started",
    "severity": "Informational",
```

```
    "description": "Service software update [R20200330-p1] started.
  }
}
```

Service software update completed

OpenSearch Service sends this event when a service software update has completed.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Amazon OpenSearch Service Software Update Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Service Software Update",
    "status": "Completed",
    "severity": "Informational",
    "description": "Service software update [R20200330-p1] completed."
  }
}
```

Service software update cancelled

OpenSearch Service sends this event when a service software update has been cancelled.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Amazon OpenSearch Service Software Update Notification",
  "source": "aws.es",
  "account": "123456789012",
```

```
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Service Software Update",
    "status": "Cancelled",
    "severity": "Informational",
    "description": "The scheduled service software update [R20200330-p1] has been
 cancelled as a
                   newer update is available. Please schedule the latest update."
  }
}
```

Scheduled service software update cancelled

OpenSearch Service sends this event when a service software update that was previously scheduled for the domain has been cancelled.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Amazon OpenSearch Service Software Update Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Service Software Update",
    "status": "Cancelled",
    "severity": "Informational",
    "description": "The scheduled service software update [R20200330-p1] has been
 cancelled."
  }
}
```

Service software update unexecuted

OpenSearch Service sends this event when it can't initiate a service software update.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Amazon OpenSearch Service Software Update Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Service Software Update",
    "status": "Unexecuted",
    "severity": "Informational",
    "description": "The scheduled service software update [R20200330-p1] cannot be
 started. Reason: [reason]"
  }
}
```

Service software update failed

OpenSearch Service sends this event when a service software update fails.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Amazon OpenSearch Service Software Update Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Service Software Update",
    "status": "Failed",
```

```
    "severity": "High",
    "description": "Installation of service software update [R20200330-p1] failed.
 [reason].
  }
}
```

Service software update required

OpenSearch Service sends this event when a service software update is required. For more information, see the section called “Optional versus required updates”.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Amazon OpenSearch Service Software Update Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2016-11-01T13:12:22Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Service Software Update",
    "status": "Required",
    "severity": "High",
    "description": "Service software update [R20200330-p1] available. Update
                    will be automatically installed after [21st May 2023] if no
                    action is taken. Service Software Deployment Mechanism: Blue/Green.
                    For more information on deployment configuration, please see:
                    https://docs.aws.amazon.com/opensearch-service/latest/
developerguide/managedomains-configuration-changes.html"
  }
}
```

Auto-Tune events

OpenSearch Service sends events to EventBridge when one of the following Auto-Tune events occur.

Auto-Tune pending

OpenSearch Service sends this event when Auto-Tune has identified tuning recommendations for improved cluster performance and availability. You'll only see this event for domains with Auto-Tune disabled.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "3acb26c8-397c-4c89-a80a-ce672a864c55",
  "detail-type": "Amazon OpenSearch Service Auto-Tune Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2020-10-30T22:06:31Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Auto-Tune Event",
    "severity": "Informational",
    "status": "Pending",
    "description": "Auto-Tune recommends the following new settings for your
 domain: { JVM Heap size : 60%}. Enable Auto-Tune to improve cluster stability and
 performance.",
    "scheduleTime": "{iso8601-timestamp}"
  }
}
```

Auto-Tune started

OpenSearch Service sends this event when Auto-Tune begins to apply new settings to your domain.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "3acb26c8-397c-4c89-a80a-ce672a864c55",
  "detail-type": "Amazon OpenSearch Service Auto-Tune Notification",
```

```
  "source": "aws.es",
  "account": "123456789012",
  "time": "2020-10-30T22:06:31Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Auto-Tune Event",
    "severity": "Informational",
    "status": "Started",
    "scheduleTime": "{iso8601-timestamp}",
    "startTime": "{iso8601-timestamp}",
    "description" : "Auto-Tune is applying the following settings to your domain: { JVM
 Heap size : 60%}."
  }
}
```

Auto-Tune requires a scheduled blue/green deployment

OpenSearch Service sends this event when Auto-Tune has identified tuning recommendations that require a scheduled blue/green deployment.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "3acb26c8-397c-4c89-a80a-ce672a864c55",
  "detail-type": "Amazon OpenSearch Service Auto-Tune Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2020-10-30T22:06:31Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Auto-Tune Event",
    "severity": "Low",
    "status": "Pending",
    "startTime": "{iso8601-timestamp}",
    "description": "Auto-Tune has identified the following settings for your domain
 that require a blue/green deployment: { JVM Heap size : 60%}.
                    You can schedule the deployment for your preferred time."
  }
```

```
}
```

Auto-Tune cancelled

OpenSearch Service sends this event when Auto-Tune schedule has been cancelled because there is no pending tuning recommendations.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "3acb26c8-397c-4c89-a80a-ce672a864c55",
  "detail-type": "Amazon OpenSearch Service Auto-Tune Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2020-10-30T22:06:31Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Auto-Tune Event",
    "severity": "Low",
    "status": "Cancelled",
    "scheduleTime": "{iso8601-timestamp}",
    "description": "Auto-Tune has cancelled the upcoming blue/green deployment."
  }
}
```

Auto-Tune completed

OpenSearch Service sends this event when Auto-Tune has completed the blue/green deployment and the cluster is operational with new JVM settings in place.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "3acb26c8-397c-4c89-a80a-ce672a864c55",
  "detail-type": "Amazon OpenSearch Service Auto-Tune Notification",
```

```
  "source": "aws.es",
  "account": "123456789012",
  "time": "2020-10-30T22:06:31Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Auto-Tune Event",
    "severity": "Informational",
    "status": "Completed",
    "completionTime": "{iso8601-timestamp}",
    "description": "Auto-Tune has completed the blue/green deployment and successfully
 applied the following settings: { JVM Heap size : 60%}."
  }
}
```

Auto-Tune disabled and changes reverted

OpenSearch Service sends this event when Auto-Tune has been disabled and the applied changes were rolled back.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "3acb26c8-397c-4c89-a80a-ce672a864c55",
  "detail-type": "Amazon OpenSearch Service Auto-Tune Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2020-10-30T22:06:31Z",
  "region": "us-east-1",
  "resources": [ "arn:aws:es:us-east-1:123456789012:domain/test-domain" ],
  "detail": {
    "event": "Auto-Tune Event",
    "severity": "Informational",
    "status": "Completed",
    "description": "Auto-Tune is now disabled. All settings have been reverted. Auto-
Tune will continue to evaluate
                    cluster performance and provide recommendations.",
    "completionTime": "{iso8601-timestamp}"
  }
}
```

Auto-Tune disabled and changes retained

OpenSearch Service sends this event when Auto-Tune has been disabled and the applied changes were retained.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "3acb26c8-397c-4c89-a80a-ce672a864c55",
  "detail-type": "Amazon OpenSearch Service Auto-Tune Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2020-10-30T22:06:31Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Auto-Tune Event",
    "severity": "Informational",
    "status": "Completed",
    "description": "Auto-Tune is now disabled. The most-recent settings by Auto-Tune
 have been retained.
                    Auto-Tune will continue to evaluate cluster performance and provide
 recommendations.",
    "completionTime": "{iso8601-timestamp}"
  }
}
```

Cluster health events

OpenSearch Service sends certain events to EventBridge when your cluster's health is compromised.

Red cluster recovery started

OpenSearch Service sends this event after your cluster status has been continuously red for more than an hour. It attempts to automatically restore one or more red indexes from a snapshot in order to fix the cluster status.

Example

The following is an example event of this type:

```
{
   "version":"0",
   "id":"01234567-0123-0123-0123-012345678901",
   "detail-type":"Amazon OpenSearch Service Cluster Status Notification",
   "source":"aws.es",
   "account":"123456789012",
   "time":"2016-11-01T13:12:22Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:es:us-east-1:123456789012:domain/test-domain"
   ],
   "detail":{
      "event":"Automatic Snapshot Restore for Red Indices",
      "status":"Started",
      "severity":"High",
      "description":"Your cluster status is red. We have started automatic snapshot
 restore for the red indices.
                    No action is needed from your side. Red indices [red-index-0, red-
index-1]"
   }
}
```

Red cluster recovery partially completed

OpenSearch Service sends this event when it was only able to restore a subset of red indexes from a snapshot while attempting to fix a red cluster status.

Example

The following is an example event of this type:

```
{
   "version":"0",
   "id":"01234567-0123-0123-0123-012345678901",
   "detail-type":"Amazon OpenSearch Service Cluster Status Notification",
   "source":"aws.es",
   "account":"123456789012",
   "time":"2016-11-01T13:12:22Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:es:us-east-1:123456789012:domain/test-domain"
```

```
   ],
   "detail":{
      "event":"Automatic Snapshot Restore for Red Indices",
      "status":"Partially Restored",
      "severity":"High",
      "description":"Your cluster status is red. We were able to restore the following
 Red indices from
                   snapshot: [red-index-0]. Indices not restored: [red-index-1].
 Please refer https://docs.aws.amazon.com/opensearch-service/latest/developerguide/
handling-errors.html#handling-errors-red-cluster-status for troubleshooting steps."
   }
}
```

Red cluster recovery failed

OpenSearch Service sends this event when it fails to restore any indexes while attempting to fix a red cluster status.

Example

The following is an example event of this type:

```
{
   "version":"0",
   "id":"01234567-0123-0123-0123-012345678901",
   "detail-type":"Amazon OpenSearch Service Cluster Status Notification",
   "source":"aws.es",
   "account":"123456789012",
   "time":"2016-11-01T13:12:22Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:es:us-east-1:123456789012:domain/test-domain"
   ],
   "detail":{
      "event":"Automatic Snapshot Restore for Red Indices",
      "status":"Failed",
      "severity":"High",
      "description":"Your cluster status is red. We were unable to restore the Red
 indices automatically.
                   Indices not restored: [red-index-0, red-index-1]. Please refer
 https://docs.aws.amazon.com/opensearch-service/latest/developerguide/handling-
errors.html#handling-errors-red-cluster-status for troubleshooting steps."
   }
```

```
}
```

Shards to be deleted

OpenSearch Service sends this event when it has attempted to automatically fix your red cluster status after it was continuously red for 14 days, but one or more indexes remains red. After 7 more days (21 total days of being continuously red), OpenSearch Service proceeds to delete unassigned shards on all red indexes.

Example

The following is an example event of this type:

```
{
   "version":"0",
   "id":"01234567-0123-0123-0123-012345678901",
   "detail-type":"Amazon OpenSearch Service Cluster Status Notification",
   "source":"aws.es",
   "account":"123456789012",
   "time":"2022-04-09T10:36:48Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:es:us-east-1:123456789012:domain/test-domain"
   ],
   "detail":{
      "severity":"Medium",
      "description":"Your cluster status is red. Please fix the red indices as soon as
 possible.
                    If not fixed by 2022-04-12 01:51:47+00:00, we will delete all
 unassigned shards,
                    the unit of storage and compute, for these red indices to recover
 your domain and make it green.
                    Please refer to https://docs.aws.amazon.com/opensearch-service/
latest/developerguide/handling-errors.html#handling-errors-red-cluster-status for
 troubleshooting steps.
                    test_data, test_data1",
      "event":"Automatic Snapshot Restore for Red Indices",
      "status":"Shard(s) to be deleted"
   }
}
```

Shards deleted

OpenSearch Service sends this event after your cluster status has been continuously red for 21 days. It proceeds to delete the unassigned shards (storage and compute) on all red indexes. For details, see the section called “Automatic remediation of red clusters”.

Example

The following is an example event of this type:

```
{
   "version":"0",
   "id":"01234567-0123-0123-0123-012345678901",
   "detail-type":"Amazon OpenSearch Service Cluster Status Notification",
   "source":"aws.es",
   "account":"123456789012",
   "time":"2022-04-09T10:54:48Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:es:us-east-1:123456789012:domain/test-domain"
   ],
   "detail":{
      "severity":"High",
      "description":"We have deleted unassinged shards, the unit of storage and
 compute, in
                    red indices: index-1, index-2 because these indices were red for
 more than
                    21 days and could not be restored with the automated restore
 process.
                    Please refer to https://docs.aws.amazon.com/opensearch-service/
latest/developerguide/handling-errors.html#handling-errors-red-cluster-status for
 troubleshooting steps.",
      "event":"Automatic Snapshot Restore for Red Indices",
      "status":"Shard(s) deleted"
   }
}
```

High shard count warning

OpenSearch Service sends this event when the average shard count across your hot data nodes has exceeded 90% of the recommended default limit of 1,000. Although later versions of Elasticsearch

and OpenSearch support a configurable max shard count per node limit, we recommend you have no more than 1,000 shards per node. See Choosing the number of shards.

Example

The following is an example event of this type:

```
{
  "version":"0",
  "id":"01234567-0123-0123-0123-012345678901",
  "detail-type":"Amazon OpenSearch Service Notification",
  "source":"aws.es",
  "account":"123456789012",
  "time":"2016-11-01T13:12:22Z",
  "region":"us-east-1",
  "resources":["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail":{
     "event":"High Shard Count",
     "status":"Warning",
     "severity":"Low",
     "description":"One or more data nodes have close to 1000 shards. To ensure optimum
 performance and stability of your
                    cluster, please refer to the best practice guidelines - https://
docs.aws.amazon.com/opensearch-service/latest/developerguide/sizing-domains.html#bp-
sharding."
  }
}
```

Shard count limit exceeded

OpenSearch Service sends this event when the average shard count across your hot data nodes has exceeded the recommended default limit of 1,000. Although later versions of Elasticsearch and OpenSearch support a configurable max shard count per node limit, we recommend you have no more than 1,000 shards per node. See Choosing the number of shards.

Example

The following is an example event of this type:

```
{
  "version":"0",
```

```
  "id":"01234567-0123-0123-0123-012345678901",
  "detail-type":"Amazon OpenSearch Service Notification",
  "source":"aws.es",
  "account":"123456789012",
  "time":"2016-11-01T13:12:22Z",
  "region":"us-east-1",
  "resources":["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail":{
     "event":"High Shard Count",
     "status":"Warning",
     "severity":"Medium",
     "description":"One or more data nodes have more than 1000 shards. To ensure
 optimum performance and stability of your
                   cluster, please refer to the best practice guidelines - https://
docs.aws.amazon.com/opensearch-service/latest/developerguide/sizing-domains.html#bp-
sharding."
  }
}
```

Low disk space

OpenSearch Service sends this event when one or more nodes in your cluster has less than 25% of available storage space, or less than 25 GB.

Example

The following is an example event of this type:

```
{
  "version":"0",
  "id":"01234567-0123-0123-0123-012345678901",
  "detail-type":"Amazon OpenSearch Service Notification",
  "source":"aws.es",
  "account":"123456789012",
  "time":"2017-12-01T13:12:22Z",
  "region":"us-east-1",
  "resources":["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail":{
     "event":"Low Disk Space",
     "status":"Warning",
     "severity":"Medium",
     "description":"One or more data nodes in your cluster has less than 25% of storage
 space or less than 25GB.
```

```
                    Your cluster will be blocked for writes at 20% or 20GB. Please refer
 to the documentation for more information - https://docs.aws.amazon.com/opensearch-
service/latest/developerguide/handling-errors.html#troubleshooting-cluster-block"
  }
}
```

Low disk watermark breach

OpenSearch Service sends this event when all nodes in your cluster have less than 10% of available storage space, or less than 10 GB. When all nodes breach the low disk watermark, any new index results in a yellow cluster, and when all nodes fall below the high disk watermark, it will lead to a red cluster.

Example

The following is an example event of this type:

```
{
  "version":"0",
  "id":"01234567-0123-0123-0123-012345678901",
  "detail-type":"Amazon OpenSearch Service Notification",
  "source":"aws.es",
  "account":"123456789012",
  "time":"2017-12-01T13:12:22Z",
  "region":"us-east-1",
  "resources":["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail":{
     "event":"Low Disk Watermark Breach",
     "status":"Warning",
     "severity":"Medium",
     "description":"Low Disk Watermark threshold is about to be breached. Once the
 threshold is breached, new index creation will be blocked on all
                    nodes to prevent the cluster status from turning red. Please
 increase disk size to suit your storage needs. For more information,
                    see https://docs.aws.amazon.com/opensearch-service/latest/
developerguide/handling-errors.html#troubleshooting-cluster-block".
  }
}
```

EBS burst balance below 70%

OpenSearch Service sends this event when the EBS burst balance on one or more data nodes falls below 70%. EBS burst balance depletion can cause widespread cluster unavailability and throttling of I/O requests, which can lead to high latencies and timeouts on indexing and search requests. For steps to fix this issue, see the section called “Low EBS burst balance”.

Example

The following is an example event of this type:

```
{
  "version":"0",
  "id":"01234567-0123-0123-0123-012345678901",
  "detail-type":"Amazon OpenSearch Service Notification",
  "source":"aws.es",
  "account":"123456789012",
  "time":"2017-12-01T13:12:22Z",
  "region":"us-east-1",
  "resources":["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail":{
     "event":"EBS Burst Balance",
     "status":"Warning",
     "severity":"Medium",
     "description":"EBS burst balance on one or more data nodes is below 70%.
                    Follow https://docs.aws.amazon.com/opensearch-service/latest/
developerguide/handling-errors.html#handling-errors-low-ebs-burst
                    to fix this issue."
  }
}
```

EBS burst balance below 20%

OpenSearch Service sends this event when the EBS burst balance on one or more data nodes falls below 20%. EBS burst balance depletion can cause widespread cluster unavailability and throttling of I/O requests, which can lead to high latencies and timeouts on indexing and search requests. For steps to fix this issue, see the section called “Low EBS burst balance”.

Example

The following is an example event of this type:

```
{
  "version":"0",
  "id":"01234567-0123-0123-0123-012345678901",
  "detail-type":"Amazon OpenSearch Service Notification",
  "source":"aws.es",
  "account":"123456789012",
  "time":"2017-12-01T13:12:22Z",
  "region":"us-east-1",
  "resources":["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail":{
     "event":"EBS Burst Balance",
     "status":"Warning",
     "severity":"High",
     "description":"EBS burst balance on one or more data nodes is below 20%.
                    Follow https://docs.aws.amazon.com/opensearch-service/latest/
developerguide/handling-errors.html#handling-errors-low-ebs-burst
                    to fix this issue.
  }
}
```

Disk throughput throttle

OpenSearch Service sends this event when read and write requests to your domain are being throttled due to the throughput limitations of your EBS volumes or EC2 instance. If you receive this notification, consider scaling up your volumes or instances following AWS recommended best practices. If your volume type is gp2, increase the volume size. If your volume type is gp3, provision more throughput. You can also check that your instance base and maximum EBS throughput are greater than or equal to the provisioned volume throughput, and can scale up accordingly.

Example

The following is an example event of this type:

```
{
  "version":"0",
  "id":"01234567-0123-0123-0123-012345678901",
  "detail-type":"Amazon OpenSearch Service Notification",
  "source":"aws.es",
  "account":"123456789012",
  "time":"2017-12-01T13:12:22Z",
  "region":"us-east-1",
  "resources":["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
```

```
  "detail":{
     "event":"Disk Throughput Throttle",
     "status":"Warning",
     "severity":"Medium",
     "description":"Your domain is experiencing throttling due to instance or volume
 throughput limitations.
                   Please consider scaling your domain to suit your throughput needs.
 In July 2023, we improved
                   the accuracy of throughput throttle calculation by replacing ‘Max
 volume throughput’ with
                   ‘Provisioned volume throughput’. Please refer to the documentation
 for more information."
  }
}
```

Large shard size

OpenSearch Service sends this event when one or more shards in your cluster has exceeded either 50GiB or 65GiB. To ensure optimum cluster performance and stability, reduce shard sizes.

For more information, see the sharding best practices.

Example

The following is an example event of this type:

```
{
  "version":"0",
  "id":"01234567-0123-0123-0123-012345678901",
  "detail-type":"Amazon OpenSearch Service Notification",
  "source":"aws.es",
  "account":"123456789012",
  "time":"2017-12-01T13:12:22Z",
  "region":"us-east-1",
  "resources":["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail":{
     "event":"Large Shard Size",
     "status":"Warning",
     "severity":"Medium",
     "description":"One or more shards are larger than 65GiB. To ensure optimum cluster
 performance and stability, reduce shard sizes.
                   For more information, see https://docs.aws.amazon.com/opensearch-
service/latest/developerguide/monitoring-events.html#monitoring-events-large-shard-
size."
```

```
  }
}
```

High JVM usage

OpenSearch Service sends this event when the `JVMMemoryPressure` metric for your domain has exceeded 80%. If it exceeds 92% for 30 minutes, all write operations to your cluster will be blocked. To ensure optimum cluster stability, reduce traffic to the cluster or scale your domain to provide sufficient memory for your workload.

Example

The following is an example event of this type:

```
{
  "version":"0",
  "id":"01234567-0123-0123-0123-012345678901",
  "detail-type":"Amazon OpenSearch Service Notification",
  "source":"aws.es",
  "account":"123456789012",
  "time":"2017-12-01T13:12:22Z",
  "region":"us-east-1",
  "resources":["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail":{
     "event":"High JVM Usage",
     "status":"Warning",
     "severity":"High",
     "description":"JVM memory pressure has exceeded 80%. If it exceeds 92% for 30
 minutes, all write operations to your cluster
                    will be blocked. To ensure optimum cluster stability, reduce
 traffic to the cluster or use larger instance types.
                    For more information, see https://docs.aws.amazon.com/opensearch-
service/latest/developerguide/monitoring-events.html#monitoring-events-high-jvm."
  }
}
```

Insufficient GC

OpenSearch Service sends this event when maximum JVM is above 70% and difference between the maximum and minimum is less than 30%. This may indicate that the JVM is unable to reclaim sufficient memory during garbage collection cycles for your workload. This can lead to increasingly

slower responses and higher latencies; and in some cases even node drops due to timed out health checks. To ensure optimum cluster stability, reduce traffic to the cluster or scale your domain to provide sufficient memory for your workload.

Example

The following is an example event of this type:

```
{
  "version":"0",
  "id":"01234567-0123-0123-0123-012345678901",
  "detail-type":"Amazon OpenSearch Service Notification",
  "source":"aws.es",
  "account":"123456789012",
  "time":"2017-12-01T13:12:22Z",
  "region":"us-east-1",
  "resources":["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail":{
     "event":"Insufficient GC",
     "status":"Warning",
     "severity":"Medium",
     "description":"Maximum JVM is above 70% and JVM range is less than 30%. This may
 indicate insufficient garbage collection for your workload.
                   For more information, see https://docs.aws.amazon.com/opensearch-
service/latest/developerguide/monitoring-events.html#monitoring-events-insufficient-
gc."
  }
}
```

Custom index routing warning

OpenSearch Service sends this event when your domain is in processing state and contains indices with custom index.routing.allocation settings which can cause blue-green deployments to get stuck. Verify settings are applied properly.

Example

The following is an example event of this type:

```
{
  "version":"0",
```

```
  "id":"01234567-0123-0123-0123-012345678901",
  "detail-type":"Amazon OpenSearch Service Notification",
  "source":"aws.es",
  "account":"123456789012",
  "time":"2017-12-01T13:12:22Z",
  "region":"us-east-1",
  "resources":["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail":{
     "event":"Custom Index Routing Warning",
     "status":"Warning",
     "severity":"Medium",
     "description":"Your domain is in processing state and contains indice(s) with
 custom index.routing.allocation
                   settings which can cause blue-green deployments to get stuck.
 Verify settings are applied properly.
                   For more information, see https://docs.aws.amazon.com/opensearch-
service/latest/developerguide/monitoring-events.html#monitoring-events-index-routing."
  }
}
```

Failed shard lock

OpenSearch Service sends this event when your domain is unhealthy due to unassigned shards with `[ShardLockObtainFailedException]`. For more information, see How do I resolve the in-memory shard lock exception in Amazon OpenSearch Service?

Example

The following is an example event of this type:

```
{
  "version":"0",
  "id":"01234567-0123-0123-0123-012345678901",
  "detail-type":"Amazon OpenSearch Service Notification",
  "source":"aws.es",
  "account":"123456789012",
  "time":"2017-12-01T13:12:22Z",
  "region":"us-east-1",
  "resources":["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail":{
     "event":"Failed Shard Lock",
     "status":"Warning",
     "severity":"Medium",
```

```
     "description":"Your domain is unhealthy due to unassigned shards with
 [ShardLockObtainFailedException]. For more information,
                    see https://docs.aws.amazon.com/opensearch-service/latest/
developerguide/monitoring-events.html#monitoring-events-failed-shard-lock."
}
```

VPC endpoint events

OpenSearch Service sends certain events to EventBridge related to AWS PrivateLink interface endpoints.

VPC endpoint creation failed

OpenSearch Service sends this event when it's unable to create a requested VPC endpoint. This error might occur because you've reached the limit on the number of VPC endoints allowed within a Region. You will also see this error if a specified subnet or security group doesn't exist.

Example

The following is an example event of this type:

```
{
   "version":"0",
   "id":"01234567-0123-0123-0123-012345678901",
   "detail-type":"Amazon OpenSearch Service VPC Endpoint Notification",
   "source":"aws.es",
   "account":"123456789012",
   "time":"2016-11-01T13:12:22Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:es:us-east-1:123456789012:domain/test-domain"
   ],
   "detail":{
      "event":"VPC Endpoint Create Validation",
      "status":"Failed",
      "severity":"High",
      "description":"Unable to create VPC endpoint aos-0d4c74c0342343 for domain
                    arn:aws:es:eu-south-1:123456789012:domain/my-domain due to the
 following validation failures: You've reached the limit on the
                    number of VPC endpoints that you can create in the AWS Region."
   }
}
```

VPC endpoint update failed

OpenSearch Service sends this event when it's unable to delete a requested VPC endpoint.

Example

The following is an example event of this type:

```
{
   "version":"0",
   "id":"01234567-0123-0123-0123-012345678901",
   "detail-type":"Amazon OpenSearch Service VPC Endpoint Notification",
   "source":"aws.es",
   "account":"123456789012",
   "time":"2016-11-01T13:12:22Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:es:us-east-1:123456789012:domain/test-domain"
   ],
   "detail":{
      "event":"VPC Endpoint Update Validation",
      "status":"Failed",
      "severity":"High",
      "description":"Unable to update VPC endpoint aos-0d4c74c0342343 for domain
                     arn:aws:es:eu-south-1:123456789012:domain/my-domain due to the
 following validation failures: <failure message>."
   }
}
```

VPC endpoint deletion failed

OpenSearch Service sends this event when it's unable to delete a requested VPC endpoint.

Example

The following is an example event of this type:

```
{
   "version":"0",
   "id":"01234567-0123-0123-0123-012345678901",
   "detail-type":"Amazon OpenSearch Service VPC Endpoint Notification",
   "source":"aws.es",
```

```
   "account":"123456789012",
   "time":"2016-11-01T13:12:22Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:es:us-east-1:123456789012:domain/test-domain"
   ],
   "detail":{
      "event":"VPC Endpoint Delete Validation",
      "status":"Failed",
      "severity":"High",
      "description":"Unable to delete VPC endpoint aos-0d4c74c0342343 for domain
                    arn:aws:es:eu-south-1:123456789012:domain/my-domain due to the
 following validation failures: Specified subnet doesn't exist."
   }
}
```

Node retirement events

OpenSearch Service sends events to EventBridge when one of the following node retirement events occur.

Node retirement scheduled

OpenSearch Service sends this event when a node retirement has been scheduled.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Amazon OpenSearch Service Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2023-04-07T10:07:33Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Node Retirement Notification",
    "status": "Scheduled",
    "severity": "Medium",
```

```
    "description": "An automated action to retire and replace a node has been scheduled
 on your domain.
                    The node will be replaced in the next off-peak window. For more
 information, see
                    https://docs.aws.amazon.com/opensearch-service/latest/
developerguide/monitoring-events.html."
  }
}
```

Node retirement completed

OpenSearch Service sends this event when a node retirement has completed.

Example

The following is an example event of this type:

```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Amazon OpenSearch Service Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2023-04-07T10:07:33Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Node Retirement Notification",
    "status": "Completed",
    "severity": "Medium",
    "description": "The node has been retired and replaced with a new node."
  }
}
```

Node retirement failed

OpenSearch Service sends this event when a node retirement fails.

Example

The following is an example event of this type:

```
{
```

```
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "Amazon OpenSearch Service Notification",
  "source": "aws.es",
  "account": "123456789012",
  "time": "2023-04-07T10:07:33Z",
  "region": "us-east-1",
  "resources": ["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail": {
    "event": "Node Retirement Notification",
    "status": "Failed",
    "severity": "Medium",
    "description": "Node retirement failed. No actions are required from your end. We
 will automatically
                   retry replacing the node."
  }
}
```

Degraded node retirement events

OpenSearch Service sends these events when a node replacement is required due to degraded hardware on a node.

Degraded node retirement notification

OpenSearch Service sends this event when the automated action to retire and replace a degraded node has been scheduled for your domain.

Example

The following is an example event of this type:

```
{
   "version":"0",
   "id":"db233454-aad1-7676-3b15-10a84b052baa",
   "detail-type":"Amazon OpenSearch Service Notification",
   "source":"aws.es",
   "account":"123456789012",
   "time":"2024-01-11T08:16:06Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:es:us-east-1:123456789012:domain/test-node-replacement"
```

```
   ],
   "detail":{
      "severity":"Medium",
      "description":"An automated action to retire and replace a node has
 been scheduled on your domain. For more information, please see https://
docs.aws.amazon.com/opensearch-service/latest/developerguide/monitoring-events.html.",
      "event":"Degraded Node Retirement Notification",
      "status":"Scheduled"
   }
}
```

Degraded node retirement complete

OpenSearch Service sends this event when a degraded node has been retired and replaced with a new node.

Example

The following is an example event of this type:

```
{
   "version":"0",
   "id":"7444215c-90f9-a52d-bcda-e85973a9a762",
   "detail-type":"Amazon OpenSearch Service Notification",
   "source":"aws.es",
   "account":"123456789012",
   "time":"2024-01-11T10:20:30Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:es:us-east-1:123456789012:domain/test-node-replacement"
   ],
   "detail":{
      "severity":"Medium",
      "description":"The node has been retired and replaced with a new node.",
      "event":"Degraded Node Retirement Notification",
      "status":"Completed"
   }
}
```

Degraded node retirement failed

OpenSearch Service sends this event if the degraded node retirement failed.

Example

The following is an example event of this type:

```
{
   "version":"0",
   "id":"c328e9bb-93b9-c0b2-b17a-df527fdf96b6",
   "detail-type":"Amazon OpenSearch Service Notification",
   "source":"aws.es",
   "account":"123456789012",
   "time":"2024-01-11T08:31:38Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:es:us-east-1:123456789012:domain/test-node-replacement"
   ],
   "detail":{
      "severity":"Medium",
      "description":"Node retirement failed. No actions are required from your end. We
 will automatically re-try replacing the node.",
      "event":"Degraded Node Retirement Notification",
      "status":"Failed"
   }
}
```

Domain error events

OpenSearch Service sends events to EventBridge when one of the following domain errors occur.

Domain update validation failure

OpenSearch Service sends this event if it encounters one or more validation failures when attempting to update or perform a configuration change on a domain. For steps to resolve these failures, see the section called “Troubleshooting validation errors”.

Example

The following is an example event of this type:

```
{
   "version":"0",
   "id":"01234567-0123-0123-0123-012345678901",
   "detail-type":"Amazon OpenSearch Service Domain Update Notification",
   "source":"aws.es",
```

```
   "account":"123456789012",
   "time":"2016-11-01T13:12:22Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:es:us-east-1:123456789012:domain/test-domain"
   ],
   "detail":{
      "event":"Domain Update Validation",
      "status":"Failed",
      "severity":"High",
      "description":"Unable to perform updates to your domain due to the following
 validation failures: <failures>
                   Please see the documentation for more information https://
docs.aws.amazon.com/opensearch-service/latest/developerguide/managedomains-
configuration-changes.html#validation"
   }
}
```

KMS key inaccessible

OpenSearch Service sends this event when it can't access your AWS KMS key.

Example

The following is an example event of this type:

```
{
  "version":"0",
  "id":"01234567-0123-0123-0123-012345678901",
  "detail-type":"Domain Error Notification",
  "source":"aws.es",
  "account":"123456789012",
  "time":"2016-11-01T13:12:22Z",
  "region":"us-east-1",
  "resources":["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail":{
     "event":"KMS Key Inaccessible",
     "status":"Error",
     "severity":"High",
     "description":"The KMS key associated with this domain is inaccessible. You are at
 risk of losing access to your domain.
                   For more information, please refer to https://docs.aws.amazon.com/
opensearch-service/latest/developerguide/encryption-at-rest.html#disabled-key."
```

```
  }
}
```

Domain isolation

OpenSearch Service sends this event when your domain becomes isolated and can't received, read, or write requests because it is unreachable by the network.

Example

The following is an example event of this type:

```
{
  "version":"0",
  "id":"01234567-0123-0123-0123-012345678901",
  "detail-type":"Amazon OpenSearch Service Notification",
  "source":"aws.es",
  "account":"123456789012",
  "time":"2023-11-01T13:12:22Z",
  "region":"us-east-1",
  "resources":["arn:aws:es:us-east-1:123456789012:domain/test-domain"],
  "detail":{
     "event":"Domain Isolation Notification",
     "status":"Error",
     "severity":"High",
     "description":"Your OpenSearch Service domain has been isolated. An isolated
 domain is unreachable by network and cannot receive, read, or write requests. For more
 information and assistance, please contact AWS Support at https://docs.aws.amazon.com/
opensearch-service/latest/developerguide/encryption-at-rest.html#disabled-key."
  }
}
```

Tutorial: Listening for Amazon OpenSearch Service EventBridge events

In this tutorial, you set up a simple AWS Lambda function that listens for Amazon OpenSearch Service events and writes them to a CloudWatch Logs log stream.

Prerequisites

This tutorial assumes that you have an existing OpenSearch Service domain. If you haven't created a domain, follow the steps in *Creating and managing domains* to create one.

Step 1: Create the Lambda function

In this procedure, you create a simple Lambda function to serve as a target for OpenSearch Service event messages.

To create a target Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose **Create function** and **Author from scratch**.
3. For **Function name**, enter **event-handler**.
4. For **Runtime**, choose **Python 3.8**.
5. Choose **Create function**.
6. In the **Function code** section, edit the sample code to match the following example:

```
import json

def lambda_handler(event, context):
    if event["source"] != "aws.es":
        raise ValueError("Function only supports input from events with a source
 type of: aws.es")

    print(json.dumps(event))
```

   This is a simple Python 3.8 function that prints the events sent by OpenSearch Service. If everything is configured correctly, at the end of this tutorial, the event details appear in the CloudWatch Logs log stream that's associated with this Lambda function.
7. Choose **Deploy**.

Step 2: Register an event rule

In this step, you create an EventBridge rule that captures events from your OpenSearch Service domains. This rule captures all events within the account where it's defined. The event messages themselves contain information about the event source, including the domain from which it originated. You can use this information to filter and sort events programmatically.

To create an EventBridge rule

1. Open the EventBridge console at https://console.aws.amazon.com/events/.

2. Choose **Create rule**.
3. Name the rule **event-rule**.
4. Choose **Next**.
5. For the event pattern, select **AWS services**, **Amazon OpenSearch Service**, and **All Events**. This pattern applies across all of your OpenSearch Service domains and to every OpenSearch Service event. Alternatively, you can create a more specific pattern to filter out some results.
6. Press **Next**.
7. For the target, choose **Lambda function**. In the function dropdown, choose **event-handler**.
8. Press **Next**.
9. Skip the tags and press **Next** again.
10. Review the configuration and choose **Create rule**.

Step 3: Test your configuration

The next time you receive a notification in the **Notifications** section of the OpenSearch Service console, if everything is configured properly, your Lambda function is triggered and it writes the event data to a CloudWatch Logs log stream for the function.

To test your configuration

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. On the navigation pane, choose **Logs** and select the log group for your Lambda function (for example, **/aws/lambda/event-handler**).
3. Select a log stream to view the event data.

Tutorial: Sending Amazon SNS alerts for available software updates

In this tutorial, you configure an Amazon EventBridge event rule that captures notifications for available service software updates in Amazon OpenSearch Service and sends you an email notification through Amazon Simple Notification Service (Amazon SNS).

Prerequisites

This tutorial assumes that you have an existing OpenSearch Service domain. If you haven't created a domain, follow the steps in *Creating and managing domains* to create one.

Step 1: Create and subscribe to an Amazon SNS topic

Configure an Amazon SNS topic to serve as an event target for your new event rule.

To create an Amazon SNS target

1. Open the Amazon SNS console at https://console.aws.amazon.com/sns/v3/home.
2. Choose **Topics** and **Create topic**.
3. For the job type, choose **Standard**, and name the job **software-update**.
4. Choose **Create topic**.
5. After the topic is created, choose **Create subscription**.
6. For **Protocol**, choose **Email**. For **Endpoint**, enter an email address that you currently have access to and choose **Create subscription**.
7. Check your email account and wait to receive a subscription confirmation email message. When you receive it, choose **Confirm subscription**.

Step 2: Register an event rule

Next, register an event rule that captures only service software update events.

To create an event rule

1. Open the EventBridge console at https://console.aws.amazon.com/events/.
2. Choose **Create rule**.
3. Name the rule **softwareupdate-rule**.
4. Choose **Next**.
5. For the event pattern, select **AWS services**, **Amazon OpenSearch Service**, and **Amazon OpenSearch Service Software Update Notification**. This pattern matches any service software update event from OpenSearch Service. For more information about event patterns, see Amazon EventBridge event patterns in the *Amazon EventBridge User Guide*.
6. Optionally, you can filter to only specific severities. For the severities of each event, see the section called “Service software update events”.
7. Choose **Next**.
8. For the target, choose **SNS topic** and select **software-update**.

9. Choose **Next**.
10. Skip the tags and choose **Next**.
11. Review the rule configuration and choose **Create rule**.

The next time you receive a notification from OpenSearch Service about an available service software update, if everything is configured properly, Amazon SNS should send you an email alert about the update.

Monitoring Amazon OpenSearch Service API calls with AWS CloudTrail

Amazon OpenSearch Service integrates with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in OpenSearch Service. CloudTrail captures all configuration API calls for OpenSearch Service as events.

> **Note**
>
> CloudTrail only captures calls to the Configuration API, such as `CreateDomain` and `GetUpgradeStatus`. CloudTrail doesn't capture calls to the OpenSearch APIs, such as `_search` and `_bulk`. For these calls, see the section called "Monitoring audit logs".

The captured calls include calls from the OpenSearch Service console, AWS CLI, or an AWS SDK. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for OpenSearch Service. If you don't configure a trail, you can still view the most recent events on the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to OpenSearch Service, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Amazon OpenSearch Service information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in OpenSearch Service, that activity is recorded in a CloudTrail event along with other AWS service

events in **Event history**. You can view, search, and download recent events in your AWS account account. For more information, see Viewing events with CloudTrail event history.

For an ongoing record of events in your AWS account account, including events for OpenSearch Service, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Creating a trail for your AWS account
- AWS service integrations with CloudTrail Logs
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple Regions and Receiving CloudTrail log files from multiple accounts

All OpenSearch Service configuration API actions are logged by CloudTrail and are documented in the Amazon OpenSearch Service API Reference.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials
- Whether the request was made with temporary security credentials for a role or federated user
- Whether the request was made by another AWS service

For more information, see the CloudTrail userIdentity Element.

Understanding Amazon OpenSearch Service log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateDomain` operation:

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/test-user",
    "accountId": "123456789012",
    "accessKeyId": "access-key",
    "userName": "test-user",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2018-08-21T21:59:11Z"
      }
    },
    "invokedBy": "signin.amazonaws.com"
  },
  "eventTime": "2018-08-21T22:00:05Z",
  "eventSource": "es.amazonaws.com",
  "eventName": "CreateDomain",
  "awsRegion": "us-west-1",
  "sourceIPAddress": "123.123.123.123",
  "userAgent": "signin.amazonaws.com",
  "requestParameters": {
    "engineVersion": "OpenSearch_1.0",
    "clusterConfig": {
      "instanceType": "m4.large.search",
      "instanceCount": 1
    },
    "snapshotOptions": {
      "automatedSnapshotStartHour": 0
    },
    "domainName": "test-domain",
    "encryptionAtRestOptions": {},
    "eBSOptions": {
      "eBSEnabled": true,
      "volumeSize": 10,
      "volumeType": "gp2"
    },
```

```
    "accessPolicies": "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect\":\"Allow
\",\"Principal\":{\"AWS\":[\"123456789012\"]},\"Action\":[\"es:*\"],\"Resource\":
\"arn:aws:es:us-west-1:123456789012:domain/test-domain/*\"}]}",
    "advancedOptions": {
      "rest.action.multi.allow_explicit_index": "true"
    }
  },
  "responseElements": {
    "domainStatus": {
      "created": true,
      "clusterConfig": {
        "zoneAwarenessEnabled": false,
        "instanceType": "m4.large.search",
        "dedicatedMasterEnabled": false,
        "instanceCount": 1
      },
      "cognitoOptions": {
        "enabled": false
      },
      "encryptionAtRestOptions": {
        "enabled": false
      },
      "advancedOptions": {
        "rest.action.multi.allow_explicit_index": "true"
      },
      "upgradeProcessing": false,
      "snapshotOptions": {
        "automatedSnapshotStartHour": 0
      },
      "eBSOptions": {
        "eBSEnabled": true,
        "volumeSize": 10,
        "volumeType": "gp2"
      },
      "engineVersion": "OpenSearch_1.0",
      "processing": true,
      "aRN": "arn:aws:es:us-west-1:123456789012:domain/test-domain",
      "domainId": "123456789012/test-domain",
      "deleted": false,
      "domainName": "test-domain",
      "accessPolicies": "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect\":\"Allow
\",\"Principal\":{\"AWS\":\"arn:aws:iam::123456789012:root\"},\"Action\":\"es:*\",
\"Resource\":\"arn:aws:es:us-west-1:123456789012:domain/test-domain/*\"}]}"
    }
```

```
  },
  "requestID": "12345678-1234-1234-1234-987654321098",
  "eventID": "87654321-4321-4321-4321-987654321098",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

Security in Amazon OpenSearch Service

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon OpenSearch Service, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using OpenSearch Service. The following topics show you how to configure OpenSearch Service to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your OpenSearch Service resources.

Topics

- Using service-linked roles for Amazon OpenSearch Service

Data protection in Amazon OpenSearch Service

The AWS shared responsibility model applies to data protection in Amazon OpenSearch Service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with OpenSearch Service or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption of data at rest for Amazon OpenSearch Service

OpenSearch Service domains offer encryption of data at rest, a security feature that helps prevent unauthorized access to your data. The feature 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 enabled, the feature encrypts the following aspects of a domain:

- All indexes (including those in UltraWarm storage)
- OpenSearch logs
- Swap files
- All other data in the application directory
- Automated snapshots

The following are *not* encrypted when you enable encryption of data at rest, but you can take additional steps to protect them:

- Manual snapshots: You currently can't use AWS KMS keys to encrypt manual snapshots. You can, however, use server-side encryption with S3-managed keys or KMS keys to encrypt the bucket you use as a snapshot repository. For instructions, see the section called “Registering a manual snapshot repository”.
- Slow logs and error logs: If you publish logs and want to encrypt them, you can encrypt their CloudWatch Logs log group using the same AWS KMS key as the OpenSearch Service domain. For more information, see Encrypt log data in CloudWatch Logs using AWS KMS in the *Amazon CloudWatch Logs User Guide*.

> **Note**
>
> You can't enable encryption at rest on an existing domain if UltraWarm or cold storage is enabled on the domain. You must first disable UltraWarm or cold storage, enable encryption at rest, and then re-enable UltraWarm or cold storage. If you want to retain indexes in UltraWarm or cold storage, you must move them to hot storage before disabling UltraWarm or cold storage.

OpenSearch Service supports only symmetric encryption KMS keys, not asymmetric ones. To learn how to create symmetric keys, see Creating keys in the *AWS Key Management Service Developer Guide*.

Regardless of whether encryption at rest is enabled, all domains automatically encrypt custom packages using AES-256 and OpenSearch Service-managed keys.

Permissions

To use the OpenSearch Service console to configure encryption of data at rest, you must have read permissions to AWS KMS, such as the following identity-based policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:List*",
        "kms:Describe*"
      ],
      "Resource": "*"
    }
  ]
}
```

If you want to use a key other than the AWS owned key, you must also have permissions to create grants for the key. These permissions typically take the form of a resource-based policy that you specify when you create the key.

If you want to keep your key exclusive to OpenSearch Service, you can add the kms:ViaService condition to that key policy:

```
"Condition": {
  "StringEquals": {
    "kms:ViaService": "es.us-west-1.amazonaws.com"
  },
  "Bool": {
    "kms:GrantIsForAWSResource": "true"
  }
}
```

For more information, see Using key policies in AWS KMS in the *AWS Key Management Service Developer Guide*.

Enabling encryption of data at rest

Encryption of data at rest on new domains requires either OpenSearch or Elasticsearch 5.1 or later. Enabling it on existing domains requires either OpenSearch or Elasticsearch 6.7 or later.

To enable encryption of data at rest (console)

1. Open the domain in the AWS console, then choose **Actions** and **Edit security configuration**.
2. Under **Encryption**, select **Enable encryption of data at rest**.
3. Choose an AWS KMS key to use, then choose **Save changes**.

You can also enable encryption through the configuration API. The following request enables encryption of data at rest on an existing domain:

```
{
   "ClusterConfig":{
      "EncryptionAtRestOptions":{
         "Enabled": true,
         "KmsKeyId":"arn:aws:kms:us-east-1:123456789012:alias/my-key"
      }
   }
}
```

Disabled or deleted KMS key

If you disable or delete the key that you used to encrypt a domain, the domain becomes inaccessible. OpenSearch Service sends you a notification informing you that it can't access the KMS key. Re-enable the key immediately to access your domain.

The OpenSearch Service team can't help you recover your data if your key is deleted. AWS KMS deletes keys only after a waiting period of at least seven days. If your key is pending deletion, either cancel deletion or take a manual snapshot of the domain to prevent loss of data.

Disabling encryption of data at rest

After you configure a domain to encrypt data at rest, you can't disable the setting. Instead, you can take a manual snapshot of the existing domain, create another domain, migrate your data, and delete the old domain.

Monitoring domains that encrypt data at rest

Domains that encrypt data at rest have two additional metrics: `KMSKeyError` and `KMSKeyInaccessible`. These metrics appear only if the domain encounters a problem with your encryption key. For full descriptions of these metrics, see the section called "Cluster metrics". You can view them using either the OpenSearch Service console or the Amazon CloudWatch console.

> **Tip**
>
> Each metric represents a significant problem for a domain, so we recommend that you create CloudWatch alarms for both. For more information, see the section called "Recommended CloudWatch alarms".

Other considerations

- Automatic key rotation preserves the properties of your AWS KMS keys, so the rotation has no effect on your ability to access your OpenSearch data. Encrypted OpenSearch Service domains don't support manual key rotation, which involves creating a new key and updating any references to the old key. To learn more, see Rotating keys in the *AWS Key Management Service Developer Guide*.
- Certain instance types don't support encryption of data at rest. For details, see the section called "Supported instance types".
- Domains that encrypt data at rest use a different repository name for their automated snapshots. For more information, see the section called "Restoring snapshots".
- While we highly recommend enabling encryption at rest, it can add additional CPU overhead and a few milliseconds of latency. Most use cases aren't sensitive to these differences, however, and the magnitude of impact depends on the configuration of your cluster, clients, and usage profile.

Node-to-node encryption for Amazon OpenSearch Service

Node-to-node encryption provides an additional layer of security on top of the default features of Amazon OpenSearch Service.

Each OpenSearch Service domain—regardless of whether the domain uses VPC access—resides within its own, dedicated VPC. This architecture prevents potential attackers from intercepting traffic between OpenSearch nodes and keeps the cluster secure. By default, however, traffic within the VPC is unencrypted. Node-to-node encryption enables TLS 1.2 encryption for all communications within the VPC.

If you send data to OpenSearch Service over HTTPS, node-to-node encryption helps ensure that your data remains encrypted as OpenSearch distributes (and redistributes) it throughout the cluster. If data arrives unencrypted over HTTP, OpenSearch Service encrypts it after it reaches the cluster. You can require that all traffic to the domain arrive over HTTPS using the console, AWS CLI, or configuration API.

Node-to-node encryption is *required* if you enable fine-grained access control.

Enabling node-to-node encryption

Node-to-node encryption on new domains requires any version of OpenSearch, or Elasticsearch 6.0 or later. Enabling node-to-node encryption on existing domains requires any version of OpenSearch, or Elasticsearch 6.7 or later. Choose the existing domain in the AWS console, **Actions**, and **Edit security configuration**.

Alternatively, you can use the AWS CLI or configuration API. For more information, see the AWS CLI Command Reference and OpenSearch Service API reference.

Disabling node-to-node encryption

After you configure a domain to use node-to-node encryption, you can't disable the setting. Instead, you can take a manual snapshot of the encrypted domain, create another domain, migrate your data, and delete the old domain.

Identity and Access Management in Amazon OpenSearch Service

Amazon OpenSearch Service offers several ways to control access to your domains. This topic covers the various policy types, how they interact with each other, and how to create your own custom policies.

> **Important**
>
> VPC support introduces some additional considerations to OpenSearch Service access control. For more information, see the section called "About access policies on VPC domains".

Types of policies

OpenSearch Service supports three types of access policies:

- the section called "Resource-based policies"
- the section called "Identity-based policies"
- the section called "IP-based policies"

Resource-based policies

You add a resource-based policy, often called the domain access policy, when you create a domain. These policies specify which actions a principal can perform on the domain's *subresources* (with the exception of cross-cluster search). Subresources include OpenSearch indexes and APIs. The Principal element specifies the accounts, users, or roles that are allowed access. The Resource element specifies which subresources these principals can access.

For example, the following resource-based policy grants `test-user` full access (`es:*`) to the subresources on `test-domain`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Principal": {
        "AWS": [
          "arn:aws:iam::123456789012:user/test-user"
        ]
      },
      "Action": [
        "es:*"
      ],
      "Resource": "arn:aws:es:us-west-1:987654321098:domain/test-domain/*"
    }
  ]
}
```

Two important considerations apply to this policy:

- These privileges apply only to this domain. Unless you create similar policies on other domains, `test-user` can only access `test-domain`.
- The trailing `/*` in the `Resource` element is significant and indicates that resource-based policies only apply to the domain's subresources, not the domain itself. In resource-based policies, the `es:*` action is equivalent to `es:ESHttp*`.

  For example, `test-user` can make requests against an index (`GET https://search-test-domain.us-west-1.es.amazonaws.com/test-index`), but can't update the domain's configuration (`POST https://es.us-west-1.amazonaws.com/2021-01-01/opensearch/domain/test-domain/config`). Note the difference between the two endpoints. Accessing the configuration API requires an identity-based policy.

You can specify a partial index name by adding a wildcard. This example identifies any indexes beginning with `commerce`:

```
arn:aws:es:us-west-1:987654321098:domain/test-domain/commerce*
```

In this case, the wildcard means that `test-user` can make requests to indexes within `test-domain` that have names that begin with `commerce`.

To further restrict `test-user`, you can apply the following policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::123456789012:user/test-user"
        ]
      },
      "Action": [
        "es:ESHttpGet"
      ],
      "Resource": "arn:aws:es:us-west-1:987654321098:domain/test-domain/commerce-data/
_search"
    }
  ]
}
```

Now `test-user` can perform only one operation: searches against the `commerce-data` index. All other indexes within the domain are inaccessible, and without permissions to use the `es:ESHttpPut` or `es:ESHttpPost` actions, `test-user` can't add or modify documents.

Next, you might decide to configure a role for power users. This policy gives `power-user-role` access to the HTTP GET and PUT methods for all URIs in the index:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::123456789012:role/power-user-role"
        ]
      },
      "Action": [
        "es:ESHttpGet",
        "es:ESHttpPut"
      ],
      "Resource": "arn:aws:es:us-west-1:987654321098:domain/test-domain/commerce-data/
*"
    }
  ]
}
```

If your domain is in a VPC or uses fine-grained access control, you can use an open domain access policy. Otherwise, your domain access policy must contain some restriction, either by principal or IP address.

For information about all available actions, see the section called "Policy element reference". For far more granular control over your data, use an open domain access policy with fine-grained access control.

Identity-based policies

Unlike resource-based policies, which are a part of each OpenSearch Service domain, you attach identity-based policies to users or roles using the AWS Identity and Access Management (IAM) service. Just like resource-based policies, identity-based policies specify who can access a service, which actions they can perform, and if applicable, the resources on which they can perform those actions.

While they certainly don't have to be, identity-based policies tend to be more generic. They often govern only the configuration API actions a user can perform. After you have these policies in place, you can use resource-based policies (or fine-grained access control) in OpenSearch Service to offer users access to OpenSearch indexes and APIs.

> **Note**
>
> Users with the AWS managed `AmazonOpenSearchServiceReadOnlyAccess` policy can't see cluster health status on the console. To allow them to see cluster health status (and other OpenSearch data), add the `es:ESHttpGet` action to an access policy and attach it to their accounts or roles.

Because identity-based policies attach to users or roles (principals), the JSON doesn't specify a principal. The following policy grants access to actions that begin with `Describe` and `List`. This combination of actions provides read-only access to domain configurations, but not to the data stored in the domain itself:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "es:Describe*",
```

```
        "es:List*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

An administrator might have full access to OpenSearch Service and all data stored on all domains:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "es:*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Identity-based policies let you use tags to control access to the configuration API. The following policy, for example, lets attached principals view and update a domain's configuration if the domain has the `team:devops` tag:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Action": [
      "es:UpdateDomainConfig",
      "es:DescribeDomain",
      "es:DescribeDomainConfig"
    ],
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
      "ForAnyValue:StringEquals": {
        "aws:ResourceTag/team": [
          "devops"
        ]
```

```
      }
    }
  }]
}
```

You can also use tags to control access to the OpenSearch API. Tag-based policies for the OpenSearch API only apply to HTTP methods. For example, the following policy lets attached principals send GET and PUT requests to the OpenSearch API if the domain has the `environment:production` tag:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Action": [
      "es:ESHttpGet",
      "es:ESHttpPut"
    ],
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
      "ForAnyValue:StringEquals": {
        "aws:ResourceTag/environment": [
          "production"
        ]
      }
    }
  }]
}
```

For more granular control of the OpenSearch API, consider using fine-grained access control.

> **Note**
>
> After you add one or more OpenSearch APIs to any tag-based policy, you must perform a single tag operation (such as adding, removing, or modifying a tag) in order for the changes to take effect on a domain. You must be on service software R20211203 or later to include OpenSearch API operations in tag-based policies.

OpenSearch Service supports the `RequestTag` and `TagKeys` global condition keys for the configuration API, not the OpenSearch API. These conditions only apply to API calls that include

tags within the request, such as `CreateDomain`, `AddTags`, and `RemoveTags`. The following policy lets attached principals create domains, but only if they include the `team:it` tag in the request:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "es:CreateDomain",
      "es:AddTags"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "aws:RequestTag/team": [
          "it"
        ]
      }
    }
  }
}
```

For more details on using tags for access control and the differences between resource-based and identity-based policies, see the IAM User Guide.

IP-based policies

IP-based policies restrict access to a domain to one or more IP addresses or CIDR blocks. Technically, IP-based policies are not a distinct type of policy. Instead, they are just resource-based policies that specify an anonymous principal and include a special Condition element.

The primary appeal of IP-based policies is that they allow unsigned requests to an OpenSearch Service domain, which lets you use clients like curl and OpenSearch Dashboards or access the domain through a proxy server. To learn more, see the section called "Using a proxy to access OpenSearch Service from Dashboards".

> **Note**
>
> If you enabled VPC access for your domain, you can't configure an IP-based policy. Instead, you can use security groups to control which IP addresses can access the domain. For more information, see the section called "About access policies on VPC domains".

The following policy grants all HTTP requests that originate from the specified IP range access to `test-domain`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "es:ESHttp*"
      ],
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "192.0.2.0/24"
          ]
        }
      },
      "Resource": "arn:aws:es:us-west-1:987654321098:domain/test-domain/*"
    }
  ]
}
```

If your domain has a public endpoint and doesn't use fine-grained access control, we recommend combining IAM principals and IP addresses. This policy grants `test-user` HTTP access only if the request originates from the specified IP range:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "AWS": [
        "arn:aws:iam::987654321098:user/test-user"
      ]
    },
    "Action": [
      "es:ESHttp*"
    ],
```

```
    "Condition": {
      "IpAddress": {
        "aws:SourceIp": [
          "192.0.2.0/24"
        ]
      }
    },
    "Resource": "arn:aws:es:us-west-1:987654321098:domain/test-domain/*"
  }]
}
```

Making and signing OpenSearch Service requests

Even if you configure a completely open resource-based access policy, *all* requests to the OpenSearch Service configuration API must be signed. If your policies specify IAM roles or users, requests to the OpenSearch APIs also must be signed using AWS Signature Version 4. The signing method differs by API:

- To make calls to the OpenSearch Service configuration API, we recommend that you use one of the AWS SDKs. The SDKs greatly simplify the process and can save you a significant amount of time compared to creating and signing your own requests. The configuration API endpoints use the following format:

  ```
  es.region.amazonaws.com/2021-01-01/
  ```

  For example, the following request makes a configuration change to the `movies` domain, but you have to sign it yourself (not recommended):

  ```
  POST https://es.us-east-1.amazonaws.com/2021-01-01/opensearch/domain/movies/config
  {
    "ClusterConfig": {
      "InstanceType": "c5.xlarge.search"
    }
  }
  ```

  If you use one of the SDKs, such as Boto 3, the SDK automatically handles the request signing:

  ```
  import boto3

  client = boto3.client(es)
  ```

```
response = client.update_domain_config(
  DomainName='movies',
  ClusterConfig={
    'InstanceType': 'c5.xlarge.search'
  }
)
```

For a Java code sample, see the section called “Using the AWS SDKs”.

- To make calls to the OpenSearch APIs, you must sign your own requests. The OpenSearch APIs use the following format:

```
domain-id.region.es.amazonaws.com
```

For example, the following request searches the `movies` index for *thor*:

```
GET https://my-domain.us-east-1.es.amazonaws.com/movies/_search?q=thor
```

> **Note**
>
> The service ignores parameters passed in URLs for HTTP POST requests that are signed with Signature Version 4.

When policies collide

Complexities arise when policies disagree or make no explicit mention of a user. Understanding how IAM works in the *IAM User Guide* provides a concise summary of policy evaluation logic:

- By default, all requests are denied.
- An explicit allow overrides this default.
- An explicit deny overrides any allows.

For example, if a resource-based policy grants you access to a domain subresource (an OpenSearch index or API), but an identity-based policy denies you access, you are denied access. If an identity-based policy grants access and a resource-based policy does not specify whether or not you should

have access, you are allowed access. See the following table of intersecting policies for a full summary of outcomes for domain subresources.

	Allowed in resource-based policy	**Denied in resource-based policy**	**Neither allowed nor denied in resource-based policy**
Allowed in identity-based policy	Allow	Deny	Allow
Denied in identity-based policy	Deny	Deny	Deny
Neither allowed nor denied in identity-based policy	Allow	Deny	Deny

Policy element reference

OpenSearch Service supports most policy elements in the IAM Policy Elements Reference, with the exception of `NotPrincipal`. The following table shows the most common elements.

JSON policy element	Summary
`Version`	The current version of the policy language is `2012-10-17` . All access policies should specify this value.
`Effect`	This element specifies whether the statement allows or denies access to the specified actions. Valid values are `Allow` or `Deny`.
`Principal`	This element specifies the AWS account or IAM role or user that is allowed or denied access to a resource and can take several forms: • **AWS accounts**: `"Principal":{"AWS": ["123456789012"]}` or `"Principal":{"AWS": ["arn:aws:iam::123456789012:root"]}` • **IAM users**: `"Principal":{"AWS": ["arn:aws:iam::123456789012:user/test-user"]}`

JSON policy element	Summary
	• **IAM roles**: `"Principal":{"AWS": ["arn:aws:iam::123456789012:role/test-role"]}`

Important

Specifying the * wildcard enables anonymous access to the domain, which we don't recommend unless you add an IP-based condition, use VPC support, or enable fine-grained access control. In addition, carefully inspect the following policies to confirm that they do not grant broad access:

- Identity-based policies attached to associated AWS principals (for example, IAM roles)
- Resource-based policies attached to associated AWS resources (for example, AWS Key Management Service KMS keys)

JSON policy element	Summary

`Action`	OpenSearch Service uses `ESHttp*` actions for OpenSearch HTTP methods. The rest of the actions apply to the configuration API.
	
	Certain `es:` actions support resource-level permissions. For example, you can give a user permissions to delete one particular domain without giving that user permissions to delete *any* domain. Other actions apply only to the service itself. For example, `es:ListDomainNames` makes no sense in the context of a single domain and thus requires a wildcard.
	
	For a list of all available actions and whether they apply to the domain subresources (`test-domain/*`), to the domain configuration (`test-domain`), or only to the service (`*`), see Actions, resources, and condition keys for Amazon OpenSearch Service in the *Service Authorization Reference*
	Resource-based policies differ from resource-level permissions. Resource-based policies are full JSON policies that attach to domains. Resource-level permissions let you restrict actions to particular domains or subresources. In practice, you can think of resource-level permissions as an optional part of a resource- or identity-based policy.
	
	While resource-level permissions for `es:CreateDomain` might seem unintuitive—after all, why give a user permissions to create a domain that already exists?—the use of a wildcard lets you enforce a simple naming scheme for your domains, such as `"Resource": "arn:aws:es:us-west-1:987654321098:domain/my-team-name-*"`.
	
	Of course, nothing prevents you from including actions alongside less restrictive resource elements, such as the following:
	
	`{`
	`  "Version": "2012-10-17",`
	`  "Statement": [`
	`  {`
	`  "Effect": "Allow",`
	`  "Action": [`

JSON policy element	Summary

	<pre>"es:ESHttpGet",
	"es:DescribeDomain"
	],
	"Resource": "*"
	}
	]
	}</pre>
	To learn more about pairing actions and resources, see the `Resource` element in this table.

`Condition`	OpenSearch Service supports most conditions that are described in AWS global condition context keys in the *IAM User Guide*. Notable exceptions include the `aws:PrincipalTag` key, which OpenSearch Service does not support.
	
	When configuring an IP-based policy, you specify the IP addresses or CIDR block as a condition, such as the following:
	
	<pre>"Condition": {
	"IpAddress": {
	"aws:SourceIp": [
	"192.0.2.0/32"
	]
	}
	}</pre>
	As noted in the section called “Identity-based policies”, the `aws:ResourceTag`, `aws:RequestTag`, and `aws:TagKeys` condition keys apply to the configuration API as well as the OpenSearch APIs.

JSON policy element	Summary

`Resource`	OpenSearch Service uses `Resource` elements in three basic ways:
	
	• For actions that apply to OpenSearch Service itself, like `es:ListDomainNames` , or to allow full access, use the following syntax:
	
	`"Resource": "*"`
	
	• For actions that involve a domain's configuration, like `es:DescribeDomain` , you can use the following syntax:
	
	`"Resource": "arn:aws:es:` *region*`:`*aws-account-id*`:domain/`*domain-name*`"`
	
	• For actions that apply to a domain's subresources, like `es:ESHttpGet` , you can use the following syntax:
	
	`"Resource": "arn:aws:es:` *region*`:`*aws-account-id*`:domain/`*domain-name*`/*"`
	
	You don't have to use a wildcard. OpenSearch Service lets you define a different access policy for each OpenSearch index or API. For example, you might limit a user's permissions to the `test-index` index:
	
	`"Resource": "arn:aws:es:` *region*`:`*aws-account-id*`:domain/`*domain-name*`/test-index"`
	
	Instead of full access to `test-index` , you might prefer to limit the policy to just the search API:
	
	`"Resource": "arn:aws:es:` *region*`:`*aws-account-id*`:domain/`*domain-name*`/test-index/_search"`
	
	You can even control access to individual documents:

JSON policy element	Summary
	`"Resource": "arn:aws:es:  region:aws-account-id:domain/domain-name /test-index/test-type/1"` Essentially, if OpenSearch expresses the subresource as a URI, you can control access to it using an access policy. For even more control over which resources a user can access, see the section called "Fine-grained access control". For details about which actions support resource-level permissions, see the `Action` element in this table.

Advanced options and API considerations

OpenSearch Service has several advanced options, one of which has access control implications: `rest.action.multi.allow_explicit_index`. At its default setting of true, it allows users to bypass subresource permissions under certain circumstances.

For example, consider the following resource-based policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::123456789012:user/test-user"
        ]
      },
      "Action": [
        "es:ESHttp*"
      ],
      "Resource": [
        "arn:aws:es:us-west-1:987654321098:domain/test-domain/test-index/*",
        "arn:aws:es:us-west-1:987654321098:domain/test-domain/_bulk"
      ]
    },
```

```
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::123456789012:user/test-user"
        ]
      },
      "Action": [
        "es:ESHttpGet"
      ],
      "Resource": "arn:aws:es:us-west-1:987654321098:domain/test-domain/restricted-
index/*"
    }
  ]
}
```

This policy grants `test-user` full access to `test-index` and the OpenSearch bulk API. It also allows `GET` requests to `restricted-index`.

The following indexing request, as you might expect, fails due to a permissions error:

```
PUT https://search-test-domain.us-west-1.es.amazonaws.com/restricted-index/movie/1
{
  "title": "Your Name",
  "director": "Makoto Shinkai",
  "year": "2016"
}
```

Unlike the index API, the bulk API lets you create, update, and delete many documents in a single call. You often specify these operations in the request body, however, rather than in the request URL. Because OpenSearch Service uses URLs to control access to domain subresources, `test-user` can, in fact, use the bulk API to make changes to `restricted-index`. Even though the user lacks `POST` permissions on the index, the following request **succeeds**:

```
POST https://search-test-domain.us-west-1.es.amazonaws.com/_bulk
{ "index" : { "_index": "restricted-index", "_type" : "movie", "_id" : "1" } }
{ "title": "Your Name", "director": "Makoto Shinkai", "year": "2016" }
```

In this situation, the access policy fails to fulfill its intent. To prevent users from bypassing these kinds of restrictions, you can change `rest.action.multi.allow_explicit_index` to false. If this value is false, all calls to the bulk, mget, and msearch APIs that specify index names in the

request body stop working. In other words, calls to `_bulk` no longer work, but calls to `test-index/_bulk` do. This second endpoint contains an index name, so you don't need to specify one in the request body.

OpenSearch Dashboards relies heavily on mget and msearch, so it is unlikely to work properly after this change. For partial remediation, you can leave `rest.action.multi.allow_explicit_index` as true and deny certain users access to one or more of these APIs.

For information about changing this setting, see the section called “Advanced cluster settings”.

Similarly, the following resource-based policy contains two subtle issues:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/test-user"
      },
      "Action": "es:ESHttp*",
      "Resource": "arn:aws:es:us-west-1:987654321098:domain/test-domain/*"
    },
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/test-user"
      },
      "Action": "es:ESHttp*",
      "Resource": "arn:aws:es:us-west-1:987654321098:domain/test-domain/restricted-
index/*"
    }
  ]
}
```

- Despite the explicit deny, `test-user` can still make calls such as `GET https://search-test-domain.us-west-1.es.amazonaws.com/_all/_search` and `GET https://search-test-domain.us-west-1.es.amazonaws.com/*/_search` to access the documents in `restricted-index`.

- Because the `Resource` element references `restricted-index/*`, `test-user` doesn't have permissions to directly access the index's documents. The user does, however, have permissions to *delete the entire index*. To prevent access and deletion, the policy instead must specify `restricted-index*`.

Rather than mixing broad allows and focused denies, the safest approach is to follow the principle of least privilege and grant only the permissions that are required to perform a task. For more information about controlling access to individual indexes or OpenSearch operations, see the section called "Fine-grained access control".

> **Important**
>
> Specifying the * wildcard enables anonymous access to your domain. It is not recommended you use the wildcard In addition, carefully inspect the following policies to confirm that they do not grant broad access:
>
> - Identity-based policies attached to associated AWS principals (for example, IAM roles)
> - Resource-based policies attached to associated AWS resources (for example, AWS Key Management Service KMS keys)

Configuring access policies

- For instructions on creating or modifying resource- and IP-based policies in OpenSearch Service, see the section called "Configuring access policies".
- For instructions on creating or modifying identity-based policies in IAM, see Creating IAM policies in the *IAM User Guide*.

Additional sample policies

Although this chapter includes many sample policies, AWS access control is a complex subject that is best understood through examples. For more, see Example IAM identity-based policies in the *IAM User Guide*.

Amazon OpenSearch Service API permissions reference

When you set up access control, you write permission policies that you can attach to an IAM identity (identity-based policies). For detailed reference information, see the following topics in the *Service Authorization Reference*:

- Actions, resources, and condition keys for OpenSearch Service.
- Actions, resources, and condition keys for OpenSearch Ingestion.

This reference contains information about which API operations can be used in an IAM policy. It also includes the AWS resource for which you can grant the permissions, and condition keys that you can include for fine-grained access control.

You specify the actions in the policy's `Action` field, the resource value in the policy's `Resource` field, and conditions in the policy's `Condition` field. To specify an action for OpenSearch Service, use the `es:` prefix followed by the API operation name (for example, `es:CreateDomain`). To specify an action for OpenSearch Ingestion, use the `osis:` prefix followed by the API operation (for example, `osis:CreatePipeline`).

AWS managed policies for Amazon OpenSearch Service

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AmazonOpenSearchDirectQueryGlueCreateAccess

Grants Amazon OpenSearch Service Direct Query Service access to the `CreateDatabase`, `CreatePartition`,`CreateTable`, and `BatchCreatePartition` AWS Glue API.

You can find the AmazonOpenSearchDirectQueryGlueCreateAccess policy in the IAM console.

AmazonOpenSearchServiceFullAccess

Grants full access to the OpenSearch Service configuration API operations and resources for an AWS account.

You can find the AmazonOpenSearchServiceFullAccess policy in the IAM console.

AmazonOpenSearchServiceReadOnlyAccess

Grants read-only access to all OpenSearch Service resources for an AWS account.

You can find the AmazonOpenSearchServiceReadOnlyAccess policy in the IAM console.

AmazonOpenSearchServiceRolePolicy

You can't attach `AmazonOpenSearchServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows OpenSearch Service to access account resources. For more information, see the section called “Permissions”.

You can find the AmazonOpenSearchServiceRolePolicy policy in the IAM console.

AmazonOpenSearchServiceCognitoAccess

Provides the minimum Amazon Cognito permissions necessary to enable Cognito authentication.

You can find the AmazonOpenSearchServiceCognitoAccess policy in the IAM console.

AmazonOpenSearchIngestionServiceRolePolicy

You can't attach `AmazonOpenSearchIngestionServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows OpenSearch Ingestion to enable VPC access for ingestion pipelines, create tags, and publish ingestion-related CloudWatch metrics to your account. For more information, see the section called “Using service-linked roles”.

You can find the AmazonOpenSearchIngestionServiceRolePolicy policy in the IAM console.

OpenSearchIngestionSelfManagedVpcePolicy

You can't attach `OpenSearchIngestionSelfManagedVpcePolicy` to your IAM entities. This policy is attached to a service-linked role that allows OpenSearch Ingestion to enable self-managed VPC access for ingestion pipelines, create tags, and publish ingestion-related CloudWatch metrics to your account. For more information, see the section called “Using service-linked roles”.

You can find the OpenSearchIngestionSelfManagedVpcePolicy policy in the IAM console.

AmazonOpenSearchIngestionFullAccess

Grants full access to the OpenSearch Ingestion API operations and resources for an AWS account.

You can find the AmazonOpenSearchIngestionFullAccess policy in the IAM console.

AmazonOpenSearchIngestionReadOnlyAccess

Grants read-only access to all OpenSearch Ingestion resources for an AWS account.

You can find the AmazonOpenSearchIngestionReadOnlyAccess policy in the IAM console.

AmazonOpenSearchServerlessServiceRolePolicy

Provides the minimum Amazon CloudWatch permissions necessary to send OpenSearch Serverless metric data to CloudWatch.

You can find the AmazonOpenSearchServerlessServiceRolePolicy policy in the IAM console.

OpenSearch Service updates to AWS managed policies

View details about updates to AWS managed policies for OpenSearch Service since this service began tracking changes.

Change	Description	Date
Updated `AmazonOpenSearchServerlessServiceRolePolicy`	Added the Sid `AllowAOSSCloudwatchMetrics` to the policy `AmazonOpenSearchServerlessServiceRolePolicy` . A Sid is a statement ID that	12 July 2024

Change	Description	Date
	acts as an optional identifier for the policy statement.	
Added `OpenSearchIngestionSelfManagedVpcePolicy`	A new policy that allows OpenSearch Ingestion to enable self-managed VPC access for ingestion pipelines, create tags, and publish ingestion-related CloudWatch metrics to your account. For the policy JSON, see the IAM console.	12 June 2024
Added`AmazonOpenSearchDirectQueryGlueCreateAccess`	Grants Amazon OpenSearch Service Direct Query Service access to the `CreateDatabase` , `CreatePartition` ,`CreateTable` , and `BatchCreatePartition` AWS Glue API.	6 May 2024
Updated `AmazonOpenSearchServiceRolePolicy` and `AmazonElasticsearchServiceRolePolicy`	Added the permissions necessary for the service-linked role to assign and unassign IPv6 addresses. The deprecated Elasticsearch policy has also been updated to ensure backwards compatibility.	18 October 2023

Change	Description	Date
Added `AmazonOpenSearchIngestionServiceRolePolicy`	A new policy that allows OpenSearch Ingestion to enable VPC access for ingestion pipelines, create tags, and publish ingestion-related CloudWatch metrics to your account. For the policy JSON, see the IAM console.	26 April 2023
Added `AmazonOpenSearchIngestionFullAccess`	A new policy that grants full access to the OpenSearch Ingestion API operations and resources for an AWS account. For the policy JSON, see the IAM console.	26 April 2023
Added `AmazonOpenSearchIngestionReadOnlyAccess`	A new policy that grants read-only access to all OpenSearch Ingestion resources for an AWS account. For the policy JSON, see the IAM console.	26 April 2023

Change	Description	Date
Added `AmazonOpenSearchServerlessServiceRolePolicy`	A new policy that provides the minimum permissions necessary to send OpenSearch Serverless metric data to Amazon CloudWatch. For the policy JSON, see the IAM console.	29 November 2022
Updated `AmazonOpenSearchServiceRolePolicy` and `AmazonElasticsearchServiceRolePolicy`	Added the permissions necessary for the service-linked role to create OpenSearch Service-managed VPC endpoints. Some actions can only be performed when the request contains the tag `OpenSearchManaged=true`. The deprecated Elasticsearch policy has also been updated to ensure backwards compatibility.	7 November 2022

Change	Description	Date
Updated `AmazonOpenSearchServiceRolePolicy` and `AmazonElasticsearchServiceRolePolicy`	Added support for the `PutMetricData` action, which is required to publish OpenSearch cluster metrics to Amazon CloudWatch. The deprecated Elasticsearch policy has also been updated to ensure backwards compatibility. For the policy JSON, see the IAM console.	12 September 2022
Updated `AmazonOpenSearchServiceRolePolicy` and `AmazonElasticsearchServiceRolePolicy`	Added support for the `acm` resource type. The policy provides the minimum AWS Certificate Manager (ACM) read-only permission necessary for the service-linked role to verify and validate ACM resources in order to create and update custom endpoint enabled domains. The deprecated Elasticsearch policy has also been updated to ensure backwards compatibility.	28 July 2022

Change	Description	Date
Updated AmazonOpenSearchServiceCognitoAccess and AmazonESCognitoAccess	Added support for the UpdateUserPoolClient action, which is required to set Cognito user pool configuration during upgrade from Elasticsearch to OpenSearch. Corrected permissions for the SetIdentityPoolRoles action to allow access to all resources. The deprecated Elasticsearch policy has also been updated to ensure backwards compatibility.	20 December 2021
Updated AmazonOpenSearchServiceRolePolicy	Added support for the security-group resource type. The policy provides the minimum Amazon EC2 and Elastic Load Balancing permissions necessary for the service-linked role to enable VPC access.	9 September 2021

Change	Description	Date
• Added `AmazonOpenSearchServiceFullAccess` • Deprecated `AmazonESFullAccess`	This new policy is meant to replace the old policy. Both policies provide full access to the OpenSearch Service configuration API and all HTTP methods for the OpenSearch APIs. Fine-grained access control and resource-based policies can still restrict access.	7 September 2021
• Added `AmazonOpenSearchServiceReadOnlyAccess` • Deprecated `AmazonESReadOnlyAccess`	This new policy is meant to replace the old policy. Both policies provide read-only access to the OpenSearch Service configuration API (`es:Describe*`, `es:List*`, and `es:Get*`) and *no* access to the HTTP methods for the OpenSearch APIs.	7 September 2021
• Added `AmazonOpenSearchServiceCognitoAccess` • Deprecated `AmazonESCognitoAccess`	This new policy is meant to replace the old policy. Both policies provide the minimum Amazon Cognito permissions necessary to enable Cognito authentication.	7 September 2021

Change	Description	Date
• Added AmazonOpenSearchServiceRolePolicy • Deprecated `AmazonElasticsearchServiceRolePolicy`	This new policy is meant to replace the old policy. Both policies provide the minimum Amazon EC2 and Elastic Load Balancing permissions necessary for the service-linked role to enable VPC access.	7 September 2021
Started tracking changes	Amazon OpenSearch Service now tracks changes to AWS-managed policies.	7 September 2021

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that Amazon OpenSearch Service gives another service to the resource. If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both global condition context keys to limit permissions. If you use both global condition context keys and the `aws:SourceArn` value contains the account ID, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The value of `aws:SourceArn` must be the ARN of the OpenSearch Service domain.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcards (*) for the unknown portions of the ARN. For example, `arn:aws:es:*:123456789012:*`.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in OpenSearch Service to prevent the confused deputy problem.

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Sid":"ConfusedDeputyPreventionExamplePolicy",
      "Effect":"Allow",
      "Principal":{
         "Service":"es.amazonaws.com"
      },
      "Action":"sts:AssumeRole",
      "Condition":{
         "StringEquals":{
            "aws:SourceAccount":"123456789012"
         },
         "ArnLike":{
            "aws:SourceArn":"arn:aws:es:region:123456789012:domain/my-domain"
         }
      }
   }
}
```

Fine-grained access control in Amazon OpenSearch Service

Fine-grained access control offers additional ways of controlling access to your data on Amazon OpenSearch Service. For example, depending on who makes the request, you might want a search to return results from only one index. You might want to hide certain fields in your documents or exclude certain documents altogether.

Fine-grained access control offers the following benefits:

- Role-based access control
- Security at the index, document, and field level

- OpenSearch Dashboards multi-tenancy
- HTTP basic authentication for OpenSearch and OpenSearch Dashboards

Topics

The bigger picture: fine-grained access control and OpenSearch Service security

Amazon OpenSearch Service security has three main layers:

Network

The first security layer is the network, which determines whether requests reach an OpenSearch Service domain. If you choose **Public access** when you create a domain, requests from any internet-connected client can reach the domain endpoint. If you choose **VPC access**, clients must connect to the VPC (and the associated security groups must permit it) for a request to reach the endpoint. For more information, see the section called “VPC support”.

Domain access policy

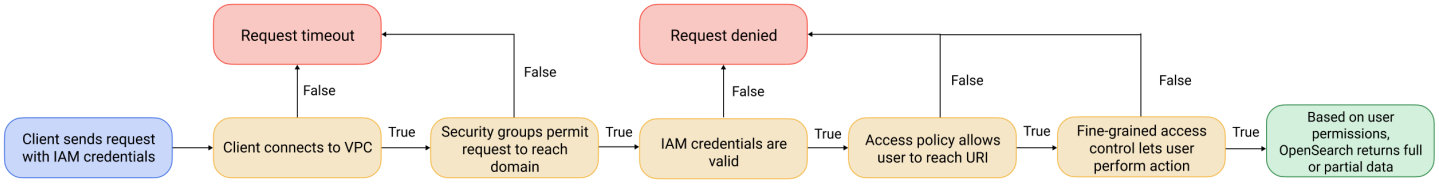
The second security layer is the domain access policy. After a request reaches a domain endpoint, the resource-based access policy allows or denies the request access to a given URI. The access policy accepts or rejects requests at the "edge" of the domain, before they reach OpenSearch itself.

Fine-grained access control

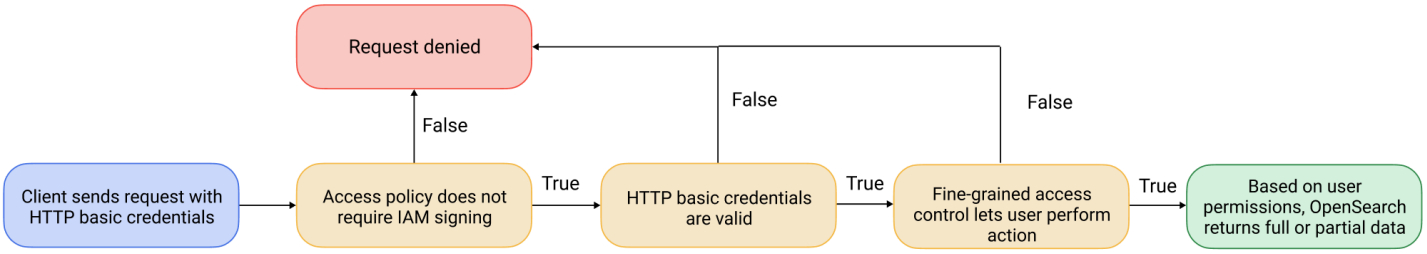
The third and final security layer is fine-grained access control. After a resource-based access policy allows a request to reach a domain endpoint, fine-grained access control evaluates the user credentials and either authenticates the user or denies the request. If fine-grained access control authenticates the user, it fetches all roles mapped to that user and uses the complete set of permissions to determine how to handle the request.

> **Note**
>
> If a resource-based access policy contains IAM roles or users, clients must send signed requests using AWS Signature Version 4. As such, access policies can conflict with fine-grained access control, especially if you use the internal user database and HTTP basic authentication. You can't sign a request with a username and password *and* IAM credentials. In general, if you enable fine-grained access control, we recommend using a domain access policy that doesn't require signed requests.

The following diagram illustrates a common configuration: a VPC access domain with fine-grained access control enabled, an IAM-based access policy, and an IAM master user.



The following diagram illustrates another common configuration: a public access domain with fine-grained access control enabled, an access policy that doesn't use IAM principals, and a master user in the internal user database.



Example

Consider a GET request to `movies/_search?q=thor`. Does the user have permissions to search the `movies` index? If so, does the user have permissions to see *all* documents within it? Should the response omit or anonymize any fields? For the master user, the response might look like this:

```
{
  "hits": {
    "total": 7,
    "max_score": 8.772789,
    "hits": [{
        "_index": "movies",
        "_type": "_doc",
        "_id": "tt0800369",
        "_score": 8.772789,
        "_source": {
          "directors": [
            "Kenneth Branagh",
            "Joss Whedon"
          ],
          "release_date": "2011-04-21T00:00:00Z",
          "genres": [
            "Action",
            "Adventure",
            "Fantasy"
          ],
          "plot": "The powerful but arrogant god Thor is cast out of Asgard to
 live amongst humans in Midgard (Earth), where he soon becomes one of their finest
 defenders.",
          "title": "Thor",
          "actors": [
            "Chris Hemsworth",
            "Anthony Hopkins",
            "Natalie Portman"
          ],
```

```
          "year": 2011
        }
      },
      ...
    ]
  }
}
```

If a user with more limited permissions issues the exact same request, the response might look like this:

```
{
  "hits": {
    "total": 2,
    "max_score": 8.772789,
    "hits": [{
        "_index": "movies",
        "_type": "_doc",
        "_id": "tt0800369",
        "_score": 8.772789,
        "_source": {
          "year": 2011,
          "release_date":
 "3812a72c6dd23eef3c750c2d99e205cbd260389461e19d610406847397ecb357",
          "plot": "The powerful but arrogant god Thor is cast out of Asgard to
 live amongst humans in Midgard (Earth), where he soon becomes one of their finest
 defenders.",
          "title": "Thor"
        }
      },
      ...
    ]
  }
}
```

The response has fewer hits and fewer fields for each hit. Also, the `release_date` field is anonymized. If a user with no permissions makes the same request, the cluster returns an error:

```
{
  "error": {
    "root_cause": [{
      "type": "security_exception",
```

```
      "reason": "no permissions for [indices:data/read/search] and User [name=limited-
user, roles=[], requestedTenant=null]"
    }],
    "type": "security_exception",
    "reason": "no permissions for [indices:data/read/search] and User [name=limited-
user, roles=[], requestedTenant=null]"
  },
  "status": 403
}
```

If a user provides invalid credentials, the cluster returns an `Unauthorized` exception.

Key concepts

As you get started with fine-grained access control, consider the following concepts:

- **Roles** – The core way of using fine-grained access control. In this case, roles are distinct from IAM roles. Roles contain any combination of permissions: cluster-wide, index-specific, document level, and field level.
- **Mapping** – After you configure a role, you *map* it to one or more users. For example, you might map three roles to a single user: one role that provides access to Dashboards, one that provides read-only access to `index1`, and one that provides write access to `index2`. Or you could include all of those permissions in a single role.
- **Users** – People or applications that make requests to the OpenSearch cluster. Users have credentials—either IAM access keys or a username and password—that they specify when they make requests.

About the master user

The *master user* in OpenSearch Service is either a username and password combination, or an IAM principal, that has full permissions to the underlying OpenSearch cluster. A user is considered a master user if they have all access to the OpenSearch cluster along with the ability to create internal users, roles, and role mappings within OpenSearch Dashboards.

A master user created in the OpenSearch Service console or through the CLI is automatically mapped to two predefined roles:

- `all_access` – Provides full access to all cluster-wide operations, permission to write to all cluster indexes, and permission to write to all tenants.

- `security_manager` – Provides access to the Security plugin and management of users and permissions.

With these two roles, the user gains access to the **Security** tab in OpenSearch Dashboards, where they can manage users and permissions. If you create another internal user and only map it to the `all_access` role, the user doesn't have access to the **Security** tab. You can create additional master users by explicitly mapping them to both the `all_access` and `security_manager` roles. For instructions, see the section called "Additional master users".

When you create a master user for your domain, you can specify either an existing *IAM principal*, or create a master user within the *internal user database*. Consider the following when deciding which to use:

- **IAM principal** – If you choose an IAM principal for your master user, all requests to the cluster must be signed using AWS Signature Version 4.

  OpenSearch Service doesn't take any of the IAM principal's permissions into consideration. The IAM user or role serves purely for *authentication*. The policies on that user or role have no bearing on the *authorization* of the master user. Authorization is handled through the various permissions in the OpenSearch Security plugin.

  For example, you can assign zero *IAM* permissions to an IAM principal, and as long as the machine or person can authenticate to that user or role, they have the power of the master user in OpenSearch Service.

  We recommend IAM if you want to use the same users on multiple clusters, if you want to use Amazon Cognito to access Dashboards, or if you have OpenSearch clients that support Signature Version 4 signing.
- **Internal user database** – If you create a master in the internal user database (with a username and password combination), you can use HTTP basic authentication (as well as IAM credentials) to make requests to the cluster. Most clients support basic authentication, including curl, which also supports AWS Signature Version 4 with the --aws-sigv4 option. The internal user database is stored in an OpenSearch index, so you can't share it with other clusters.

  We recommend the internal user database if you don't need to reuse users across multiple clusters, if you want to use HTTP basic authentication to access Dashboards (rather than Amazon Cognito), or if you have clients that only support basic authentication. The internal user database is the simplest way to get started with OpenSearch Service.

Enabling fine-grained access control

Enable fine-grained access control using the console, AWS CLI, or configuration API. For steps, see *Creating and managing domains*.

Fine-grained access control requires OpenSearch or Elasticsearch 6.7 or later. It also requires HTTPS for all traffic to the domain, Encryption of data at rest, and node-to-node encryption. Depending on how you configure the advanced features of fine-grained access control, additional processing of your requests may require compute and memory resources on individual data nodes. After you enable fine-grained access control, you can't disable it.

Enabling fine-grained access control on existing domains

You can enable fine-grained access control on existing domains running OpenSearch or Elasticsearch 6.7 or later.

To enable fine-grained access control on an existing domain (console)

1. Select your domain and choose **Actions** and **Edit security configuration**.
2. Select **Enable fine-grained access control**.
3. Choose how to create the master user:
   - If you want to use IAM for user management, choose **Set IAM ARN as master user** and specify the ARN for an IAM role.
   - If you want to use the internal user database, choose **Create master user** and specify a username and password.
4. (Optional) Select **Enable migration period for open/IP-based access policy**. This setting enables a 30-day transition period during which your existing users can continue to access the domain without disruptions, and existing open and IP-based access policies will continue to work with your domain. During this migration period, we recommend that administrators create the necessary roles and map them to users for the domain. If you use identity-based policies instead of an open or IP-based access policy, you can disable this setting.

   You also need to update your clients to work with fine-grained access control during the migration period. For example, if you map IAM roles with fine-grained access control, you must update your clients to start signing requests with AWS Signature Version 4. If you configure HTTP basic authentication with fine-grained access control, you must update your clients to provide appropriate basic authentication credentials in requests.

During the migration period, users who access the OpenSearch Dashboards endpoint for the domain will land directly on the **Discover** page rather than the login page. Administrators and master users can choose **Login** to log in with admin credentials and configure role mappings.

> **Important**
>
> **OpenSearch Service automatically disables the migration period after 30 days**. We recommend ending it as soon as you create the necessary roles and map them to users. After the migration period ends, you can't re-enable it.

5. Choose **Save changes**.

The change triggers a blue/green deployment during which the cluster health becomes red, but all cluster operations remain unaffected.

To enable fine-grained access control on an existing domain (CLI)

Set `AnonymousAuthEnabled` to `true` to enable the migration period with fine-grained access control:

```
aws opensearch update-domain-config --domain-name test-domain --region us-east-1 \
      --advanced-security-options '{ "Enabled": true,
 "InternalUserDatabaseEnabled":true, "MasterUserOptions": {"MasterUserName":"master-
username","MasterUserPassword":"master-password"},"AnonymousAuthEnabled": true}'
```

About the default_role

Fine-grained access control requires role mapping. If your domain uses identity-based access policies, OpenSearch Service automatically maps your users to a new role called **default_role** in order to help you properly migrate existing users. This temporary mapping ensures that your users can still successfully send IAM-signed GET and PUT requests until you create your own role mappings.

The role does not add any security vulnerabilities or flaws to your OpenSearch Service domain. We recommend deleting the default role as soon as you set up your own roles and map them accordingly.

Migration scenarios

The following table describes the behavior for each authentication method before and after enabling fine-grained access control on an existing domain, and the steps administrators must take to properly map their users to roles:

Authentication method	Before enabling fine-grained access control	After enabling fine-grained access control	Administrator tasks
Identity-based policies	All users satisfying the IAM policy can access the domain.	You don't need to enable the migration period. OpenSearch Service automatically maps all users that satisfy the IAM policy to the **default_role** so that they can continue to access the domain.	1. Create custom role mappings on the domain. 2. Delete the **default_role**.
IP-based policies	All users from the allowed IP addresses or CIDR blocks can access the domain.	During the 30-day migration period, all users from the allowed IP addresses or CIDR blocks can continue to access the domain.	1. Create custom role mappings on the domain. 2. Update your clients to either provide basic authentication credentials or IAM credentials, depending on your role mapping configuration. 3. Disable the migration period. Users from the allowed IP addresses or CIDR blocks sending requests without basic authentication or IAM credentials will lose access to the domain.

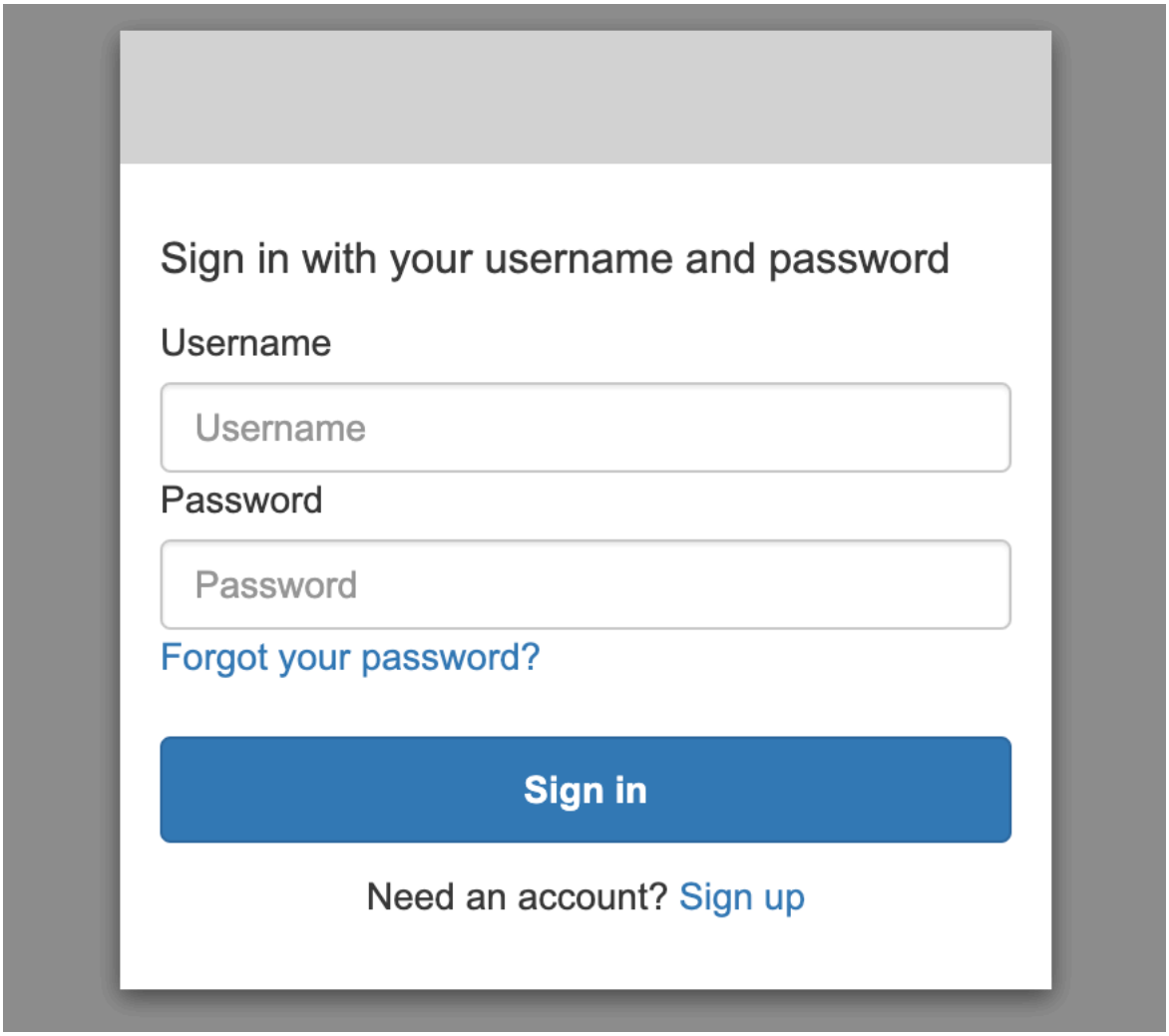
Authentication method	Before enabling fine-grained access control	After enabling fine-grained access control	Administrator tasks
Open access policies	All users over the internet can access the domain.	During the 30-day migration period, all users over the internet can continue to access to domain.	1. Create role mappings on the domain. 2. Update your clients to either provide basic authentication credentials or IAM credentials, depending on your role mapping configuration. 3. Disable the migration period. Users sending requests without basic authentication or IAM credentials will lose access to the domain.

Accessing OpenSearch Dashboards as the master user

Fine-grained access control has an OpenSearch Dashboards plugin that simplifies management tasks. You can use Dashboards to manage users, roles, mappings, action groups, and tenants. The OpenSearch Dashboards sign-in page and underlying authentication method differs, however, depending on how you manage users and configured your domain.

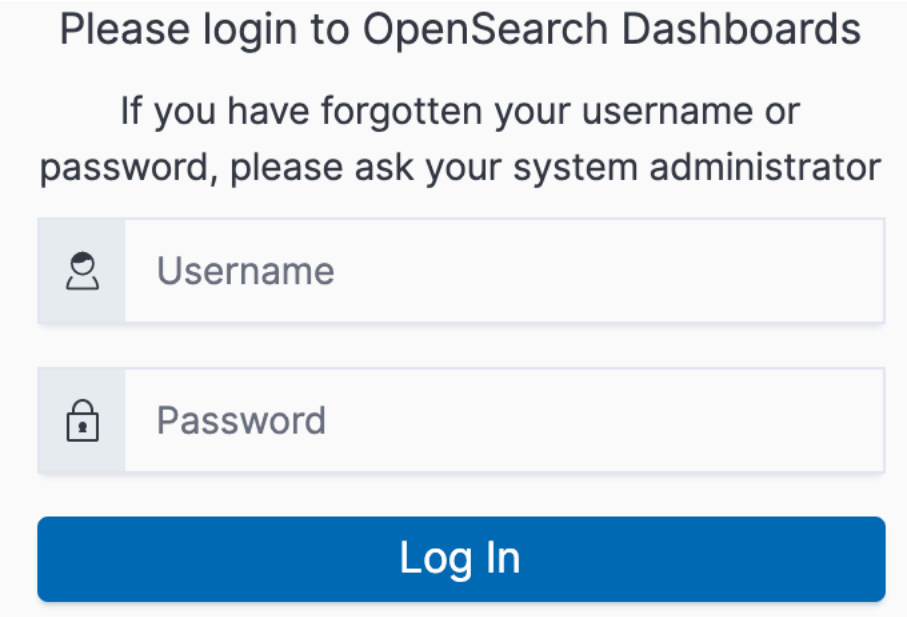
- If you want to use IAM for user management, use the section called "Amazon Cognito authentication for OpenSearch Dashboards" to access Dashboards. Otherwise, Dashboards shows a nonfunctional sign-in page. See the section called "Limitations".

  With Amazon Cognito authentication, one of the assumed roles from the identity pool must match the IAM role that you specified for the master user. For more information about this configuration, see the section called "(Optional) Configuring granular access" and the section called "Tutorial: Fine-grained access control with Cognito authentication".



- If you choose to use the internal user database, you can sign in to Dashboards with your master username and password. You must access Dashboards over HTTPS. Amazon Cognito and SAML authentication for Dashboards both replace this login screen.

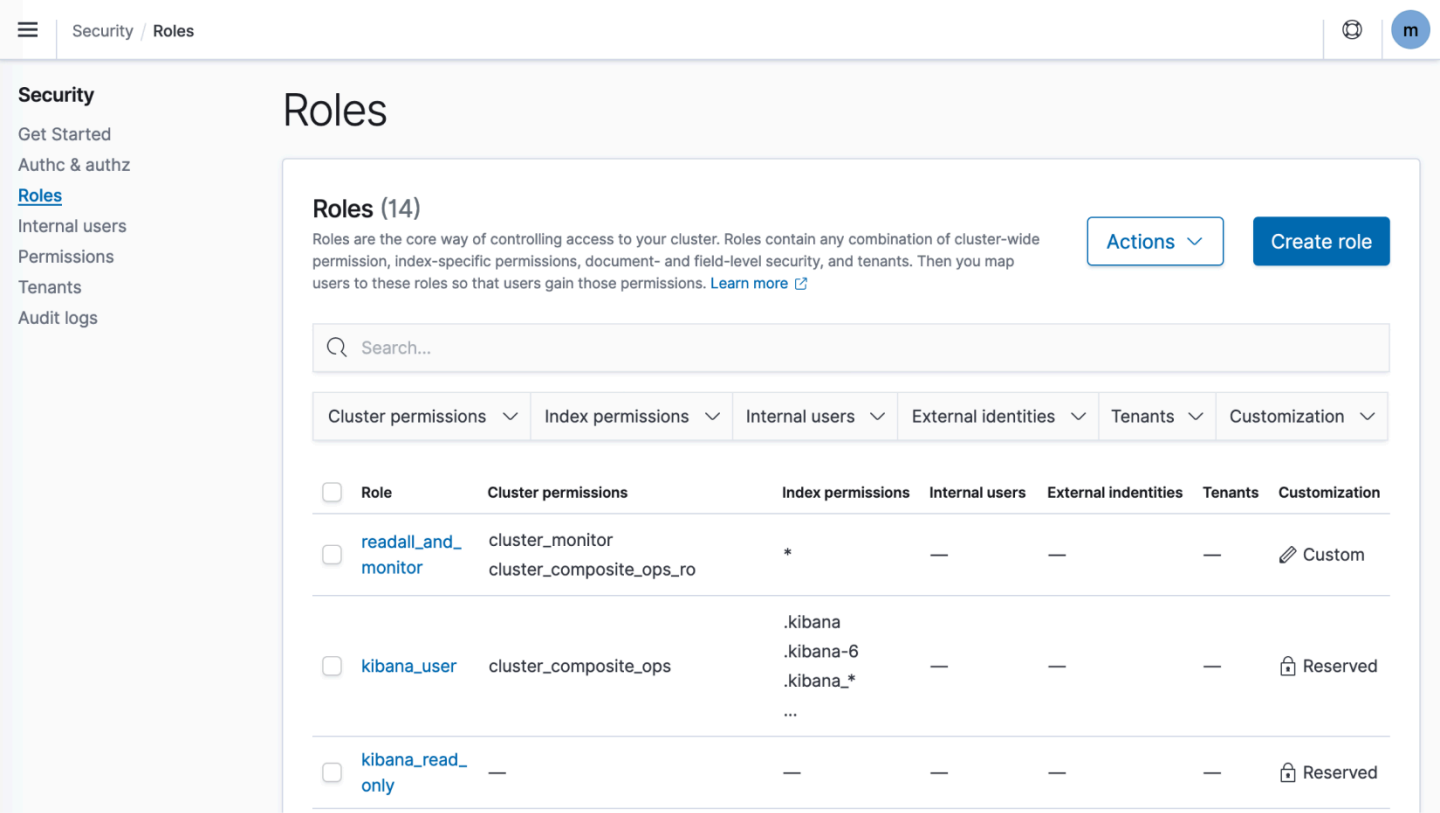
  For more information about this configuration, see the section called “Tutorial: Internal user database with basic authentication”.



- If you choose to use SAML authentication, you can sign in using credentials from an external identity provider. For more information, see the section called "SAML authentication for OpenSearch Dashboards".

Managing permissions

As noted in the section called "Key concepts", you manage fine-grained access control permissions using roles, users, and mappings. This section describes how to create and apply those resources. We recommend that you sign in to Dashboards as the master user to perform these operations.



> **Note**
>
> The permissions that you choose to grant to your users vary widely based on use case. We cannot feasibly cover all scenarios in this documentation. As you're determining which permissions to grant your users, make sure to reference the OpenSearch cluster and index permissions mentioned in the following sections, and always follow the principle of least privilege.

Creating roles

You can create new roles for fine-grained access control using OpenSearch Dashboards or the `_plugins/_security` operation in the REST API. For more information, see Create roles.

Fine-grained access control also includes a number of predefined roles. Clients such as OpenSearch Dashboards and Logstash make a wide variety of requests to OpenSearch, which can make it hard to manually create roles with the minimum set of permissions. For example, the `opensearch_dashboards_user` role includes the permissions that a user needs to work with index patterns, visualizations, dashboards, and tenants. We recommend mapping it to any user

or backend role that accesses Dashboards, along with additional roles that allow access to other indices.

Amazon OpenSearch Service doesn't offer the following OpenSearch roles:

- `observability_full_access`
- `observability_read_access`
- `reports_read_access`
- `reports_full_access`

Amazon OpenSearch Service offers several roles that aren't available with OpenSearch:

- `ultrawarm_manager`
- `ml_full_access`
- `cold_manager`
- `notifications_full_access`
- `notifications_read_access`

Cluster-level security

Cluster-level permissions include the ability to make broad requests such as `_mget`, `_msearch`, and `_bulk`, monitor health, take snapshots, and more. Manage these permissions using the **Cluster Permissions** section when creating a role. For a full list of cluster-level permissions, see Cluster permissions.

Rather than individual permissions, you can often achieve your desired security posture using a combination of the default action groups. For a list of cluster-level action groups, see Cluster-level.

Index-level security

Index-level permissions include the ability to create new indices, search indices, read and write documents, delete documents, manage aliases, and more. Manage these permissions using the **Index Permissions** section when creating a role. For a full list of index-level permissions, see Index permissions.

Rather than individual permissions, you can often achieve your desired security posture using a combination of the default action groups. For a list of index-level action groups, see Index-level.

Document-level security

Document-level security lets you restrict which documents in an index a user can see. When creating a role, specify an index pattern and an OpenSearch query. Any users that you map to that role can see only the documents that match the query. Document-level security affects the number of hits that you receive when you search.

For more information, see Document-level security.

Field-level security

Field-level security lets you control which document fields a user can see. When creating a role, add a list of fields to either include or exclude. If you include fields, any users you map to that role can see only those fields. If you exclude fields, they can see all fields *except* the excluded ones. Field-level security affects the number of fields included in hits when you search.

For more information, see Field-level security.

Field masking

Field masking is an alternative to field-level security that lets you anonymize the data in a field rather than remove it altogether. When creating a role, add a list of fields to mask. Field masking affects whether you can see the contents of a field when you search.

> **Tip**
>
> If you apply the standard masking to a field, OpenSearch Service uses a secure, random hash that can cause inaccurate aggregation results. To perform aggregations on masked fields, use pattern-based masking instead.

Creating users

If you enabled the internal user database, you can create users using OpenSearch Dashboards or the `_plugins/_security` operation in the REST API. For more information, see Create users.

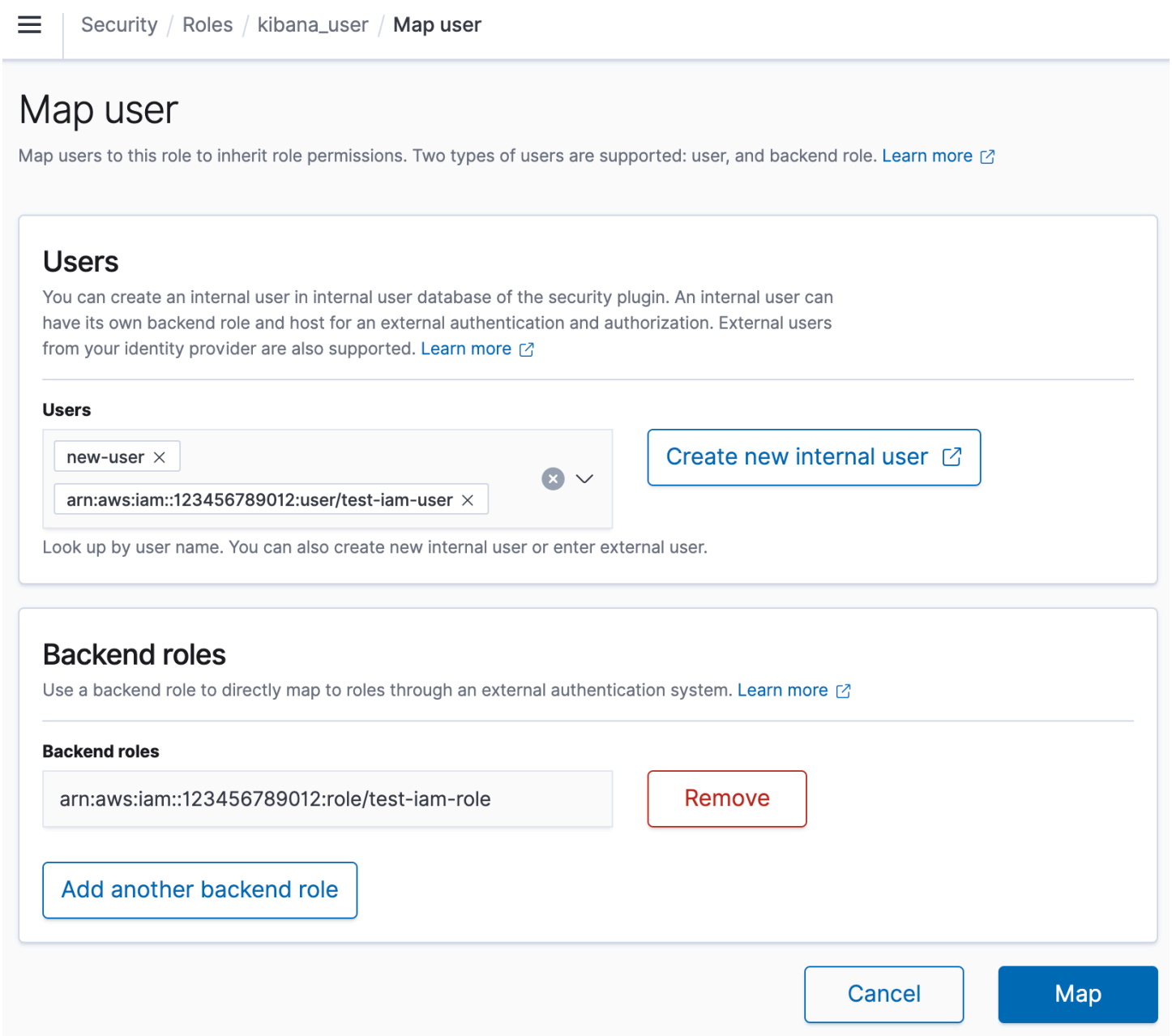
If you chose IAM for your master user, ignore this portion of Dashboards. Create IAM roles instead. For more information, see the IAM User Guide.

Mapping roles to users

Role mapping is the most critical aspect of fine-grained access control. Fine-grained access control has some predefined roles to help you get started, but unless you map roles to users, every request to the cluster ends in a permissions error.

Backend roles can help simplify the role mapping process. Rather than mapping the same role to 100 individual users, you can map the role to a single backend role that all 100 users share. Backend roles can be IAM roles or arbitrary strings.

- Specify users, user ARNs, and Amazon Cognito user strings in the **Users** section. Cognito user strings take the form of `Cognito/user-pool-id/username`.
- Specify backend roles and IAM role ARNs in the **Backend roles** section.



You can map roles to users using OpenSearch Dashboards or the `_plugins/_security` operation in the REST API. For more information, see Map users to roles.

Creating action groups

Action groups are sets of permissions that you can reuse across different resources. You can create new action groups using OpenSearch Dashboards or the `_plugins/_security` operation in the REST API, although the default action groups suffice for most use cases. For more information about the default action groups, see Default action groups.

OpenSearch Dashboards multi-tenancy

Tenants are spaces for saving index patterns, visualizations, dashboards, and other Dashboards objects. Dashboards multi-tenancy lets you safely share your work with other Dashboards users (or keep it private) and dynamically configure tenants. You can control which roles have access to a tenant and whether those roles have read or write access. The Global tenant is the default. To learn more, see OpenSearch Dashboards multi-tenancy.

To view your current tenant or change tenants

1. Navigate to OpenSearch Dashboards and sign in.
2. Select your user icon in the upper-right and choose **Switch tenants**.
3. Verify your tenant before creating visualizations or dashboards. If you want to share your work with all other Dashboards users, choose **Global**. To share your work with a subset of Dashboards users, choose a different shared tenant. Otherwise, choose **Private**.

> **Note**
>
> OpenSearch Dashboards maintains a separate index for each tenant, and creates an index template called `tenant_template`. Do not delete or modify the `tenant_template` index, as it could cause OpenSearch Dashboards to malfunction if the tenant index mapping is misconfigured.

Recommended configurations

Due to how fine-grained access control interacts with other security features, we recommend several fine-grained access control configurations that work well for most use cases.

Description	Master user	Domain access policy
Use IAM credentials for calls to the OpenSearch APIs, and use SAML authentication to access Dashboards. Manage	IAM role or user	`{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow",`

Description	Master user	Domain access policy

fine-grained access control roles using Dashboards or the REST API.		<pre>"Principal": {
		"AWS": "*"
		},
		"Action": "es:ESHttp*",
		"Resource": " *domain-arn* /*"
		}
		]
		}</pre>

Use IAM credentials or basic authentication for calls to the OpenSearch APIs. Manage fine-grained access control roles using Dashboards or the REST API.
 	
This configuration offers a lot of flexiblity, especially if you have OpenSearch clients that only support basic authentication.
 	
If you have an existing identity provider, use SAML authentication to access Dashboards. Otherwise, manage Dashboards users in the internal user database.
	Username and password	<pre>{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Effect": "Allow",
		"Principal": {
		"AWS": "*"
		},
		"Action": "es:ESHttp*",
		"Resource": " *domain-arn* /*"
		}
		]
		}</pre>

Description	Master user	Domain access policy
Use IAM credentials for calls to the OpenSearch APIs, and use Amazon Cognito to access Dashboards. Manage fine-grained access control roles using Dashboards or the REST API.	IAM role or user	`{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": { "AWS": "*" }, "Action": "es:ESHttp*", "Resource": " domain-arn /*" } ] }`
Use IAM credentials for calls to the OpenSearch APIs, and block most access to Dashboards. Manage fine-grained access control roles using the REST API.	IAM role or user	`{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": { "AWS": "*" }, "Action": "es:ESHttp*", "Resource": " domain-arn /*" }, { "Effect": "Deny", "Principal": { "AWS": "*" }, "Action": "es:ESHttp*", "Resource": " domain-arn /_dashboards*" } ] }`

Limitations

Fine-grained access control has several important limitations:

- The `hosts` aspect of role mappings, which maps roles to hostnames or IP addresses, doesn't work if the domain is within a VPC. You can still map roles to users and backend roles.
- If you choose IAM for the master user and don't enable Amazon Cognito or SAML authentication, Dashboards displays a nonfunctional sign-in page.
- If you choose IAM for the master user, you can still create users in the internal user database. Because HTTP basic authentication is not enabled under this configuration, however, any requests signed with those user credentials are rejected.
- If you use SQL to query an index that you don't have access to, you receive a "no permissions" error. If the index doesn't exist, you receive a "no such index" error. This difference in error messages means that you can confirm the existence of an index if you happen to guess its name.

  To minimize the issue, don't include sensitive information in index names. To deny all access to SQL, add the following element to your domain access policy:

```
{
  "Effect": "Deny",
  "Principal": {
    "AWS": [
      "*"
    ]
  },
  "Action": [
    "es:*"
  ],
  "Resource": "arn:aws:es:us-east-1:123456789012:domain/my-domain/_plugins/_sql"
}
```

- If your domain version is 2.3 or higher and you have fine-grained access control enabled, setting `max_clause_count` to 1 causes issues with your domain. We recommend setting this account to a higher number.
- If you are enabling fine-grained access control in a domain where fine-grained access control is not set up, for data sources created for direct query, you need to setup fine-grained access control roles yourself. For more information on how to set up fine-grained access roles, see Creating Amazon OpenSearch Service data source integrations with Amazon S3.

Modifying the master user

If you forget the details of the master user, you can reconfigure it using the console, AWS CLI, or configuration API.

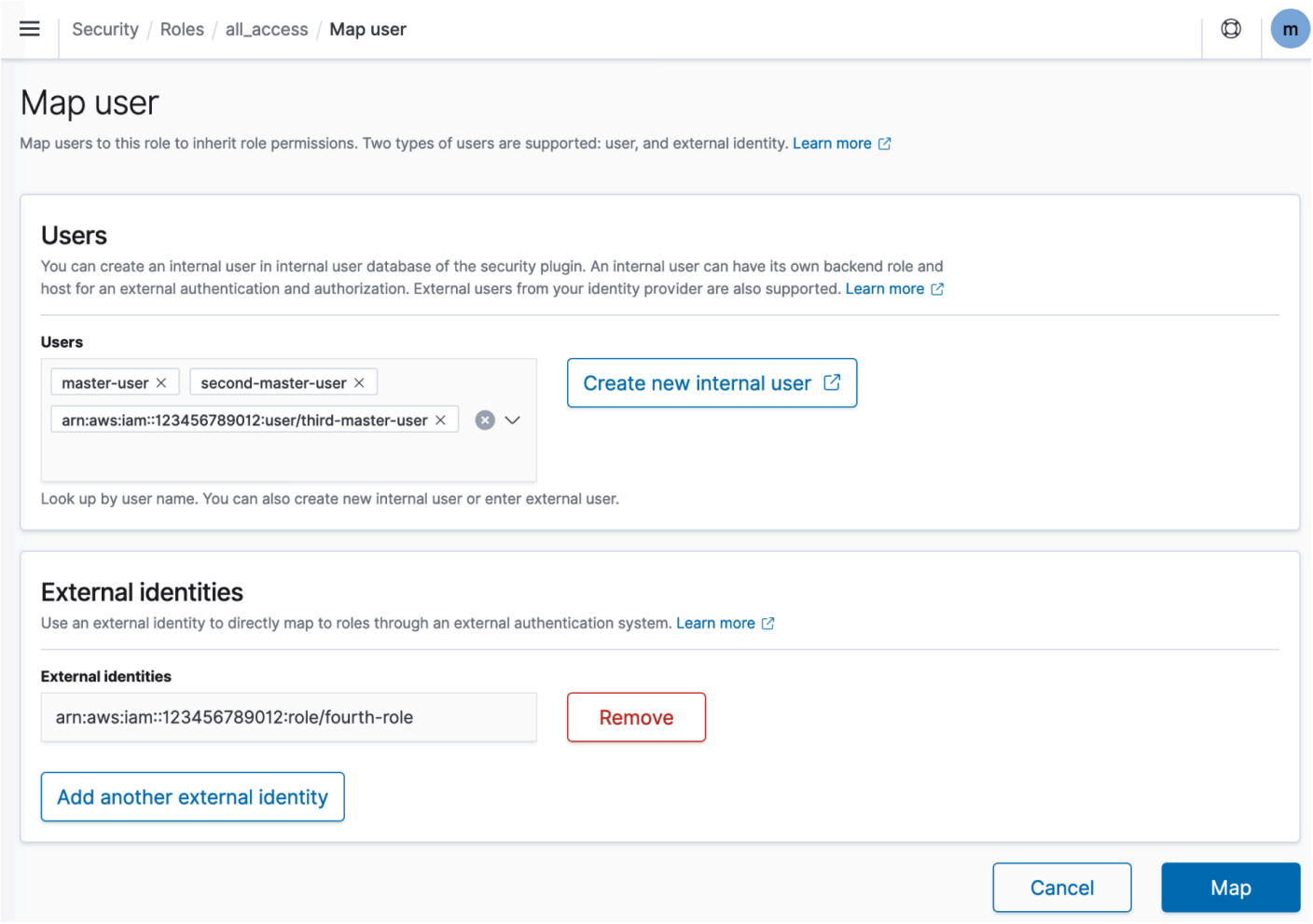
To modify the master user (console)

1. Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home/.
2. Choose your domain and choose **Actions**, **Edit security configuration**.
3. Choose either **Set IAM ARN as master user** or **Create master user**.
   - If you previously used an IAM master user, fine-grained access control re-maps the `all_access` role to the new IAM ARN that you specify.
   - If you previously used the internal user database, fine-grained access control creates a new master user. You can use the new master user to delete the old one.
   - Switching from the internal user database to an IAM master user does *not* delete any users from the internal user database. Instead, it just disables HTTP basic authentication. Manually delete users from the internal user database, or keep them in case you ever need to reenable HTTP basic authentication.
4. Choose **Save changes**.

Additional master users

You designate a master user when you create a domain, but if you want, you can use this master user to create additional master users. You have two options: OpenSearch Dashboards or the REST API.

- In Dashboards, choose **Security**, **Roles**, and then map the new master user to the `all_access` and `security_manager` roles.



- To use the REST API, send the following requests:

```
PUT _plugins/_security/api/rolesmapping/all_access
{
  "backend_roles": [
    "arn:aws:iam::123456789012:role/fourth-master-user"
  ],
  "hosts": [],
  "users": [
    "master-user",
    "second-master-user",
    "arn:aws:iam::123456789012:user/third-master-user"
  ]
}
```

```
PUT _plugins/_security/api/rolesmapping/security_manager
{
```

```
  "backend_roles": [
    "arn:aws:iam::123456789012:role/fourth-master-user"
  ],
  "hosts": [],
  "users": [
    "master-user",
    "second-master-user",
    "arn:aws:iam::123456789012:user/third-master-user"
  ]
}
```

These requests *replace* the current role mappings, so perform `GET` requests first so that you can include all current roles in the `PUT` requests. The REST API is especially useful if you can't access Dashboards and want to map an IAM role from Amazon Cognito to the `all_access` role.

Manual snapshots

Fine-grained access control introduces some additional complications with taking manual snapshots. To register a snapshot repository—even if you use HTTP basic authentication for all other purposes—you must map the `manage_snapshots` role to an IAM role that has `iam:PassRole` permissions to assume `TheSnapshotRole`, as defined in the section called "Prerequisites".

Then use that IAM role to send a signed request to the domain, as outlined in the section called "Registering a manual snapshot repository".

Integrations

If you use other AWS services with OpenSearch Service, you must provide the IAM roles for those services with appropriate permissions. For example, Firehose delivery streams often use an IAM role called `firehose_delivery_role`. In Dashboards, create a role for fine-grained access control, and map the IAM role to it. In this case, the new role needs the following permissions:

```
{
  "cluster_permissions": [
    "cluster_composite_ops",
    "cluster_monitor"
  ],
  "index_permissions": [{
    "index_patterns": [
```

```
      "firehose-index*"
    ],
    "allowed_actions": [
      "create_index",
      "manage",
      "crud"
    ]
  }]
}
```

Permissions vary based on the actions each service performs. An AWS IoT rule or AWS Lambda function that indexes data likely needs similar permissions to Firehose, while a Lambda function that only performs searches can use a more limited set.

REST API differences

The fine-grained access control REST API differs slightly depending on your OpenSearch/Elasticsearch version. Prior to making a PUT request, make a GET request to verify the expected request body. For example, a GET request to `_plugins/_security/api/user` returns all users, which you can then modify and use to make valid PUT requests.

On Elasticsearch 6.*x*, requests to create users look like this:

```
PUT _opendistro/_security/api/user/new-user
{
  "password": "some-password",
  "roles": ["new-backend-role"]
}
```

On OpenSearch or Elasticsearch 7.x, requests look like this (change `_plugins` to `_opendistro` if using Elasticsearch):

```
PUT _plugins/_security/api/user/new-user
{
  "password": "some-password",
  "backend_roles": ["new-backend-role"]
}
```

Further, tenants are properties of roles in Elasticsearch 6.*x*:

```
GET _opendistro/_security/api/roles/all_access
```

```
{
  "all_access": {
    "cluster": ["UNLIMITED"],
    "tenants": {
      "admin_tenant": "RW"
    },
    "indices": {
      "*": {
        "*": ["UNLIMITED"]
      }
    },
    "readonly": "true"
  }
}
```

In OpenSearch and Elasticsearch 7.x, they're objects with their own URI (change `_plugins` to `_opendistro` if using Elasticsearch)::

```
GET _plugins/_security/api/tenants

{
  "global_tenant": {
    "reserved": true,
    "hidden": false,
    "description": "Global tenant",
    "static": false
  }
}
```

For documentation on the OpenSearch REST API, see the Security plugin API reference.

Tip

If you use the internal user database, you can use curl to make requests and test your domain. Try the following sample commands:

```
curl -XGET -u 'master-user:master-user-password' 'domain-endpoint/_search'
curl -XGET -u 'master-user:master-user-password' 'domain-endpoint/_plugins/
_security/api/user'
```

Tutorial: Configure a domain with an IAM master user and Amazon Cognito authentication

This tutorial covers a popular Amazon OpenSearch Service use case for fine-grained access control: an IAM master user with Amazon Cognito authentication for OpenSearch Dashboards.

In the tutorial, we'll configure a *master* IAM role and a *limited* IAM role, which we'll then associate with users in Amazon Cognito. The master user can then sign in to OpenSearch Dashboards, map the limited user to a role, and use fine-grained access control to limit the user's permissions.

Although these steps use the Amazon Cognito user pool for authentication, this same basic process works for any Cognito authentication provider that lets you assign different IAM roles to different users.

You'll complete the following steps in this tutorial:

1. Create master and limited IAM roles
2. Create a domain with Cognito authentication
3. Configure a Cognito user pool and identity pool
4. Map roles in OpenSearch Dashboards
5. Test the permissions

Step 1: Create master and limited IAM roles

Navigate to the AWS Identity and Access Management (IAM) console and create two separate roles:

- `MasterUserRole` – The master user, which will have full permissions to the cluster and manage roles and role mappings.

- `LimitedUserRole` – A more restricted role, which you'll grant limited access to as the master user.

For instructions to create the roles, see Creating a role using custom trust policies.

Both roles must have the following trust policy, which allows your Cognito identity pool to assume the roles:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Federated": "cognito-identity.amazonaws.com"
    },
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
      "StringEquals": {
        "cognito-identity.amazonaws.com:aud": "{identity-pool-id}"
      },
      "ForAnyValue:StringLike": {
        "cognito-identity.amazonaws.com:amr": "authenticated"
      }
    }
  }]
}
```

Note

Replace `identity-pool-id` with the unique identifier of your Amazon Cognito identity pool. For example, `us-east-1:0c6cdba7-3c3c-443b-a958-fb9feb207aa6`.

Step 2: Create a domain with Cognito authentication

Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home/ and create a domain with the following settings:

- OpenSearch 1.0 or later, or Elasticsearch 7.8 or later
- Public access

- Fine-grained access control enabled with `MasterUserRole` as the master user (created in the previous step)
- Amazon Cognito authentication enabled for OpenSearch Dashboards. For instructions to enable Cognito authentication and select a user and identity pool, see the section called "Configuring a domain to use Amazon Cognito authentication".
- The following domain access policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{account-id}:root"
      },
      "Action": [
        "es:ESHttp*"
      ],
      "Resource": "arn:aws:es:{region}:{account-id}:domain/{domain-name}/*"
    }
  ]
}
```

- HTTPS required for all traffic to the domain
- Node-to-node encryption
- Encryption of data at rest

Step 3: Configure Cognito users

While your domain is being created, configure the master and limited users within Amazon Cognito by following Create a user pool in the *Amazon Cognito Developer Guide*. Lastly, configure your identity pool by following the steps in Create an identity pool in Amazon Cognito. The user pool and identity pool must be in the same AWS Region.

Step 4: Map roles in OpenSearch Dashboards

Now that your users are configured, you can sign in to OpenSearch Dashboards as the master user and map users to roles.

1. Go back to the OpenSearch Service console and navigate to the OpenSearch Dashboards URL for the domain you created. The URL follows this format: *domain-endpoint*/_dashboards/.
2. Sign in with the `master-user` credentials.
3. Choose **Add sample data** and add the sample flight data.
4. In the left navigation pane, choose **Security**, **Roles**, **Create role**.
5. Name the role `new-role`.
6. For **Index**, specify `opensearch_dashboards_sample_data_fli*` (`kibana_sample_data_fli*` on Elasticsearch domains).
7. For **Index permissions**, choose **read**.
8. For **Document level security**, specify the following query:

```
{
  "match": {
    "FlightDelay": true
  }
}
```

9. For field-level security, choose **Exclude** and specify `FlightNum`.
10. For **Anonymization**, specify `Dest`.
11. Choose **Create**.
12. Choose **Mapped users**, **Manage mapping**. Add the Amazon Resource Name (ARN) for `LimitedUserRole` as an external identity and choose **Map**.
13. Return to the list of roles and choose **opensearch_dashboards_user**. Choose **Mapped users**, **Manage mapping**. Add the ARN for `LimitedUserRole` as a backend role and choose **Map**.

Step 5: Test the permissions

When your roles are mapped correctly, you can sign in as the limited user and test the permissions.

1. In a new, private browser window, navigate to the OpenSearch Dashboards URL for the domain, sign in using the `limited-user` credentials, and choose **Explore on my own**.
2. Go to **Dev Tools** and run the default search:

```
GET _search
```

```
{
  "query": {
    "match_all": {}
  }
}
```

Note the permissions error. `limited-user` doesn't have permissions to run cluster-wide searches.

3. Run another search:

```
GET opensearch_dashboards_sample_data_flights/_search
{
  "query": {
    "match_all": {}
  }
}
```

Note that all matching documents have a `FlightDelay` field of `true`, an anonymized `Dest` field, and no `FlightNum` field.

4. In your original browser window, signed in as `master-user`, choose **Dev Tools**, and then perform the same searches. Note the difference in permissions, number of hits, matching documents, and included fields.

Tutorial: Configure a domain with the internal user database and HTTP basic authentication

This tutorial covers another popular fine-grained access control use case: a master user in the internal user database and HTTP basic authentication for OpenSearch Dashboards. The master user can then sign in to OpenSearch Dashboards, create an internal user, map the user to a role, and use fine-grained access control to limit the user's permissions.

You'll complete the following steps in this tutorial:

1. Create a domain with a master user
2. Configure an internal user in OpenSearch Dashboards
3. Map roles in OpenSearch Dashboards
4. Test the permissions

Step 1: Create a domain

Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home/ and create a domain with the following settings:

- OpenSearch 1.0 or later, or Elasticsearch 7.9 or later
- Public access
- Fine-grained access control with a master user in the internal user database (`TheMasterUser` for the rest of this tutorial)
- Amazon Cognito authentication for Dashboards *disabled*
- The following access policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{account-id}:root"
      },
      "Action": [
        "es:ESHttp*"
      ],
      "Resource": "arn:aws:es:{region}:{account-id}:domain/{domain-name}/*"
    }
  ]
}
```

- HTTPS required for all traffic to the domain
- Node-to-node encryption
- Encryption of data at rest

Step 2: Create an internal user in OpenSearch Dashboards

Now that you have a domain, you can sign in to OpenSearch Dashboards and create an internal user.

1. Go back to the OpenSearch Service console and navigate to the OpenSearch Dashboards URL for the domain you created. The URL follows this format: *domain-endpoint*/_dashboards/.
2. Sign in with the `TheMasterUser`.
3. Choose **Add sample data** and add the sample flight data.
4. In the left navigation pane, choose **Security**, **Internal users**, **Create internal user**.
5. Name the user `new-user` and specify a password. Then choose **Create**.

Step 3: Map roles in OpenSearch Dashboards

Now that your user is configured, you can map your user to a role.

1. Stay in the **Security** section of OpenSearch Dashboards and choose **Roles**, **Create role**.
2. Name the role `new-role`.
3. For **Index**, specify `opensearch_dashboards_sample_data_fli*` (`kibana_sample_data_fli*` on Elasticsearch domains) for the index pattern.
4. For the action group, choose **read**.
5. For **Document level security**, specify the following query:

```
{
  "match": {
    "FlightDelay": true
  }
}
```

6. For field-level security, choose **Exclude** and specify `FlightNum`.
7. For **Anonymization**, specify `Dest`.
8. Choose **Create**.
9. Choose **Mapped users**, **Manage mapping**. Then add `new-user` to **Users** and choose **Map**.
10. Return to the list of roles and choose **opensearch_dashboards_user**. Choose **Mapped users**, **Manage mapping**. Then add `new-user` to **Users** and choose **Map**.

Step 4: Test the permissions

When your roles are mapped correctly, you can sign in as the limited user and test the permissions.

1. In a new, private browser window, navigate to the OpenSearch Dashboards URL for the domain, sign in using the `new-user` credentials, and choose **Explore on my own**.
2. Go to **Dev Tools** and run the default search:

   ```
   GET _search
   {
     "query": {
       "match_all": {}
     }
   }
   ```

   Note the permissions error. `new-user` doesn't have permissions to run cluster-wide searches.
3. Run another search:

   ```
   GET dashboards_sample_data_flights/_search
   {
     "query": {
       "match_all": {}
     }
   }
   ```

   Note that all matching documents have a `FlightDelay` field of `true`, an anonymized `Dest` field, and no `FlightNum` field.
4. In your original browser window, signed in as `TheMasterUser`, choose **Dev Tools** and perform the same searches. Note the difference in permissions, number of hits, matching documents, and included fields.

Compliance validation for Amazon OpenSearch Service

Third-party auditors assess the security and compliance of Amazon OpenSearch Service as part of multiple AWS compliance programs. These programs include SOC, PCI, and HIPAA.

If you have compliance requirements, consider using any version of OpenSearch or Elasticsearch 6.0 or later. Earlier versions of Elasticsearch don't offer a combination of encryption of data at rest and node-to-node encryption and are unlikely to meet your needs. You might also consider using any version of OpenSearch or Elasticsearch 6.7 or later if fine-grained access control is important to your use case. Regardless, choosing a particular OpenSearch or Elasticsearch version when you create a domain does not guarantee compliance.

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security Compliance & Governance – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- HIPAA Eligible Services Reference – Lists HIPAA eligible services. Not all AWS services are HIPAA eligible.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in Amazon OpenSearch Service

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, OpenSearch Service offers several features to help support your data resiliency and backup needs:

- Multi-AZ domains and replica shards
- Automated and manual snapshots

JWT authentication and authorization for Amazon OpenSearch Service

Amazon OpenSearch Service now allows you to use JSON Web Tokens (JWTs) for authentication and authorization. JWTs are JSON-based access tokens used to grant single sign-on (SSO) access. You can use JWTs in OpenSearch Service to create single sign-on tokens to validate requests to your OpenSearch Service domain. To use JWTs, you must have fine-grained access control enabled, and you must provide a valid RSA or ECDSA PEM formatted public key. For more information on fine-grained access control, see Fine-grained access control in Amazon OpenSearch Service.

You can configure JSON Web Tokens by using the OpenSearch Service console, the AWS Command Line Interface (AWS CLI), or the AWS SDKs.

Considerations

Before you use JWTs with Amazon OpenSearch Service you must consider the following:

- Due to the size of RSA public keys in PEM formatting, we recommend using the AWS console to configure JWT authentication and authorization.
- You must provide valid users and roles when specifying the subjects and roles fields for your JWTs, otherwise, requests will be denied.

- OpenSearch 2.11 is the earliest compatible version that can be used for JWT authentication.

Modifying the domain access policy

Before you can configure your domain to use JWT authentication and authorization, you must update your domain access policy to allow JWT users to access the domain. Otherwise, all incoming JWT authorized requests are denied. The recommended domain access policy to provide full access to the sub resources (/*) is:

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "es:ESHttp*",
      "Resource": "domain-arn/*"
    }
  ]
}
```

Configuring JWT authentication and authorization

You can enable JWT authentication and authorization during the domain creation process or by updating an existing domain. The set-up steps vary slightly depending on which option you choose.

The following steps explain how to configure an existing domain for JWT authentication and authorization in the OpenSearch Service console:

1. Under **Domain configuration**, navigate to **JWT authentication and authorization for OpenSearch**, select **Enable JWT authentication and authorization**.
2. Configure the public key to use for your domain. To do this, you can either upload a PEM file, containing a public key, or manually enter it.

> **Note**
>
> If the uploaded or entered key is not valid, a warning will appear above the text box specifying the issue.

3. (Optional) Under Additional settings, you can configure the following optional fields

   - **Subject key** — you can leave this field empty to use the default `sub` key for your JWTs.
   - **Roles key** — you can leave this field empty to use the default `roles` key for your JWTs.

   After you've made your changes, save your domain.

Using a JWT to send a test request

After creating a new JWT with a specified a subject and role pair, you can send a test request. To do this, use the private key to sign your request through the tool that created the JWT. OpenSearch Service is able to validate the incoming request by verifying this signature.

> **Note**
>
> If you specified a custom subject key or roles key for your JWT, you must use the correct claims names for your JWT.

The following is an example of how to use a JWT token to access OpenSearch Service through your domain's search endpoint:

```
curl -XGET "$search_endpoint" -H "Authorization: Bearer <JWT>"
```

Configuring JWT authentication and authorization (AWS CLI)

The following AWS CLI command enables JWT authentication and authorization for OpenSearch provided that the domain exists:

```
aws opensearch update-domain-config --domain-name <your_domain_name> --advanced-
security-options '{"JWTOptions":{"Enabled":true, "PublicKey": "<your_public_key>",
 "SubjectKey": "<your_subject_key>", "RolesKey": "<your_roles_key>"}}'
```

Configuring JWT authentication and authorization (configuration via API)

The following request to the configuration API enables JWT authentication and authorization for OpenSearch on an existing domain:

```
POST https://es.us-east-1.amazonaws.com/2021-01-01/opensearch/domain/my-domain/config
{
  "AdvancedSecurityOptions": {
    "JWTOptions": {
      "Enabled": true,
      "PublicKey": "public-key",
      "RolesKey": "optional-roles-key",
      "SubjectKey": "optional-subject-key"
   }
  }
}
```

Generating a key pair

In order to configure JWTs for your OpenSearch domain, you will need to provide a public key in Privacy-Enhanced Mail (PEM) format. Amazon OpenSearch Service currently supports two asymetric encryption algorithms when using JWTs: RSA and ECDSA.

To create an RSA key pair using the common openssl library, follow these steps:

1. `openssl genrsa -out privatekey.pem 2048`
2. `openssl rsa -in privatekey.pem -pubout -out publickey.pem`

In this example, the `publickey.pem` file contains the public key for use with Amazon OpenSearch Service, while `privatekey.pem` contains the private for signing the JWTs sent to the service. Additionally, you have the option to convert the private key into the commonly used `pkcs8` format if you need that to generate your JWTs.

If you use the upload button to add a PEM file directly to the console, the file must have a `.pem` extension, other file extensions such as `.crt`,`.cert`, or `.key` are not supported at this time.

Infrastructure security in Amazon OpenSearch Service

As a managed service, Amazon OpenSearch Service is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud

Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access OpenSearch Service through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

You use AWS published API calls to access the OpenSearch Service configuration API through the network. To configure the minimum required TLS version to accept, specify the `TLSSecurityPolicy` value in the domain endpoint options:

```
aws opensearch update-domain-config --domain-name my-domain --domain-endpoint-options
 '{"TLSSecurityPolicy": "Policy-Min-TLS-1-2-2019-07"}'
```

For details, see the AWS CLI command reference.

Depending on your domain configuration, you might also need to sign requests to the OpenSearch APIs. For more information, see the section called “Making and signing OpenSearch Service requests”.

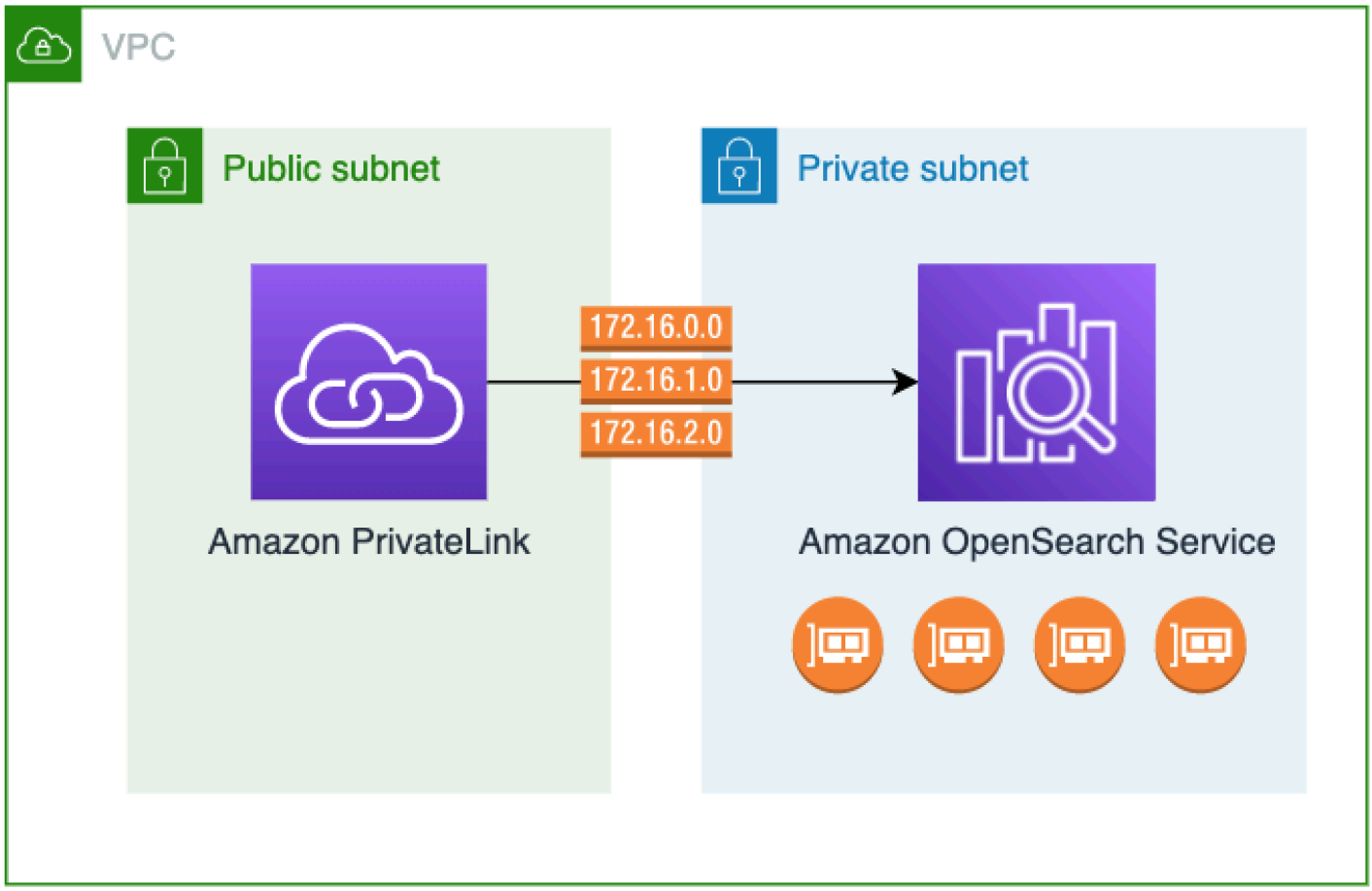
OpenSearch Service supports public access domains, which can receive requests from any internet-connected device, and VPC access domains, which are isolated from the public internet.

Access Amazon OpenSearch Service using an OpenSearch Service-managed VPC endpoint (AWS PrivateLink)

You can access an Amazon OpenSearch Service domain by setting up an OpenSearch Service-managed VPC endpoint (powered by AWS PrivateLink). These endpoints create a private connection between your VPC and Amazon OpenSearch Service. You can access OpenSearch Service VPC domains as if they were in your VPC, without the use of an internet gateway, NAT

device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to access OpenSearch Service.

You can configure OpenSearch Service domains to expose additional endpoints running on public or private subnets within the same VPC, different VPC, or different AWS accounts. This enables you to add an additional layer of security to access your domains regardless of where they run, with no infrastructure to manage. The following diagram illustrates OpenSearch Service-managed VPC endpoints within the same VPC:



You establish this private connection by creating an OpenSearch Service-managed *interface VPC endpoint*, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you enable for the interface VPC endpoint. These are service-managed network interfaces that serve as the entry point for traffic destined for OpenSearch Service. Standard AWS PrivateLink interface endpoint pricing applies for OpenSearch Service-managed VPC endpoints billed under AWS PrivateLink.

You can create VPC endpoints for domains running all versions of OpenSearch and legacy Elasticsearch. For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Considerations and limitations for OpenSearch Service

Before you set up an interface VPC endpoint for OpenSearch Service, review Considerations in the *AWS PrivateLink Guide*.

When using OpenSearch Service-managed VPC endpoints, consider the following:

- You can only use interface VPC endpoints to connect to VPC domains. Public domains aren't supported.
- VPC endpoints can only connect to domains within the same AWS Region.
- HTTPS is the only supported protocol for VPC endpoints. HTTP is not allowed.
- OpenSearch Service supports making calls to all of the supported OpenSearch API operations through an interface VPC endpoint.
- You can configure a maximum of 50 endpoints per account, and a maximum of 10 endpoints per domain. A single domain can have a maximum of 10 authorized principals.
- You currently can't use AWS CloudFormation to create interface VPC endpoints.
- You can only create interface VPC endpoints through the OpenSearch Service console or using the OpenSearch Service API. You can't create interface VPC endpoints for OpenSearch Service using the Amazon VPC console.
- OpenSearch Service-managed VPC endpoints aren't accessible from the internet. An OpenSearch Service-managed VPC endpoint is accessible only within the VPC where the endpoint is provisioned or any VPCs peered with the VPC where the endpoint is provisioned, as permitted by the route tables and security groups.
- VPC endpoint policies are not supported for OpenSearch Service. You can associate a security group with the endpoint network interfaces to control traffic to OpenSearch Service through the interface VPC endpoint.
- Your service-linked role must be in the same AWS account that you use to create the VPC endpoint.
- To create, update, and delete the OpenSearch Service VPC endpoint, you must have the following Amazon EC2 permissions in addition to your Amazon OpenSearch Service permissions:
  - `ec2:CreateVpcEndpoint`

- `ec2:DescribeVpcEndpoints`
- `ec2:ModifyVpcEndpoint`
- `ec2:DeleteVpcEndpoints`
- `ec2:CreateTags`
- `ec2:DescribeTags`
- `ec2:DescribeSubnets`
- `ec2:DescribeSecurityGroups`
- `ec2:DescribeVpcs`

> **Note**
>
> Currently, you can't limit VPC endpoint creation to OpenSearch Service. We're working to make this possible in a future update.

Provide access to a domain

If the VPC that you want to access your domain is in another AWS account, you need to authorize it from the owner's account before you can create an interface VPC endpoint.

To allow a VPC in another AWS account to access your domain

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home/.
2. In the navigation pane, choose **Domains** and open the domain that you want to provide access to.
3. Go to the **VPC endpoints** tab, which shows the accounts and corresponding VPCs that have access to your domain.
4. Choose **Authorize principal**.
5. Enter the AWS account ID of the account that will access your domain. This step authorizes the specified account to create VPC endpoints against the domain.
6. Choose **Authorize**.

Create an interface VPC endpoint for a VPC domain

You can create an interface VPC endpoint for OpenSearch Service using either the OpenSearch Service console or the AWS Command Line Interface (AWS CLI).

To create an interface VPC endpoint for an OpenSearch Service domain

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home/.
2. In the left navigation pane, choose **VPC endpoints**.
3. Choose **Create endpoint**.
4. Select whether to connect a domain in the current AWS account or another AWS account.
5. Select the domain that you connect to with this endpoint. If the domain is in the current AWS account, use the dropdown to choose the domain. If the domain is in a different account, enter the Amazon Resource Name (ARN) of the domain to connect to. To choose a domain in a different account, the owner needs to provide you access to the domain.
6. For **VPC**, select the VPC from which you'll access OpenSearch Service.
7. For **Subnets**, select one or more subnets from which you'll access OpenSearch Service.
8. For **Security groups**, select the security groups to associate with the endpoint network interfaces. This is a critical step in which you limit what ports, protocols, and sources for inbound traffic that you're authorizing into your endpoint. The security group rules must allow the resources that will use the VPC endpoint to communicate with OpenSearch Service to communicate with the endpoint network interface.
9. Choose **Create endpoint**. The endpoint should be active within 2-5 minutes.

Working with OpenSearch Service-managed VPC endpoints using the configuration API

Use the following API operations to create and manage OpenSearch Service-managed VPC endpoints.

- CreateVpcEndpoint
- ListVpcEndpoints
- UpdateVpcEndpoint
- DeleteVpcEndpoint

Use the following API operations to manage endpoint access to VPC domains:

- AuthorizeVpcEndpointAccess
- ListVpcEndpointAccess
- ListVpcEndpointsForDomain
- RevokeVpcEndpointAccess

SAML authentication for OpenSearch Dashboards

SAML authentication for OpenSearch Dashboards lets you use your existing identity provider to offer single sign-on (SSO) for Dashboards on Amazon OpenSearch Service domains running OpenSearch or Elasticsearch 6.7 or later. To use SAML authentication, you must enable fine-grained access control.

Rather than authenticating through Amazon Cognito or the internal user database, SAML authentication for OpenSearch Dashboards lets you use third-party identity providers to log in to Dashboards, manage fine-grained access control, search your data, and build visualizations. OpenSearch Service supports providers that use the SAML 2.0 standard, such as Okta, Keycloak, Active Directory Federation Services (ADFS), Auth0, and AWS IAM Identity Center.

SAML authentication for Dashboards is only for accessing OpenSearch Dashboards through a web browser. Your SAML credentials do *not* let you make direct HTTP requests to the OpenSearch or Dashboards APIs.

SAML configuration overview

This documentation assumes that you have an existing identity provider and some familiarity with it. We can't provide detailed configuration steps for your exact provider, only for your OpenSearch Service domain.

The OpenSearch Dashboards login flow can take one of two forms:

- **Service provider (SP) initiated**: You navigate to Dashboards (for example, `https://my-domain.us-east-1.es.amazonaws.com/_dashboards`), which redirects you to the login screen. After you log in, the identity provider redirects you to Dashboards.
- **Identity provider (IdP) initiated**: You navigate to your identity provider, log in, and choose OpenSearch Dashboards from an application directory.

OpenSearch Service provides two single sign-on URLs, SP-initiated and IdP-initiated, but you only need the one that matches your desired OpenSearch Dashboards login flow.

Regardless of which authentication type you use, the goal is to log in through your identity provider and receive a SAML assertion that contains your username (required) and any backend roles (optional, but recommended). This information allows fine-grained access control to assign permissions to SAML users. In external identity providers, backend roles are typically called "roles" or "groups."

Considerations

Consider the following when you configure SAML authentication:

- Due to the size of the IdP metadata file, we highly recommend using the AWS console to configure SAML authentication.
- Domains only support one Dashboards authentication method at a time. If you have Amazon Cognito authentication for OpenSearch Dashboards enabled, you must disable it before you can enable SAML authentication.
- If you use a network load balancer with SAML, you must first create a custom endpoint. For more information, see ???.
- Service Control Policies (SCP) will not be applicable or evaluated in case of non-IAM identities (like SAML in Amazon OpenSearch Serverless & SAML and basic internal user authorization for Amazon OpenSearch Service).

SAML authentication for VPC domains

SAML doesn't require direct communication between your identity provider and your service provider. Therefore, even if your OpenSearch domain is hosted within a private VPC, you can still use SAML as long as your browser can communicate with both your OpenSearch cluster and your identity provider. Your browser essentially acts as the intermediary between your identity provider and your service provider. For a useful diagram that explains the SAML authentication flow, see the Okta documentation.

Modifying the domain access policy

Before you configure SAML authentication, you must update the domain access policy to allow SAML users to access the domain. Otherwise, you'll see access denied errors.

We recommend the following domain access policy, which provides full access to the subresources (/*) on the domain:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "es:ESHttp*",
      "Resource": "domain-arn/*"
    }
  ]
}
```

To make the policy more restrictive, you can add an IP address condition to the policy. This conditions limits access to only the specified IP address range or subnet. For example, the following policy allows access only from the 192.0.2.0/24 subnet:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "es:ESHttp*"
      ],
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "192.0.2.0/24"
          ]
        }
      },
      "Resource": "domain-arn/*"
    }
  ]
```

```
}
```

> **Note**
>
> An open domain access policy requires fine-grained access control to be enabled on your domain, otherwise you see the following error:
> `To protect domains with public access, a restrictive policy or fine-grained access control is required.`
> If you have a master user or internal user configured with a robust password, keeping the policy open while using fine-grained access control might be acceptable from a security standpoint. For more information, see ???.

Configuring SP- or IdP-initiated authentication

These steps explain how to enable SAML authentication with SP-initiated *or* IdP-initiated authentication for OpenSearch Dashboards. For the extra step required to enable both, see Configuring both SP- and IdP-initiated authentication.

Step 1: Enable SAML authentication

You can enable SAML authentication either during domain creation, or by choosing **Actions**, **Edit security configuration** on an existing domain. The following steps vary slightly depending on which one you choose.

Within the domain configuration, under **SAML authentication for OpenSearch Dashboards/ Kibana**, select **Enable SAML authentication**.

Step 2: Configure your identity provider

Perform the following steps depending on when you're configuring SAML authentication.

If you're creating a new domain

If you're in the process of creating a new domain, OpenSearch Service can't yet generate a service provider entity ID or SSO URLs. Your identity provider requires these values in order to properly enable SAML authentication, but they can only be generated after the domain is created. To work around this interdependency during domain creation, you can provide temporary values into your

IdP configuration to generate the required metadata and then update them once your domain is active.

If you're using a custom endpoint, you can infer what the URLs will be. For example, if your custom endpoint is `www.custom-endpoint.com`, the service provider entity ID will be `www.custom-endpoint.com`, the IdP-initiated SSO URL will be `www.custom-endpoint.com/_dashboards/_opendistro/_security/saml/acs/idpinitiated`, and the SP-initiated SSO URL will be `www.custom-endpoint.com/_dashboards/_opendistro/_security/saml/acs`. You can use the values to configure your identity provider before the domain is created. See the next section for examples.

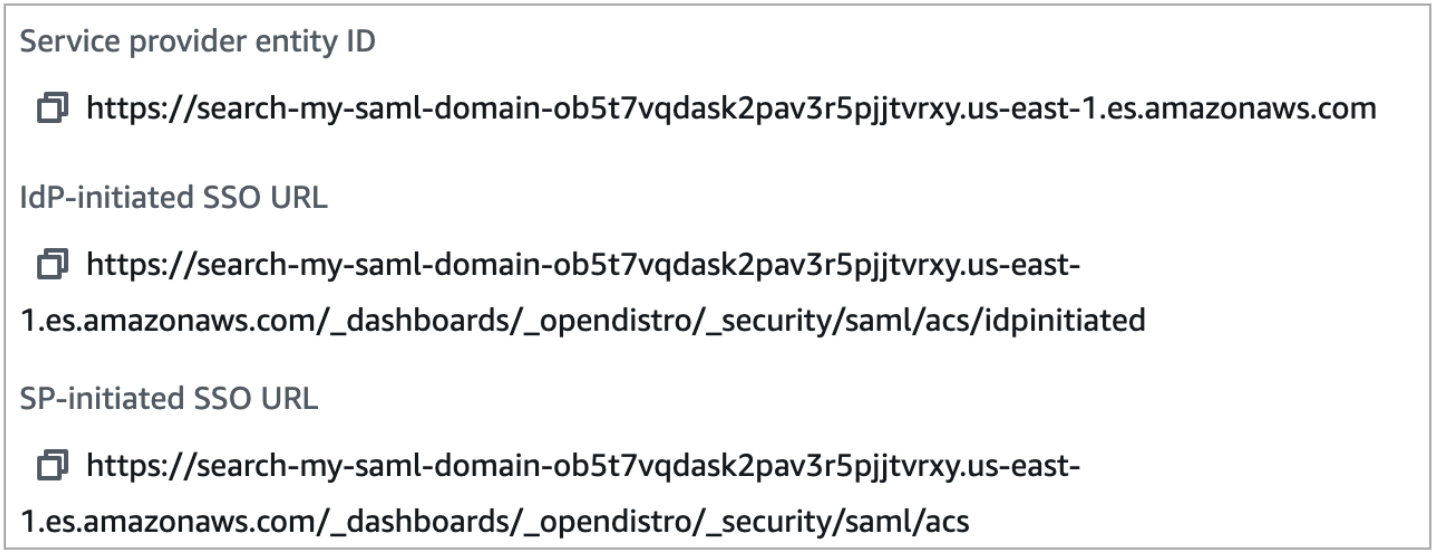
> **Note**
>
> You can not sign in with a dual stack endpoint because the FQDN of a HTTP request is different than the FQDN of a SAML request. An OpenSearch administrator will need to set up a custom endpoint and set the CNAME value to dual stack endpoint if you would like to sign in using a dual stack endpoint.

If you're not using a custom endpoint, you can enter *temporary* values into your IdP to generate the required metadata, and then update them later after the domain is active.

For example, within Okta, you can enter `https://temp-endpoint.amazonaws.com` into the **Single sign on URL** and **Audience URI (SP Entity ID)** fields, which enables you to generate the metadata. Then, after the domain is active, you can retrieve the correct values from OpenSearch Service and update them in Okta. For instructions, see the section called “Step 6: Update your IdP URLs”.

If you're editing an existing domain

If you're enabling SAML authentication on an existing domain, copy the service provider entity ID and one of the SSO URLs. For guidance on which URL to use, see the section called “SAML configuration overview”.



Use the values to configure your identity provider. This is the most complex part of the process, and unfortunately, terminology and steps vary wildly by provider. Consult your provider's documentation.

In Okta, for example, you create a SAML 2.0 web application. For **Single sign on URL**, specify the SSO URL. For **Audience URI (SP Entity ID)**, specify the SP entity ID.

Rather than users and backend roles, Okta has users and groups. For **Group Attribute Statements**, we recommend that you add `role` to the **Name** field and the regular expression `.+` to the **Filter** field. This statement tells the Okta identity provider to include all user groups under the `role` field of the SAML assertion after a user authenticates.

In IAM Identity Center, you specify the SP entity ID as the **Application SAML audience**. You also need to specify the following attribute mappings: `Subject=${user:subject}:format=unspecified` and `Role=${user:groups}:format=uri`.

In Auth0, you create a regular web application and enable the SAML 2.0 add-on. In Keycloak, you create a client.

Step 3: Import IdP metadata

After you configure your identity provider, it generates an IdP metadata file. This XML file contains information about the provider, such as a TLS certificate, single sign-on endpoints, and the identity provider's entity ID.

Copy the contents of the IdP metadata file and paste it into the **Metadata from IdP** field in the OpenSearch Service console. Alternately, choose **Import from XML file** and upload the file. The metadata file should look something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<md:EntityDescriptor entityID="entity-id"
 xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata">
  <md:IDPSSODescriptor WantAuthnRequestsSigned="false"
 protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <md:KeyDescriptor use="signing">
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:X509Data>
          <ds:X509Certificate>tls-certificate</ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
    </md:KeyDescriptor>
    <md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</
md:NameIDFormat>
    <md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress</
md:NameIDFormat>
    <md:SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
 Location="idp-sso-url"/>
    <md:SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
Redirect" Location="idp-sso-url"/>
  </md:IDPSSODescriptor>
</md:EntityDescriptor>
```

Step 4: Configure SAML fields

After you input your IdP metadata, configure the following additional fields within the OpenSearch Service console:

- **IdP entity ID** – Copy the value of the `entityID` property from your metadata file and paste it into this field. Many identity providers also display this value as part of a post-configuration summary. Some providers call it the "issuer".
- **SAML master username** and **SAML master backend role** – The user and/or backend role that you specify receive full permissions to the cluster, equivalent to a new master user, but can only use those permissions within OpenSearch Dashboards.

  In Okta, for example, you might have a user `jdoe` who belongs to the group `admins`. If you add `jdoe` to the **SAML master username** field, only that user receives full permissions. If you

add `admins` to the SAML master backend role field, any user that belongs to the `admins` group receives full permissions.

> **Note**
>
> The contents of the SAML assertion must exactly match the strings that you use for the SAML master username and SAML master role. Some identity providers add a prefix before their usernames, which can cause a hard-to-diagnose mismatch. In the identity provider user interface, you might see `jdoe`, but the SAML assertion might contain `auth0|jdoe`. Always use the string from the SAML assertion.

Many identity providers let you view a sample assertion during the configuration process, and tools like SAML-tracer can help you examine and troubleshoot the contents of real assertions. Assertions look something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<saml2:Assertion ID="id67229299299259351343340162"
 IssueInstant="2020-09-22T22:03:08.633Z" Version="2.0"
  xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
  <saml2:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">idp-issuer</
saml2:Issuer>
  <saml2:Subject>
    <saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified">username</saml2:NameID>
    <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
      <saml2:SubjectConfirmationData NotOnOrAfter="2020-09-22T22:08:08.816Z"
 Recipient="domain-endpoint/_dashboards/_opendistro/_security/saml/acs"/>
    </saml2:SubjectConfirmation>
  </saml2:Subject>
  <saml2:Conditions NotBefore="2020-09-22T21:58:08.816Z"
 NotOnOrAfter="2020-09-22T22:08:08.816Z">
    <saml2:AudienceRestriction>
      <saml2:Audience>domain-endpoint</saml2:Audience>
    </saml2:AudienceRestriction>
  </saml2:Conditions>
  <saml2:AuthnStatement AuthnInstant="2020-09-22T19:54:37.274Z">
    <saml2:AuthnContext>

 <saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport<
saml2:AuthnContextClassRef>
```

```
    </saml2:AuthnContext>
  </saml2:AuthnStatement>
  <saml2:AttributeStatement>
    <saml2:Attribute Name="role" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:unspecified">
      <saml2:AttributeValue
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:type="xs:string">GroupName Match Matches regex ".+" (case-sensitive)
      </saml2:AttributeValue>
    </saml2:Attribute>
  </saml2:AttributeStatement>
</saml2:Assertion>
```

Step 5: (Optional) Configure additional settings

Under **Additional settings**, configure the following optional fields:

- **Subject key** – You can leave this field empty to use the `NameID` element of the SAML assertion for the username. If your assertion doesn't use this standard element and instead includes the username as a custom attribute, specify that attribute here.
- **Roles key** – If you want to use backend roles (recommended), specify an attribute from the assertion in this field, such as `role` or `group`. This is another situation in which tools like SAML-tracer can help.
- **Session time to live** – By default, OpenSearch Dashboards logs users out after 24 hours. You can configure this value to any number between 60 and 1,440 (24 hours) by specifying a new value.

After you're satisfied with your configuration, save the domain.

Step 6: Update your IdP URLs

If you enabled SAML authentication while creating a domain, you had to specify temporary URLs within your IdP in order to generate the XML metadata file. After the domain status changes to `Active`, you can get the correct URLs and modify your IdP.

To retrieve the URLs, select the domain and choose **Actions**, **Edit security configuration**. Under **SAML authentication for OpenSearch Dashboards/Kibana**, you can find the correct service provider entity ID and SSO URLs. Copy the values and use them to configure your identity provider, replacing the temporary URLs that you provided in step 2.

Step 7: Map SAML users to roles

Once your domain status is Active and your IdP is configured correctly, navigate to OpenSearch Dashboards.

- If you chose the SP-initiated URL, navigate to *domain-endpoint*/_dashboards. To log in to a specific tenant directly, you can append ?security_tenant=*tenant-name* to the URL.
- If you chose the IdP-initiated URL, navigate to your identity provider's application directory.

In both cases, log in as either the SAML master user or a user who belongs to the SAML master backend role. To continue the example from step 7, log in as either jdoe or a member of the admins group.

After OpenSearch Dashboards loads, choose **Security**, **Roles**. Then, map roles to allow other users to access OpenSearch Dashboards.

For example, you might map your trusted colleague jroe to the all_access and security_manager roles. You might also map the backend role analysts to the readall and opensearch_dashboards_user roles.

If you prefer to use the API rather than OpenSearch Dashboards, see the following sample request:

```
PATCH _plugins/_security/api/rolesmapping
[
  {
    "op": "add", "path": "/security_manager", "value": { "users": ["master-user",
 "jdoe", "jroe"], "backend_roles": ["admins"] }
  },
  {
    "op": "add", "path": "/all_access", "value": { "users": ["master-user", "jdoe",
 "jroe"], "backend_roles": ["admins"] }
  },
  {
    "op": "add", "path": "/readall", "value": { "backend_roles": ["analysts"] }
  },
  {
    "op": "add", "path": "/opensearch_dashboards_user", "value": { "backend_roles":
 ["analysts"] }
  }
]
```

Configuring both SP- and IdP-initiated authentication

If you want to configure both SP- and IdP-initiated authentication, you must do so through your identity provider. For example, in Okta, you can perform the following steps:

1. Within your SAML application, go to **General**, **SAML settings**.
2. For the **Single sign on URL**, provide your *IdP*-initiated SSO URL. For example, `https://search-domain-hash/_dashboards/_opendistro/_security/saml/acs/idpinitiated`.
3. Enable **Allow this app to request other SSO URLs**.
4. Under **Requestable SSO URLs**, add one or more *SP*-initiated SSO URLs. For example, `https://search-domain-hash/_dashboards/_opendistro/_security/saml/acs`.

Configuring SAML authentication (AWS CLI)

The following AWS CLI command enables SAML authentication for OpenSearch Dashboards on an existing domain:

```
aws opensearch update-domain-config \
  --domain-name my-domain \
  --advanced-security-options '{"SAMLOptions":{"Enabled":true,"MasterUserName":"my-idp-user","MasterBackendRole":"my-idp-group-or-role","Idp":{"EntityId":"entity-id","MetadataContent":"metadata-content-with-quotes-escaped"},"RolesKey":"optional-roles-key","SessionTimeoutMinutes":180,"SubjectKey":"optional-subject-key"}}'
```

You must escape all quotes and newline characters in the metadata XML. For example, use `<KeyDescriptor use=\"signing\">\n` instead of `<KeyDescriptor use="signing">` and a line break. For detailed information about using the AWS CLI, see the AWS CLI Command Reference.

Configuring SAML authentication (configuration API)

The following request to the configuration API enables SAML authentication for OpenSearch Dashboards on an existing domain:

```
POST https://es.us-east-1.amazonaws.com/2021-01-01/opensearch/domain/my-domain/config
{
  "AdvancedSecurityOptions": {
    "SAMLOptions": {
```

```
      "Enabled": true,
      "MasterUserName": "my-idp-user",
      "MasterBackendRole": "my-idp-group-or-role",
      "Idp": {
        "EntityId": "entity-id",
        "MetadataContent": "metadata-content-with-quotes-escaped"
      },
      "RolesKey": "optional-roles-key",
      "SessionTimeoutMinutes": 180,
      "SubjectKey": "optional-subject-key"
    }
  }
}
```

You must escape all quotes and newline characters in the metadata XML. For example, use `<KeyDescriptor use=\"signing\">\n` instead of `<KeyDescriptor use="signing">` and a line break. For detailed information about using the configuration API, see the OpenSearch Service API reference.

SAML troubleshooting

Error	Details
Your request: '*/some/path* ' is not allowed.	Verify that you provided the correct SSO URL (step 3) to your identity provider.
Please provide valid identity provider metadata document to enable SAML.	Your IdP metadata file does not conform to the SAML 2.0 standard. Check for errors using a validatio n tool.
SAML configuration options aren't visible in the console.	Update to the latest service software.
SAML configuration error: Something went wrong while retrieving the SAML configuration, please check your settings.	This generic error can occur for many reasons. • Check that you provided your identity provider with the correct SP entity ID and SSO URL. • Regenerate the IdP metadata file, and verify the IdP entity ID. Add any updated metadata in the AWS console.

Error	Details
	• Verify that your domain access policy allows access to OpenSearch Dashboards and `_plugins/_security/*` . In general, we recommend an open access policy for domains that use fine-grai ned access control. • Consult your identity provider's documentation for steps on configuring SAML.
Missing role: No roles available for this user, please contact your system administrator.	You successfully authenticated, but the username and any backend roles from the SAML assertion are not mapped to any roles and thus have no permissio ns. These mappings are case-sensitive. Your system administrator can verify the contents of your SAML assertion using a tool like SAML-trac er, and then check your role mapping using the following request: `GET _plugins/_security/api/rolesmapping`
Your browser continuously redirects or receives HTTP 500 errors when trying to access OpenSearch Dashboards.	These errors can occur if your SAML assertion contains a large number of roles totaling approxima tely 1,500 characters. For example, if you pass 80 roles, the average length of which is 20 characters, you might exceed the size limit for cookies in your web browser. Starting with OpenSearch version 2.7, SAML assertion supports roles up to 5000 characters.
You can't log out of ADFS.	ADFS requires all logout request to be signed, which OpenSearch Service doesn't support. Remove `<SingleLogoutService />` from the IdP metadata file to force OpenSearch Service to use its own internal logout mechanism.

Error	Details
`Could not find entity descriptor for __PATH__.`	The entity ID of the IdP provided in the metadata XML to OpenSearch Service is different than the one in the SAML response. To fix this, make sure that they match. Enable **CW Application Error logs** on your domain to find the error message to debug the SAML integration issue.
`Signature validation failed. SAML response rejected.`	OpenSearch Service is unable to verify the signature in the SAML response using the certificate of the IdP provided in metadata XML. This could either be a manual error, or your IdP has rotated its certifica te. Update the latest certificate from your IdP in the metadata XML provided to OpenSearch Service through the AWS Management Console.
`__PATH__ is not a valid audience for this response.`	The audience field in the SAML response doesn't match the domain endpoint. To fix this error, update the SP audience field to match your domain endpoint. If you've enabled custom endpoints, the audience field should match your custom endpoint. Enable **CW Application Error logs** on your domain to find the error message to debug the SAML integrati on issue.
Your browser receives a HTTP 400 error with `Invalid Request Id` in the response.	This error generally happens if you've configured the IdP-initiated URL with the format *<DashboardsURL>* `/_opendistro/_security/saml/acs`. Instead, configure the URL with the format *<DashboardsURL>* `/_opendistro/_security/saml/acs/idpinitiated` .

Error	Details
The response was received at __PATH__ instead of __PATH__.	The destination field in SAML response doesn't match one of the following URL formats: • *<DashboardsURL>* /_opendistro/_security/saml/acs • *<DashboardsURL>* /_opendistro/_security/saml/acs/idpinitiated . Depending on the login flow you use (SP-initiated or IdP-initiated), enter in a destination field that matches one of the OpenSearch URLs.
The response has an InResponseTo attribute, while no InResponseTo was expected.	You're using the IdP-initiated URL for an SP-initiated login flow. Use the SP-initiated URL instead.

Disabling SAML authentication

To disable SAML authentication for OpenSearch Dashboards (console)

1. Choose the domain, **Actions**, and **Edit security configuration**.
2. Uncheck **Enable SAML authentication**.
3. Choose **Save changes**.
4. After the domain finishes processing, verify the fine-grained access control role mapping with the following request:

   ```
   GET _plugins/_security/api/rolesmapping
   ```

   Disabling SAML authentication for Dashboards does *not* remove the mappings for the SAML master username and/or the SAML master backend role. If you want to remove these mappings, log in to Dashboards using the internal user database (if enabled), or use the API to remove them:

   ```
   PUT _plugins/_security/api/rolesmapping/all_access
   ```

```
{
  "users": [
    "master-user"
  ]
}
```

IAM Identity Center Trusted Identity Propagation Support for Amazon OpenSearch Service

You can now use your centrally configured AWS IAM Identity Center principals (users and groups) via Trusted Identity Propagation to access OpenSearch domains through OpenSearch Service applications . In order to enable IAM Identity Center support for Amazon OpenSearch Service, you will need to enable use of IAM Identity Center To learn more on how to do this, see What is IAM Identity Center?. See How to associate OpenSearch domain as datasource in OpenSearch applications? for details.

You can configure IAM Identity Center by using the OpenSearch Service console, the AWS Command Line Interface (AWS CLI), or the AWS SDKs.

> **Note**
>
> IAM Identity Center principals are not supported through Dashboards (co-located with cluster). They are only supported through Centralized OpenSearch user interface (Dashboards).

Considerations

Before you use IAM Identity Center with Amazon OpenSearch Service you must consider the following:

- IAM Identity Center is enabled in the account.
- The OpenSearch domain version is 1.3 or later.
- Fine Grained Access Control is enabled on the domain.
- Domain should be in the same region as IAM Identity Center instance.
- Domain and OpenSearch application should belong to same AWS account.

Modifying the domain access policy

Before you configure IAM Identity Center, you must update the domain access policy or the permissions of the IAM role configured in OpenSearch applications for Trusted Identity Propagation.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "IAM Role configured in OpenSearch application"
            },
            "Action": "es:ESHttp*",
            "Resource": "domain-arn/*"
        },
        {
        ... // Any other permissions
        }
    ]
}
```

Configuring IAM Identity Center authentication and authorization (Console)

You can enable IAM Identity Center authentication and authorization during the domain creation process or by updating an existing domain. The set-up steps vary slightly depending on which option you choose.

The following steps explain how to configure an existing domain for IAM Identity Center authentication and authorization in the Amazon OpenSearch Service console:

1. Under **Domain configuration**, navigate to **Security Configuration**, choose Edit and navigate to IAM Identity Center Authentication section and select **Enable API access authenticated with IAM Identity Center**.
2. Select the SubjectKey and Roles key as follows.

   - **Subject key** - choose one of UserId (default), UserName and Email to use corresponding attribute as principal accessing the domain.

- **Roles key** - choose one of GroupId (default) and GroupName to use corresponding attribute values as backend role for fine-grained-access-control for all the groups associated with the IdC principal.

After you've made your changes, save your domain.

Configuring Fine Grained Access Control

Once you have enabled IAM Identity Center option on your OpenSearch domain, you can configure access to IAM Identity Center principals by creating role mapping to the **backend role**. The backend role value for the principal is based on the IdC principal's group membership and the RolesKey configuration of GroupId or GroupName.

> **Note**
>
> Amazon OpenSearch Service can support up to **100 groups** for a single user. If you try to use more than the number of allowed instances, you will experience inconsistency with your fine-grained-access-control authorization processing and receive a 403error message.

Configuring IAM Identity Center authentication and authorization (CLI)

```
  aws opensearch update-domain-config \
      --domain-name my-domain \
      --identity-center-options '{"EnabledAPIAccess": true,
 "IdentityCenterInstanceARN": "instance arn",  "SubjectKey": "UserId/UserName/
UserEmail" , "RolesKey": "GroupId/GroupName"}'
```

Disabling IAM Identity Center authentication on the domain

To disable IAM Identity Center on your OpenSearch domain:

1. Choose the domain, **Actions**, and **Edit security configuration**.
2. Uncheck **Enable API access authenticated with IAM Identity Center**.
3. Choose **Save changes**.
4. After the domain finishes processing, remove role mappings added for IdC principals

To disable IAM Identity Center through CLI, you can use following

```
aws opensearch update-domain-config \
    --domain-name my-domain \
    --identity-center-options '{"EnabledAPIAccess": false}'
```

Configuring Amazon Cognito authentication for OpenSearch Dashboards

You can authenticate and protect your Amazon OpenSearch Service default installation of OpenSearch Dashboards using Amazon Cognito. Amazon Cognito authentication is optional and available only for domains using OpenSearch or Elasticsearch 5.1 or later. If you don't configure Amazon Cognito authentication, you can still protect Dashboards using an IP-based access policy and a proxy server, HTTP basic authentication, or SAML.

Much of the authentication process occurs in Amazon Cognito, but this section offers guidelines and requirements for configuring Amazon Cognito resources to work with OpenSearch Service domains. Standard pricing applies to all Amazon Cognito resources.

Tip

The first time you configure a domain to use Amazon Cognito authentication for OpenSearch Dashboards, we recommend using the console. Amazon Cognito resources are extremely customizable, and the console can help you identify and understand the features that matter to you.

Topics

- (Optional) Customizing the sign-in page
- (Optional) Configuring advanced security
- Testing
- Quotas
- Common configuration issues
- Disabling Amazon Cognito authentication for OpenSearch Dashboards
- Deleting domains that use Amazon Cognito authentication for OpenSearch Dashboards

Prerequisites

Before you can configure Amazon Cognito authentication for OpenSearch Dashboards, you must fulfill several prerequisites. The OpenSearch Service console helps streamline the creation of these resources, but understanding the purpose of each resource helps with configuration and troubleshooting. Amazon Cognito authentication for Dashboards requires the following resources:

- Amazon Cognito user pool
- Amazon Cognito identity pool
- IAM role that has the `AmazonOpenSearchServiceCognitoAccess` policy attached (`CognitoAccessForAmazonOpenSearch`)

> **Note**
>
> The user pool and identity pool must be in the same AWS Region. You can use the same user pool, identity pool, and IAM role to add Amazon Cognito authentication for Dashboards to multiple OpenSearch Service domains. To learn more, see the section called "Quotas".

About the user pool

User pools have two main features: create and manage a directory of users, and let users sign up and log in. For instructions to create a user pool, see Create a User Pool in the *Amazon Cognito Developer Guide*.

When you create a user pool to use with OpenSearch Service, consider the following:

- Your Amazon Cognito user pool must have a domain name. OpenSearch Service uses this domain name to redirect users to a login page for accessing Dashboards. Other than a domain name, the user pool doesn't require any non-default configuration.
- You must specify the pool's required standard attributes—attributes like name, birth date, email address, and phone number. You can't change these attributes after you create the user pool, so choose the ones that matter to you at this time.
- While creating your user pool, choose whether users can create their own accounts, the minimum password strength for accounts, and whether to enable multi-factor authentication. If you plan to use an external identity provider, these settings are inconsequential. Technically, you can enable the user pool as an identity provider *and* enable an external identity provider, but most people prefer one or the other.

User pool IDs take the form of *region*_*ID*. If you plan to use the AWS CLI or an AWS SDK to configure OpenSearch Service, make note of the ID.

About the identity pool

Identity pools let you assign temporary, limited-privilege roles to users after they log in. For instructions about creating an identity pool, see Identity Pools in the *Amazon Cognito Developer Guide*. When you create an identity pool to use with OpenSearch Service, consider the following:

- If you use the Amazon Cognito console, you must select the **Enable access to unauthenticated identities** check box to create the identity pool. After you create the identity pool and configure the OpenSearch Service domain, Amazon Cognito disables this setting.
- You don't need to add external identity providers to the identity pool. When you configure OpenSearch Service to use Amazon Cognito authentication, it configures the identity pool to use the user pool that you just created.
- After you create the identity pool, you must choose unauthenticated and authenticated IAM roles. These roles specify the access policies that users have before and after they log in. If you use the Amazon Cognito console, it can create these roles for you. After you create the authenticated role, make note of the ARN, which takes the form of `arn:aws:iam::`*`123456789012`*`:role/Cognito_`*`identitypoolname`*`Auth_Role`.

Identity pool IDs take the form of *region*`:`*ID*`-`*ID*`-`*ID*`-`*ID*`-`*ID*. If you plan to use the AWS CLI or an AWS SDK to configure OpenSearch Service, make note of the ID.

About the CognitoAccessForAmazonOpenSearch role

OpenSearch Service needs permissions to configure the Amazon Cognito user and identity pools and use them for authentication. You can use `AmazonOpenSearchServiceCognitoAccess`, which is an AWS-managed policy, for this purpose. `AmazonESCognitoAccess` is a legacy policy that was replaced by `AmazonOpenSearchServiceCognitoAccess` when the service was renamed to Amazon OpenSearch Service. Both policies provide the minimum Amazon Cognito permissions necessary to enable Cognito authentication. For the policy JSON, see the IAM console.

If you use the console to create or configure your OpenSearch Service domain, it creates an IAM role for you and attaches the `AmazonOpenSearchServiceCognitoAccess` policy (or the `AmazonESCognitoAccess` policy if it's an Elasticsearch domain) to the role. The default name for this role is `CognitoAccessForAmazonOpenSearch`.

The role permissions policies `AmazonOpenSearchServiceCognitoAccess` and `AmazonESCognitoAccess` both allow OpenSearch Service to complete the following actions on all identity and user pools:

- Action: `cognito-idp:DescribeUserPool`
- Action: `cognito-idp:CreateUserPoolClient`
- Action: `cognito-idp:DeleteUserPoolClient`
- Action: `cognito-idp:UpdateUserPoolClient`
- Action: `cognito-idp:DescribeUserPoolClient`
- Action: `cognito-idp:AdminInitiateAuth`
- Action: `cognito-idp:AdminUserGlobalSignOut`
- Action: `cognito-idp:ListUserPoolClients`
- Action: `cognito-identity:DescribeIdentityPool`
- Action: `cognito-identity:SetIdentityPoolRoles`
- Action: `cognito-identity:GetIdentityPoolRoles`

If you use the AWS CLI or one of the AWS SDKs, you must create your own role, attach the policy, and specify the ARN for this role when you configure your OpenSearch Service domain. The role must have the following trust relationship:

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "opensearchservice.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

For instructions, see Creating a Role to Delegate Permissions to an AWS Service and Attaching and Detaching IAM Policies in the *IAM User Guide*.

Configuring a domain to use Amazon Cognito authentication

After you complete the prerequisites, you can configure an OpenSearch Service domain to use Amazon Cognito for Dashboards.

> **Note**
>
> Amazon Cognito is not available in all AWS Regions. For a list of supported Regions, see AWS Regions and Endpoints. You don't need to use the same Region for Amazon Cognito that you use for OpenSearch Service.

Configuring Amazon Cognito authentication (console)

Because it creates the CognitoAccessForAmazonOpenSearch role for you, the console offers the simplest configuration experience. In addition to the standard OpenSearch Service permissions, you need the following set of permissions to use the console to create a domain that uses Amazon Cognito authentication for OpenSearch Dashboards.

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeVpcs",
```

```
        "cognito-identity:ListIdentityPools",
        "cognito-idp:ListUserPools",
        "iam:CreateRole",
        "iam:AttachRolePolicy"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:GetRole",
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::123456789012:role/service-
role/CognitoAccessForAmazonOpenSearch"
    }
  ]
}
```

For instructions to add permissions to an identity (user, user group, or role), see Adding IAM identity permissions (console).

If `CognitoAccessForAmazonOpenSearch` already exists, you need fewer permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeVpcs",
        "cognito-identity:ListIdentityPools",
        "cognito-idp:ListUserPools"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:GetRole",
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::123456789012:role/service-
role/CognitoAccessForAmazonOpenSearch"
```

```
    }
  ]
}
```

To configure Amazon Cognito authentication for Dashboards (console)

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home/.
2. Under **Domains**, select the domain you want to configure.
3. Choose **Actions**, **Edit security configuration**.
4. Select **Enable Amazon Cognito authentication**.
5. For **Region**, select the AWS Region that contains your Amazon Cognito user pool and identity pool.
6. For **Cognito user pool**, select a user pool or create one. For guidance, see the section called "About the user pool".
7. For **Cognito identity pool**, select an identity pool or create one. For guidance, see the section called "About the identity pool".

   > **Note**
   >
   > The **Create user pool** and **Create identity pool** links direct you to the Amazon Cognito console and require you to create these resources manually. The process is not automatic. To learn more, see the section called "Prerequisites".

8. For **IAM role name**, use the default value of `CognitoAccessForAmazonOpenSearch` (recommended) or enter a new name. To learn more about the purpose of this role, see the section called "About the CognitoAccessForAmazonOpenSearch role".
9. Choose **Save changes**.

After your domain finishes processing, see the section called "Allowing the authenticated role" and the section called "Configuring identity providers" for additional configuration steps.

Configuring Amazon Cognito authentication (AWS CLI)

Use the `--cognito-options` parameter to configure your OpenSearch Service domain. The following syntax is used by both the `create-domain` and `update-domain-config` commands:

```
--cognito-options Enabled=true,UserPoolId="user-pool-id",IdentityPoolId="identity-pool-
id",RoleArn="arn:aws:iam::123456789012:role/CognitoAccessForAmazonOpenSearch"
```

Example

The following example creates a domain in the `us-east-1` Region that enables Amazon Cognito authentication for Dashboards using the `CognitoAccessForAmazonOpenSearch` role and provides domain access to `Cognito_Auth_Role`:

```
aws opensearch create-domain --domain-name my-domain --region us-east-1 --access-
policies '{ "Version":"2012-10-17", "Statement":[{"Effect":"Allow","Principal":{"AWS":
 ["arn:aws:iam::123456789012:role/
Cognito_Auth_Role"]},"Action":"es:ESHttp*","Resource":"arn:aws:es:us-
east-1:123456789012:domain/*" }]}' --engine-version "OpenSearch_1.0"
 --cluster-config InstanceType=m4.xlarge.search,InstanceCount=1
 --ebs-options EBSEnabled=true,VolumeSize=10 --cognito-options
 Enabled=true,UserPoolId="us-east-1_123456789",IdentityPoolId="us-
east-1:12345678-1234-1234-1234-123456789012",RoleArn="arn:aws:iam::123456789012:role/
CognitoAccessForAmazonOpenSearch"
```

After your domain finishes processing, see the section called "Allowing the authenticated role" and the section called "Configuring identity providers" for additional configuration steps.

Configuring Amazon Cognito Authentication (AWS SDKs)

The AWS SDKs (except the Android and iOS SDKs) support all the operations that are defined in the Amazon OpenSearch Service API Reference, including the `CognitoOptions` parameter for the `CreateDomain` and `UpdateDomainConfig` operations. For more information about installing and using the AWS SDKs, see AWS Software Development Kits.

After your domain finishes processing, see the section called "Allowing the authenticated role" and the section called "Configuring identity providers" for additional configuration steps.

Allowing the authenticated role

By default, the authenticated IAM role that you configured by following the guidelines in the section called "About the identity pool" does not have the necessary privileges to access OpenSearch Dashboards. You must provide the role with additional permissions.

> **Note**
>
> If you configured fine-grained access control and use an open or IP-based access policy, you can skip this step.

You can include these permissions in an identity-based policy, but unless you want authenticated users to have access to all OpenSearch Service domains, a resource-based policy attached to a single domain is the better approach.

For the `Principal`, specify the ARN of the Cognito authenticated role that you configured with the guidelines in the section called "About the identity pool".

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "AWS":[
               "arn:aws:iam::123456789012:role/Cognito_identitypoolnameAuth_Role"
            ]
         },
         "Action":[
            "es:ESHttp*"
         ],
         "Resource":"arn:aws:es:region:123456789012:domain/domain-name/*"
      }
   ]
}
```

For instructions about adding a resource-based policy to an OpenSearch Service domain, see the section called "Configuring access policies".

Configuring identity providers

When you configure a domain to use Amazon Cognito authentication for Dashboards, OpenSearch Service adds an app client to the user pool and adds the user pool to the identity pool as an authentication provider.

> **Warning**
>
> Don't rename or delete the app client.

Depending on how you configured your user pool, you might need to create user accounts manually, or users might be able to create their own. If these settings are acceptable, you don't need to take further action. Many people, however, prefer to use external identity providers.

To enable a SAML 2.0 identity provider, you must provide a SAML metadata document. To enable social identity providers like Login with Amazon, Facebook, and Google, you must have an app ID and app secret from those providers. You can enable any combination of identity providers.

The easiest way to configure your user pool is to use the Amazon Cognito console. For instructions, see Using Federation from a User Pool and Specifying Identity Provider Settings for Your User Pool App in the *Amazon Cognito Developer Guide*.

(Optional) Configuring granular access

You might have noticed that the default identity pool settings assign every user who logs in the same IAM role (`Cognito_`*`identitypool`*`Auth_Role`), which means that every user can access the same AWS resources. If you want to use fine-grained access control with Amazon Cognito—for example, if you want your organization's analysts to have read-only access to several indices, but developers to have write access to all indices—you have two options:

- Create user groups and configure your identity provider to choose the IAM role based on the user's authentication token (recommended).
- Configure your identity provider to choose the IAM role based on one or more rules.

For a walkthrough that includes fine-grained access control, see the section called “Tutorial: Fine-grained access control with Cognito authentication”.

> **Important**
>
> Just like the default role, Amazon Cognito must be part of each additional role's trust relationship. For details, see Creating Roles for Role Mapping in the *Amazon Cognito Developer Guide*.

User groups and tokens

When you create a user group, you choose an IAM role for members of the group. For information about creating groups, see User Groups in the *Amazon Cognito Developer Guide*.

After you create one or more user groups, you can configure your authentication provider to assign users their groups' roles rather than the identity pool's default role. Select **Choose role from token**, then choose either **Use default Authenticated role** or **DENY** to specify how the identity pool handles users who aren't part of a group.

Rules

Rules are essentially a series of `if` statements that Amazon Cognito evaluates sequentially. For example, if a user's email address contains `@corporate`, Amazon Cognito assigns that user `Role_A`. If a user's email address contains `@subsidiary`, it assigns that user `Role_B`. Otherwise, it assigns the user the default authenticated role.

To learn more, see Using Rule-Based Mapping to Assign Roles to Users in the *Amazon Cognito Developer Guide*.

(Optional) Customizing the sign-in page

You can use the Amazon Cognito console to upload a custom logo and make CSS changes to the sign-in page. For instructions and a full list of CSS properties, see Specifying App UI Customization Settings for Your User Pool in the *Amazon Cognito Developer Guide*.

(Optional) Configuring advanced security

Amazon Cognito user pools support advanced security features like multi-factor authentication, compromised credential checking, and adaptive authentication. To learn more, see Managing Security in the *Amazon Cognito Developer Guide*.

Testing

After you're satisfied with your configuration, verify that the user experience meets your expectations.

To access OpenSearch Dashboards

1. Navigate to `https://`*`opensearch-domain`*`/_dashboards` in a web browser. To log in to a specific tenant directly, append `?security_tenant=`*`tenant-name`* to the URL.

2. Sign in using your preferred credentials.
3. After OpenSearch Dashboards loads, configure at least one index pattern. Dashboards uses these patterns to identity which indices that you want to analyze. Enter *, choose **Next step**, and then choose **Create index pattern**.
4. To search or explore your data, choose **Discover**.

If any step of this process fails, see the section called "Common configuration issues" for troubleshooting information.

Quotas

Amazon Cognito has soft limits on many of its resources. If you want to enable Dashboards authentication for a large number of OpenSearch Service domains, review Quotas in Amazon Cognito and request limit increases as necessary.

Each OpenSearch Service domain adds an app client to the user pool, which adds an authentication provider to the identity pool. If you enable OpenSearch Dashboards authentication for more than 10 domains, you might encounter the "maximum Amazon Cognito user pool providers per identity pool" limit. If you exceed a limit, any OpenSearch Service domains that you try to configure to use Amazon Cognito authentication for Dashboards can get stuck in a configuration state of **Processing**.

Common configuration issues

The following tables list common configuration issues and solutions.

Configuring OpenSearch Service

Issue	Solution
`OpenSearch Service can't create the role` (console)	You don't have the correct IAM permissions. Add the permissions specified in the section called "Configuring Amazon Cognito authentication (console)".
`User is not authorized to perform: iam:PassRole on resource CognitoAc`	You don't have `iam:PassRole` permissions for the CognitoAccessForAmazonOpenSearch role. Attach the following policy to your account: `{`

Issue	Solution

cessForAmazonOpenSearch (console)	<pre>{
	  "Version": "2012-10-17",
	  "Statement": [
	    {
	      "Effect": "Allow",
	      "Action": [
	        "iam:PassRole"
	      ],
	      "Resource": "arn:aws:iam:: *123456789012:role/service-role/CognitoAccessForAmazonOpenSearch* "
	    }
	  ]
	}</pre> Alternately, you can attach the `IAMFullAccess` policy.

`User is not authorized to perform: cognito-identity:ListIdentityPools on resource`	You don't have read permissions for Amazon Cognito. Attach the `AmazonCognitoReadOnly` policy to your account.
`An error occurred (ValidationException) when calling the CreateDomain operation: OpenSearch Service must be allowed to use the passed role`	OpenSearch Service isn't specified in the trust relationship of the `CognitoAccessForAmazonOpenSearch` role. Check that your role uses the trust relationship that is specified in the section called "About the CognitoAccessForAmazonOpenSearch role". Alternately, use the console to configure Amazon Cognito authentication. The console creates a role for you.
`An error occurred (ValidationException) when calling the CreateDomain operation: User is not authorized to perform: cognito-idp: action on resource: user pool`	The role specified in `--cognito-options` does not have permissions to access Amazon Cognito. Check that the role has the AWS managed `AmazonOpenSearchServiceCognitoAccess` policy attached. Alternately, use the console to configure Amazon Cognito authentication. The console creates a role for you.

Issue	Solution
`An error occurred (ValidationException) when calling the CreateDomain operation: User pool does not exist`	OpenSearch Service can't find the user pool. Confirm that you created one and have the correct ID. To find the ID, you can use the Amazon Cognito console or the following AWS CLI command: `aws cognito-idp list-user-pools --max-results 60 --region region`
`An error occurred (ValidationException) when calling the CreateDomain operation: IdentityPool not found`	OpenSearch Service can't find the identity pool. Confirm that you created one and have the correct ID. To find the ID, you can use the Amazon Cognito console or the following AWS CLI command: `aws cognito-identity list-identity-pools --max-results 60 --region region`
`An error occurred (ValidationException) when calling the CreateDomain operation: Domain needs to be specified for user pool`	The user pool does not have a domain name. You can configure one using the Amazon Cognito console or the following AWS CLI command: `aws cognito-idp create-user-pool-domain --domain name --user-pool-id id`

Accessing OpenSearch Dashboards

Issue	Solution
The login page doesn't show my preferred identity providers.	Check that you enabled the identity provider for the OpenSearch Service app client as specified in the section called “Configuring identity providers”.
The login page doesn't look as if it's associated with my organization.	See the section called “(Optional) Customizing the sign-in page”.

Issue	Solution
My login credentials don't work.	Check that you have configured the identity provider as specified in the section called “Configuring identity providers”. If you use the user pool as your identity provider, check that the account exists on the Amazon Cognito console.
OpenSearch Dashboards either doesn't load at all or doesn't work properly.	The Amazon Cognito authenticated role needs `es:ESHttp*` permissions for the domain (`/*`) to access and use Dashboards. Check that you added an access policy as specified in the section called “Allowing the authenticated role”.
When I sign out of OpenSearch Dashboards from one tab, the remaining tabs display a message stating that the refresh token has been revoked.	When you sign out of an OpenSearch Dashboards session while using Amazon Cognito authentication, OpenSearch Service runs an AdminUserGlobalSignOut operation, which signs you out of *all* active OpenSearch Dashboards sessions.

Issue	Solution
`Invalid identity pool configuration. Check assigned IAM roles for this pool.`	Amazon Cognito doesn't have permissions to assume the IAM role on behalf of the authenticated user. Modify the trust relationship for the role to include: `{ "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Principal": { "Federated": "cognito-identity.amazonaws.com" }, "Action": "sts:AssumeRoleWithWebIdentity", "Condition": { "StringEquals": { "cognito-identity.amazonaws.com:aud" : " identity-pool-id " }, "ForAnyValue:StringLike": { "cognito-identity.amazonaws.com:amr" : "authenticated" } } }] }`
`Token is not from a supported provider of this identity pool.`	This uncommon error can occur when you remove the app client from the user pool. Try opening Dashboards in a new browser session.

Disabling Amazon Cognito authentication for OpenSearch Dashboards

Use the following procedure to disable Amazon Cognito authentication for Dashboards.

To disable Amazon Cognito authentication for Dashboards (console)

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home/.
2. Under **Domains**, choose the domain you want to configure.
3. Choose **Actions**, **Edit security configuration**.
4. Deselect **Enable Amazon Cognito authentication**.
5. Choose **Save changes**.

> **Important**
>
> If you no longer need the Amazon Cognito user pool and identity pool, delete them. Otherwise, you continue to incur charges.

Deleting domains that use Amazon Cognito authentication for OpenSearch Dashboards

To prevent domains that use Amazon Cognito authentication for Dashboards from becoming stuck in a configuration state of **Processing**, delete OpenSearch Service domains *before* deleting their associated Amazon Cognito user and identity pools.

Using service-linked roles for Amazon OpenSearch Service

Amazon OpenSearch Service uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to OpenSearch Service. Service-linked roles are predefined by OpenSearch Service and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up OpenSearch Service easier because you don't have to manually add the necessary permissions. OpenSearch Service defines the permissions of its service-linked roles, and unless defined otherwise, only OpenSearch Service can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity. For updates to service-linked roles and permissions policies, see Document history for Amazon OpenSearch Service.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Topics

Using service-linked roles to create VPC domains and direct query data sources

Amazon OpenSearch Service uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to OpenSearch Service. Service-linked roles are predefined by OpenSearch Service and include all the permissions that the service requires to call other AWS services on your behalf.

OpenSearch Service uses the service-linked role named **AWSServiceRoleForAmazonOpenSearchService**, which provides the minimum Amazon EC2 and Elastic Load Balancing permissions necessary for the role to enable VPC access for a domain or a direct query data source.

Legacy Elasticsearch role

Amazon OpenSearch Service uses a service-linked role called `AWSServiceRoleForAmazonOpenSearchService`. Your accounts might also contain a legacy service-linked role called `AWSServiceRoleForAmazonElasticsearchService`, which works with the deprecated Elasticsearch API endpoints.

If the legacy Elasticsearch role doesn't exist in your account, OpenSearch Service automatically creates a new OpenSearch service-linked role the first time you create an OpenSearch domain. Otherwise your account continues to use the Elasticsearch role. In order for this automatic creation to succeed, you must have permissions for the `iam:CreateServiceLinkedRole` action.

Permissions

The `AWSServiceRoleForAmazonOpenSearchService` service-linked role trusts the following services to assume the role:

- `opensearchservice.amazonaws.com`

The role permissions policy named `AmazonOpenSearchServiceRolePolicy` allows OpenSearch Service to complete the following actions on the specified resources:

- Action: `acm:DescribeCertificate` on *
- Action: `cloudwatch:PutMetricData` on *
- Action: `ec2:CreateNetworkInterface` on *
- Action: `ec2:DeleteNetworkInterface` on *
- Action: `ec2:DescribeNetworkInterfaces` on *
- Action: `ec2:ModifyNetworkInterfaceAttribute` on *
- Action: `ec2:DescribeSecurityGroups` on *
- Action: `ec2:DescribeSubnets` on *
- Action: `ec2:DescribeVpcs` on *
- Action: `ec2:CreateTags` on all network interfaces and VPC endpoints
- Action: `ec2:DescribeTags` on *
- Action: `ec2:CreateVpcEndpoint` on all VPCs, security groups, subnets, and route tables, as well as all VPC endpoints when the request contains the tag `OpenSearchManaged=true`
- Action: `ec2:ModifyVpcEndpoint` on all VPCs, security groups, subnets, and route tables, as well as all VPC endpoints when the request contains the tag `OpenSearchManaged=true`
- Action: `ec2:DeleteVpcEndpoints` on all endpoints when the request contains the tag `OpenSearchManaged=true`
- Action: `ec2:AssignIpv6Addresses` on *
- Action: `ec2:UnAssignIpv6Addresses` on *
- Action: `elasticloadbalancing:AddListenerCertificates` on *
- Action: `elasticloadbalancing:RemoveListenerCertificates` on *

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating the service-linked role

You don't need to manually create a service-linked role. When you create a VPC-enabled domain or a direct query data source using the AWS Management Console, OpenSearch Service creates the service-linked role for you. In order for this automatic creation to succeed, you must have permissions for the `iam:CreateServiceLinkedRole` action.

You can also use the IAM console, the IAM CLI, or the IAM API to create a service-linked role manually. For more information, see Creating a service-linked role in the *IAM User Guide*.

Editing the service-linked role

OpenSearch Service doesn't let you edit the `AWSServiceRoleForAmazonOpenSearchService` service-linked role. After you create a service-linked role, you can't change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting the service-linked role

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don’t have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

Cleaning up the service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**. Then choose the name (not the check box) of the `AWSServiceRoleForAmazonOpenSearchService` role.
3. On the **Summary** page for the selected role, choose the **Access Advisor** tab.
4. On the **Access Advisor** tab, review recent activity for the service-linked role.

> **Note**
>
> If you're unsure whether OpenSearch Service is using the `AWSServiceRoleForAmazonOpenSearchService` role, you can try to delete the role. If the service is using the role, then the deletion fails and you can view the resources using the role. If the role is being used, then you must wait for the session to end before you can delete the role, and/or delete the resources using the role. You cannot revoke the session for a service-linked role.

Manually deleting a service-linked role

Delete service-linked roles from the IAM console, API, or AWS CLI. For instructions, see Deleting a service-linked role in the *IAM User Guide*.

Using service-linked roles to create OpenSearch Serverless collections

OpenSearch Serverless uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to OpenSearch Service. Service-linked roles are predefined by OpenSearch Service and include all the permissions that the service requires to call other AWS services on your behalf.

OpenSearch Serverless uses the service-linked role named **AWSServiceRoleForAmazonOpenSearchServerless**, which provides the permissions necessary for the role to publish serverless-related CloudWatch metrics to your account. The role permissions policy associated with AWSServiceRoleForAmazonOpenSearchServerless is named `AmazonOpenSearchServerlessServiceRolePolicy`. For more information about the policy, see AmazonOpenSearchServerlessServiceRolePolicy in the *AWS Managed Policy Reference Guide*.

Service-linked role permissions for OpenSearch Serverless

OpenSearch Serverless uses the service-linked role named AWSServiceRoleForAmazonOpenSearchServerless, which allows OpenSearch Serverless to call AWS services on your behalf.

The AWSServiceRoleForAmazonOpenSearchServerless service-linked role trusts the following services to assume the role:

- `observability.aoss.amazonaws.com`

The role permissions policy named `AmazonOpenSearchServerlessServiceRolePolicy` allows OpenSearch Serverless to complete the following actions on the specified resources:

- Action: `cloudwatch:PutMetricData` on all AWS resources

> **Note**
>
> The policy includes the condition key `{"StringEquals": {"cloudwatch:namespace": "AWS/AOSS"}}`, which means that the service-linked role can only send metric data to the `AWS/AOSS` CloudWatch namespace.

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating the service-linked role for OpenSearch Serverless

You don't need to manually create a service-linked role. When you create an OpenSearch Serverless collection in the AWS Management Console, the AWS CLI, or the AWS API, OpenSearch Serverless creates the service-linked role for you.

> **Note**
>
> The first time you create a collection, you must be assigned the `iam:CreateServiceLinkedRole` in an identity-based policy.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create an OpenSearch Serverless collection, OpenSearch Serverless creates the service-linked role for you again.

You can also use the IAM console to create a service-linked role with the **Amazon OpenSearch Serverless** use case. In the AWS CLI or the AWS API, create a service-linked role with the `observability.aoss.amazonaws.com` service name:

```
aws iam create-service-linked-role --aws-service-name
 "observability.aoss.amazonaws.com"
```

For more information, see Creating a service-linked role in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

Editing the service-linked role for OpenSearch Serverless

OpenSearch Serverless does not allow you to edit the AWSServiceRoleForAmazonOpenSearchServerless service-linked role. After you create a service-linked role, you can't change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting the service-linked role for OpenSearch Serverless

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. This prevents you from having an unused entity that isn't actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

To delete the AWSServiceRoleForAmazonOpenSearchServerless, you must first delete all OpenSearch Serverless collections in your AWS account.

> **Note**
>
> If OpenSearch Serverless is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAmazonOpenSearchServerless service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported Regions for OpenSearch Serverless service-linked roles

OpenSearch Serverless supports using the AWSServiceRoleForAmazonOpenSearchServerless service-linked role in every Region where OpenSearch Serverless is available. For a list of supported Regions, see Amazon OpenSearch Serverless endpoints and quotas in the *AWS General Reference*.

Using service-linked roles to create OpenSearch Ingestion pipelines

Amazon OpenSearch Ingestion uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to OpenSearch Ingestion. Service-linked roles are predefined by OpenSearch Ingestion and include all the permissions that the service requires to call other AWS services on your behalf.

OpenSearch Ingestion uses the service-linked role named **AWSServiceRoleForAmazonOpenSearchIngestionService**, except when you use a self-managed VPC, in which case it uses the service-linked role named **AWSServiceRoleForOpensearchIngestionSelfManagedVpce**. The attached policy provides the permissions necessary for the role to create a virtual private cloud (VPC) between your account and OpenSearch Ingestion, and to publish CloudWatch metrics to your account.

Permissions

The `AWSServiceRoleForAmazonOpenSearchIngestionService` service-linked role trusts the following services to assume the role:

- `osis.amazon.com`

The role permissions policy named `AmazonOpenSearchIngestionServiceRolePolicy` allows OpenSearch Ingestion to complete the following actions on the specified resources:

- Action: `ec2:DescribeSubnets` on *
- Action: `ec2:DescribeSecurityGroups` on *
- Action: `ec2:DeleteVpcEndpoints` on *
- Action: `ec2:CreateVpcEndpoint` on *
- Action: `ec2:DescribeVpcEndpoints` on *
- Action: `ec2:CreateTags` on `arn:aws:ec2:*:*:network-interface/*`
- Action: `cloudwatch:PutMetricData` on `cloudwatch:namespace": "AWS/OSIS"`

The `AWSServiceRoleForOpensearchIngestionSelfManagedVpce` service-linked role trusts the following services to assume the role:

- `self-managed-vpce.osis.amazon.com`

The role permissions policy named `OpenSearchIngestionSelfManagedVpcePolicy` allows OpenSearch Ingestion to complete the following actions on the specified resources:

- Action: `ec2:DescribeSubnets` on *
- Action: `ec2:DescribeSecurityGroups` on *
- Action: `ec2:DescribeVpcEndpoints` on *
- Action: `cloudwatch:PutMetricData` on `cloudwatch:namespace": "AWS/OSIS"`

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating the service-linked role for OpenSearch Ingestion

You don't need to manually create a service-linked role. When you create an OpenSearch Ingestion pipeline in the AWS Management Console, the AWS CLI, or the AWS API, OpenSearch Ingestion creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create an OpenSearch Ingestion pipeline, OpenSearch Ingestion creates the service-linked role for you again.

Editing the service-linked role for OpenSearch Ingestion

OpenSearch Ingestion does not allow you to edit the `AWSServiceRoleForAmazonOpenSearchIngestionService` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting the service-linked role for OpenSearch Ingestion

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

> **Note**
>
> If OpenSearch Ingestion is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete OpenSearch Ingestion resources used by the `AWSServiceRoleForAmazonOpenSearchIngestionService` or `AWSServiceRoleForOpensearchIngestionSelfManagedVpce` role

1. Navigate to the Amazon OpenSearch Service console and choose **Ingestion**.
2. Delete all pipelines. For instructions, see the section called “Deleting pipelines”.

Delete the service-linked role for OpenSearch Ingestion

You can use the OpenSearch Ingestion console to delete a service-linked role.

To delete a service-linked role (console)

1. Navigate to the IAM console.
2. Choose **Roles** and search for the **AWSServiceRoleForAmazonOpenSearchIngestionService** or **AWSServiceRoleForOpensearchIngestionSelfManagedVpce** role.
3. Select the role and choose **Delete**.

Sample code for Amazon OpenSearch Service

This chapter contains common sample code for working with Amazon OpenSearch Service: HTTP request signing in a variety of programming languages, compressing HTTP request bodies, and using the AWS SDKs to create domains.

Topics

Elasticsearch client compatibility

The latest versions of the Elasticsearch clients might include license or version checks that artificially break compatibility. The following table includes recommendations around which versions of those clients to use for best compatibility with OpenSearch Service.

> **Important**
>
> These client versions are out of date and are not updated with the latest dependencies, including Log4j. We highly recommend using the OpenSearch versions of the clients when possible.

Client	Recommended version
Java low-level REST client	7.13.4
Java high-level REST client	7.13.4
Python Elasticsearch client	7.13.4
Ruby Elasticsearch client	7.13.3
Node.js Elasticsearch client	7.13.0

Compressing HTTP requests in Amazon OpenSearch Service

You can compress HTTP requests and responses in Amazon OpenSearch Service domains using gzip compression. Gzip compression can help you reduce the size of your documents and lower bandwidth utilization and latency, thereby leading to improved transfer speeds.

Gzip compression is supported for all domains running OpenSearch or Elasticsearch 6.0 or later. Some OpenSearch clients have built-in support for gzip compression, and many programming languages have libraries that simplify the process.

Enabling gzip compression

Not to be confused with similar OpenSearch settings, `http_compression.enabled` is specific to OpenSearch Service and enables or disables gzip compression on a domain. Domains running OpenSearch or Elasticsearch 7.*x* have the gzip compression enabled by default, whereas domains running Elasticsearch 6.*x* have it disabled by default.

To enable gzip compression, send the following request:

```
PUT _cluster/settings
{
  "persistent" : {
    "http_compression.enabled": true
  }
}
```

Requests to `_cluster/settings` must be uncompressed, so you might need to use a separate client or standard HTTP request to update cluster settings.

To confirm that you successfully enabled gzip compression, send the following request:

```
GET _cluster/settings?include_defaults=true
```

Make sure you see the following setting in the response:

```
...
"http_compression": {
  "enabled": "true"
}
...
```

Required headers

When including a gzip-compressed request body, keep the standard `Content-Type: application/json` header, and add the `Content-Encoding: gzip` header. To accept a gzip-compressed response, add the `Accept-Encoding: gzip` header, as well. If an OpenSearch client supports gzip compression, it likely includes these headers automatically.

Sample code (Python 3)

The following sample uses opensearch-py to perform the compression and send the request. This code signs the request using your IAM credentials.

```
from opensearchpy import OpenSearch, RequestsHttpConnection
from requests_aws4auth import AWS4Auth
import boto3

host = '' # e.g. my-test-domain.us-east-1.es.amazonaws.com
region = '' # e.g. us-west-1
service = 'es'
credentials = boto3.Session().get_credentials()
awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region, service,
 session_token=credentials.token)

# Create the client.
search = OpenSearch(
    hosts = [{'host': host, 'port': 443}],
    http_auth = awsauth,
    use_ssl = True,
    verify_certs = True,
    http_compress = True, # enables gzip compression for request bodies
    connection_class = RequestsHttpConnection
)

document = {
  "title": "Moneyball",
  "director": "Bennett Miller",
  "year": "2011"
}

# Send the request.
print(search.index(index='movies', id='1', body=document, refresh=True))
```

```
# print(search.index(index='movies', doc_type='_doc', id='1', body=document,
 refresh=True))
```

Alternately, you can specify the proper headers, compress the request body yourself, and use a standard HTTP library like Requests. This code signs the request using HTTP basic credentials, which your domain might support if you use fine-grained access control.

```
import requests
import gzip
import json

base_url = '' # The domain with https:// and a trailing slash. For example, https://my-
test-domain.us-east-1.es.amazonaws.com/
auth = ('master-user', 'master-user-password') # For testing only. Don't store
 credentials in code.

headers = {'Accept-Encoding': 'gzip', 'Content-Type': 'application/json',
           'Content-Encoding': 'gzip'}

document = {
  "title": "Moneyball",
  "director": "Bennett Miller",
  "year": "2011"
}

# Compress the document.
compressed_document = gzip.compress(json.dumps(document).encode())

# Send the request.
path = 'movies/_doc?refresh=true'
url = base_url + path
response = requests.post(url, auth=auth, headers=headers, data=compressed_document)
print(response.status_code)
print(response.text)
```

Using the AWS SDKs to interact with Amazon OpenSearch Service

This section includes examples of how to use the AWS SDKs to interact with the Amazon OpenSearch Service configuration API. These code samples show how to create, update, and delete OpenSearch Service domains.

Java

This section includes examples for versions 1 and 2 of the AWS SDK for Java.

Version 2

This example uses the OpenSearchClientBuilder constructor from version 2 of the AWS SDK for Java to create an OpenSearch domain, update its configuration, and delete it. Uncomment the calls to `waitForDomainProcessing` (and comment the call to `deleteDomain`) to allow the domain to come online and be useable.

```
package com.example.samples;

import java.util.concurrent.TimeUnit;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.opensearch.OpenSearchClient;
import software.amazon.awssdk.services.opensearch.model.ClusterConfig;
import software.amazon.awssdk.services.opensearch.model.EBSOptions;
import software.amazon.awssdk.services.opensearch.model.CognitoOptions;
import software.amazon.awssdk.services.opensearch.model.NodeToNodeEncryptionOptions;
import software.amazon.awssdk.services.opensearch.model.CreateDomainRequest;
import software.amazon.awssdk.services.opensearch.model.CreateDomainResponse;
import software.amazon.awssdk.services.opensearch.model.DescribeDomainRequest;
import software.amazon.awssdk.services.opensearch.model.UpdateDomainConfigRequest;
import software.amazon.awssdk.services.opensearch.model.UpdateDomainConfigResponse;
import software.amazon.awssdk.services.opensearch.model.DescribeDomainResponse;
import software.amazon.awssdk.services.opensearch.model.DeleteDomainRequest;
import software.amazon.awssdk.services.opensearch.model.DeleteDomainResponse;
import software.amazon.awssdk.services.opensearch.model.OpenSearchException;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;

/**
 * Sample class demonstrating how to use the Amazon Web Services SDK for Java to
 create, update,
 * and delete Amazon OpenSearch Service domains.
 */

public class OpenSearchSample {

    public static void main(String[] args) {

     String domainName = "my-test-domain";
```

```
    // Build the client using the default credentials chain.
       // You can use the CLI and run `aws configure` to set access key, secret
       // key, and default region.

       OpenSearchClient client = OpenSearchClient.builder()
         // Unnecessary, but lets you use a region different than your default.
         .region(Region.US_EAST_1)
         // Unnecessary, but if desired, you can use a different provider chain.
         .credentialsProvider(DefaultCredentialsProvider.create())
                    .build();

       // Create a new domain, update its configuration, and delete it.
       createDomain(client, domainName);
       //waitForDomainProcessing(client, domainName);
       updateDomain(client, domainName);
       //waitForDomainProcessing(client, domainName);
       deleteDomain(client, domainName);
   }

   /**
    * Creates an Amazon OpenSearch Service domain with the specified options.
    * Some options require other Amazon Web Services resources, such as an Amazon
Cognito user pool
    * and identity pool, whereas others require just an instance type or instance
    * count.
    *
    * @param client
    *            The client to use for the requests to Amazon OpenSearch Service
    * @param domainName
    *            The name of the domain you want to create
    */

   public static void createDomain(OpenSearchClient client, String domainName) {

    // Create the request and set the desired configuration options

       try {

           ClusterConfig clusterConfig = ClusterConfig.builder()
                   .dedicatedMasterEnabled(true)
                   .dedicatedMasterCount(3)
                   // Small, inexpensive instance types for testing. Not
recommended for production.
                   .dedicatedMasterType("t2.small.search")
```

```
                    .instanceType("t2.small.search")
                    .instanceCount(5)
                    .build();

            // Many instance types require EBS storage.
            EBSOptions ebsOptions = EBSOptions.builder()
                    .ebsEnabled(true)
                    .volumeSize(10)
                    .volumeType("gp2")
                    .build();

            NodeToNodeEncryptionOptions encryptionOptions =
 NodeToNodeEncryptionOptions.builder()
                    .enabled(true)
                    .build();

            CreateDomainRequest createRequest = CreateDomainRequest.builder()
                    .domainName(domainName)
                    .engineVersion("OpenSearch_1.0")
                    .clusterConfig(clusterConfig)
                    .ebsOptions(ebsOptions)
                    .nodeToNodeEncryptionOptions(encryptionOptions)
                    // You can uncomment this line and add your account ID, a
 username, and the
                    // domain name to add an access policy.
                    // .accessPolicies("{\"Version\":\"2012-10-17\",
\"Statement\":[{\"Effect\":\"Allow\",\"Principal\":{\"AWS\":
[\"arn:aws:iam::123456789012:user/user-name\"]},\"Action\":[\"es:*\"],\"Resource\":
\"arn:aws:es:region:123456789012:domain/domain-name/*\"}]}")
                    .build();

            // Make the request.
            System.out.println("Sending domain creation request...");
            CreateDomainResponse createResponse =
 client.createDomain(createRequest);
            System.out.println("Domain status:
 "+createResponse.domainStatus().toString());
            System.out.println("Domain ID:
 "+createResponse.domainStatus().domainId());


        } catch (OpenSearchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
```

```
        }
    }

    /**
     * Updates the configuration of an Amazon OpenSearch Service domain with the
     * specified options. Some options require other Amazon Web Services resources,
such as an
     * Amazon Cognito user pool and identity pool, whereas others require just an
     * instance type or instance count.
     *
     * @param client
     *             The client to use for the requests to Amazon OpenSearch Service
     * @param domainName
     *             The name of the domain to update
     */

    public static void updateDomain(OpenSearchClient client, String domainName) {

     // Updates the domain to use three data instances instead of five.
        // You can uncomment the Cognito line and fill in the strings to enable
Cognito
        // authentication for OpenSearch Dashboards.

        try {

            ClusterConfig clusterConfig = ClusterConfig.builder()
                    .instanceCount(5)
                    .build();

            CognitoOptions cognitoOptions = CognitoOptions.builder()
                    .enabled(true)
                    .userPoolId("user-pool-id")
                    .identityPoolId("identity-pool-id")
                    .roleArn("role-arn")
                    .build();

            UpdateDomainConfigRequest updateRequest =
UpdateDomainConfigRequest.builder()
                    .domainName(domainName)
                    .clusterConfig(clusterConfig)
                    //.cognitoOptions(cognitoOptions)
                    .build();

            System.out.println("Sending domain update request...");
```

```
            UpdateDomainConfigResponse updateResponse =
client.updateDomainConfig(updateRequest);
            System.out.println("Domain config:
"+updateResponse.domainConfig().toString());


        } catch (OpenSearchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    /**
     * Deletes an Amazon OpenSearch Service domain. Deleting a domain can take
     * several minutes.
     *
     * @param client
     *             The client to use for the requests to Amazon OpenSearch Service
     * @param domainName
     *             The name of the domain that you want to delete
     */

    public static void deleteDomain(OpenSearchClient client, String domainName) {

        try {

            DeleteDomainRequest deleteRequest = DeleteDomainRequest.builder()
                    .domainName(domainName)
                    .build();

            System.out.println("Sending domain deletion request...");
            DeleteDomainResponse deleteResponse =
client.deleteDomain(deleteRequest);
            System.out.println("Domain status: "+deleteResponse.toString());


        } catch (OpenSearchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    /**
```

```
    * Waits for the domain to finish processing changes. New domains typically take
 15-30 minutes
    * to initialize, but can take longer depending on the configuration. Most
 updates to existing domains
    * take a similar amount of time. This method checks every 15 seconds and
 finishes only when
    * the domain's processing status changes to false.
    *
    * @param client
    *            The client to use for the requests to Amazon OpenSearch Service
    * @param domainName
    *            The name of the domain that you want to check
    */

   public static void waitForDomainProcessing(OpenSearchClient client, String
 domainName) {
       // Create a new request to check the domain status.
       DescribeDomainRequest describeRequest = DescribeDomainRequest.builder()
               .domainName(domainName)
               .build();

       // Every 15 seconds, check whether the domain is processing.
       DescribeDomainResponse describeResponse =
 client.describeDomain(describeRequest);
       while (describeResponse.domainStatus().processing()) {
           try {
               System.out.println("Domain still processing...");
               TimeUnit.SECONDS.sleep(15);
               describeResponse = client.describeDomain(describeRequest);
           } catch (InterruptedException e) {
               e.printStackTrace();
           }
       }

       // Once we exit that loop, the domain is available
       System.out.println("Amazon OpenSearch Service has finished processing
 changes for your domain.");
       System.out.println("Domain description: "+describeResponse.toString());
   }
}
```

Version 1

This example uses the AWSElasticsearchClientBuilder constructor from version 1 of the AWS SDK for Java to create a legacy Elasticsearch domain, update its configuration, and delete it. Uncomment the calls to `waitForDomainProcessing` (and comment the call to `deleteDomain`) to allow the domain to come online and be useable.

```
package com.amazonaws.samples;

import java.util.concurrent.TimeUnit;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.elasticsearch.AWSElasticsearch;
import com.amazonaws.services.elasticsearch.AWSElasticsearchClientBuilder;
import com.amazonaws.services.elasticsearch.model.CreateElasticsearchDomainRequest;
import com.amazonaws.services.elasticsearch.model.CreateElasticsearchDomainResult;
import com.amazonaws.services.elasticsearch.model.DeleteElasticsearchDomainRequest;
import com.amazonaws.services.elasticsearch.model.DeleteElasticsearchDomainResult;
import
 com.amazonaws.services.elasticsearch.model.DescribeElasticsearchDomainRequest;
import com.amazonaws.services.elasticsearch.model.DescribeElasticsearchDomainResult;
import com.amazonaws.services.elasticsearch.model.EBSOptions;
import com.amazonaws.services.elasticsearch.model.ElasticsearchClusterConfig;
import com.amazonaws.services.elasticsearch.model.ResourceNotFoundException;
import
 com.amazonaws.services.elasticsearch.model.UpdateElasticsearchDomainConfigRequest;
import
 com.amazonaws.services.elasticsearch.model.UpdateElasticsearchDomainConfigResult;
import com.amazonaws.services.elasticsearch.model.VolumeType;

/**
 * Sample class demonstrating how to use the Amazon Web Services SDK for Java to
 create, update,
 * and delete Amazon OpenSearch Service domains.
 */

public class OpenSearchSample {

    public static void main(String[] args) {

        final String domainName = "my-test-domain";

        // Build the client using the default credentials chain.
```

```
        // You can use the CLI and run `aws configure` to set access key, secret
        // key, and default region.
        final AWSElasticsearch client = AWSElasticsearchClientBuilder
                .standard()
                // Unnecessary, but lets you use a region different than your
default.
                .withRegion(Regions.US_WEST_2)
                // Unnecessary, but if desired, you can use a different provider
chain.
                .withCredentials(new DefaultAWSCredentialsProviderChain())
                .build();

        // Create a new domain, update its configuration, and delete it.
        createDomain(client, domainName);
        // waitForDomainProcessing(client, domainName);
        updateDomain(client, domainName);
        // waitForDomainProcessing(client, domainName);
        deleteDomain(client, domainName);
    }

    /**
     * Creates an Amazon OpenSearch Service domain with the specified options.
     * Some options require other Amazon Web Services resources, such as an Amazon
Cognito user pool
     * and identity pool, whereas others require just an instance type or instance
     * count.
     *
     * @param client
     *            The client to use for the requests to Amazon OpenSearch Service
     * @param domainName
     *            The name of the domain you want to create
     */
    private static void createDomain(final AWSElasticsearch client, final String
domainName) {

        // Create the request and set the desired configuration options
        CreateElasticsearchDomainRequest createRequest = new
CreateElasticsearchDomainRequest()
                .withDomainName(domainName)
                .withElasticsearchVersion("7.10")
                .withElasticsearchClusterConfig(new ElasticsearchClusterConfig()
                        .withDedicatedMasterEnabled(true)
                        .withDedicatedMasterCount(3)
```

```
                        // Small, inexpensive instance types for testing. Not
 recommended for production
                        // domains.
                        .withDedicatedMasterType("t2.small.elasticsearch")
                        .withInstanceType("t2.small.elasticsearch")
                        .withInstanceCount(5))
                // Many instance types require EBS storage.
                .withEBSOptions(new EBSOptions()
                        .withEBSEnabled(true)
                        .withVolumeSize(10)
                        .withVolumeType(VolumeType.Gp2));
                // You can uncomment this line and add your account ID, a username,
 and the
                // domain name to add an access policy.
                // .withAccessPolicies("{\"Version\":\"2012-10-17\",
\"Statement\":[{\"Effect\":\"Allow\",\"Principal\":{\"AWS\":
[\"arn:aws:iam::123456789012:user/user-name\"]},\"Action\":[\"es:*\"],\"Resource\":
\"arn:aws:es:region:123456789012:domain/domain-name/*\"}]}")


        // Make the request.
        System.out.println("Sending domain creation request...");
        CreateElasticsearchDomainResult createResponse =
 client.createElasticsearchDomain(createRequest);
        System.out.println("Domain creation response from Amazon OpenSearch
 Service:");
        System.out.println(createResponse.getDomainStatus().toString());
    }

    /**
     * Updates the configuration of an Amazon OpenSearch Service domain with the
     * specified options. Some options require other Amazon Web Services resources,
 such as an
     * Amazon Cognito user pool and identity pool, whereas others require just an
     * instance type or instance count.
     *
     * @param client
     *            The client to use for the requests to Amazon OpenSearch Service
     * @param domainName
     *            The name of the domain to update
     */
    private static void updateDomain(final AWSElasticsearch client, final String
 domainName) {
        try {
```

```
            // Updates the domain to use three data instances instead of five.
            // You can uncomment the Cognito lines and fill in the strings to enable
Cognito
            // authentication for OpenSearch Dashboards.
            final UpdateElasticsearchDomainConfigRequest updateRequest = new
UpdateElasticsearchDomainConfigRequest()
                    .withDomainName(domainName)
                    // .withCognitoOptions(new CognitoOptions()
                            // .withEnabled(true)
                            // .withUserPoolId("user-pool-id")
                            // .withIdentityPoolId("identity-pool-id")
                            // .withRoleArn("role-arn")
                    .withElasticsearchClusterConfig(new ElasticsearchClusterConfig()
                            .withInstanceCount(3));

            System.out.println("Sending domain update request...");
            final UpdateElasticsearchDomainConfigResult updateResponse = client
                    .updateElasticsearchDomainConfig(updateRequest);
            System.out.println("Domain update response from Amazon OpenSearch
Service:");
            System.out.println(updateResponse.toString());
        } catch (ResourceNotFoundException e) {
            System.out.println("Domain not found. Please check the domain name.");
        }
    }

    /**
     * Deletes an Amazon OpenSearch Service domain. Deleting a domain can take
     * several minutes.
     *
     * @param client
     *             The client to use for the requests to Amazon OpenSearch Service
     * @param domainName
     *             The name of the domain that you want to delete
     */
    private static void deleteDomain(final AWSElasticsearch client, final String
domainName) {
        try {
            final DeleteElasticsearchDomainRequest deleteRequest = new
DeleteElasticsearchDomainRequest()
                    .withDomainName(domainName);

            System.out.println("Sending domain deletion request...");
```

```
            final DeleteElasticsearchDomainResult deleteResponse =
client.deleteElasticsearchDomain(deleteRequest);
            System.out.println("Domain deletion response from Amazon OpenSearch
Service:");
            System.out.println(deleteResponse.toString());
        } catch (ResourceNotFoundException e) {
            System.out.println("Domain not found. Please check the domain name.");
        }
    }

    /**
     * Waits for the domain to finish processing changes. New domains typically take
15-30 minutes
     * to initialize, but can take longer depending on the configuration. Most
updates to existing domains
     * take a similar amount of time. This method checks every 15 seconds and
finishes only when
     * the domain's processing status changes to false.
     *
     * @param client
     *            The client to use for the requests to Amazon OpenSearch Service
     * @param domainName
     *            The name of the domain that you want to check
     */
    private static void waitForDomainProcessing(final AWSElasticsearch client, final
String domainName) {
        // Create a new request to check the domain status.
        final DescribeElasticsearchDomainRequest describeRequest = new
DescribeElasticsearchDomainRequest()
                .withDomainName(domainName);

        // Every 15 seconds, check whether the domain is processing.
        DescribeElasticsearchDomainResult describeResponse =
client.describeElasticsearchDomain(describeRequest);
        while (describeResponse.getDomainStatus().isProcessing()) {
            try {
                System.out.println("Domain still processing...");
                TimeUnit.SECONDS.sleep(15);
                describeResponse =
client.describeElasticsearchDomain(describeRequest);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
```

```
        // Once we exit that loop, the domain is available
        System.out.println("Amazon OpenSearch Service has finished processing
 changes for your domain.");
        System.out.println("Domain description response from Amazon OpenSearch
 Service:");
        System.out.println(describeResponse.toString());
    }
}
```

Python

This example uses the OpenSearchService low-level Python client from the AWS SDK for Python (Boto) to create a domain, update its configuration, and delete it.

```
import boto3
import botocore
from botocore.config import Config
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default region.

my_config = Config(
    # Optionally lets you specify a region other than your default.
    region_name='us-west-2'
)

client = boto3.client('opensearch', config=my_config)

domainName = 'my-test-domain'  # The name of the domain


def createDomain(client, domainName):
    """Creates an Amazon OpenSearch Service domain with the specified options."""
    response = client.create_domain(
        DomainName=domainName,
        EngineVersion='OpenSearch_1.0',
        ClusterConfig={
            'InstanceType': 't2.small.search',
            'InstanceCount': 5,
```

```
            'DedicatedMasterEnabled': True,
            'DedicatedMasterType': 't2.small.search',
            'DedicatedMasterCount': 3
        },
        # Many instance types require EBS storage.
        EBSOptions={
            'EBSEnabled': True,
            'VolumeType': 'gp2',
            'VolumeSize': 10
        },
        AccessPolicies="{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect\":\"Allow
\",\"Principal\":{\"AWS\":[\"arn:aws:iam::123456789012:user/user-name\"]},\"Action\":
[\"es:*\"],\"Resource\":\"arn:aws:es:us-west-2:123456789012:domain/my-test-domain/*
\"}]}",
        NodeToNodeEncryptionOptions={
            'Enabled': True
        }
    )
    print("Creating domain...")
    print(response)


def updateDomain(client, domainName):
    """Updates the domain to use three data nodes instead of five."""
    try:
        response = client.update_domain_config(
            DomainName=domainName,
            ClusterConfig={
                'InstanceCount': 3
            }
        )
        print('Sending domain update request...')
        print(response)

    except botocore.exceptions.ClientError as error:
        if error.response['Error']['Code'] == 'ResourceNotFoundException':
            print('Domain not found. Please check the domain name.')
        else:
            raise error


def deleteDomain(client, domainName):
    """Deletes an OpenSearch Service domain. Deleting a domain can take several
 minutes."""
```

```
    try:
        response = client.delete_domain(
            DomainName=domainName
        )
        print('Sending domain deletion request...')
        print(response)

    except botocore.exceptions.ClientError as error:
        if error.response['Error']['Code'] == 'ResourceNotFoundException':
            print('Domain not found. Please check the domain name.')
        else:
            raise error


def waitForDomainProcessing(client, domainName):
    """Waits for the domain to finish processing changes."""
    try:
        response = client.describe_domain(
            DomainName=domainName
        )
        # Every 15 seconds, check whether the domain is processing.
        while response["DomainStatus"]["Processing"] == True:
            print('Domain still processing...')
            time.sleep(15)
            response = client.describe_domain(
                DomainName=domainName)

        # Once we exit the loop, the domain is available.
        print('Amazon OpenSearch Service has finished processing changes for your
 domain.')
        print('Domain description:')
        print(response)

    except botocore.exceptions.ClientError as error:
        if error.response['Error']['Code'] == 'ResourceNotFoundException':
            print('Domain not found. Please check the domain name.')
        else:
            raise error


def main():
    """Create a new domain, update its configuration, and delete it."""
    createDomain(client, domainName)
    waitForDomainProcessing(client, domainName)
```

```
    updateDomain(client, domainName)
    waitForDomainProcessing(client, domainName)
    deleteDomain(client, domainName)
```

Node

This example uses the version 3 of the SDK for JavaScript in Node.js OpenSearch client to create a domain, update its configuration, and delete it.

```
var {
    OpenSearchClient,
    CreateDomainCommand,
    DescribeDomainCommand,
    UpdateDomainConfigCommand,
    DeleteDomainCommand
} = require("@aws-sdk/client-opensearch");
var sleep = require('sleep');

var client = new OpenSearchClient();

var domainName = 'my-test-domain'

// Create a new domain, update its configuration, and delete it.
createDomain(client, domainName)
waitForDomainProcessing(client, domainName)
updateDomain(client, domainName)
waitForDomainProcessing(client, domainName)
deleteDomain(client, domainName)

async function createDomain(client, domainName) {
    // Creates an Amazon OpenSearch Service domain with the specified options.
    var command = new CreateDomainCommand({
        DomainName: domainName,
        EngineVersion: 'OpenSearch_1.0',
        ClusterConfig: {
        'InstanceType': 't2.small.search',
        'InstanceCount': 5,
        'DedicatedMasterEnabled': 'True',
        'DedicatedMasterType': 't2.small.search',
        'DedicatedMasterCount': 3
        },
        EBSOptions:{
            'EBSEnabled': 'True',
```

```
            'VolumeType': 'gp2',
            'VolumeSize': 10
        },
        AccessPolicies: "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect\":\"Allow
\",\"Principal\":{\"AWS\":[\"arn:aws:iam::123456789012:user/user-name\"]},\"Action\":
[\"es:*\"],\"Resource\":\"arn:aws:es:us-east-1:123456789012:domain/my-test-domain/*
\"}]}",
         NodeToNodeEncryptionOptions:{
            'Enabled': 'True'
        }
    });
    const response = await client.send(command);
    console.log("Creating domain...");
    console.log(response);
}

async function updateDomain(client, domainName) {
    // Updates the domain to use three data nodes instead of five.
    var command = new UpdateDomainConfigCommand({
        DomainName: domainName,
        ClusterConfig: {
        'InstanceCount': 3
        }
    });
    const response = await client.send(command);
    console.log('Sending domain update request...');
    console.log(response);
}

async function deleteDomain(client, domainName) {
    // Deletes an OpenSearch Service domain. Deleting a domain can take several
 minutes.
    var command = new DeleteDomainCommand({
        DomainName: domainName
    });
    const response = await client.send(command);
    console.log('Sending domain deletion request...');
    console.log(response);
}

async function waitForDomainProcessing(client, domainName) {
    // Waits for the domain to finish processing changes.
    try {
        var command = new DescribeDomainCommand({
```

```
            DomainName: domainName
        });
        var response = await client.send(command);

        while (response.DomainStatus.Processing == true) {
            console.log('Domain still processing...')
            await sleep(15000) // Wait for 15 seconds, then check the status again
            function sleep(ms) {
                return new Promise((resolve) => {
                    setTimeout(resolve, ms);
                });
            }
            var response = await client.send(command);
        }
        // Once we exit the loop, the domain is available.
        console.log('Amazon OpenSearch Service has finished processing changes for your
 domain.');
        console.log('Domain description:');
        console.log(response);

    } catch (error) {
        if (error.name === 'ResourceNotFoundException') {
            console.log('Domain not found. Please check the domain name.');
            }
    };
}
```

Indexing data in Amazon OpenSearch Service

Because Amazon OpenSearch Service uses a REST API, numerous methods exist for indexing documents. You can use standard clients like curl or any programming language that can send HTTP requests. To further simplify the process of interacting with it, OpenSearch Service has clients for many programming languages. Advanced users can skip directly to the section called "Loading streaming data into OpenSearch Service".

We strongly recommend that you use Amazon OpenSearch Ingestion to ingest data, which is a fully managed data collector built within OpenSearch Service. For more information, see Amazon OpenSearch Ingestion.

For an introduction to indexing, see the OpenSearch documentation.

Naming restrictions for indexes

OpenSearch Service indexes have the following naming restrictions:

- All letters must be lowercase.
- Index names cannot begin with _ or -.
- Index names can't contain spaces, commas, :, ", *, +, /, \, |, ?, #, >, or <.

Don't include sensitive information in index, type, or document ID names. OpenSearch Service uses these names in its Uniform Resource Identifiers (URIs). Servers and applications often log HTTP requests, which can lead to unnecessary data exposure if URIs contain sensitive information:

```
2018-10-03T23:39:43 198.51.100.14 200 "GET https://opensearch-domain/dr-jane-doe/flu-patients-2018/202-555-0100/ HTTP/1.1"
```

Even if you don't have permissions to view the associated JSON document, you could infer from this fake log line that one of Dr. Doe's patients with a phone number of 202-555-0100 had the flu in 2018.

If OpenSearch Service detects a real or percieved IP address in an index name (for example, `my-index-12.34.56.78.91`), it masks the IP address. A call to `_cat/indices` yields the following response:

```
green open my-index-x.x.x.x.91    soY19tBERoKo71WcEScidw 5 1 0 0   2kb  1kb
```

To prevent unnecessary confusion, avoid including IP addresses in index names.

Reducing response size

Responses from the _index and _bulk APIs contain quite a bit of information. This information can be useful for troubleshooting requests or for implementing retry logic, but can use considerable bandwidth. In this example, indexing a 32 byte document results in a 339 byte response (including headers):

```
PUT opensearch-domain/more-movies/_doc/1
{"title": "Back to the Future"}
```

Response

```
{
  "_index": "more-movies",
  "_type": "_doc",
  "_id": "1",
  "_version": 4,
  "result": "updated",
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "_seq_no": 3,
  "_primary_term": 1
}
```

This response size might seem minimal, but if you index 1,000,000 documents per day—approximately 11.5 documents per second—339 bytes per response works out to 10.17 GB of download traffic per month.

If data transfer costs are a concern, use the filter_path parameter to reduce the size of the OpenSearch Service response, but be careful not to filter out fields that you need in order to identify or retry failed requests. These fields vary by client. The filter_path parameter works for all OpenSearch Service REST APIs, but is especially useful with APIs that you call frequently, such as the _index and _bulk APIs:

```
PUT opensearch-domain/more-movies/_doc/1?filter_path=result,_shards.total
{"title": "Back to the Future"}
```

Response

```
{
  "result": "updated",
  "_shards": {
    "total": 2
  }
}
```

Instead of including fields, you can exclude fields with a `-` prefix. `filter_path` also supports wildcards:

```
POST opensearch-domain/_bulk?filter_path=-took,-items.index._*
{ "index": { "_index": "more-movies", "_id": "1" } }
{"title": "Back to the Future"}
{ "index": { "_index": "more-movies", "_id": "2" } }
{"title": "Spirited Away"}
```

Response

```
{
  "errors": false,
  "items": [
    {
      "index": {
        "result": "updated",
        "status": 200
      }
    },
    {
      "index": {
        "result": "updated",
        "status": 200
      }
    }
  ]
}
```

Index codecs

Index codecs determine how the stored fields on an index are compressed and stored on disk. The index codec is controlled by the static `index.codec` setting, which specifies the compression algorithm. This setting impacts the index shard size and operation performance.

For a list of supported codecs and their performance characteristics, see Supported codecs in the OpenSearch documentation.

When you choose an index codec, consider the following:

- To avoid the challenges of changing the codec setting of an existing index, test a representative workload in a non-production environment before using a new codec setting. For more information, see Changing an index codec.
- You can't use Zstandard compression codecs (`"index.codec": "zstd"` or `"index.codec": "zstd_no_dict"`) for k-NN or Security Analytics indexes.

Loading streaming data into Amazon OpenSearch Service

You can use OpenSearch Ingestion to directly load streaming data into your Amazon OpenSearch Service domain, without needing to use third-party solutions. To send data to OpenSearch Ingestion, you configure your data producers and the service automatically delivers the data to the domain or collection that you specify. To get started with OpenSearch Ingestion, see the section called “Tutorial: Ingest data into a collection”.

You can still use other sources to load streaming data, such as Amazon Data Firehose and Amazon CloudWatch Logs, which have built-in support for OpenSearch Service. Others, like Amazon S3, Amazon Kinesis Data Streams, and Amazon DynamoDB, use AWS Lambda functions as event handlers. The Lambda functions respond to new data by processing it and streaming it to your domain.

> **Note**
>
> Lambda supports several popular programming languages and is available in most AWS Regions. For more information, see Getting started with Lambda in the *AWS Lambda Developer Guide* and AWS service endpoints in the *AWS General Reference*.

Loading streaming data from OpenSearch Ingestion

You can use Amazon OpenSearch Ingestion to load data into an OpenSearch Service domain. You configure your data producers to send data to OpenSearch Ingestion, and it automatically delivers the data to the collection that you specify. You can also configure OpenSearch Ingestion to transform your data before delivering it. For more information, see *Amazon OpenSearch Ingestion*.

Loading streaming data from Amazon S3

You can use Lambda to send data to your OpenSearch Service domain from Amazon S3. New data that arrives in an S3 bucket triggers an event notification to Lambda, which then runs your custom code to perform the indexing.

This method of streaming data is extremely flexible. You can index object metadata, or if the object is plaintext, parse and index some elements of the object body. This section includes some unsophisticated Python sample code that uses regular expressions to parse a log file and index the matches.

Prerequisites

Before proceeding, you must have the following resources.

Prerequisite	Description
Amazon S3 bucket	For more information, see Create your first S3 bucket in the *Amazon Simple Storage Service User Guide*. The bucket must reside in the same Region as your OpenSearch Service domain.
OpenSearch Service domain	The destination for data after your Lambda function processes it. For more information, see the section called " Creating OpenSearch Service domains".

Create the Lambda deployment package

Deployment packages are ZIP or JAR files that contain your code and its dependencies. This section includes Python sample code. For other programming languages, see Lambda deployment packages in the *AWS Lambda Developer Guide*.

1. Create a directory. In this sample, we use the name `s3-to-opensearch`.
2. Create a file within the directory named `sample.py`:

```
import boto3
import re
import requests
from requests_aws4auth import AWS4Auth

region = '' # e.g. us-west-1
service = 'es'
credentials = boto3.Session().get_credentials()
awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region, service,
 session_token=credentials.token)

host = '' # the OpenSearch Service domain, e.g. https://search-mydomain.us-
west-1.es.amazonaws.com
index = 'lambda-s3-index'
datatype = '_doc'
url = host + '/' + index + '/' + datatype

headers = { "Content-Type": "application/json" }

s3 = boto3.client('s3')

# Regular expressions used to parse some simple log lines
ip_pattern = re.compile('(\d+\.\d+\.\d+\.\d+)')
time_pattern = re.compile('\[(\d+\/\w\w\w\/\d\d\d\d:\d\d:\d\d:\d\d\s-\d\d\d\d)\]')
message_pattern = re.compile('\"(.+)\"')

# Lambda execution starts here
def handler(event, context):
    for record in event['Records']:

        # Get the bucket name and key for the new file
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']

        # Get, read, and split the file into lines
        obj = s3.get_object(Bucket=bucket, Key=key)
        body = obj['Body'].read()
        lines = body.splitlines()

        # Match the regular expressions to each line and index the JSON
```

```
        for line in lines:
            line = line.decode("utf-8")
            ip = ip_pattern.search(line).group(1)
            timestamp = time_pattern.search(line).group(1)
            message = message_pattern.search(line).group(1)

            document = { "ip": ip, "timestamp": timestamp, "message": message }
            r = requests.post(url, auth=awsauth, json=document, headers=headers)
```

Edit the variables for `region` and `host`.

3. Install pip if you haven't already, then install the dependencies to a new `package` directory:

```
cd s3-to-opensearch

pip install --target ./package requests
pip install --target ./package requests_aws4auth
```

All Lambda execution environments have Boto3 installed, so you don't need to include it in your deployment package.

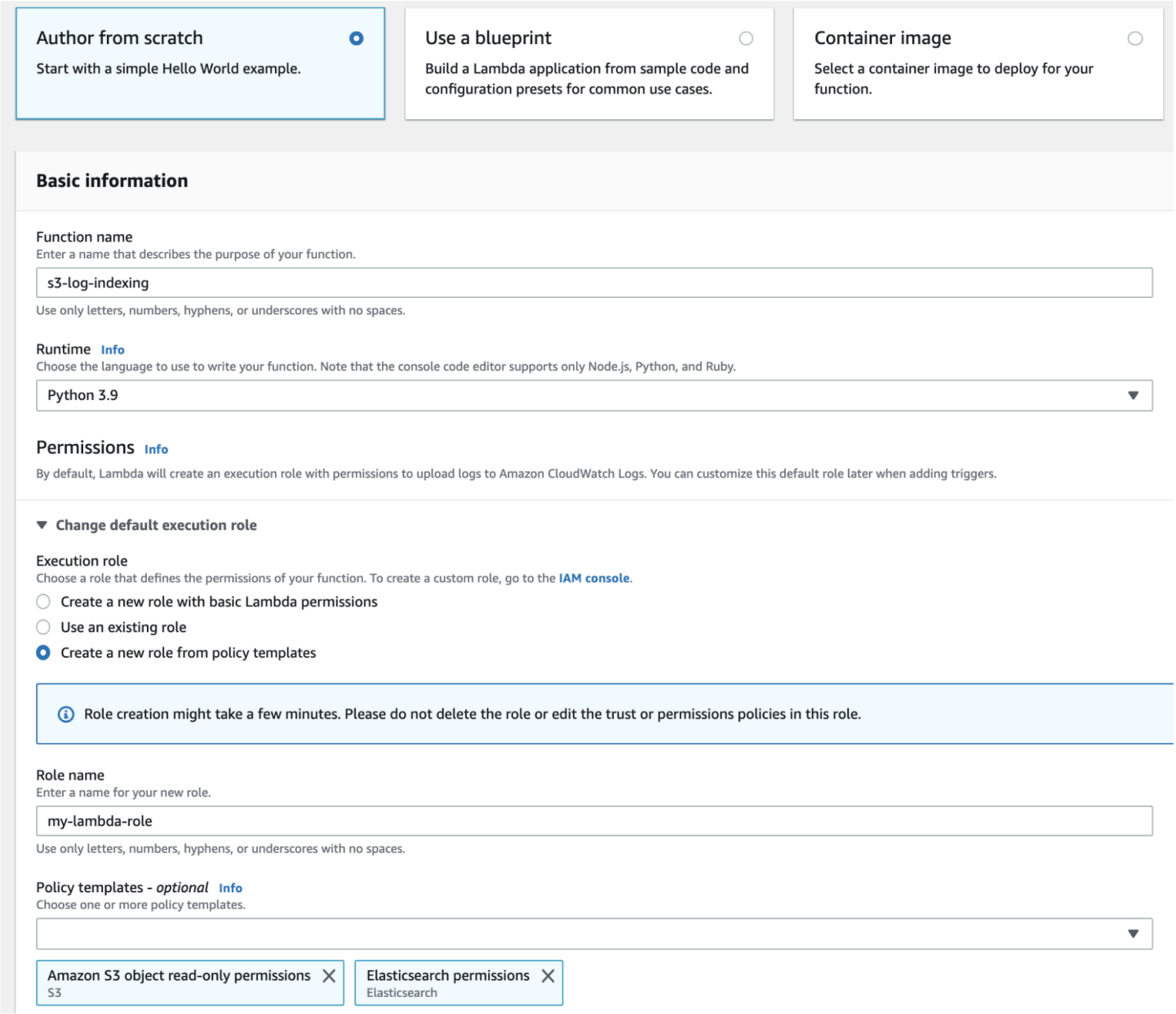
4. Package the application code and dependencies:

```
cd package
zip -r ../lambda.zip .

cd ..
zip -g lambda.zip sample.py
```

Create the Lambda function

After you create the deployment package, you can create the Lambda function. When you create a function, choose a name, runtime (for example, Python 3.8), and IAM role. The IAM role defines the permissions for your function. For detailed instructions, see Create a Lambda function with the console in the *AWS Lambda Developer Guide*.

This example assumes you're using the console. Choose Python 3.9 and a role that has S3 read permissions and OpenSearch Service write permissions, as shown in the following screenshot:



After you create the function, you must add a trigger. For this example, we want the code to run whenever a log file arrives in an S3 bucket:

1. Choose **Add trigger** and select **S3**.
2. Choose your bucket.
3. For **Event type**, choose **PUT**.
4. For **Prefix**, type `logs/`.
5. For **Suffix**, type `.log`.
6. Acknowledge the recursive invocation warning and choose **Add**.

Finally, you can upload your deployment package:

1. Choose **Upload from** and **.zip file**, then follow the prompts to upload your deployment package.
2. After the upload finishes, edit the **Runtime settings** and change the **Handler** to `sample.handler`. This setting tells Lambda the file (`sample.py`) and method (`handler`) that it should run after a trigger.

At this point, you have a complete set of resources: a bucket for log files, a function that runs whenever a log file is added to the bucket, code that performs the parsing and indexing, and an OpenSearch Service domain for searching and visualization.

Testing the Lambda Function

After you create the function, you can test it by uploading a file to the Amazon S3 bucket. Create a file named `sample.log` using following sample log lines:

```
12.345.678.90 - [10/Oct/2000:13:55:36 -0700] "PUT /some-file.jpg"
12.345.678.91 - [10/Oct/2000:14:56:14 -0700] "GET /some-file.jpg"
```

Upload the file to the `logs` folder of your S3 bucket. For instructions, see Upload an object to your bucket in the *Amazon Simple Storage Service User Guide*.

Then use the OpenSearch Service console or OpenSearch Dashboards to verify that the `lambda-s3-index` index contains two documents. You can also make a standard search request:

```
GET https://domain-name/lambda-s3-index/_search?pretty
{
  "hits" : {
    "total" : 2,
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "lambda-s3-index",
        "_type" : "_doc",
        "_id" : "vTYXaWIBJWV_TTkEuSDg",
        "_score" : 1.0,
        "_source" : {
          "ip" : "12.345.678.91",
          "message" : "GET /some-file.jpg",
```

```
          "timestamp" : "10/Oct/2000:14:56:14 -0700"
        }
      },
      {
        "_index" : "lambda-s3-index",
        "_type" : "_doc",
        "_id" : "vjYmaWIBJWV_TTkEuCAB",
        "_score" : 1.0,
        "_source" : {
          "ip" : "12.345.678.90",
          "message" : "PUT /some-file.jpg",
          "timestamp" : "10/Oct/2000:13:55:36 -0700"
        }
      }
    ]
  }
}
```

Loading streaming data from Amazon Kinesis Data Streams

You can load streaming data from Kinesis Data Streams to OpenSearch Service. New data that arrives in the data stream triggers an event notification to Lambda, which then runs your custom code to perform the indexing. This section includes some unsophisticated Python sample code.

Prerequisites

Before proceeding, you must have the following resources.

Prerequisite	Description
Amazon Kinesis Data Stream	The event source for your Lambda function. To learn more, see Kinesis Data Streams.
OpenSearch Service Domain	The destination for data after your Lambda function processes it. For more information, see the section called “ Creating OpenSearch Service domains”
IAM Role	This role must have basic OpenSearch Service, Kinesis, and Lambda permissions, such as the following: `{`

Prerequisite	Description

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "es:ESHttpPost",
        "es:ESHttpPut",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "kinesis:GetShardIterator",
        "kinesis:GetRecords",
        "kinesis:DescribeStream",
        "kinesis:ListStreams"
      ],
      "Resource": "*"
    }
  ]
}
```

The role must have the following trust relationship:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

To learn more, see Creating IAM roles in the *IAM User Guide*.

Create the Lambda function

Follow the instructions in the section called "Create the Lambda deployment package", but create a directory named `kinesis-to-opensearch` and use the following code for `sample.py`:

```
import base64
import boto3
import json
import requests
from requests_aws4auth import AWS4Auth

region = '' # e.g. us-west-1
service = 'es'
credentials = boto3.Session().get_credentials()
awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region, service,
 session_token=credentials.token)

host = '' # the OpenSearch Service domain, e.g. https://search-mydomain.us-
west-1.es.amazonaws.com
index = 'lambda-kine-index'
datatype = '_doc'
url = host + '/' + index + '/' + datatype + '/'

headers = { "Content-Type": "application/json" }

def handler(event, context):
    count = 0
    for record in event['Records']:
        id = record['eventID']
        timestamp = record['kinesis']['approximateArrivalTimestamp']

        # Kinesis data is base64-encoded, so decode here
        message = base64.b64decode(record['kinesis']['data'])

        # Create the JSON document
        document = { "id": id, "timestamp": timestamp, "message": message }
        # Index the document
        r = requests.put(url + id, auth=awsauth, json=document, headers=headers)
        count += 1
    return 'Processed ' + str(count) + ' items.'
```

Edit the variables for `region` and `host`.

Install pip if you haven't already, then use the following commands to install your dependencies:

```
cd kinesis-to-opensearch

pip install --target ./package requests
pip install --target ./package requests_aws4auth
```

Then follow the instructions in the section called "Create the Lambda function", but specify the IAM role from the section called "Prerequisites" and the following settings for the trigger:

- **Kinesis stream**: your Kinesis stream
- **Batch size**: 100
- **Starting position**: Trim horizon

To learn more, see What is Amazon Kinesis Data Streams? in the *Amazon Kinesis Data Streams Developer Guide*.

At this point, you have a complete set of resources: a Kinesis data stream, a function that runs after the stream receives new data and indexes that data, and an OpenSearch Service domain for searching and visualization.

Test the Lambda Function

After you create the function, you can test it by adding a new record to the data stream using the AWS CLI:

```
aws kinesis put-record --stream-name test --data "My test data." --partition-key
 partitionKey1 --region us-west-1
```

Then use the OpenSearch Service console or OpenSearch Dashboards to verify that `lambda-kine-index` contains a document. You can also use the following request:

```
GET https://domain-name/lambda-kine-index/_search
{
  "hits" : [
    {
      "_index": "lambda-kine-index",
      "_type": "_doc",
      "_id":
 "shardId-000000000000:495835116157626994950129608214214566865294366804960870 42",
```

```
      "_score": 1,
      "_source": {
        "timestamp": 1523648740.051,
        "message": "My test data.",
        "id":
 "shardId-000000000000:495835116157626994950129608214214566865294366804960870 42"
      }
    }
  ]
}
```

Loading streaming data from Amazon DynamoDB

You can use AWS Lambda to send data to your OpenSearch Service domain from Amazon DynamoDB. New data that arrives in the database table triggers an event notification to Lambda, which then runs your custom code to perform the indexing.

Prerequisites

Before proceeding, you must have the following resources.

Prerequisite	Description
DynamoDB table	The table contains your source data. For more information, see Basic Operations on DynamoDB Tables in the *Amazon DynamoDB Developer Guide*. The table must reside in the same Region as your OpenSearch Service domain and have a stream set to **New image**. To learn more, see Enabling a Stream.
OpenSearch Service domain	The destination for data after your Lambda function processes it. For more information, see the section called " Creating OpenSearch Service domains".

IAM role	This role must have basic OpenSearch Service, DynamoDB, and Lambda execution permissions, such as the following:
	
	`{`
	`  "Version": "2012-10-17",`
	`  "Statement": [`

Prerequisite	Description

```
    {
      "Effect": "Allow",
      "Action": [
        "es:ESHttpPost",
        "es:ESHttpPut",
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator",
        "dynamodb:ListStreams",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

The role must have the following trust relationship:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

To learn more, see Creating IAM roles in the *IAM User Guide*.

Create the Lambda function

Follow the instructions in the section called “Create the Lambda deployment package”, but create a directory named `ddb-to-opensearch` and use the following code for `sample.py`:

```
import boto3
import requests
from requests_aws4auth import AWS4Auth

region = '' # e.g. us-east-1
service = 'es'
credentials = boto3.Session().get_credentials()
awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region, service,
 session_token=credentials.token)

host = '' # the OpenSearch Service domain, e.g. https://search-mydomain.us-
west-1.es.amazonaws.com
index = 'lambda-index'
datatype = '_doc'
url = host + '/' + index + '/' + datatype + '/'

headers = { "Content-Type": "application/json" }

def handler(event, context):
    count = 0
    for record in event['Records']:
        # Get the primary key for use as the OpenSearch ID
        id = record['dynamodb']['Keys']['id']['S']

        if record['eventName'] == 'REMOVE':
            r = requests.delete(url + id, auth=awsauth)
        else:
            document = record['dynamodb']['NewImage']
            r = requests.put(url + id, auth=awsauth, json=document, headers=headers)
        count += 1
    return str(count) + ' records processed.'
```

Edit the variables for `region` and `host`.

Install pip if you haven't already, then use the following commands to install your dependencies:

```
cd ddb-to-opensearch

pip install --target ./package requests
pip install --target ./package requests_aws4auth
```

Then follow the instructions in the section called "Create the Lambda function", but specify the IAM role from the section called "Prerequisites" and the following settings for the trigger:

- **Table**: your DynamoDB table
- **Batch size**: 100
- **Starting position**: Trim horizon

To learn more, see Process New Items with DynamoDB Streams and Lambda in the *Amazon DynamoDB Developer Guide*.

At this point, you have a complete set of resources: a DynamoDB table for your source data, a DynamoDB stream of changes to the table, a function that runs after your source data changes and indexes those changes, and an OpenSearch Service domain for searching and visualization.

Test the Lambda function

After you create the function, you can test it by adding a new item to the DynamoDB table using the AWS CLI:

```
aws dynamodb put-item --table-name test --item '{"director": {"S": "Kevin
 Costner"},"id": {"S": "00001"},"title": {"S": "The Postman"}}' --region us-west-1
```

Then use the OpenSearch Service console or OpenSearch Dashboards to verify that `lambda-index` contains a document. You can also use the following request:

```
GET https://domain-name/lambda-index/_doc/00001
{
    "_index": "lambda-index",
    "_type": "_doc",
    "_id": "00001",
    "_version": 1,
    "found": true,
    "_source": {
        "director": {
            "S": "Kevin Costner"
        },
        "id": {
            "S": "00001"
        },
        "title": {
```

```
            "S": "The Postman"
        }
    }
}
```

Loading streaming data from Amazon Data Firehose

Firehose supports OpenSearch Service as a delivery destination. For instructions about how to load streaming data into OpenSearch Service, see Creating a Kinesis Data Firehose Delivery Stream and Choose OpenSearch Service for Your Destination in the *Amazon Data Firehose Developer Guide*.

Before you load data into OpenSearch Service, you might need to perform transforms on the data. To learn more about using Lambda functions to perform this task, see Amazon Kinesis Data Firehose Data Transformation in the same guide.

As you configure a delivery stream, Firehose features a "one-click" IAM role that gives it the resource access it needs to send data to OpenSearch Service, back up data on Amazon S3, and transform data using Lambda. Because of the complexity involved in creating such a role manually, we recommend using the provided role.

Loading streaming data from Amazon CloudWatch

You can load streaming data from CloudWatch Logs to your OpenSearch Service domain by using a CloudWatch Logs subscription. For information about Amazon CloudWatch subscriptions, see Real-time processing of log data with subscriptions. For configuration information, see Streaming CloudWatch Logs data to Amazon OpenSearch Service in the *Amazon CloudWatch Developer Guide*.

Loading streaming data from AWS IoT

You can send data from AWS IoT using rules. To learn more, see the OpenSearch action in the *AWS IoT Developer Guide*.

Loading data into Amazon OpenSearch Service with Logstash

The open source version of Logstash (Logstash OSS) provides a convenient way to use the bulk API to upload data into your Amazon OpenSearch Service domain. The service supports all standard Logstash input plugins, including the Amazon S3 input plugin. OpenSearch Service supports the logstash-output-opensearch output plugin, which supports both basic authentication and IAM credentials. The plugin works with version 8.1 and lower of Logstash OSS.

Configuration

Logstash configuration varies based on the type of authentication your domain uses.

No matter which authentication method you use, you must set `ecs_compatibility` to `disabled` in the output section of the configuration file. Logstash 8.0 introduced a breaking change where all plugins are run in ECS compatibility mode by default. You must override the default value to maintain legacy behavior.

Fine-grained access control configuration

If your OpenSearch Service domain uses fine-grained access control with HTTP basic authentication, configuration is similar to any other OpenSearch cluster. This example configuration file takes its input from the open source version of Filebeat (Filebeat OSS):

```
input {
  beats  {
    port => 5044
  }
}

output {
  opensearch {
    hosts       => "https://domain-endpoint:443"
    user        => "my-username"
    password    => "my-password"
    index       => "logstash-logs-%{+YYYY.MM.dd}"
    ecs_compatibility => disabled
    ssl_certificate_verification => false
  }
}
```

Configuration varies by Beats application and use case, but your Filebeat OSS configuration might look like this:

```
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /path/to/logs/dir/*.log
filebeat.config.modules:
```

```
  path: ${path.config}/modules.d/*.yml
  reload.enabled: false
setup.ilm.enabled: false
setup.ilm.check_exists: false
setup.template.settings:
  index.number_of_shards: 1
output.logstash:
  hosts: ["logstash-host:5044"]
```

IAM configuration

If your domain uses an IAM-based domain access policy or fine-grained access control with a master user, you must sign all requests to OpenSearch Service using IAM credentials. The following identity-based policy grants all HTTP requests to your domain's subresources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "es:ESHttp*"
      ],
      "Resource": "arn:aws:es:region:aws-account-id:domain/domain-name/*"
    }
  ]
}
```

To set up your Logstash configuration, change your configuration file to use the plugin for its output. This example configuration file takes its input from files in an S3 bucket:

```
input {
  s3 {
    bucket => "amzn-s3-demo-"
    region => "us-east-1"
  }
}

output {
  opensearch {
    hosts => ["domain-endpoint:443"]
    auth_type => {
```

```
      type => 'aws_iam'
      aws_access_key_id => 'your-access-key'
      aws_secret_access_key => 'your-secret-key'
      region => 'us-east-1'
      }
      index  => "logstash-logs-%{+YYYY.MM.dd}"
      ecs_compatibility => disabled
  }
}
```

If you don't want to provide your IAM credentials within the configuration file, you can export them (or run `aws configure`):

```
export AWS_ACCESS_KEY_ID="your-access-key"
export AWS_SECRET_ACCESS_KEY="your-secret-key"
export AWS_SESSION_TOKEN="your-session-token"
```

If your OpenSearch Service domain is in a VPC, the Logstash OSS machine must be able to connect to the VPC and have access to the domain through the VPC security groups. For more information, see the section called “About access policies on VPC domains”.

Searching data in Amazon OpenSearch Service

There are several common methods for searching documents in Amazon OpenSearch Service, including URI searches and request body searches. OpenSearch Service offers additional functionality that improves the search experience, such as custom packages, SQL support, and asynchronous search. For a comprehensive OpenSearch search API reference, see the OpenSearch documentation.

> **Note**
>
> The following sample requests work with OpenSearch APIs. Some requests might not work with older Elasticsearch versions.

Topics

URI searches

Universal Resource Identifier (URI) searches are the simplest form of search. In a URI search, you specify the query as an HTTP request parameter:

```
GET https://search-my-domain.us-west-1.es.amazonaws.com/_search?q=house
```

A sample response might look like the following:

```
{
  "took": 25,
  "timed_out": false,
  "_shards": {
    "total": 10,
    "successful": 10,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 85,
      "relation": "eq",
    },
    "max_score": 6.6137657,
    "hits": [
      {
        "_index": "movies",
        "_type": "movie",
        "_id": "tt0077975",
        "_score": 6.6137657,
        "_source": {
          "directors": [
            "John Landis"
          ],
          "release_date": "1978-07-27T00:00:00Z",
          "rating": 7.5,
          "genres": [
            "Comedy",
            "Romance"
          ],
          "image_url": "http://ia.media-imdb.com/images/M/
MV5BMTY2OTQxNTc1OF5BMl5BanBnXkFtZTYwNjA3NjI5._V1_SX400_.jpg",
```

```
          "plot": "At a 1962 College, Dean Vernon Wormer is determined to expel the
 entire Delta Tau Chi Fraternity, but those troublemakers have other plans for him.",
          "title": "Animal House",
          "rank": 527,
          "running_time_secs": 6540,
          "actors": [
            "John Belushi",
            "Karen Allen",
            "Tom Hulce"
          ],
          "year": 1978,
          "id": "tt0077975"
        }
      },
      ...
    ]
  }
}
```

By default, this query searches all fields of all indices for the term *house*. To narrow the search, specify an index (`movies`) and a document field (`title`) in the URI:

```
GET https://search-my-domain.us-west-1.es.amazonaws.com/movies/_search?q=title:house
```

You can include additional parameters in the request, but the supported parameters provide only a small subset of the OpenSearch search options. The following request returns 20 results (instead of the default of 10) and sorts by year (rather than by `_score`):

```
GET https://search-my-domain.us-west-1.es.amazonaws.com/movies/_search?
q=title:house&size=20&sort=year:desc
```

Request body searches

To perform more complex searches, use the HTTP request body and the OpenSearch domain-specific language (DSL) for queries. The query DSL lets you specify the full range of OpenSearch search options.

Note

You can't include Unicode special characters in a text field value, or the value will be parsed as multiple values separated by the special character. This incorrect parsing can lead to unintentional filtering of documents and potentially compromise control over their access. For more information, see A note on Unicode special characters in text fields in the OpenSearch documentation.

The following `match` query is similar to the final URI search example:

```
POST https://search-my-domain.us-west-1.es.amazonaws.com/movies/_search
{
  "size": 20,
  "sort": {
    "year": {
      "order": "desc"
    }
  },
  "query": {
    "query_string": {
      "default_field": "title",
      "query": "house"
    }
  }
}
```

Note

The `_search` API accepts HTTP `GET` and `POST` for request body searches, but not all HTTP clients support adding a request body to a `GET` request. `POST` is the more universal choice.

In many cases, you might want to search several fields, but not all fields. Use the `multi_match` query:

```
POST https://search-my-domain.us-west-1.es.amazonaws.com/movies/_search
{
  "size": 20,
  "query": {
```

```
    "multi_match": {
      "query": "house",
      "fields": ["title", "plot", "actors", "directors"]
    }
  }
}
```

Boosting fields

You can improve search relevancy by "boosting" certain fields. Boosts are multipliers that weigh matches in one field more heavily than matches in other fields. In the following example, a match for *john* in the `title` field influences `_score` twice as much as a match in the `plot` field and four times as much as a match in the `actors` or `directors` fields. The result is that films like *John Wick* and *John Carter* are near the top of the search results, and films starring John Travolta are near the bottom.

```
POST https://search-my-domain.us-west-1.es.amazonaws.com/movies/_search
{
  "size": 20,
  "query": {
    "multi_match": {
      "query": "john",
      "fields": ["title^4", "plot^2", "actors", "directors"]
    }
  }
}
```

Search result highlighting

The `highlight` option tells OpenSearch to return an additional object inside of the `hits` array if the query matched one or more fields:

```
POST https://search-my-domain.us-west-1.es.amazonaws.com/movies/_search
{
  "size": 20,
  "query": {
    "multi_match": {
      "query": "house",
      "fields": ["title^4", "plot^2", "actors", "directors"]
    }
  },
```

```
  "highlight": {
    "fields": {
      "plot": {}
    }
  }
}
```

If the query matched the content of the `plot` field, a hit might look like the following:

```
{
  "_index": "movies",
  "_type": "movie",
  "_id": "tt0091541",
  "_score": 11.276199,
  "_source": {
    "directors": [
      "Richard Benjamin"
    ],
    "release_date": "1986-03-26T00:00:00Z",
    "rating": 6,
    "genres": [
      "Comedy",
      "Music"
    ],
    "image_url": "http://ia.media-imdb.com/images/M/
MV5BMTIzODEzODE2OF5BMl5BanBnXkFtZTcwNjQ3ODcyMQ@@._V1_SX400_.jpg",
    "plot": "A young couple struggles to repair a hopelessly dilapidated house.",
    "title": "The Money Pit",
    "rank": 4095,
    "running_time_secs": 5460,
    "actors": [
      "Tom Hanks",
      "Shelley Long",
      "Alexander Godunov"
    ],
    "year": 1986,
    "id": "tt0091541"
  },
  "highlight": {
    "plot": [
      "A young couple struggles to repair a hopelessly dilapidated <em>house</em>."
    ]
  }
```

```
}
```

By default, OpenSearch wraps the matching string in tags, provides up to 100 characters of context around the match, and breaks content into sentences by identifying punctuation marks, spaces, tabs, and line breaks. All of these settings are customizable:

```
POST https://search-my-domain.us-west-1.es.amazonaws.com/movies/_search
{
  "size": 20,
  "query": {
    "multi_match": {
      "query": "house",
      "fields": ["title^4", "plot^2", "actors", "directors"]
    }
  },
  "highlight": {
    "fields": {
      "plot": {}
    },
    "pre_tags": "<strong>",
    "post_tags": "</strong>",
    "fragment_size": 200,
    "boundary_chars": ".,!? "
  }
}
```

Count API

If you're not interested in the contents of your documents and just want to know the number of matches, you can use the `_count` API instead of the `_search` API. The following request uses the `query_string` query to identify romantic comedies:

```
POST https://search-my-domain.us-west-1.es.amazonaws.com/movies/_count
{
  "query": {
    "query_string": {
      "default_field": "genres",
      "query": "romance AND comedy"
    }
  }
}
```

A sample response might look like the following:

```
{
  "count": 564,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  }
}
```

Paginating search results

If you need to display a large number of search results, you can implement pagination using several different methods.

Point in time

The point in time (PIT) feature is a type of search that lets you run different queries against a dataset that's fixed in time. This is the preferred pagination method in OpenSearch, especially for deep pagination. You can use PIT with OpenSearch Service version 2.5 and later. For more information about PIT, see ???.

The `from` and `size` parameters

The simplest way to paginate is with the `from` and `size` parameters. The following request returns results 20–39 of the zero-indexed list of search results:

```
POST https://search-my-domain.us-west-1.es.amazonaws.com/movies/_search
{
  "from": 20,
  "size": 20,
  "query": {
    "multi_match": {
      "query": "house",
      "fields": ["title^4", "plot^2", "actors", "directors"]
    }
  }
}
```

For more information about search pagination, see Paginate results in the OpenSearch documentation.

Dashboards Query Language

You can use the Dashboards Query Language (DQL) to search for data and visualizations in OpenSearch Dashboards. DQL uses four primary query types: *terms*, *Boolean*, *date and range*, and *nested field*.

Terms query

A terms query requires you to specify the term that you're searching for.

To perform a terms query, enter the following:

```
host:www.example.com
```

Boolean query

You can use the Boolean operators `AND`, `or`, and `not` to combine multiple queries.

To perform a Boolean query, paste the following:

```
host.keyword:www.example.com and response.keyword:200
```

Date and range query

You can use a date and range query to find a date before or after your query.

- > indicates a search for a date after your specified date.
- < indicates a search for a date before your specified date.

```
@timestamp > "2020-12-14T09:35:33"
```

Nested field query

If you have a document with nested fields, you have to specify which parts of the document that you want to retrieve. The following is a sample document that contains nested fields:

```
{"NBA players":[
    {"player-name": "Lebron James",
      "player-position": "Power forward",
      "points-per-game": "30.3"
    },
    {"player-name": "Kevin Durant",
      "player-position": "Power forward",
      "points-per-game": "27.1"
    },
    {"player-name": "Anthony Davis",
      "player-position": "Power forward",
      "points-per-game": "23.2"
    },
    {"player-name": "Giannis Antetokounmpo",
      "player-position": "Power forward",
      "points-per-game":"29.9"
    }
  ]
}
```

To retrieve a specific field using DQL, paste the following:

```
NBA players: {player-name: Lebron James}
```

To retrieve multiple objects from the nested document, paste the following:

```
NBA players: {player-name: Lebron James} and NBA players: {player-name: Giannis
 Antetokounmpo}
```

To search within a range, paste the following:

```
NBA players: {player-name: Lebron James} and NBA players: {player-name: Giannis
 Antetokounmpo and < 30}
```

If your document has an object nested within another object, you can still retrieve data by specifying all of the levels. To do this, paste the following:

```
Top-Power-forwards.NBA players: {player-name:Lebron James}
```

Custom packages for Amazon OpenSearch Service

Amazon OpenSearch Service lets you upload custom dictionary files, such as stop words and synonyms, and also provides several pre-packaged, optional plugins that you can associate with your domain. The generic term for both these types of files is *packages*.

Dictionary files improve your search results by telling OpenSearch to ignore certain high-frequency words or to treat terms like "frozen custard," "gelato," and "ice cream" as equivalent. They can also improve stemming, such as in the Japanese (kuromoji) Analysis plugin.

Optional plugins can provide added functionality to your domain. For example, you can use the Amazon Personalize plugin to give you personalized search results. Optional plugins use the `ZIP-PLUGIN` package type. For more information about optional plugins, see the section called “Plugins by engine version”.

Topics

Package permissions requirements

Users without administrator access require certain AWS Identity and Access Management (IAM) actions in order to manage packages:

- `es:CreatePackage` - create a package in an OpenSearch Service Region
- `es:DeletePackage` - delete a package from an OpenSearch Service Region
- `es:AssociatePackage` - associate a package to a domain
- `es:DissociatePackage` - dissociate a package from a domain

You also need permissions on the Amazon S3 bucket path or object where the custom package resides.

Grant all permission within IAM, not in the domain access policy. For more information, see the section called “Identity and Access Management”.

Uploading packages to Amazon S3

This section covers how to up upload custom dictionary packages, since optional plugin packages are already pre-installed. Before you can associate a custom dictionary with your domain, you must upload it to an Amazon S3 bucket. For instructions, see Uploading objects in the *Amazon Simple Storage Service User Guide*. Supported plugins don't need to be uploaded.

If your dictionary contains sensitive information, specify server-side encryption with S3-managed keys when you upload it. OpenSearch Service can't access files on S3 that you protect using an AWS KMS key.

After you upload the file, make note of its S3 path. The path format is `s3://`*`bucket-name`*`/`*`file-path`*`/`*`file-name`*.

You can use the following synonyms file for testing purposes. Save it as `synonyms.txt`.

```
danish, croissant, pastry
ice cream, gelato, frozen custard
sneaker, tennis shoe, running shoe
basketball shoe, hightop
```

Certain dictionaries, such as Hunspell dictionaries, use multiple files and require their own directories on the file system. At this time, OpenSearch Service only supports single-file dictionaries.

Importing and associating packages

The console is the simplest way to import a custom dictionary into OpenSearch Service. When you import a dictionary from Amazon S3, OpenSearch Service stores its own copy of the package and automatically encrypts that copy using AES-256 with OpenSearch Service-managed keys.

Optional plugins are already pre-installed in OpenSearch Service so you don't need to upload them yourself, but you do need to associate a plugin with a domain. Available plugins are listed on the **Packages** screen in the console.

Import and associate a package with a domain with the AWS Management Console

1. In the Amazon OpenSearch Service console, choose **Packages**.
2. Choose **Import package**.
3. Give the custom dictionary a descriptive name.
4. Provide the S3 path to the file, and then choose **Submit**.
5. Return to the **Packages** screen.
6. When the package status is **Available**, select it. Optional plugins will automatically be **Available**.
7. Choose **Associate to a domain**.
8. Select a domain, and then choose **Associate**.
9. In the navigation pane, choose your domain and go to the **Packages** tab.
10. If the package is a custom dictionary, note the ID when the package becomes **Available**. Use `analyzers/id` as the file path in requests to OpenSearch.

Alternately, use the AWS CLI, SDKs, or configuration API to import and associate packages. For more information, see the AWS CLI Command Reference and Amazon OpenSearch Service API Reference.

Using packages with OpenSearch

This section covers how to use both types of packages: custom dictionaries and optional plugins.

Using custom dictionaries

After you associate a file with a domain, you can use it in parameters such as `synonyms_path`, `stopwords_path`, and `user_dictionary` when you create tokenizers and token filters. The exact parameter varies by object. Several objects support `synonyms_path` and `stopwords_path`, but `user_dictionary` is exclusive to the kuromoji plugin.

For the IK (Chinese) Analysis plugin, you can upload a custom dictionary file as a custom package and associate it to a domain, and the plugin automatically picks it up without requiring a `user_dictionary` parameter. If your file is a synonyms file, use the `synonyms_path` parameter.

The following example adds a synonyms file to a new index:

```
PUT my-index
```

```
{
  "settings": {
    "index": {
      "analysis": {
        "analyzer": {
          "my_analyzer": {
            "type": "custom",
            "tokenizer": "standard",
            "filter": ["my_filter"]
          }
        },
        "filter": {
          "my_filter": {
            "type": "synonym",
            "synonyms_path": "analyzers/F111111111",
            "updateable": true
          }
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "description": {
        "type": "text",
        "analyzer": "standard",
        "search_analyzer": "my_analyzer"
      }
    }
  }
}
```

This request creates a custom analyzer for the index that uses the standard tokenizer and a synonym token filter.

- Tokenizers break streams of characters into *tokens* (typically words) based on some set of rules. The simplest example is the whitespace tokenizer, which breaks the preceding characters into a token each time it encounters a whitespace character. A more complex example is the standard tokenizer, which uses a set of grammar-based rules to work across many languages.
- Token filters add, modify, or delete tokens. For example, a synonym token filter adds tokens when it finds a word in the synonyms list. The stop token filter removes tokens when finds a word in the stop words list.

This request also adds a text field (`description`) to the mapping and tells OpenSearch to use the new analyzer as its search analyzer. You can see that it still uses the standard analyzer as its index analyzer.

Finally, note the line `"updateable": true` in the token filter. This field only applies to search analyzers, not index analyzers, and is critical if you later want to update the search analyzer automatically.

For testing purposes, add some documents to the index:

```
POST _bulk
{ "index": { "_index": "my-index", "_id": "1" } }
{ "description": "ice cream" }
{ "index": { "_index": "my-index", "_id": "2" } }
{ "description": "croissant" }
{ "index": { "_index": "my-index", "_id": "3" } }
{ "description": "tennis shoe" }
{ "index": { "_index": "my-index", "_id": "4" } }
{ "description": "hightop" }
```

Then search them using a synonym:

```
GET my-index/_search
{
  "query": {
    "match": {
      "description": "gelato"
    }
  }
}
```

In this case, OpenSearch returns the following response:

```
{
  "hits": {
    "total": {
      "value": 1,
      "relation": "eq"
    },
    "max_score": 0.99463606,
    "hits": [{
```

```
      "_index": "my-index",
      "_type": "_doc",
      "_id": "1",
      "_score": 0.99463606,
      "_source": {
        "description": "ice cream"
      }
    }]
  }
}
```

Tip

Dictionary files use Java heap space proportional to their size. For example, a 2 GiB dictionary file might consume 2 GiB of heap space on a node. If you use large files, ensure that your nodes have enough heap space to accommodate them. Monitor the `JVMMemoryPressure` metric, and scale your cluster as necessary.

Using optional plugins

OpenSearch Service lets you associate pre-installed, optional OpenSearch plugins to use with your domain. An optional plugin package is compatible with a specific OpenSearch version, and can only be associated to domains with that version. The list of available packages for your domain includes all supported plugins that are compatible with your domain version. After you associate a plugin to a domain, an installation process on the domain begins. Then, you can reference and use the plugin when you make requests to OpenSearch Service.

Associating and dissociating a plugin requires a blue/green deployment. For more information, see the section called “Changes that usually cause blue/green deployments”.

Optional plugins include language analyzers and customized search results. For example, the Amazon Personalize Search Ranking plugin uses machine learning to personalize search results for your customers. For more information about this plugin, see Personalizing search results from OpenSearch. For a list of all supported plugins, see the section called “Plugins by engine version”.

Sudachi plugin

For the Sudachi plugin, when you reassociate a dictionary file, it doesn't immediately reflect on the domain. The dictionary refreshes when the next blue/green deployment runs on the domain as

part of a configuration change or other update. Alternatively, you can create a new package with the updated data, create a new index using this new package, reindex the existing index to the new index, and then delete the old index. If you prefer to use the reindexing approach, use an index alias so that there's no disruption to your traffic.

Additionally, the Sudachi plugin only supports binary Sudachi dictionaries, which you can upload with the CreatePackage API operation. For information on the pre-built system dictionary and process for compiling user dictionaries, see the Sudachi documentation.

The following example demonstrates how to use system and user dictionaries with the Sudachi tokenizer. You must upload these dictionaries as custom packages with type `TXT-DICTIONARY` and provide their package IDs in the additional settings.

```
PUT sudachi_sample
{
  "settings": {
    "index": {
      "analysis": {
        "tokenizer": {
          "sudachi_tokenizer": {
            "type": "sudachi_tokenizer",
            "additional_settings": "{\"systemDict\": \"<system-dictionary-package-
id>\",\"userDict\": [\"<user-dictionary-package-id>\"]}"
        }
        },
        "analyzer": {
          "sudachi_analyzer": {
            "filter": ["my_searchfilter"],
            "tokenizer": "sudachi_tokenizer",
            "type": "custom"
          }
        },
        "filter":{
          "my_searchfilter": {
            "type": "sudachi_split",
            "mode": "search"
          }
        }
      }
    }
  }
}
```

Updating packages

This section only covers how to update a custom dictionary package, because optional plugin packages are already updated for you. Uploading a new version of a dictionary to Amazon S3 does *not* automatically update the package on Amazon OpenSearch Service. OpenSearch Service stores its own copy of the file, so if you upload a new version to S3, you must manually update it.

Each of your associated domains stores *its* own copy of the file, as well. To keep search behavior predictable, domains continue to use their current package version until you explicitly update them. To update a custom package, modify the file in Amazon S3 Control, update the package in OpenSearch Service, and then apply the update.

Update a package with the AWS Management Console

1. In the OpenSearch Service console, choose **Packages**.
2. Choose a package and **Update**.
3. Provide the S3 path to the file, and then choose **Update package**.
4. Return to the **Packages** screen.
5. When the package status changes to **Available**, select it. Then choose one or more associated domains, **Apply update**, and confirm. Wait for the association status to change to **Active**.
6. The next steps vary depending on how you configured your indices:

   - If your domain is running OpenSearch or Elasticsearch 7.8 or later, and only uses search analyzers with the updateable field set to true, you don't need to take any further action. OpenSearch Service automatically updates your indices using the _plugins/_refresh_search_analyzers API.
   - If your domain is running Elasticsearch 7.7 or earlier, uses index analyzers, or doesn't use the `updateable` field, see the section called “Manual index updates for dictionaries”.

Although the console is the simplest method, you can also use the AWS CLI, SDKs, or configuration API to update OpenSearch Service packages. For more information, see the AWS CLI Command Reference and Amazon OpenSearch Service API Reference.

Update a package with the AWS SDK

Instead of manually updating a package in the console, you can use the SDKs to automate the update process. The following sample Python script uploads a new package file to Amazon S3,

updates the package in OpenSearch Service, and applies the new package to the specified domain. After confirming the update was successful, it makes a sample call to OpenSearch demonstrating the new synonyms have been applied.

You must provide values for `host`, `region`, `file_name`, `bucket_name`, `s3_key`, `package_id`, `domain_name`, and `query`.

```
from requests_aws4auth import AWS4Auth
import boto3
import requests
import time
import json
import sys

host = ''  # The OpenSearch domain endpoint with https:// and a trailing slash. For
 example, https://my-test-domain.us-east-1.es.amazonaws.com/
region = ''  # For example, us-east-1
file_name = ''  # The path to the file to upload
bucket_name = ''  # The name of the S3 bucket to upload to
s3_key = ''  # The name of the S3 key (file name) to upload to
package_id = ''  # The unique identifier of the OpenSearch package to update
domain_name = ''  # The domain to associate the package with
query = ''  # A test query to confirm the package has been successfully updated

service = 'es'
credentials = boto3.Session().get_credentials()
client = boto3.client('opensearch')
awsauth = AWS4Auth(credentials.access_key, credentials.secret_key,
                   region, service, session_token=credentials.token)


def upload_to_s3(file_name, bucket_name, s3_key):
    """Uploads file to S3"""
    s3 = boto3.client('s3')
    try:
        s3.upload_file(file_name, bucket_name, s3_key)
        print('Upload successful')
        return True
    except FileNotFoundError:
        sys.exit('File not found. Make sure you specified the correct file path.')


def update_package(package_id, bucket_name, s3_key):
```

```
    """Updates the package in OpenSearch Service"""
    print(package_id, bucket_name, s3_key)
    response = client.update_package(
        PackageID=package_id,
        PackageSource={
            'S3BucketName': bucket_name,
            'S3Key': s3_key
        }
    )
    print(response)


def associate_package(package_id, domain_name):
    """Associates the package to the domain"""
    response = client.associate_package(
        PackageID=package_id, DomainName=domain_name)
    print(response)
    print('Associating...')


def wait_for_update(domain_name, package_id):
    """Waits for the package to be updated"""
    response = client.list_packages_for_domain(DomainName=domain_name)
    package_details = response['DomainPackageDetailsList']
    for package in package_details:
        if package['PackageID'] == package_id:
            status = package['DomainPackageStatus']
            if status == 'ACTIVE':
                print('Association successful.')
                return
            elif status == 'ASSOCIATION_FAILED':
                sys.exit('Association failed. Please try again.')
            else:
                time.sleep(10)  # Wait 10 seconds before rechecking the status
                wait_for_update(domain_name, package_id)


def sample_search(query):
    """Makes a sample search call to OpenSearch"""
    path = '_search'
    params = {'q': query}
    url = host + path
    response = requests.get(url, params=params, auth=awsauth)
    print('Searching for ' + '"' + query + '"')
```

```
print(response.text)
```

Note

If you receive a "package not found" error when you run the script using the AWS CLI, it likely means Boto3 is using whichever Region is specified in ~/.aws/config, which isn't the Region your S3 bucket is in. Either run `aws configure` and specify the correct Region, or explicitly add the Region to the client:

```
client = boto3.client('opensearch', region_name='us-east-1')
```

Manual index updates for dictionaries

Manual index updates only apply to custom dictionaries, not optional plugins. To use an updated dictionary, you must manually update your indexes if you meet any of the following conditions:

- Your domain runs Elasticsearch 7.7 or earlier.
- You use custom packages as index analyzers.
- You use custom packages as search analyzers, but don't include the updateable field.

To update analyzers with the new package files, you have two options:

- Close and open any indexes that you want to update:

```
POST my-index/_close
POST my-index/_open
```

- Reindex the indexes. First, create an index that uses the updated synonyms file (or an entirely new file). Note that only UTF-8 is supported.

```
PUT my-new-index
{
  "settings": {
    "index": {
      "analysis": {
        "analyzer": {
          "synonym_analyzer": {
            "type": "custom",
```

```
            "tokenizer": "standard",
            "filter": ["synonym_filter"]
          }
        },
        "filter": {
          "synonym_filter": {
            "type": "synonym",
            "synonyms_path": "analyzers/F222222222"
          }
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "description": {
        "type": "text",
        "analyzer": "synonym_analyzer"
      }
    }
  }
}
```

Then reindex the old index to that new index:

```
POST _reindex
{
  "source": {
    "index": "my-index"
  },
  "dest": {
    "index": "my-new-index"
  }
}
```

If you frequently update index analyzers, use index aliases to maintain a consistent path to the latest index:

```
POST _aliases
{
  "actions": [
    {
```

```
      "remove": {
        "index": "my-index",
        "alias": "latest-index"
      }
    },
    {
      "add": {
        "index": "my-new-index",
        "alias": "latest-index"
      }
    }
  ]
}
```

If you don't need the old index, delete it:

```
DELETE my-index
```

Dissociating and removing packages

Dissociating a package, whether it's a custom dictionary or optional plugin, from a domain means that you can no longer use that package when you create new indexes. After a package is dissociated, existing indexes that were using the package can no longer use it. You must remove the package from any index before you can dissociate it, otherwise the dissociation fails.

The console is the simplest way to dissociate a package from a domain and remove it from OpenSearch Service. Removing a package from OpenSearch Service does *not* remove it from its original location on Amazon S3.

Dissociate a package from a domain with the AWS Management Console

1. Go to https://aws.amazon.com, and then choose **Sign In to the Console**.
2. Under **Analytics**, choose **Amazon OpenSearch Service**.
3. In the navigation pane, choose your domain, and then choose the **Packages** tab.
4. Select a package, **Actions**, and then choose **Dissociate**. Confirm your choice.
5. Wait for the package to disappear from the list. You might need to refresh your browser.
6. If you want to use the package with other domains, stop here. To continue with removing the package (if it's a custom dictionary), choose **Packages** in the navigation pane.

7. Select the package and choose **Delete**.

Alternately, use the AWS CLI, SDKs, or configuration API to dissociate and remove packages. For more information, see the AWS CLI Command Reference and Amazon OpenSearch Service API Reference.

Querying your Amazon OpenSearch Service data with SQL

You can use SQL to query your Amazon OpenSearch Service, rather than using the JSON-based OpenSearch query DSL. Querying with SQL is useful if you're already familiar with the language or want to integrate your domain with an application that uses it. SQL support is available on domains running OpenSearch or Elasticsearch 6.5 or higher.

> **Note**
>
> This documentation describes version compatibility between OpenSearch Service and various versions of the SQL plugin, as well as the JDBC and ODBC driver. See the open source OpenSearch documentation for information about the syntax for basic and complex queries, functions, metadata queries, and aggregate functions.

Use the following table to find the version of the SQL plugin that's supported by each OpenSearch and Elasticsearch version.

OpenSearch

OpenSearch version	SQL plugin version	Notable features
2.13.0	2.13.0.0	
2.11.0	2.11.0.0	Add support for PPL language and queries
2.9.0	2.9.0.0	Add Spark connector, and support table and PromQL functions
2.7.0	2.7.0.0	Add `datasource` API
2.5.0	2.5.0.0	

OpenSearch version	SQL plugin version	Notable features
2.3.0	2.3.0.0	Add `maketime` and `makedate` datetime functions
1.3.0	1.3.0.0	Support default query limit size, and IN clause to select from within a value list
1.2.0	1.2.0.0	Add new protocol for visualization response format
1.1.0	1.1.0.0	Support match function as filter in SQL and PPL
1.0.0	1.0.0.0	Support querying a data stream

Open Distro for Elasticsearch

Elasticsearch version	SQL plugin version	Notable features
7.10	1.13.0	NULL FIRST and LAST for window functions, CAST() function, SHOW and DESCRIBE commands
7.9	1.11.0	Add additional date/time functions, ORDER BY keyword
7.8	1.9.0	
7.7	1.8.0	
7.3	1.3.0	Multiple string and number operators
7.1	1.1.0	

Sample call

To query your data with SQL, send HTTP requests to `_sql` using the following format:

```
POST domain-endpoint/_plugins/_sql
{
  "query": "SELECT * FROM my-index LIMIT 50"
```

```
}
```

> **Note**
>
> If your domain is running Elasticsearch rather than OpenSearch, the format is `_opendistro/_sql`.

Notes and differences

Calls to `_plugins/_sql` include index names in the request body, so they have the same access policy considerations as the bulk, mget, and msearch operations. As always, follow the principle of least privilege when you grant permissions to API operations.

For security considerations related to using SQL with fine-grained access control, see the section called "Fine-grained access control".

The OpenSearch SQL plugin includes many tunable settings. In OpenSearch Service, use the `_cluster/settings` path, not the plugin settings path (`_plugins/_query/settings`):

```
PUT _cluster/settings
{
  "transient" : {
    "plugins.sql.enabled" : true
  }
}
```

For legacy Elasticsearch domains, replace `plugins` with `opendistro`:

```
PUT _cluster/settings
{
  "transient" : {
    "opendistro.sql.enabled" : true
  }
}
```

SQL Workbench

The SQL Workbench is an OpenSearch Dashboards user interface that lets you run on-demand SQL queries, translate SQL into its REST equivalent, and view and save results as text, JSON, JDBC, or CSV. For more information, see Query Workbench.

SQL CLI

The SQL CLI is a standalone Python application that you can launch with the `opensearchsql` command. For steps to install, configure, and use, see SQL CLI.

JDBC driver

The Java Database Connectivity (JDBC) driver lets you integrate OpenSearch Service domains with your favorite business intelligence (BI) applications. To download the driver, click here. For more information, see the GitHub repository.

The following tables summarize version compatibility for the driver.

OpenSearch

OpenSearch version	JDBC driver version
2.13	1.1.0.1
2.11	1.1.0.1
2.9	1.1.0.1
2.7	1.1.0.1
2.5	1.1.0.1
2.3	1.1.0.1
1.3	1.1.0.1
1.2	1.1.0.1
1.1	1.1.0.1
1.0	1.1.0.1

Open Distro for Elasticsearch

Elasticsearch version	JDBC driver version
7.10	1.13.0
7.9	1.11.0
7.8	1.9.0
7.7	1.8.0
7.4	1.4.0
7.1	1.0.0
6.8	0.9.0
6.7	0.9.0
6.5	0.9.0

ODBC driver

The Open Database Connectivity (ODBC) driver is a read-only ODBC driver for Windows and macOS that lets you connect business intelligence and data visualization applications like Microsoft Excel to the SQL plugin.

You can download an example working driver file on the OpenSearch artifacts page. For information about installing the driver, see the SQL repository on GitHub.

k-Nearest Neighbor (k-NN) search in Amazon OpenSearch Service

Short for its associated *k-nearest neighbors* algorithm, k-NN for Amazon OpenSearch Service lets you search for points in a vector space and find the "nearest neighbors" for those points by Euclidean distance or cosine similarity. Use cases include recommendations (for example, an "other songs you might like" feature in a music application), image recognition, and fraud detection.

Note

This documentation describes version compatibility between OpenSearch Service and various versions of the k-NN plugin, as well as limitations when using the plugin with managed OpenSearch Service. For comprehensive documentation of the k-NN plugin, including simple and complex examples, parameter references, and the complete API reference for the plugin, see the open source OpenSearch documentation. The open source documentation also covers performance tuning and k-NN-specific cluster settings.

Use the following tables to find the version of the k-NN plugin running on your Amazon OpenSearch Service domain. Each k-NN plugin version corresponds to an OpenSearch or Elasticsearch version.

OpenSearch

OpenSearch version	k-NN plugin version	Notable features
2.13	2.13.0.0	
2.11	2.11.0.0	Added support for `ignore_unmapped` in k-NN queries
2.9	2.9.0.0	Implemented k-NN byte vectors and efficient filtering with the Faiss engine
2.7	2.7.0.0	
2.5	2.5.0.0	Extended SystemIndexPlugin for k-NN model system index, added Lucene-specific file extensions to core HybridFS
2.3	2.3.0.0	
1.3	1.3.0.0	
1.2	1.2.0.0	Added support for the Faiss library
1.1	1.1.0.0	

OpenSearch version	k-NN plugin version	Notable features
1.0	1.0.0.0	Renamed REST APIs while supporting backwards compatibility, renamed namespace from `opendistro` to `opensearch`

Elasticsearch

Elasticsearch version	k-NN plugin version	Notable features
7.1	1.3.0.0	Euclidean distance
7.4	1.4.0.0	
7.7	1.8.0.0	Cosine similarity
7.8	1.9.0.0	
7.9	1.11.0.0	Warmup API, custom scoring
7.10	1.13.0.0	Hamming distance, L1 Norm distance, Painless scripting

Getting started with k-NN

To use k-NN, you must create an index with the `index.knn` setting and add one or more fields of the `knn_vector` data type.

```
PUT my-index
{
  "settings": {
    "index.knn": true
  },
  "mappings": {
    "properties": {
      "my_vector1": {
        "type": "knn_vector",
        "dimension": 2
```

```
      },
      "my_vector2": {
        "type": "knn_vector",
        "dimension": 4
      }
    }
  }
}
```

The knn_vector data type supports a single list of up to 10,000 floats, with the number of floats defined by the required dimension parameter. After you create the index, add some data to it.

```
POST _bulk
{ "index": { "_index": "my-index", "_id": "1" } }
{ "my_vector1": [1.5, 2.5], "price": 12.2 }
{ "index": { "_index": "my-index", "_id": "2" } }
{ "my_vector1": [2.5, 3.5], "price": 7.1 }
{ "index": { "_index": "my-index", "_id": "3" } }
{ "my_vector1": [3.5, 4.5], "price": 12.9 }
{ "index": { "_index": "my-index", "_id": "4" } }
{ "my_vector1": [5.5, 6.5], "price": 1.2 }
{ "index": { "_index": "my-index", "_id": "5" } }
{ "my_vector1": [4.5, 5.5], "price": 3.7 }
{ "index": { "_index": "my-index", "_id": "6" } }
{ "my_vector2": [1.5, 5.5, 4.5, 6.4], "price": 10.3 }
{ "index": { "_index": "my-index", "_id": "7" } }
{ "my_vector2": [2.5, 3.5, 5.6, 6.7], "price": 5.5 }
{ "index": { "_index": "my-index", "_id": "8" } }
{ "my_vector2": [4.5, 5.5, 6.7, 3.7], "price": 4.4 }
{ "index": { "_index": "my-index", "_id": "9" } }
{ "my_vector2": [1.5, 5.5, 4.5, 6.4], "price": 8.9 }
```

Then you can search the data using the knn query type.

```
GET my-index/_search
{
  "size": 2,
  "query": {
    "knn": {
      "my_vector2": {
        "vector": [2, 3, 5, 6],
        "k": 2
```

```
      }
    }
  }
}
```

In this case, k is the number of neighbors you want the query to return, but you must also include the `size` option. Otherwise, you get k results for each shard (and each segment) rather than k results for the entire query. k-NN supports a maximum k value of 10,000.

If you mix the knn query with other clauses, you might receive fewer than k results. In this example, the `post_filter` clause reduces the number of results from 2 to 1.

```
GET my-index/_search
{
  "size": 2,
  "query": {
    "knn": {
      "my_vector2": {
        "vector": [2, 3, 5, 6],
        "k": 2
      }
    }
  },
  "post_filter": {
    "range": {
      "price": {
        "gte": 6,
        "lte": 10
      }
    }
  }
}
```

If you need to handle a large volume of queries while maintaining optimal performance, you can use the `_msearch` API to construct a bulk search with JSON and send a single request to perform multiple searches:

```
GET _msearch
{ "index": "my-index"}
{ "query": { "knn": {"my_vector2":{"vector": [2, 3, 5, 6],"k":2 }} } }
{ "index": "my-index", "search_type": "dfs_query_then_fetch"}
```

```
{ "query": { "knn": {"my_vector1":{"vector": [2, 3],"k":2 }} } }
```

The following video demonstrates how to set up bulk vector searches for K-NN queries.

k-NN differences, tuning, and limitations

OpenSearch lets you modify all k-NN settings using the `_cluster/settings` API. On OpenSearch Service, you can change all settings except `knn.memory.circuit_breaker.enabled` and `knn.circuit_breaker.triggered`. k-NN statistics are included as Amazon CloudWatch metrics.

In particular, check the `KNNGraphMemoryUsage` metric on each data node against the `knn.memory.circuit_breaker.limit` statistic and the available RAM for the instance type. OpenSearch Service uses half of an instance's RAM for the Java heap (up to a heap size of 32 GiB). By default, k-NN uses up to 50% of the remaining half, so an instance type with 32 GiB of RAM can accommodate 8 GiB of graphs (32 * 0.5 * 0.5). Performance can suffer if graph memory usage exceeds this value.

You can migrate a k-NN index created on version 2.x or later to UltraWarm or cold storage on a domain with version 2.17 or later.

Clear cache api and warmup apis for k-NN indices are blocked for warm indices. When the first query is initiated for the index, it downloads the graph files from Amazon S3 and loads the graph to memory. Similarly, when TTL is expired for the graphs, the files are automatically evicted from memory.

Cross-cluster search in Amazon OpenSearch Service

Cross-cluster search in Amazon OpenSearch Service lets you perform queries and aggregations across multiple connected domains. It often makes more sense to use multiple smaller domains instead of a single large domain, especially when you're running different types of workloads.

Workload-specific domains enable you to perform the following tasks:

- Optimize each domain by choosing instance types for specific workloads.
- Establish fault-isolation boundaries across workloads. This means that if one of your workloads fails, the fault is contained within that specific domain and doesn't impact your other workloads.
- Scale more easily across domains.

Cross-cluster search supports OpenSearch Dashboards, so you can create visualizations and dashboards across all your domains. You pay standard AWS data transfer charges for search results transferred between domains.

> **Note**
>
> Open source OpenSearch also has documentation for cross-cluster search. Setup differs significantly for open source clusters compared to managed Amazon OpenSearch Service domains. Most notably, in OpenSearch Service, you configure cross-cluster connections using the AWS Management Console rather than through cURL. In addition, the managed service uses AWS Identity and Access Management (IAM) for cross-cluster authentication in addition to fine-grained access control. Therefore, we recommend using this documentation, rather than the open source OpenSearch documentation, to configure cross-cluster search for your domains.

Topics

Limitations

Cross-cluster search has several important limitations:

- You can't connect an Elasticsearch domain to an OpenSearch domain.
- You can't connect to self-managed OpenSearch/Elasticsearch clusters.
- To connect domains across Regions, both domains must be on Elasticsearch 7.10 or later or OpenSearch.

- A domain can have a maximum of 20 outgoing connections. Similarly, a domain can have a maximum of 20 incoming connections. In other words, one domain can connect to a maximum of 20 other domains.
- The source domain must be on the same or a higher version than the destination domain. If you set up a bidirectional connection between two domains and you want to upgrade one or both of them, you must first delete one of the connections.
- You can't use custom dictionaries or SQL with cross-cluster search.
- You can't use AWS CloudFormation to connect domains.
- You can't use cross-cluster search on M3 or burstable (T2 and T3) instances.

Cross-cluster search prerequisites

Before you set up cross-cluster search, make sure that your domains meet the following requirements:

- Two OpenSearch domains, or Elasticsearch domains on version 6.7 or later
- Fine-grained access control enabled
- Node-to-node encryption enabled

Cross-cluster search pricing

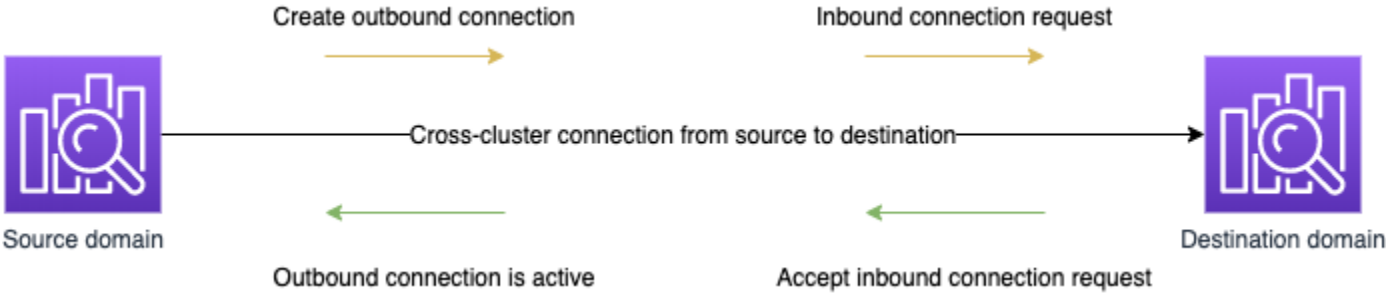
There is no additional charge for searching across domains.

Setting up a connection

The “source” domain refers to the domain that a cross-cluster search request originates from. In other words, the source domain is the one that you send the initial search request to.

The “destination” domain is the domain that the source domain queries.

A cross-cluster connection is unidirectional from the source to the destination domain. This means that the destination domain can’t query the source domain. However, you can set up another connection in the opposite direction.



The source domain creates an "outbound" connection to the destination domain. The destination domain receives an "inbound" connection request from the source domain.

To set up a connection

1. On your domain dashboard, choose a domain and go to the **Connections** tab.
2. In the **Outbound connections** section, choose **Request**.
3. For **Connection alias**, enter a name for your connection.
4. Choose between connecting to a domain in your AWS account and Region or in another account or Region.
   - To connect to a cluster in your AWS account and Region, select the domain from the dropdown menu and choose **Request**.
   - To connect to a cluster in another AWS account or Region, select the ARN of the remote domain and choose **Request**. To connect domains across Regions, both domains must be running Elasticsearch version 7.10 or later or OpenSearch.
5. To skip unavailable clusters for cluster queries, select **Skip unavailable**. This setting ensures that your cross-cluster queries return partial results despite failures on one or more remote clusters.
6. Cross-cluster search first validates the connection request to make sure the prerequisites are met. If the domains are found to be incompatible, the connection request enters the `Validation failed` state.
7. After the connection request is validated successfully, it is sent to the destination domain, where it needs to be approved. Until this approval happens, the connection remains in a `Pending acceptance` state. When the connection request is accepted at the destination domain, the state changes to `Active` and the destination domain becomes available for queries.

- The domain page shows you the overall domain health and instance health details of your destination domain. Only domain owners have the flexibility to create, view, remove, and monitor connections to or from their domains.

After the connection is established, any traffic that flows between the nodes of the connected domains is encrypted. If you connect a VPC domain to a non-VPC domain and the non-VPC domain is a public endpoint that can receive traffic from the internet, the cross-cluster traffic between the domains is still encrypted and secure.

Removing a connection

Removing a connection stops any cross-cluster operation on its indexes.

1. On your domain dashboard, go to the **Connections** tab.
2. Select the domain connections that you want to remove and choose **Delete**, then confirm deletion.

You can perform these steps on either the source or destination domain to remove the connection. After you remove the connection, it's still visible with a `Deleted` status for a period of 15 days.

You can't delete a domain with active cross-cluster connections. To delete a domain, first remove all incoming and outgoing connections from that domain. This ensures you take into account the cross-cluster domain users before deleting the domain.

Setting up security and sample walkthrough

1. You send a cross-cluster search request to the source domain.
2. The source domain evaluates that request against its domain access policy. Because cross-cluster search requires fine-grained access control, we recommend an open access policy on the source domain.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
```

```
          "*"
        ]
      },
      "Action": [
        "es:ESHttp*"
      ],
      "Resource": "arn:aws:es:region:account:domain/src-domain/*"
    }
  ]
}
```

> **Note**
>
> If you include remote indexes in the path, you must URL-encode the URI in the domain ARN. For example, use `arn:aws:es:us-east-1:123456789012:domain/my-domain/local_index,dst%3Aremote_index` rather than `arn:aws:es:us-east-1:123456789012:domain/my-domain/local_index,dst:remote_index`.

If you choose to use a restrictive access policy in addition to fine-grained access control, your policy must allow access to `es:ESHttpGet` at a minimum.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::123456789012:user/test-user"
        ]
      },
      "Action": "es:ESHttpGet",
      "Resource": "arn:aws:es:region:account:domain/src-domain/*"
    }
  ]
}
```

3. Fine-grained access control on the source domain evaluates the request:

   - Is the request signed with valid IAM or HTTP basic credentials?

- If so, does the user have permission to perform the search and access the data?

If the request only searches data on the destination domain (for example, `dest-alias:dest-index/_search`), you only need permissions on the destination domain.

If the request searches data on both domains (for example, `source-index,dest-alias:dest-index/_search`), you need permissions on both domains.

In fine-grained access control, users must have the `indices:admin/shards/search_shards` permission in addition to standard `read` or `search` permissions for the relevant indexes.

4. The source domain passes the request to the destination domain. The destination domain evaluates this request against its domain access policy. You must include the `es:ESCrossClusterGet` permission on the destination domain:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "es:ESCrossClusterGet",
      "Resource": "arn:aws:es:region:account:domain/dst-domain"
    }
  ]
}
```

Make sure that the `es:ESCrossClusterGet` permission is applied for `/dst-domain` and not `/dst-domain/*`.

However, this minimum policy only allows cross-cluster searches. To perform other operations, such as indexing documents and performing standard searches, you need additional permissions. We recommend the following policy on the destination domain:

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "*"
        ]
      },
      "Action": [
        "es:ESHttp*"
      ],
      "Resource": "arn:aws:es:region:account:domain/dst-domain/*"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "es:ESCrossClusterGet",
      "Resource": "arn:aws:es:region:account:domain/dst-domain"
    }
  ]
}
```

> **Note**
>
> All cross-cluster search requests between domains are encrypted in transit by default as part of node-to-node encryption.

5. The destination domain performs the search and returns the results to the source domain.
6. The source domain combines its own results (if any) with the results from the destination domain and returns them to you.
7. We recommend Postman for testing requests:

   - On the destination domain, index a document:

```
POST https://dst-domain.us-east-1.es.amazonaws.com/books/_doc/1

{
  "Dracula": "Bram Stoker"
}
```

- To query this index from the source domain, include the connection alias of the destination domain within the query.

```
GET https://src-domain.us-east-1.es.amazonaws.com/<connection_alias>:books/
_search

{
    ...
  "hits": [
    {
      "_index": "source-destination:books",
      "_type": "_doc",
      "_id": "1",
      "_score": 1,
      "_source": {
        "Dracula": "Bram Stoker"
      }
    }
  ]
}
```

  You can find the connection alias on the **Connections** tab on your domain dashboard.

- If you set up a connection between `domain-a -> domain-b` with connection alias `cluster_b` and `domain-a -> domain-c` with connection alias `cluster_c`, search `domain-a`, `domain-b`, and `domain-c` as follows:

```
GET https://src-domain.us-east-1.es.amazonaws.com/
local_index,cluster_b:b_index,cluster_c:c_index/_search
{
  "query": {
    "match": {
      "user": "domino"
    }
  }
}
```

  Response

```
{
  "took": 150,
  "timed_out": false,
```

```
  "_shards": {
    "total": 3,
    "successful": 3,
    "failed": 0,
    "skipped": 0
  },
  "_clusters": {
    "total": 3,
    "successful": 3,
    "skipped": 0
  },
  "hits": {
    "total": 3,
    "max_score": 1,
    "hits": [
      {
        "_index": "local_index",
        "_type": "_doc",
        "_id": "0",
        "_score": 1,
        "_source": {
          "user": "domino",
          "message": "Lets unite the new mutants",
          "likes": 0
        }
      },
      {
        "_index": "cluster_b:b_index",
        "_type": "_doc",
        "_id": "0",
        "_score": 2,
        "_source": {
          "user": "domino",
          "message": "I'm different",
          "likes": 0
        }
      },
      {
        "_index": "cluster_c:c_index",
        "_type": "_doc",
        "_id": "0",
        "_score": 3,
        "_source": {
          "user": "domino",
```

```
          "message": "So am I",
          "likes": 0
        }
      }
    ]
  }
}
```

If you did not choose to skip unavailable clusters in your connection setup, all destination clusters that you search must be available for your search request to run successfully. Otherwise, the whole request fails—even if one of the domains is not available, no search results are returned.

OpenSearch Dashboards

You can visualize data from multiple connected domains in the same way as from a single domain, except that you must access the remote indexes using `connection-alias:index`. So, your index pattern must match `connection-alias:index`.

Learning to Rank for Amazon OpenSearch Service

OpenSearch uses a probabilistic ranking framework called BM-25 to calculate relevance scores. If a distinctive keyword appears more frequently in a document, BM-25 assigns a higher relevance score to that document. This framework, however, doesn't take into account user behavior like click-through data, which can further improve relevance.

Learning to Rank is an open-source plugin that lets you use machine learning and behavioral data to tune the relevance of documents. It uses models from the XGBoost and Ranklib libraries to rescore the search results. The Elasticsearch LTR plugin was initially developed by OpenSource Connections, with significant contributions by Wikimedia Foundation, Snagajob Engineering, Bonsai, and Yelp Engineering. The OpenSearch version of the plugin is derived from the Elasticsearch LTR plugin.

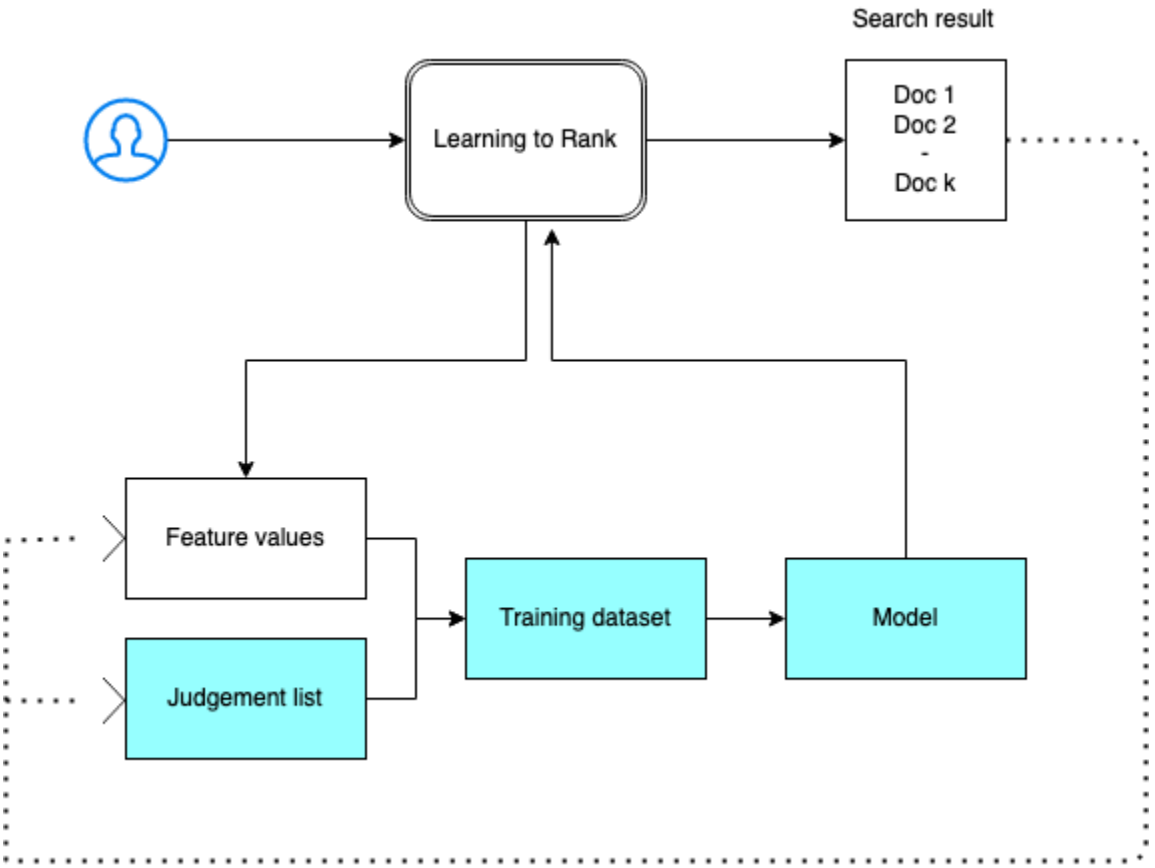
Learning to Rank requires OpenSearch or Elasticsearch 7.7 or later. To use the Learning to Rank plugin, you must have full admin permissions. To learn more, see the section called "Modifying the master user".

> **Note**
>
> This documentation provides a general overview of the Learning to Rank plugin and helps you get started using it. Full documentation, including detailed steps and API descriptions, is available in the Learning to Rank documentation.

Topics

Getting started with Learning to Rank

You need to provide a judgment list, prepare a training dataset, and train the model outside of Amazon OpenSearch Service. The parts in blue occur outside of OpenSearch Service:



Step 1: Initialize the plugin

To initialize the Learning to Rank plugin, send the following request to your OpenSearch Service domain:

```
PUT _ltr
```

```
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : ".ltrstore"
}
```

This command creates a hidden `.ltrstore` index that stores metadata information such as feature sets and models.

Step 2: Create a judgment list

> **Note**
>
> You must perform this step outside of OpenSearch Service.

A judgment list is a collection of examples that a machine learning model learns from. Your judgment list should include keywords that are important to you and a set of graded documents for each keyword.

In this example, we have a judgment list for a movie dataset. A grade of 4 indicates a perfect match. A grade of 0 indicates the worst match.

Grade	Keyword	Doc ID	Movie name
4	rambo	7555	Rambo
3	rambo	1370	Rambo III
3	rambo	1369	Rambo: First Blood Part II

Grade	Keyword	Doc ID	Movie name
3	rambo	1368	First Blood

Prepare your judgment list in the following format:

```
4 qid:1 # 7555 Rambo
3 qid:1 # 1370 Rambo III
3 qid:1 # 1369 Rambo: First Blood Part II
3 qid:1 # 1368 First Blood

where qid:1 represents "rambo"
```

For a more complete example of a judgment list, see movie judgments.

You can create this judgment list manually with the help of human annotators or infer it programmatically from analytics data.

Step 3: Build a feature set

A feature is a field that corresponds to the relevance of a document—for example, `title`, `overview`, `popularity score` (number of views), and so on.

Build a feature set with a Mustache template for each feature. For more information about features, see Working with Features.

In this example, we build a `movie_features` feature set with the `title` and `overview` fields:

```
POST _ltr/_featureset/movie_features
{
  "featureset" : {
      "name" : "movie_features",
      "features" : [
        {
          "name" : "1",
          "params" : [
            "keywords"
          ],
          "template_language" : "mustache",
          "template" : {
            "match" : {
```

```
              "title" : "{{keywords}}"
            }
          }
        },
        {
          "name" : "2",
          "params" : [
            "keywords"
          ],
          "template_language" : "mustache",
          "template" : {
            "match" : {
              "overview" : "{{keywords}}"
            }
          }
        }
      ]
    }
}
```

If you query the original `.ltrstore` index, you get back your feature set:

```
GET _ltr/_featureset
```

Step 4: Log the feature values

The feature values are the relevance scores calculated by BM-25 for each feature.

Combine the feature set and judgment list to log the feature values. For more information about logging features, see Logging Feature Scores.

In this example, the `bool` query retrieves the graded documents with the filter, and then selects the feature set with the `sltr` query. The `ltr_log` query combines the documents and the features to log the corresponding feature values:

```
POST tmdb/_search
{
  "_source": {
    "includes": [
      "title",
      "overview"
    ]
```

```
  },
  "query": {
    "bool": {
      "filter": [
        {
          "terms": {
            "_id": [
              "7555",
              "1370",
              "1369",
              "1368"
            ]
          }
        },
        {
          "sltr": {
            "_name": "logged_featureset",
            "featureset": "movie_features",
            "params": {
              "keywords": "rambo"
            }
          }
        }
      ]
    }
  },
  "ext": {
    "ltr_log": {
      "log_specs": {
        "name": "log_entry1",
        "named_query": "logged_featureset"
      }
    }
  }
}
```

A sample response might look like the following:

```
{
  "took" : 7,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
```

```
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 4,
      "relation" : "eq"
    },
    "max_score" : 0.0,
    "hits" : [
      {
        "_index" : "tmdb",
        "_type" : "movie",
        "_id" : "1368",
        "_score" : 0.0,
        "_source" : {
          "overview" : "When former Green Beret John Rambo is harassed by local law
enforcement and arrested for vagrancy, the Vietnam vet snaps, runs for the hills and
rat-a-tat-tats his way into the action-movie hall of fame. Hounded by a relentless
sheriff, Rambo employs heavy-handed guerilla tactics to shake the cops off his tail.",
          "title" : "First Blood"
        },
        "fields" : {
          "_ltrlog" : [
            {
              "log_entry1" : [
                {
                  "name" : "1"
                },
                {
                  "name" : "2",
                  "value" : 10.558305
                }
              ]
            }
          ]
        },
        "matched_queries" : [
          "logged_featureset"
        ]
      },
      {
        "_index" : "tmdb",
```

```
        "_type" : "movie",
        "_id" : "7555",
        "_score" : 0.0,
        "_source" : {
          "overview" : "When governments fail to act on behalf of captive missionaries,
ex-Green Beret John James Rambo sets aside his peaceful existence along the Salween
River in a war-torn region of Thailand to take action.  Although he's still haunted
by violent memories of his time as a U.S. soldier during the Vietnam War, Rambo can
hardly turn his back on the aid workers who so desperately need his help.",
          "title" : "Rambo"
        },
        "fields" : {
          "_ltrlog" : [
            {
              "log_entry1" : [
                {
                  "name" : "1",
                  "value" : 11.2569065
                },
                {
                  "name" : "2",
                  "value" : 9.936821
                }
              ]
            }
          ]
        },
        "matched_queries" : [
          "logged_featureset"
        ]
      },
      {
        "_index" : "tmdb",
        "_type" : "movie",
        "_id" : "1369",
        "_score" : 0.0,
        "_source" : {
          "overview" : "Col. Troutman recruits ex-Green Beret John Rambo for a highly
secret and dangerous mission. Teamed with Co Bao, Rambo goes deep into Vietnam to
rescue POWs. Deserted by his own team, he's left in a hostile jungle to fight for his
life, avenge the death of a woman and bring corrupt officials to justice.",
          "title" : "Rambo: First Blood Part II"
        },
        "fields" : {
```

```
          "_ltrlog" : [
            {
              "log_entry1" : [
                {
                  "name" : "1",
                  "value" : 6.334839
                },
                {
                  "name" : "2",
                  "value" : 10.558305
                }
              ]
            }
          ]
        },
        "matched_queries" : [
          "logged_featureset"
        ]
      },
      {
        "_index" : "tmdb",
        "_type" : "movie",
        "_id" : "1370",
        "_score" : 0.0,
        "_source" : {
          "overview" : "Combat has taken its toll on Rambo, but he's finally begun to
find inner peace in a monastery. When Rambo's friend and mentor Col. Trautman asks for
his help on a top secret mission to Afghanistan, Rambo declines but must reconsider
when Trautman is captured.",
          "title" : "Rambo III"
        },
        "fields" : {
          "_ltrlog" : [
            {
              "log_entry1" : [
                {
                  "name" : "1",
                  "value" : 9.425955
                },
                {
                  "name" : "2",
                  "value" : 11.262714
                }
              ]
```

```
            }
          ]
        },
        "matched_queries" : [
          "logged_featureset"
        ]
      }
    ]
  }
}
```

In the previous example, the first feature doesn't have a feature value because the keyword "rambo" doesn't appear in the title field of the document with an ID equal to 1368. This is a missing feature value in the training data.

Step 5: Create a training dataset

> **Note**
>
> You must perform this step outside of OpenSearch Service.

The next step is to combine the judgment list and feature values to create a training dataset. If your original judgment list looks like this:

```
4 qid:1 # 7555 Rambo
3 qid:1 # 1370 Rambo III
3 qid:1 # 1369 Rambo: First Blood Part II
3 qid:1 # 1368 First Blood
```

Convert it into the final training dataset, which looks like this:

```
4 qid:1 1:12.318474 2:10.573917 # 7555 rambo
3 qid:1 1:10.357875 2:11.950391 # 1370 rambo
3 qid:1 1:7.010513 2:11.220095 # 1369 rambo
3 qid:1 1:0.0 2:11.220095 # 1368 rambo
```

You can perform this step manually or write a program to automate it.

Step 6: Choose an algorithm and build the model

> **Note**
>
> You must perform this step outside of OpenSearch Service.

With the training dataset in place, the next step is to use XGBoost or Ranklib libraries to build a model. XGBoost and Ranklib libraries let you build popular models such as LambdaMART, Random Forests, and so on.

For steps to use XGBoost and Ranklib to build the model, see the XGBoost and RankLib documentation, respectively. To use Amazon SageMaker to build the XGBoost model, see XGBoost Algorithm.

Step 7: Deploy the model

After you have built the model, deploy it into the Learning to Rank plugin. For more information about deploying a model, see Uploading A Trained Model.

In this example, we build a `my_ranklib_model` model using the Ranklib library:

```
POST _ltr/_featureset/movie_features/_createmodel?pretty
{
  "model": {
    "name": "my_ranklib_model",
    "model": {
      "type": "model/ranklib",
      "definition": """## LambdaMART
## No. of trees = 10
## No. of leaves = 10
## No. of threshold candidates = 256
## Learning rate = 0.1
## Stop early = 100

<ensemble>
   <tree id="1" weight="0.1">
      <split>
         <feature>1</feature>
         <threshold>10.357875</threshold>
         <split pos="left">
            <feature>1</feature>
```

```
            <threshold>0.0</threshold>
            <split pos="left">
               <output>-2.0</output>
            </split>
            <split pos="right">
               <feature>1</feature>
               <threshold>7.010513</threshold>
               <split pos="left">
                  <output>-2.0</output>
               </split>
               <split pos="right">
                  <output>-2.0</output>
               </split>
            </split>
         </split>
         <split pos="right">
            <output>2.0</output>
         </split>
      </split>
   </tree>
   <tree id="2" weight="0.1">
      <split>
         <feature>1</feature>
         <threshold>10.357875</threshold>
         <split pos="left">
            <feature>1</feature>
            <threshold>0.0</threshold>
            <split pos="left">
               <output>-1.67031991481781</output>
            </split>
            <split pos="right">
               <feature>1</feature>
               <threshold>7.010513</threshold>
               <split pos="left">
                  <output>-1.67031991481781</output>
               </split>
               <split pos="right">
                  <output>-1.6703200340270996</output>
               </split>
            </split>
         </split>
         <split pos="right">
            <output>1.6703201532363892</output>
         </split>
```

```
      </split>
   </tree>
   <tree id="3" weight="0.1">
      <split>
         <feature>2</feature>
         <threshold>10.573917</threshold>
         <split pos="left">
            <output>1.479954481124878</output>
         </split>
         <split pos="right">
            <feature>1</feature>
            <threshold>7.010513</threshold>
            <split pos="left">
               <feature>1</feature>
               <threshold>0.0</threshold>
               <split pos="left">
                  <output>-1.4799546003341675</output>
               </split>
               <split pos="right">
                  <output>-1.479954481124878</output>
               </split>
            </split>
            <split pos="right">
               <output>-1.479954481124878</output>
            </split>
         </split>
      </split>
   </tree>
   <tree id="4" weight="0.1">
      <split>
         <feature>1</feature>
         <threshold>10.357875</threshold>
         <split pos="left">
            <feature>1</feature>
            <threshold>0.0</threshold>
            <split pos="left">
               <output>-1.3569872379302979</output>
            </split>
            <split pos="right">
               <feature>1</feature>
               <threshold>7.010513</threshold>
               <split pos="left">
                  <output>-1.3569872379302979</output>
               </split>
```

```
            <split pos="right">
               <output>-1.3569872379302979</output>
            </split>
         </split>
         <split pos="right">
            <output>1.3569873571395874</output>
         </split>
      </split>
   </tree>
   <tree id="5" weight="0.1">
      <split>
         <feature>1</feature>
         <threshold>10.357875</threshold>
         <split pos="left">
            <feature>1</feature>
            <threshold>0.0</threshold>
            <split pos="left">
               <output>-1.2721362113952637</output>
            </split>
            <split pos="right">
               <feature>1</feature>
               <threshold>7.010513</threshold>
               <split pos="left">
                  <output>-1.272136330604532</output>
               </split>
               <split pos="right">
                  <output>-1.272136330604532</output>
               </split>
            </split>
         </split>
         <split pos="right">
            <output>1.2721362113952637</output>
         </split>
      </split>
   </tree>
   <tree id="6" weight="0.1">
      <split>
         <feature>1</feature>
         <threshold>10.357875</threshold>
         <split pos="left">
            <feature>1</feature>
            <threshold>7.010513</threshold>
            <split pos="left">
```

```
            <feature>1</feature>
            <threshold>0.0</threshold>
            <split pos="left">
               <output>-1.2110036611557007</output>
            </split>
            <split pos="right">
               <output>-1.2110036611557007</output>
            </split>
         </split>
         <split pos="right">
            <output>-1.2110037803649902</output>
         </split>
      </split>
      <split pos="right">
         <output>1.2110037803649902</output>
      </split>
   </split>
</tree>
<tree id="7" weight="0.1">
   <split>
      <feature>1</feature>
      <threshold>10.357875</threshold>
      <split pos="left">
         <feature>1</feature>
         <threshold>7.010513</threshold>
         <split pos="left">
            <feature>1</feature>
            <threshold>0.0</threshold>
            <split pos="left">
               <output>-1.165616512298584</output>
            </split>
            <split pos="right">
               <output>-1.165616512298584</output>
            </split>
         </split>
         <split pos="right">
            <output>-1.165616512298584</output>
         </split>
      </split>
      <split pos="right">
         <output>1.165616512298584</output>
      </split>
   </split>
</tree>
```

```
<tree id="8" weight="0.1">
   <split>
      <feature>1</feature>
      <threshold>10.357875</threshold>
      <split pos="left">
         <feature>1</feature>
         <threshold>7.010513</threshold>
         <split pos="left">
            <feature>1</feature>
            <threshold>0.0</threshold>
            <split pos="left">
               <output>-1.131177544593811</output>
            </split>
            <split pos="right">
               <output>-1.131177544593811</output>
            </split>
         </split>
         <split pos="right">
            <output>-1.131177544593811</output>
         </split>
      </split>
      <split pos="right">
         <output>1.131177544593811</output>
      </split>
   </split>
</tree>
<tree id="9" weight="0.1">
   <split>
      <feature>2</feature>
      <threshold>10.573917</threshold>
      <split pos="left">
         <output>1.1046180725097656</output>
      </split>
      <split pos="right">
         <feature>1</feature>
         <threshold>7.010513</threshold>
         <split pos="left">
            <feature>1</feature>
            <threshold>0.0</threshold>
            <split pos="left">
               <output>-1.1046180725097656</output>
            </split>
            <split pos="right">
               <output>-1.1046180725097656</output>
```

```
               </split>
            </split>
            <split pos="right">
               <output>-1.1046180725097656</output>
            </split>
         </split>
      </split>
   </tree>
   <tree id="10" weight="0.1">
      <split>
         <feature>1</feature>
         <threshold>10.357875</threshold>
         <split pos="left">
            <feature>1</feature>
            <threshold>7.010513</threshold>
            <split pos="left">
               <feature>1</feature>
               <threshold>0.0</threshold>
               <split pos="left">
                  <output>-1.0838804244995117</output>
               </split>
               <split pos="right">
                  <output>-1.0838804244995117</output>
               </split>
            </split>
            <split pos="right">
               <output>-1.0838804244995117</output>
            </split>
         </split>
         <split pos="right">
            <output>1.0838804244995117</output>
         </split>
      </split>
   </tree>
</ensemble>
"""
    }
  }
}
```

To see the model, send the following request:

```
GET _ltr/_model/my_ranklib_model
```

Step 8: Search with learning to rank

After you deploy the model, you're ready to search.

Perform the `sltr` query with the features that you're using and the name of the model that you want to execute:

```
POST tmdb/_search
{
  "_source": {
    "includes": ["title", "overview"]
  },
  "query": {
    "multi_match": {
      "query": "rambo",
      "fields": ["title", "overview"]
    }
  },
  "rescore": {
    "query": {
      "rescore_query": {
        "sltr": {
          "params": {
            "keywords": "rambo"
          },
          "model": "my_ranklib_model"
        }
      }
    }
  }
}
```

With Learning to Rank, you see "Rambo" as the first result because we have assigned it the highest grade in the judgment list:

```
{
  "took" : 12,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
```

```
 },
 "hits" : {
   "total" : {
     "value" : 7,
     "relation" : "eq"
   },
   "max_score" : 13.096414,
   "hits" : [
     {
       "_index" : "tmdb",
       "_type" : "movie",
       "_id" : "7555",
       "_score" : 13.096414,
       "_source" : {
         "overview" : "When governments fail to act on behalf of captive missionaries,
ex-Green Beret John James Rambo sets aside his peaceful existence along the Salween
River in a war-torn region of Thailand to take action.  Although he's still haunted
by violent memories of his time as a U.S. soldier during the Vietnam War, Rambo can
hardly turn his back on the aid workers who so desperately need his help.",
         "title" : "Rambo"
       }
     },
     {
       "_index" : "tmdb",
       "_type" : "movie",
       "_id" : "1370",
       "_score" : 11.17245,
       "_source" : {
         "overview" : "Combat has taken its toll on Rambo, but he's finally begun to
find inner peace in a monastery. When Rambo's friend and mentor Col. Trautman asks for
his help on a top secret mission to Afghanistan, Rambo declines but must reconsider
when Trautman is captured.",
         "title" : "Rambo III"
       }
     },
     {
       "_index" : "tmdb",
       "_type" : "movie",
       "_id" : "1368",
       "_score" : 10.442155,
       "_source" : {
         "overview" : "When former Green Beret John Rambo is harassed by local law
enforcement and arrested for vagrancy, the Vietnam vet snaps, runs for the hills and
```

```
rat-a-tat-tats his way into the action-movie hall of fame. Hounded by a relentless
sheriff, Rambo employs heavy-handed guerilla tactics to shake the cops off his tail.",
          "title" : "First Blood"
        }
      },
      {
        "_index" : "tmdb",
        "_type" : "movie",
        "_id" : "1369",
        "_score" : 10.442155,
        "_source" : {
          "overview" : "Col. Troutman recruits ex-Green Beret John Rambo for a highly
secret and dangerous mission. Teamed with Co Bao, Rambo goes deep into Vietnam to
rescue POWs. Deserted by his own team, he's left in a hostile jungle to fight for his
life, avenge the death of a woman and bring corrupt officials to justice.",
          "title" : "Rambo: First Blood Part II"
        }
      },
      {
        "_index" : "tmdb",
        "_type" : "movie",
        "_id" : "31362",
        "_score" : 7.424202,
        "_source" : {
          "overview" : "It is 1985, and a small, tranquil Florida town is being rocked
by a wave of vicious serial murders and bank robberies. Particularly sickening to the
authorities is the gratuitous use of violence by two “Rambo” like killers who dress
themselves in military garb. Based on actual events taken from FBI files, the movie
depicts the Bureau’s efforts to track down these renegades.",
          "title" : "In the Line of Duty: The F.B.I. Murders"
        }
      },
      {
        "_index" : "tmdb",
        "_type" : "movie",
        "_id" : "13258",
        "_score" : 6.43182,
        "_source" : {
          "overview" : """Will Proudfoot (Bill Milner) is looking for an escape from
his family's stifling home life when he encounters Lee Carter (Will Poulter), the
school bully. Armed with a video camera and a copy of "Rambo: First Blood", Lee plans
to make cinematic history by filming his own action-packed video epic. Together, these
two newfound friends-turned-budding-filmmakers quickly discover that their imaginative
```

```
— and sometimes mishap-filled — cinematic adventure has begun to take on a life of its
own!""",
          "title" : "Son of Rambow"
        }
      },
      {
        "_index" : "tmdb",
        "_type" : "movie",
        "_id" : "61410",
        "_score" : 3.9719706,
        "_source" : {
          "overview" : "It's South Africa 1990. Two major events are about to happen:
The release of Nelson Mandela and, more importantly, it's Spud Milton's first year
at an elite boys only private boarding school. John Milton is a boy from an ordinary
background who wins a scholarship to a private school in Kwazulu-Natal, South Africa.
Surrounded by boys with nicknames like Gecko, Rambo, Rain Man and Mad Dog, Spud has
his hands full trying to adapt to his new home. Along the way Spud takes his first
tentative steps along the path to manhood. (The path it seems could be a rather long
road). Spud is an only child. He is cursed with parents from well beyond the lunatic
fringe and a senile granny. His dad is a fervent anti-communist who is paranoid that
the family domestic worker is running a shebeen from her room at the back of the
family home. His mom is a free spirit and a teenager's worst nightmare, whether it's
shopping for Spud's underwear in the local supermarket",
          "title" : "Spud"
        }
      }
    ]
  }
}
```

If you search without using the Learning to Rank plugin, OpenSearch returns different results:

```
POST tmdb/_search
{
  "_source": {
    "includes": ["title", "overview"]
  },
  "query": {
    "multi_match": {
      "query": "Rambo",
      "fields": ["title", "overview"]
    }
  }
```

```
}
```


```
{
  "took" : 5,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : 11.262714,
    "hits" : [
      {
        "_index" : "tmdb",
        "_type" : "movie",
        "_id" : "1370",
        "_score" : 11.262714,
        "_source" : {
          "overview" : "Combat has taken its toll on Rambo, but he's finally begun to
find inner peace in a monastery. When Rambo's friend and mentor Col. Trautman asks for
his help on a top secret mission to Afghanistan, Rambo declines but must reconsider
when Trautman is captured.",
          "title" : "Rambo III"
        }
      },
      {
        "_index" : "tmdb",
        "_type" : "movie",
        "_id" : "7555",
        "_score" : 11.2569065,
        "_source" : {
          "overview" : "When governments fail to act on behalf of captive missionaries,
ex-Green Beret John James Rambo sets aside his peaceful existence along the Salween
River in a war-torn region of Thailand to take action.  Although he's still haunted
by violent memories of his time as a U.S. soldier during the Vietnam War, Rambo can
hardly turn his back on the aid workers who so desperately need his help.",
          "title" : "Rambo"
```

```
      }
    },
    {
      "_index" : "tmdb",
      "_type" : "movie",
      "_id" : "1368",
      "_score" : 10.558305,
      "_source" : {
        "overview" : "When former Green Beret John Rambo is harassed by local law
enforcement and arrested for vagrancy, the Vietnam vet snaps, runs for the hills and
rat-a-tat-tats his way into the action-movie hall of fame. Hounded by a relentless
sheriff, Rambo employs heavy-handed guerilla tactics to shake the cops off his tail.",
        "title" : "First Blood"
      }
    },
    {
      "_index" : "tmdb",
      "_type" : "movie",
      "_id" : "1369",
      "_score" : 10.558305,
      "_source" : {
        "overview" : "Col. Troutman recruits ex-Green Beret John Rambo for a highly
secret and dangerous mission. Teamed with Co Bao, Rambo goes deep into Vietnam to
rescue POWs. Deserted by his own team, he's left in a hostile jungle to fight for his
life, avenge the death of a woman and bring corrupt officials to justice.",
        "title" : "Rambo: First Blood Part II"
      }
    },
    {
      "_index" : "tmdb",
      "_type" : "movie",
      "_id" : "13258",
      "_score" : 6.4600153,
      "_source" : {
        "overview" : """Will Proudfoot (Bill Milner) is looking for an escape from
his family's stifling home life when he encounters Lee Carter (Will Poulter), the
school bully. Armed with a video camera and a copy of "Rambo: First Blood", Lee plans
to make cinematic history by filming his own action-packed video epic. Together, these
two newfound friends-turned-budding-filmmakers quickly discover that their imaginative
– and sometimes mishap-filled – cinematic adventure has begun to take on a life of its
own!""",
        "title" : "Son of Rambow"
      }
    }
```

```
    ]
  }
}
```

Based on how well you think the model is performing, adjust the judgment list and features. Then, repeat steps 2–8 to improve the ranking results over time.

Learning to Rank API

Use the Learning to Rank operations to programmatically work with feature sets and models.

Create store

Creates a hidden `.ltrstore` index that stores metadata information such as feature sets and models.

```
PUT _ltr
```

Delete store

Deletes the hidden `.ltrstore` index and resets the plugin.

```
DELETE _ltr
```

Create feature set

Creates a feature set.

```
POST _ltr/_featureset/<name_of_features>
```

Delete feature set

Deletes a feature set.

```
DELETE _ltr/_featureset/<name_of_feature_set>
```

Get feature set

Retrieves a feature set.

```
GET _ltr/_featureset/<name_of_feature_set>
```

Create model

Creates a model.

```
POST _ltr/_featureset/<name_of_feature_set>/_createmodel
```

Delete model

Deletes a model.

```
DELETE _ltr/_model/<name_of_model>
```

Get model

Retrieves a model.

```
GET _ltr/_model/<name_of_model>
```

Get stats

Provides information about how the plugin is behaving.

```
GET _ltr/_stats
```

You can also use filters to retrieve a single stat:

```
GET _ltr/_stats/<stat>
```

Futthermore, you can limit the information to a single node in the cluster:

```
GET _ltr/_stats/<stat>/nodes/<nodeId>

{
  "_nodes" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
```

```
  },
  "cluster_name" : "873043598401:ltr-77",
  "stores" : {
    ".ltrstore" : {
      "model_count" : 1,
      "featureset_count" : 1,
      "feature_count" : 2,
      "status" : "green"
    }
  },
  "status" : "green",
  "nodes" : {
    "DjelK-_ZSfyzstO5dhGGQA" : {
      "cache" : {
        "feature" : {
          "eviction_count" : 0,
          "miss_count" : 0,
          "entry_count" : 0,
          "memory_usage_in_bytes" : 0,
          "hit_count" : 0
        },
        "featureset" : {
          "eviction_count" : 2,
          "miss_count" : 2,
          "entry_count" : 0,
          "memory_usage_in_bytes" : 0,
          "hit_count" : 0
        },
        "model" : {
          "eviction_count" : 2,
          "miss_count" : 3,
          "entry_count" : 1,
          "memory_usage_in_bytes" : 3204,
          "hit_count" : 1
        }
      },
      "request_total_count" : 6,
      "request_error_count" : 0
    }
  }
}
```

The statistics are provided at two levels, node and cluster, as specified in the following tables:

Node-level stats

Field name	Description
request_total_count	Total count of ranking requests.
request_error_count	Total count of unsuccessful requests.
cache	Statistics across all caches (features, featuresets, models). A cache hit occurs when a user queries the plugin and the model is already loaded into memory.
cache.eviction_count	Number of cache evictions.
cache.hit_count	Number of cache hits.
cache.miss_count	Number of cache misses. A cache miss occurs when a user queries the plugin and the model has not yet been loaded into memory.
cache.entry_count	Number of entries in the cache.
cache.memory_usage_in_bytes	Total memory used in bytes.
cache.cache_capacity_reached	Indicates if the cache limit is reached.

Cluster-level stats

Field name	Description
stores	Indicates where the feature sets and model metadata are stored. (The default is ".ltrstore". Otherwise, it's prefixed with ".ltrstore_", with a user supplied name).
stores.status	Status of the index.
stores.feature_sets	Number of feature sets.

Field name	Description
stores.features_count	Number of features.
stores.model_count	Number of models.
status	The plugin status based on the status of the feature store indices (red, yellow, or green) and circuit breaker state (open or closed).
cache.cache_capacity_reached	Indicates if the cache limit is reached.

Get cache stats

Returns statistics about the cache and memory usage.

```
GET _ltr/_cachestats

{
    "_nodes": {
        "total": 2,
        "successful": 2,
        "failed": 0
    },
    "cluster_name": "opensearch-cluster",
    "all": {
        "total": {
            "ram": 612,
            "count": 1
        },
        "features": {
            "ram": 0,
            "count": 0
        },
        "featuresets": {
            "ram": 612,
            "count": 1
        },
        "models": {
            "ram": 0,
            "count": 0
```

```
        }
    },
    "stores": {
        ".ltrstore": {
            "total": {
                "ram": 612,
                "count": 1
            },
            "features": {
                "ram": 0,
                "count": 0
            },
            "featuresets": {
                "ram": 612,
                "count": 1
            },
            "models": {
                "ram": 0,
                "count": 0
            }
        }
    },
    "nodes": {
        "ejF6uutERF20wOFNOXB61A": {
            "name": "opensearch1",
            "hostname": "172.18.0.4",
            "stats": {
                "total": {
                    "ram": 612,
                    "count": 1
                },
                "features": {
                    "ram": 0,
                    "count": 0
                },
                "featuresets": {
                    "ram": 612,
                    "count": 1
                },
                "models": {
                    "ram": 0,
                    "count": 0
                }
            }
```

```
        },
        "Z2RZNWRLSveVcz2c6lHf5A": {
            "name": "opensearch2",
            "hostname": "172.18.0.2",
            "stats": {
                ...
            }
        }
    }
}
```

Clear cache

Clears the plugin cache. Use this to refresh the model.

```
POST _ltr/_clearcache
```

Asynchronous search in Amazon OpenSearch Service

With asynchronous search for Amazon OpenSearch Service you can submit a search query that gets executed in the background, monitor the progress of the request, and retrieve results at a later stage. You can retrieve partial results as they become available before the search has completed. After the search finishes, save the results for later retrieval and analysis.

Asynchronous search requires OpenSearch 1.0 or later, or Elasticsearch 7.10 or later.

This documentation provides a brief overview of asynchronous search. It also discusses the limitations of using asynchronous search with a managed Amazon OpenSearch Service domain rather than an open source OpenSearch cluster. For full documentation of asynchronous search, including available settings, permissions, and a complete API reference, see Asynchronous search in the OpenSearch documentation.

Sample search call

To perform an asynchronous search, send HTTP requests to `_plugins/_asynchronous_search` using the following format:

```
POST opensearch-domain/_plugins/_asynchronous_search
```

Note

If you're using Elasticsearch 7.10 instead of an OpenSearch version, replace `_plugins` with `_opendistro` in all asynchronous search requests.

You can specify the following asynchronous search options:

Options	Description	Default value	Required
`wait_for_completion_timeout`	Specifies the amount of time that you plan to wait for the results. You can see whatever results you get within this time just like in a normal search. You can poll the remaining results based on an ID. The maximum value is 300 seconds.	1 second	No
`keep_on_completion`	Specifies whether you want to save the results in the cluster after the search is complete. You can examine the stored results at a later time.	`false`	No
`keep_alive`	Specifies the amount of time that the result is saved in the cluster. For example, 2d means that the results are stored in the cluster for 48 hours. The saved search results are deleted after this period or if the search is canceled. Note that this includes the query runtime. If the query overruns this time, the process cancels this query automatically.	12 hours	No

Sample request

```
POST _plugins/_asynchronous_search/?
pretty&size=10&wait_for_completion_timeout=1ms&keep_on_completion=true&request_cache=false
{
  "aggs": {
    "city": {
```

```
      "terms": {
        "field": "city",
        "size": 10
      }
    }
  }
}
```

> **Note**
>
> All request parameters that apply to a standard `_search` query are supported. If you're using Elasticsearch 7.10 instead of an OpenSearch version, replace `_plugins` with `_opendistro`.

Asynchronous search permissions

Asynchronous search supports fine-grained access control. For details on mixing and matching permissions to fit your use case, see Asynchronous search security.

For domains with fine-grained access control enabled, you need the following minimum permissions for a role:

```
# Allows users to use all asynchronous search functionality
asynchronous_search_full_access:
  reserved: true
  cluster_permissions:
    - 'cluster:admin/opensearch/asynchronous-search/*'
  index_permissions:
    - index_patterns:
        - '*'
      allowed_actions:
        - 'indices:data/read/search*'

# Allows users to read stored asynchronous search results
asynchronous_search_read_access:
  reserved: true
  cluster_permissions:
    - 'cluster:admin/opensearch/asynchronous-search/get'
```

For domains with fine-grained access control disabled, use your IAM access and secret key to sign all requests. You can access the results with the asynchronous search ID.

Asynchronous search settings

OpenSearch lets you change all available asynchronous search settings using the `_cluster/settings` API. In OpenSearch Service, you can only change the following settings:

- `plugins.asynchronous_search.node_concurrent_running_searches`
- `plugins.asynchronous_search.persist_search_failures`

Cross-cluster search

You can perform an asynchronous search across clusters with the following minor limitations:

- You can run an asynchronous search only on the source domain.
- You can't minimize network round trips as part of a cross-cluster search query.

If you set up a connection between `domain-a -> domain-b` with connection alias `cluster_b` and `domain-a -> domain-c` with connection alias `cluster_c`, asynchronously search `domain-a`, `domain-b`, and `domain-c` as follows:

```
POST https://src-domain.us-east-1.es.amazonaws.com/
local_index,cluster_b:b_index,cluster_c:c_index/_plugins/_asynchronous_search/?
pretty&size=10&wait_for_completion_timeout=500ms&keep_on_completion=true&request_cache=false
{
  "size": 0,
  "_source": {
    "excludes": []
  },
  "aggs": {
    "2": {
      "terms": {
        "field": "clientip",
        "size": 50,
        "order": {
          "_count": "desc"
        }
      }
    }
```

```
  },
  "stored_fields": [
    "*"
  ],
  "script_fields": {},
  "docvalue_fields": [
    "@timestamp"
  ],
  "query": {
    "bool": {
      "must": [
        {
          "query_string": {
            "query": "status:404",
            "analyze_wildcard": true,
            "default_field": "*"
          }
        },
        {
          "range": {
            "@timestamp": {
              "gte": 1483747200000,
              "lte": 1488326400000,
              "format": "epoch_millis"
            }
          }
        }
      ],
      "filter": [],
      "should": [],
      "must_not": []
    }
  }
}
```

Response

```
{
  "id" :
 "Fm9pYzJyVG91U19xb0hIQUJnMHJfRFEAAAAAAAknghQ1OWVBczNZQjVEa2dMYTBXaTdEagAAAAAAAAAB",
  "state" : "RUNNING",
  "start_time_in_millis" : 1609329314796,
  "expiration_time_in_millis" : 1609761314796
```

```
}
```

For more information, see the section called “Cross-cluster search”.

UltraWarm

Asynchronous searches with UltraWarm indexes continue to work. For more information, see the section called “UltraWarm storage”.

> **Note**
>
> You can monitor asynchronous search statistics in CloudWatch. For a full list of metrics, see the section called “Asynchronous search metrics”.

Point in time search in Amazon OpenSearch Service

Point in Time (PIT) is a type of search that lets you run different queries against a dataset that's fixed in time. Typically, when you run the same query on the same index at different points in time, you receive different results because documents are constantly indexed, updated, and deleted. With PIT, you can query against a constant state of your dataset.

The main use of PIT search is to couple it with `search_after` functionality. This is the preferred pagination method in OpenSearch, especially for deep pagination, because it operates on a dataset that is frozen in time, it is not bound to a query, and it supports consistent pagination going forward and backward. You can use PIT with a domain running OpenSearch version 2.5.

> **Note**
>
> This topic provides an overview of PIT and some things to consider when using it on a managed Amazon OpenSearch Service domain rather than a self-managed OpenSearch cluster. For full documentation of PIT, including a comprehensive API reference, see Point in Time in the open source OpenSearch documentation.

Considerations

Consider the following when you configure your PIT searches:

- If you're upgrading from domain running OpenSearch version 2.3 and need fine-grain access control on PIT actions, you need to manually add those actions and roles.
- There's no resiliency for PIT. Node reboot, node termination, blue/green deployments, and OpenSearch process restarts cause all PIT data to be lost.
- If a shard relocates during blue/green deployment, only live data segments are transferred to the new node. Segments of shards held by PIT (both exclusively and the one shared with lived data) remain on the old node.
- PIT searches currently don't work with asynchronous search.

Create a PIT

To run a PIT query, send HTTP requests to `_search/point_in_time` using the following format:

```
POST opensearch-domain/my-index/_search/point_in_time?keep_alive=time
```

You can specify the following PIT options:

Options	Description	Default value	Required
`keep_alive`	The amount of time to keep the PIT. Every time you access a PIT with a search request, the PIT lifetime is extended by the amount of time equal to the `keep_alive` parameter . This query parameter is required when you create a PIT, but optional in a search request.		Yes
`preference`	A string that specifies the node or the shard used to perform the search.	Random	No
`routing`	A string that specifies to route search requests to a specific shard.	The document's `_id`	No
`expand_wildcards`	A string that specifies type of index that can match the wildcard pattern. Supports	`open`	No

Options	Description	Default value	Required
	comma-separated values. Valid values are the following: • `all`: Match any index or data stream, including hidden ones. • `open`: Match open, non-hidden indexes or non-hidden data streams. • `closed`: Match closed, non-hidden indexes or non-hidden data streams. • `hidden`: Match hidden indexes or data streams. Must be combined with open, closed or both open and closed. • `none`: No wildcard patterns are accepted.		
`allow_partial_pit_creation`	A boolean that specifies whether to create a PIT with partial failures.	`true`	No

Sample response

```
{
    "pit_id":
 "o463QQEPbXktaW5kZXgtMDAwMDAxFnNOWU43ckt3U3IyaFVpbGE1UWEtMncAFjFyeXBsRGJmVFM2RTB6eVg1aVVqQncAA
    "_shards": {
        "total": 1,
        "successful": 1,
        "skipped": 0,
        "failed": 0
    },
    "creation_time": 1658146050064
}
```

When you create a PIT, you receive a PIT ID in the response. This is the ID that you use to perform searches with the PIT.

Point in time permissions

PIT supports fine-grained access control. If you're upgrading to an OpenSearch version 2.5 domain and need fine-grain access control, you need to manually create roles with the following permissions:

```
# Allows users to use all point in time search search functionality
point_in_time_full_access:
  reserved: true
  index_permissions:
    - index_patterns:
        - '*'
      allowed_actions:
        - "indices:data/read/point_in_time/create"
        - "indices:data/read/point_in_time/delete"
        - "indices:data/read/point_in_time/readall"
        - "indices:data/read/search"
        - "indices:monitor/point_in_time/segments"


# Allows users to use point in time search search functionality for specific index
# All type operations like list all PITs, delete all PITs are not supported in this
 case

point_in_time_index_access:
  reserved: true
  index_permissions:
    - index_patterns:
        - 'my-index-1'
      allowed_actions:
        - "indices:data/read/point_in_time/create"
        - "indices:data/read/point_in_time/delete"
        - "indices:data/read/search"
        - "indices:monitor/point_in_time/segments"
```

For domains with OpenSearch version 2.5 and above, you can use the built-in `point_in_time_full_access` role. For more information, see Security model in the OpenSearch documentation.

PIT settings

OpenSearch lets you change all available PIT settings using the `_cluster/settings` API. In OpenSearch Service, you can't currently modify settings.

Cross-cluster search

You can create PITs, search with PIT IDs, list PITs, and delete PITs across clusters with the following minor limitations:

- You can list all and delete all PITs only on the source domain.
- You can't minimize network round trips as part of a cross-cluster search query.

For more information, see the section called “Cross-cluster search”.

UltraWarm

PIT searches with UltraWarm indexes continue to work. For more information, see the section called “UltraWarm storage”.

> **Note**
>
> You can monitor PIT search statistics in CloudWatch. For a full list of metrics, see the section called “Point in time metrics”.

Semantic search in Amazon OpenSearch Service

Starting with OpenSearch version 2.9, you can use semantic search to help you understand search queries and improve search relevance. You can use semantic search in one of two ways – with neural search and with k-Nearest Neighbor (k-NN) search.

With OpenSearch Service, you can set up AI connectors for AWS services and external services. Using the console, you can also create an ML model with a AWS CloudFormation template. For more information, see the section called “CloudFormation template integrations”.

For full documentation of semantic search, including a step-by-step guide to use semantic search, see Semantic search in the open source OpenSearch documentation.

Concurrent segment search in Amazon OpenSearch Service

Starting with OpenSearch version 2.17, concurrent segment search uses a new setting to control concurrent search behavior.

- New domains created with version 2.17 have default concurrent segment search set to **auto** mode by default on nodes that are 2xl or above.
- Existing domains upgrading to 2.17 have default concurrent segment search set to **auto** based on instance type for all nodes that are 2xl or above, and if the overall CPU utilization of the cluster is below 45% in the past 1 week.
- For more information, see Concurrent segment search version 2.17.

Starting with OpenSearch version 2.13, you can use concurrent segment search to help you search segments in parallel during the query phase. For full documentation of concurrent segment search, see Concurrent segment search in the open source OpenSearch documentation. For information about Amazon CloudWatch metrics related to concurrent segment search, see Instance metrics and UltraWarm metrics.

There are a few additional limitations that apply when you use current segment search with Amazon OpenSearch Service:

- You can't enable concurrent segment search at an index level in OpenSearch Service.
- By default, OpenSearch Service uses a count of 2 slices with the max slice count mechanism.

Natural language query generation with OpenSearch

The natural language query generation feature in Amazon OpenSearch Service allows you to query your operational and security log data through natural language. OpenSearch is an ideal option to explore log data because it is a highly scalable and performant log analytics and search engine, and now you can use natural language to explore these logs. This feature allows you to identify issues without relying on OpenSearch Piped Processing Language (PPL) or having to look up data definitions when you build your queries. You can use the natural language query generation feature on OpenSearch Service domains with version 2.13 and later. You must have fine-grained access control enabled.

This feature was built with the OpenSearch Assistant Toolkit. If you want to create similar features that connect to your large language models, you can use the toolkit to configure your own agents and tools.

Prerequisites

Before you can use the natural language query generation feature, your domain must have the following:

- Version 2.13 or later.
- Service software R20240520-P4 or higher.
- Fine-grained access control enabled. For more information, see Enabling fine-grained access control.

Getting started

To start using the natural language query generation feature, make sure you have the feature enabled on your OpenSearch Service domain. This feature is enabled by default on all domains created with version 2.13 and later that have fine-grained access control enabled.

If you upgraded to OpenSearch version 2.13 before July 2, 2024, you must update your service software to (R20240520-P4) or later before you can enable natural language query generation. After you do this, you can enable the feature by selecting the **Enable natural language query generation box** checkbox under the **Artificial Intelligence (AI) and Machine Learning (ML)** section.

After you have your domain set up, navigate to the **Log Explorer** page in OpenSearch Dashboards. Choose **Event Explorer** and ask a question with the query assistant.

Configure permissions

If you enable natural language query generation on a preexisting OpenSearch Service domain, the **query_assistant_access** role might not be defined on the domain. Non-admin users must be mapped to this role in order to manage warm indexes on domains using fine-grained access control. To manually create the **query_assistant_access** role, perform the following steps:

1. In OpenSearch Dashboards, go to **Security** and choose **Roles**.
2. Choose **Create role** and configure the following cluster permissions:

- `cluster:admin/opensearch/ml/config/get`
- `cluster:admin/opensearch/ml/execute`
- `cluster:admin/opensearch/ml/predict`
- `cluster:admin/opensearch/ppl`

3. Name the role **query_assistant_access**.
4. Choose **Create role**. The **query_assistant_access** role is now available.

> **Note**
>
> You must also have the `indices:admin/mappings/get` and `read` index permissions for the indices that you want to use natural language questions with.

Configuration automation

Flow Framework is an OpenSearch plugin that provides a way to automate OpenSearch configurations for use cases such as query generation and conversational chat. Because the plugin tracks the resources that enable the natural language query generation feature, the flow framework index stores a template for each domain that uses query assist.

Flow Framework allows you to either select from a set of predefined templates, or create your own automations for machine learning connectors, tools, agents, and other components that prepare OpenSearch as a backend for generative models.

Using Dashboards (co-located with cluster) with Amazon OpenSearch Service

Dashboards (co-located with cluster) is an open-source visualization tool designed to work with OpenSearch. Amazon OpenSearch Service provides an installation of Dashboards with every OpenSearch Service domain. Dashboards runs on the hot data nodes in the domain. If you are looking for documentation on the new centralized OpenSearch user interface that supports multiple data sources at one endpoint, see Centralized OpenSearch UI (Dashboards).

You can find a link to Dashboards on your domain dashboard in the OpenSearch Service console. For domains running OpenSearch, the URL is *domain-endpoint*/_dashboards/. For domains running legacy Elasticsearch, the URL is *domain-endpoint*/_plugin/kibana.

Queries using this default Dashboards installation have a 300-second timeout.

> **Note**
>
> This documentation discusses Dashboards in the context of Amazon OpenSearch Service, including different ways to connect to it. For comprehensive documentation, including a getting started guide, instruction to create a dashboard, dashboards management, and Dashboards Query Language (DQL), see OpenSearch Dashboards in the open source OpenSearch documentation.

Controlling access to Dashboards

Dashboards does not natively support IAM users and roles, but OpenSearch Service offers several solutions for controlling access to Dashboards:

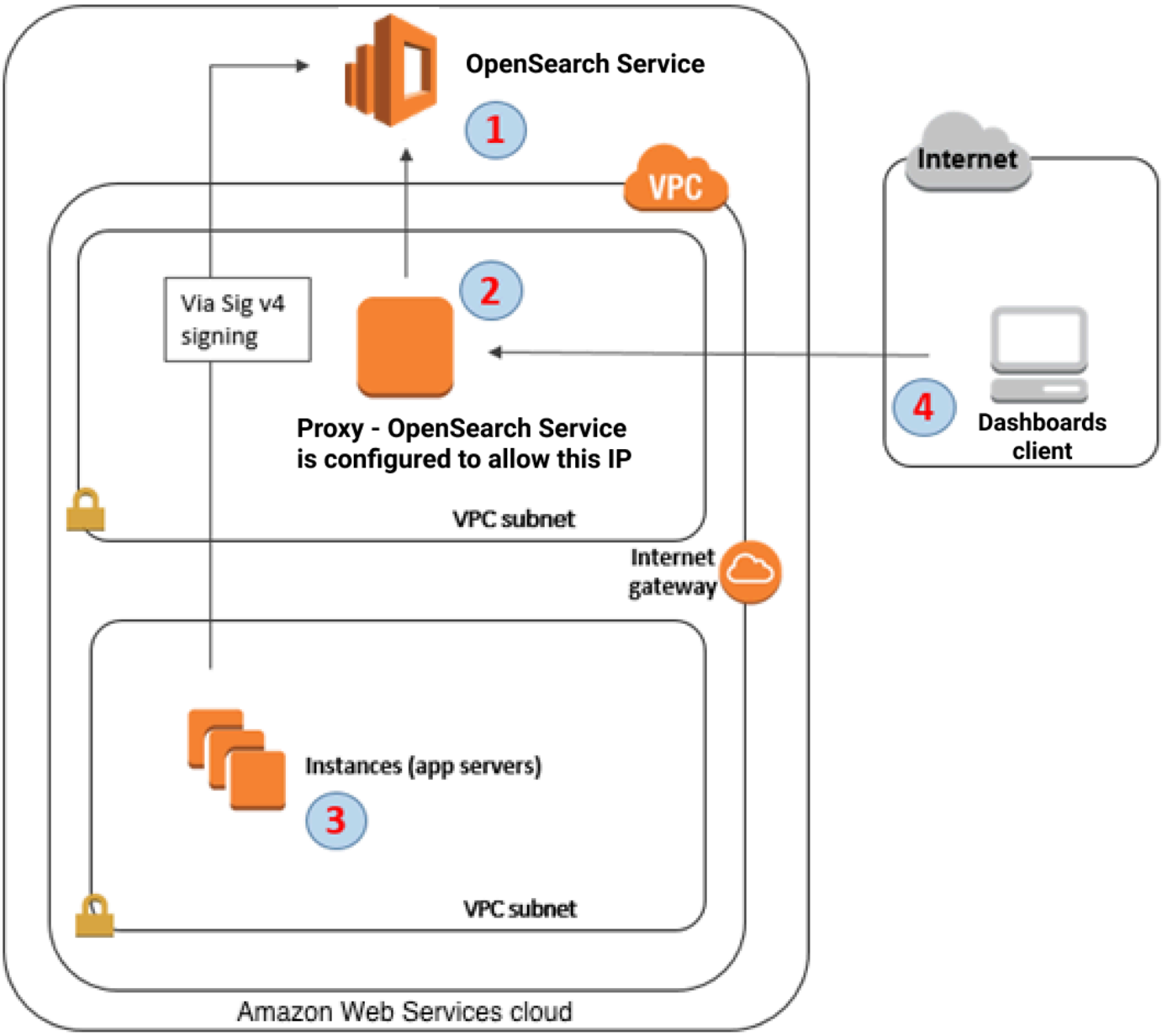
- Enable SAML authentication for Dashboards.
- Use fine-grained access control with HTTP basic authentication.
- Configure Cognito authentication for Dashboards.
- For public access domains, configure an IP-based access policy that either uses or does not use a proxy server.

- For VPC access domains, use an open access policy that either uses or does not use a proxy server, and security groups to control access. To learn more, see the section called “About access policies on VPC domains”.

Using a proxy to access OpenSearch Service from Dashboards

> **Note**
>
> This process is only applicable if your domain uses public access and you don't want to use Cognito authentication. See the section called “Controlling access to Dashboards ”.

Because Dashboards is a JavaScript application, requests originate from the user's IP address. IP-based access control might be impractical due to the sheer number of IP addresses you would need to allow in order for each user to have access to Dashboards. One workaround is to place a proxy server between Dashboards and OpenSearch Service. Then you can add an IP-based access policy that allows requests from only one IP address, the proxy's. The following diagram shows this configuration.



1. This is your OpenSearch Service domain. IAM provides authorized access to this domain. An additional, IP-based access policy provides access to the proxy server.
2. This is the proxy server, running on an Amazon EC2 instance.
3. Other applications can use the Signature Version 4 signing process to send authenticated requests to OpenSearch Service.
4. Dashboards clients connect to your OpenSearch Service domain through the proxy.

To enable this sort of configuration, you need a resource-based policy that specifies roles and IP addresses. Here's a sample policy:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Resource":"arn:aws:es:us-west-2:111111111111:domain/my-domain/*",
         "Principal":{
            "AWS":"arn:aws:iam::111111111111:role/allowedrole1"
         },
         "Action":[
            "es:ESHttpGet"
         ],
         "Effect":"Allow"
      },
      {
         "Effect":"Allow",
         "Principal":{
            "AWS":"*"
         },
         "Action":"es:*",
         "Condition":{
            "IpAddress":{
               "aws:SourceIp":[
                  "203.0.113.0/24",
                  "2001:DB8:1234:5678::/64"
               ]
            }
         },
         "Resource":"arn:aws:es:us-west-2:111111111111:domain/my-domain/*"
      }
   ]
}
```

We recommend that you configure the EC2 instance running the proxy server with an Elastic IP address. This way, you can replace the instance when necessary and still attach the same public IP address to it. To learn more, see Elastic IP Addresses in the *Amazon EC2 User Guide*.

If you use a proxy server *and* Cognito authentication, you might need to add settings for Dashboards and Amazon Cognito to avoid `redirect_mismatch` errors. See the following `nginx.conf` example:

```
server {
    listen 443;
    server_name $host;
    rewrite ^/$ https://$host/_plugin/_dashboards redirect;

    ssl_certificate           /etc/nginx/cert.crt;
    ssl_certificate_key       /etc/nginx/cert.key;

    ssl on;
    ssl_session_cache  builtin:1000  shared:SSL:10m;
    ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers HIGH:!aNULL:!eNULL:!EXPORT:!CAMELLIA:!DES:!MD5:!PSK:!RC4;
    ssl_prefer_server_ciphers on;

    location /_plugin/_dashboards {
        # Forward requests to Dashboards
        proxy_pass https://$dashboards_host/_plugin/_dashboards;

        # Handle redirects to Cognito
        proxy_redirect https://$cognito_host https://$host;

        # Update cookie domain and path
        proxy_cookie_domain $dashboards_host $host;
        proxy_cookie_path / /_plugin/_dashboards/;

        # Response buffer settings
        proxy_buffer_size 128k;
        proxy_buffers 4 256k;
        proxy_busy_buffers_size 256k;
    }

    location ~ \/(log|sign|fav|forgot|change|saml|oauth2) {
        # Forward requests to Cognito
        proxy_pass https://$cognito_host;

        # Handle redirects to Dashboards
        proxy_redirect https://$dashboards_host https://$host;

        # Update cookie domain
        proxy_cookie_domain $cognito_host $host;
    }
}
```

Note

(Optional) If you choose to provision a dedicated coordinator node, it will automatically start hosting OpenSearch Dashboards. As a result, the availability of data node resources such as CPU and memory is increased. This increased availability of data node resources can help to improve the overall resiliency of your domain.

Configuring Dashboards to use a WMS map server

The default installation of Dashboards for OpenSearch Service includes a map service, except for domains in the India and China Regions. The map service supports up to 10 zoom levels.

Regardless of your Region, you can configure Dashboards to use a different Web Map Service (WMS) server for coordinate map visualizations. Region map visualizations only support the default map service.

To configure Dashboards to use a WMS map server:

1. Open Dashboards.
2. Choose **Stack Management**.
3. Choose **Advanced Settings**.
4. Locate **visualization:tileMap:WMSdefaults**.
5. Change `enabled` to `true` and `url` to the URL of a valid WMS map server:

```
{
  "enabled": true,
  "url": "wms-server-url",
  "options": {
    "format": "image/png",
    "transparent": true
  }
}
```

6. Choose **Save changes**.

To apply the new default value to visualizations, you might need to reload Dashboards. If you have saved visualizations, choose **Options** after opening the visualization. Verify that **WMS map server** is enabled and **WMS url** contains your preferred map server, and then choose **Apply changes**.

> **Note**
>
> Map services often have licensing fees or restrictions. You are responsible for all such considerations on any map server that you specify. You might find the map services from the U.S. Geological Survey useful for testing.

Connecting a local Dashboards server to OpenSearch Service

If you already invested significant time into configuring your own Dashboards instance, you can use it instead of (or in addition to) the default Dashboards instance that OpenSearch Service provides. The following procedure works for domains that use fine-grained access control with an open access policy.

To connect a local Dashboards server to OpenSearch Service

1. On your OpenSearch Service domain, create a user with the appropriate permissions:
   a. In Dashboards, go to **Security**, **Internal users**, and choose **Create internal user**.
   b. Provide a username and password and choose **Create**.
   c. Go to **Roles** and select a role.
   d. Select **Mapped users** and choose **Manage mapping**.
   e. In **Users**, add your username and choose **Map**.
2. Download and install the appropriate version of the OpenSearch security plugin on your self-managed Dashboards OSS installation.
3. On your local Dashboards server, open the `config/opensearch_dashboards.yml` file and add your OpenSearch Service endpoint with the username and password you created earlier:

   ```
   opensearch.hosts: ['https://domain-endpoint']
   opensearch.username: 'username'
   opensearch.password: 'password'
   ```

   You can use the following sample `opensearch_dashboards.yml` file:

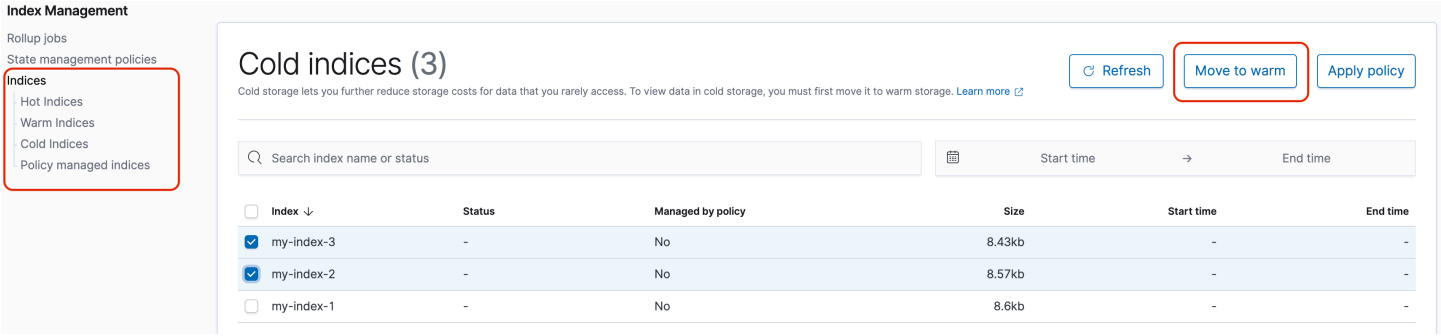
```
server.host: '0.0.0.0'

opensearch.hosts: ['https://domain-endpoint']

opensearchDashboards.index: ".username"

opensearch.ssl.verificationMode: none # if not using HTTPS

opensearch_security.auth.type: basicauth
opensearch_security.auth.anonymous_auth_enabled: false
opensearch_security.cookie.secure: false # set to true when using HTTPS
opensearch_security.cookie.ttl: 3600000
opensearch_security.session.ttl: 3600000
opensearch_security.session.keepalive: false
opensearch_security.multitenancy.enabled: false
opensearch_security.readonly_mode.roles: ['opensearch_dashboards_read_only']
opensearch_security.auth.unauthenticated_routes: []
opensearch_security.basicauth.login.title: 'Please log in using your username and
 password'

opensearch.username: 'username'
opensearch.password: 'password'
opensearch.requestHeadersWhitelist: [authorization, securitytenant,
 security_tenant]
```

To see your OpenSearch Service indexes, start your local Dashboards server, go to **Dev Tools** and run the following command:

```
GET _cat/indices
```

Managing indexes in Dashboards

The Dashboards installation on your OpenSearch Service domain provides a useful UI for managing indexes in different storage tiers on your domain. Choose **Index Management** from the Dashboards main menu to view all indexes in hot, UltraWarm, and cold storage, as well as indexes managed by Index State Management (ISM) policies. Use index management to move indexes between warm and cold storage, and to monitor migrations between the three tiers.



Note that you won't see the hot, warm, and cold index options unless you have UltraWarm and/or cold storage enabled.

Additional features

The default Dashboards installation on each OpenSearch Service domain has some additional features:

- User interfaces for the various OpenSearch plugins
- Tenants
- Reports

  Use the **Reporting** menu to generate on-demand CSV reports from the Discover page and PDF or PNG reports of dashboards or visualizations. CSV reports have a 10,000 row limit.
- Gantt charts
- Notebooks

Centralized OpenSearch user interface (Dashboards) with Amazon OpenSearch Service

OpenSearch user interface is the modernized operational analytics experience for Amazon OpenSearch Service. In comparison to the existing OpenSearch Dashboards that is hosted in individual domains or collections and supports only one data source, OpenSearch user interface is created as a web-based application and runs in the AWS cloud, so that it can be associated with data sources across multiple managed clusters, serverless collections, and connected AWS data sources such as Amazon S3. With OpenSearch user interface, you can gain a comprehensive insights across your data in a unified interface. If you are looking for documentation on OpenSearch Dashboards which are co-located with each managed cluster or collection, see Dashboards (co-located with cluster).

OpenSearch user interface introduces the concept of workspaces. A workspace is a tailored experience for common use cases such as observability and security analytics. You can create one workspace for each of your use cases or teams, and manage collaborators and data sources associated with each workspace, so that you can easily manage the access control and collaboration across your teams. In OpenSearch user interface, Discover provides a unified log exploration experience supporting popular languages like SQL and Piped-Processing-Language (PPL), in addition to offering existing support for DQL and Lucene.

To use OpenSearch user interface, you can create an OpenSearch UI application from the AWS Management Console or via the AWS Command Line Interface (CLI). You can find the list of your created OpenSearch applications under your Amazon OpenSearch Service console, Central Management section. Each OpenSearch application has its own endpoint URL and Amazon Resource Name (ARN). You can use the endpoint URL to open the OpenSearch application, and easily share it to your colleagues for collaboration. You can configure each OpenSearch application to support login with AWS Identity and Access Management Identity and Access Management (IAM) credentials and/or IAM Identity Center, and manage the user and group permissions to the application.

Note

OpenSearch user interface (Dashboards) application does not support use of IAM Identity Center applications created in a different region. To use IAM Identity Center, create

OpenSearch application in the same region as your IAM Identity Center application instance.

Creating an OpenSearch application

To create an OpenSearch Application in the console, do the following:

1. Open the AWS Management Console and go to the Amazon OpenSearch Service homepage.
2. In the left navigation window, find the OpenSearch user interface (Dashboards) tab
3. Select `Create Application`

To create an OpenSearch Application in the AWS Command Line Interface do the following:

```
aws opensearch create-application \
      --name myapplication

aws opensearch create-application \
      --name myapplication \
      --iam-identity-center-options "
            {
            \"enabled\":true,
            \"iamIdentityCenterInstanceArn\":\"arn:aws:sso:::instance/ssoins-xxxxxxxxx
\",
            \"iamRoleForIdentityCenterApplicationArn\":
\"arn:aws:iam::555555555555:role/xxxxxxxx\"
            }
      "
```

Controlling access to an OpenSearch Application

OpenSearch user interface supports both AWS Identity and Access Management (IAM) and IAM Identity Center for login. When creating an OpenSearch application, the default options is to use IAM and you can manage permissions to the OpenSearch application by IAM users. Optionally, you can opt to use IAM Identity Center that links to your existing identity providers, for user login to the OpenSearch application. To enable IAM Identity Center, click on the checkbox for “Authenticate

with IAM Identity Center" in the OpenSearch application creation workflow, and then grant IAM Identity Center users the permission to access the OpenSearch application.

You can also configure IAM and IAM Identity Center configurations through the AWS Command Line Interface. See the following example:

```
aws opensearch create-application \
      --name myapplication

aws opensearch create-application \
      --name myapplication \
      --iam-identity-center-options "
            {
            \"enabled\":true,
            \"iamIdentityCenterInstanceArn\":\"arn:aws:sso:::instance/ssoins-xxxxxxxxx
\",
            \"iamRoleForIdentityCenterApplicationArn\":
\"arn:aws:iam::555555555555:role/xxxxxxxx\"
            }
      "
```

You will also need to specify the following trust policy for their IAM role using `iamRoleForIdentityCenterApplication`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "application.opensearchservice.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:SetContext"
            ],
            "Condition": {
                "ForAllValues:ArnEquals": {
                    "sts:RequestContextProviders": "arn:aws:iam::aws:contextProvider/
IdentityCenter"
                }
            }
```

```
        }
    ]
}
```

You will need to define the following permissions policies for the role as well:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "IdentityStoreOpenSearchDomainConnectivity",
            "Effect": "Allow",
            "Action": [
                "identitystore:DescribeUser",
                "identitystore:ListGroupMembershipsForMember",
                "identitystore:DescribeGroup"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:CalledViaLast": "es.amazonaws.com"
                }
            }
        },
        {
            "Sid": "OpenSearchDomain",
            "Effect": "Allow",
            "Action": [
                "es:ESHttp*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "OpenSearchServerless", // if need to access OpenSearch serverless
 collections
            "Effect": "Allow",
            "Action": [
                "aoss:APIAccessAll"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

In addition to enabling IAM Identity Center in OpenSearch, you'll need to specify the following Trust policy for your IAM role using the `iamRoleForIdentityCenterApplication` parameter:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "application.opensearchservice.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:SetContext"
            ],
            "Condition": {
                "ForAllValues:ArnEquals": {
                    "sts:RequestContextProviders": "arn:aws:iam::aws:contextProvider/
IdentityCenter"
                }
            }
        }
    ]
}
```

Define OpenSearch Application admin

OpenSearch application admin is the defined role with permission to edit and delete an OpenSearch application. As the creator of an OpenSearch, you will by default become the first admin of the OpenSearch application. Additional admins can be added to the OpenSearch application in the AWS Management Console, either during the application creation workflow or in the "**edit application**" page, by searching for the ARN of IAM principals or the name of IAM Identity Center users in the OpenSearch application admins management search bar. Extra admins can be removed but there must be at least one admin for an OpenSearch application.

OpenSearch application admin management can also be done through the AWS Command Line Interface. Here are the examples of how to add IAM principals and IAM Identity Center users as admin while creating an OpenSearch application.

```
aws opensearch create-application \
    --name myapplication \
    --app-configs "
        {
        \"key\":\"opensearchDashboards.dashboardAdmin.users\",
        \"value\":\"arn:aws:iam::555555555555:user/xxxxxxxx\"
        }
    "

aws opensearch create-application \
    --name myapplication \
    --iam-identity-center-options "
        {
        \"enabled\":true,
        \"iamIdentityCenterInstanceArn\":\"arn:aws:sso:::instance/ssoins-xxxxxxxxx\",
        \"iamRoleForIdentityCenterApplicationArn\":\"arn:aws:iam::555555555555:role/
xxxxxxxx\"
        }
    " \
    --app-configs "
        {
        \"key\":\"opensearchDashboards.dashboardAdmin.users\",
        \"value\":\"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx\"
        }
    "
```

Here are the examples of how to update IAM principals and IAM Identity Center users as admin to an existing OpenSearch application.

```
aws opensearch update-application \
    --id myapplication \
    --app-configs "
        {
        \"key\":\"opensearchDashboards.dashboardAdmin.users\",
        \"value\":\"arn:aws:iam::555555555555:user/xxxxxxxx\"
        }
    "


aws opensearch update-application \
    --id myapplication \
    --app-configs "
        {
```

```
        \"key\":\"opensearchDashboards.dashboardAdmin.users\",
        \"value\":\"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx\"
        }
    "
```

Associate data sources with an OpenSearch application

An OpenSearch application can work with multiple data sources, including OpenSearch Service managed domains and serverless collections, as well as integrated data sources such as Amazon S3.

To associate data sources to an AWS application, click on associate resources button under the “Associated data sources” table, and follow the instructions.

Alternatively, you can use the AWS Command Line Interface to call an OpenSearch Application and update the associated data sources.

```
aws opensearch-es create-application \
      --name myapplication \
      --data-sources "[{\"dataSourceArn\": \"arn:aws:es:us-east-1:555555555555:domain/
xxxxxxxx\"}]"


aws opensearch update-application \
      --id myapplication \
      --data-sources "[{\"dataSourceArn\": \"arn:aws:es:us-east-1:555555555555:domain/
xxxxxxxx\"}]"
```

Associate with OpenSearch domains in VPC

To associate a domain within a VPC as a data source to an OpenSearch user interface Dashboards application, you need the owner of the VPC to authorize the access on the domain side.

To authorize a VPC domain from the AWS Management Console:

1. Go to your OpenSearch Service console homepage.
2. Select **Domains** on the left navigation bar, and open the specific domain in vpc.
3. Under **VPC endpoints**, select **Authorize principal**, and then **Authorized principals from other AWS**. Select **OpenSearch applications (Dashboard)** from the drop down list

To authorize VPC domain access from the AWS Command Line Interface, you can use the `authorize-vpc-endpoint-access` command

```
aws opensearch authorize-vpc-endpoint-access \
 --domain-name <domain-name> \
 --service application.opensearchservice.amazonaws.com \
 --region <region>
```

Associate with OpenSearch Serverless collections in VPC

To associate an OpenSearch Serverless collections within a VPC as a data source to OpenSearch user interface Dashboards applications, you need the owner of the VPC to specifically authorize the access by creating a new network policy and attaching it to the collection.

To create or update a new network policy to make a collection in VPC work with an OpenSearch application from the AWS Management Console:

1. Navigate to your OpenSearch Service console homepage, and select **Network policies** under **Serverless**.
2. Select on **Create network policy** or select an existing policy and select **Edit**.
3. On the configuration page, navigate to the Access type section.
4. Select Private (recommended), and then select AWS service private access.
5. From the search bar, select **`application.opensearchservice.amazonaws.com`**.
6. Select the box for **Enable access to OpenSearch endpoint** under the resource type section.
7. In the search bar for collection name, enter or select the name of the collections that you want to attach with this network policy.
8. Create or save the settings for the network policy.

To create or update a new network policy to make a collection in VPC work with an OpenSearch application from the AWS Command Line Interface, you can use the following examples:

```
% aws opensearchserverless create-security-policy \
--type network  \
--region $region \
--endpoint-url=$endpoint \
--name allow-public-service \
--policy file:/<path_to_network_policy_json_file>
```

```
{
    "securityPolicyDetail": {
        "createdDate": ******,
        "lastModifiedDate": ******,
        "name": "<network_policy_name>",
        "policy": [
            {
                "SourceVPCEs": [],
                "AllowFromPublic": false,
                "Description": "Test network policy statement",
                "Rules": [
                    {
                        "Resource": [
                            "collection/<network_policy_name>"
                        ],
                        "ResourceType": "collection"
                    }
                ],
                "SourceServices": [
                    "application.opensearchservice.amazonaws.com"
                ]
            }
        ],
        "policyVersion": "******",
        "type": "network"
    }
}
```

Alternatively, you can update an existing network policy:

```
% aws opensearchserverless update-security-policy \
--type network  \
--region $region \
--endpoint-url=$endpoint \
--name allow1-service \
--policy-version "<policy_version_from_output_of_network_policy_creation>" \
--policy file:/<path_to_network_policy_json_file>
{
    "securityPolicyDetail": {
        "createdDate": ******,
        "lastModifiedDate": ******,
        "name": "<network_policy_name>",
        "policy": [
```

```
            {
                "SourceVPCEs": [],
                "AllowFromPublic": false,
                "Description": "Test network policy statement",
                "Rules": [
                    {
                        "Resource": [
                            "collection/<network_policy_name>"
                        ],
                        "ResourceType": "collection"
                    }
                ],
                "SourceServices": [
                    "application.opensearchservice.amazonaws.com"
                ]
            }
        ],
        "policyVersion": "******",
        "type": "network"
    }
}
```

Network policy JSON file example:

```
[{
    "Description" : "Test network policy statement",
    "Rules": [{
       "ResourceType" : "collection",
        "Resource" : ["collection/<collection_name>"]
     }],
    "SourceServices" : [
          "application.opensearchservice.amazonaws.com"
      ],
      "AllowFromPublic" : false
}]
```

When the association is no longer needed, the VPC domain owner can revoke the access using the following steps:

1. Go to **OpenSearch Service** console homepage.
2. Select **Domains** on the left navigation bar, and open the specific domain in VPC.

3. Under VPC endpoints, select **AWS Service- OpenSearch Service applications (Dashboard)** from the **Authorized principals** list and choose **Revoke access**.

Creating workspaces in an OpenSearch application

Once an OpenSearch application is created with an associated data sources and user permissions, the next step is to launch the OpenSearch application to create workspaces. To do this, you can select the **Launch application** button or use the OpenSearch application URL to open the OpenSearch application homepage in a new webpage. The OpenSearch application will list all the existing workspaces in the homepage, categorized by use case.

To associate data sources to an AWS application, click on associate resources button under the "**Associate data sources**" table, and follow the instructions.

There are five workspace types currently available in OpenSearch Service, each with different features available for the specific use case.

- The **Observability workspace** is designed for gaining visibility into system health, performance, and reliability through monitoring of logs, metrics and traces. Observability workspace supports
- The **Security Analytics workspace** is designed for detecting and investigating potential security threats and vulnerabilities across your systems and data.
- The **Search workspace** is designed for quickly finding and exploring relevant information across your organization's data sources.
- The **Essentials workspace** is designed for OpenSearch Serverless as a data source, and enables analyzing data to derive insights, identify patterns and trends, and make data driven decisions.quickly finding and exploring relevant information across your organization's data sources.
- The **Analytics (all features) workspace** is designed for multi-purpose use cases and supports all the features available in OpenSearch Service UI (Dashboards).

Managing indexes in Amazon OpenSearch Service

After you add data to Amazon OpenSearch Service, you often need to reindex that data, work with index aliases, move an index to more cost-effective storage, or delete it altogether. This chapter covers UltraWarm storage, cold storage, and Index State Management. For information on the OpenSearch index APIs, see the OpenSearch documentation.

Topics

UltraWarm storage for Amazon OpenSearch Service

UltraWarm provides a cost-effective way to store large amounts of read-only data on Amazon OpenSearch Service. Standard data nodes use "hot" storage, which takes the form of instance stores or Amazon EBS volumes attached to each node. Hot storage provides the fastest possible performance for indexing and searching new data.

Rather than attached storage, UltraWarm nodes use Amazon S3 and a sophisticated caching solution to improve performance. For indexes that you are not actively writing to, query less frequently, and don't need the same performance from, UltraWarm offers significantly lower costs per GiB of data. Because warm indexes are read-only unless you return them to hot storage, UltraWarm is best-suited to immutable data, such as logs.

In OpenSearch, warm indexes behave just like any other index. You can query them using the same APIs or use them to create visualizations in OpenSearch Dashboards.

Topics

Prerequisites

UltraWarm has a few important prerequisites:

- UltraWarm requires OpenSearch or Elasticsearch 6.8 or higher.
- To use warm storage, domains must have dedicated master nodes.
- When using a Multi-AZ with Standby domain, the number of warm nodes must be a multiple of the number of Availability Zones being used.
- If your domain uses a T2 or T3 instance type for your data nodes, you can't use warm storage.
- If your index uses approximate k-NN (`"index.knn":true`), you can move it to warm storage from version 2.17 and later. Domains on versions earlier than 2.17 can upgrade to 2.17 to use this functionality, but KNN indices creaed on versions earlier than 2.x can't migrate to UltraWarm.
- If the domain uses fine-grained access control, users must be mapped to the `ultrawarm_manager` role in OpenSearch Dashboards to make UltraWarm API calls.

Note

The `ultrawarm_manager` role might not be defined on some preexisting OpenSearch Service domains. If you don't see the role in Dashboards, you need to manually create it.

Configure permissions

If you enable UltraWarm on a preexisting OpenSearch Service domain, the `ultrawarm_manager` role might not be defined on the domain. Non-admin users must be mapped to this role in order to manage warm indexes on domains using fine-grained access control. To manually create the `ultrawarm_manager` role, perform the following steps:

1. In OpenSearch Dashboards, go to **Security** and choose **Permissions**.
2. Choose **Create action group** and configure the following groups:

Group name	Permissions
`ultrawarm_cluster`	• `cluster:admin/ultrawarm/migration/list` • `cluster:monitor/nodes/stats`
`ultrawarm_index_read`	• `indices:admin/ultrawarm/migration/get` • `indices:admin/get`
`ultrawarm_index_write`	• `indices:admin/ultrawarm/migration/warm` • `indices:admin/ultrawarm/migration/hot` • `indices:monitor/stats` • `indices:admin/ultrawarm/migration/cancel`

3. Choose **Roles** and **Create role**.
4. Name the role **ultrawarm_manager**.
5. For **Cluster permissions,** select `ultrawarm_cluster` and `cluster_monitor`.
6. For **Index**, type `*`.
7. For **Index permissions**, select `ultrawarm_index_read`, `ultrawarm_index_write`, and `indices_monitor`.
8. Choose **Create**.

9. After you create the role, map it to any user or backend role that will manage UltraWarm indexes.

UltraWarm storage requirements and performance considerations

As covered in the section called "Calculating storage requirements", data in hot storage incurs significant overhead: replicas, Linux reserved space, and OpenSearch Service reserved space. For example, a 20 GiB primary shard with one replica shard requires roughly 58 GiB of hot storage.

Because it uses Amazon S3, UltraWarm incurs none of this overhead. When calculating UltraWarm storage requirements, you consider only the size of the primary shards. The durability of data in S3 removes the need for replicas, and S3 abstracts away any operating system or service considerations. That same 20 GiB shard requires 20 GiB of warm storage. If you provision an `ultrawarm1.large.search` instance, you can use all 20 TiB of its maximum storage for primary shards. See the section called "UltraWarm storage quotas" for a summary of instance types and the maximum amount of storage that each can address.

With UltraWarm, we still recommend a maximum shard size of 50 GiB. The number of CPU cores and amount of RAM allocated to each UltraWarm instance type gives you an idea of the number of shards they can simultaneously search. Note that while only primary shards count toward UltraWarm storage in S3, OpenSearch Dashboards and `_cat/indices` still report UltraWarm index size as the *total* of all primary and replica shards.

For example, each `ultrawarm1.medium.search` instance has two CPU cores and can address up to 1.5 TiB of storage on S3. Two of these instances have a combined 3 TiB of storage, which works out to approximately 62 shards if each shard is 50 GiB. If a request to the cluster only searches four of these shards, performance might be excellent. If the request is broad and searches all 62 of them, the four CPU cores might struggle to perform the operation. Monitor the `WarmCPUUtilization` and `WarmJVMMemoryPressure` UltraWarm metrics to understand how the instances handle your workloads.

If your searches are broad or frequent, consider leaving the indexes in hot storage. Just like any other OpenSearch workload, the most important step to determining if UltraWarm meets your needs is to perform representative client testing using a realistic dataset.

UltraWarm pricing

With hot storage, you pay for what you provision. Some instances require an attached Amazon EBS volume, while others include an instance store. Whether that storage is empty or full, you pay the same price.

With UltraWarm storage, you pay for what you use. An `ultrawarm1.large.search` instance can address up to 20 TiB of storage on S3, but if you store only 1 TiB of data, you're only billed for 1 TiB of data. Like all other node types, you also pay an hourly rate for each UltraWarm node. For more information, see the section called “Pricing”.

Enabling UltraWarm

The console is the simplest way to create a domain that uses warm storage. While creating the domain, choose **Enable UltraWarm data nodes** and the number of warm nodes that you want. The same basic process works on existing domains, provided they meet the prerequisites. Even after the domain state changes from **Processing** to **Active**, UltraWarm might not be available to use for several hours.

When using a Multi-AZ with Standby domain, the number of warm nodes must be a multiple of the number of Availability Zones being used. For more information, see the section called “Multi-AZ with Standby”.

You can also use the AWS CLI or configuration API to enable UltraWarm, specifically the `WarmEnabled`, `WarmCount`, and `WarmType` options in `ClusterConfig`.

> **Note**
>
> Domains support a maximum number of warm nodes. For details, see the section called “Quotas”.

Sample CLI command

The following AWS CLI command creates a domain with three data nodes, three dedicated master nodes, six warm nodes, and fine-grained access control enabled:

```
aws opensearch create-domain \
  --domain-name my-domain \
```

```
  --engine-version Opensearch_1.0 \
  --cluster-config
 InstanceCount=3,InstanceType=r6g.large.search,DedicatedMasterEnabled=true,DedicatedMasterType=
 \
  --ebs-options EBSEnabled=true,VolumeType=gp2,VolumeSize=11 \
  --node-to-node-encryption-options Enabled=true \
  --encryption-at-rest-options Enabled=true \
  --domain-endpoint-options EnforceHTTPS=true,TLSSecurityPolicy=Policy-Min-
TLS-1-2-2019-07 \
  --advanced-security-options
 Enabled=true,InternalUserDatabaseEnabled=true,MasterUserOptions='{MasterUserName=master-
user,MasterUserPassword=master-password}' \
  --access-policies '{"Version":"2012-10-17","Statement":
[{"Effect":"Allow","Principal":{"AWS":["123456789012"]},"Action":
["es:*"],"Resource":"arn:aws:es:us-west-1:123456789012:domain/my-domain/*"}]}' \
  --region us-east-1
```

For detailed information, see the AWS CLI Command Reference.

Sample configuration API request

The following request to the configuration API creates a domain with three data nodes, three dedicated master nodes, and six warm nodes with fine-grained access control enabled and a restrictive access policy:

```
POST https://es.us-east-2.amazonaws.com/2021-01-01/opensearch/domain
{
  "ClusterConfig": {
    "InstanceCount": 3,
    "InstanceType": "r6g.large.search",
    "DedicatedMasterEnabled": true,
    "DedicatedMasterType": "r6g.large.search",
    "DedicatedMasterCount": 3,
    "ZoneAwarenessEnabled": true,
    "ZoneAwarenessConfig": {
      "AvailabilityZoneCount": 3
    },
    "WarmEnabled": true,
    "WarmCount": 6,
    "WarmType": "ultrawarm1.medium.search"
  },
  "EBSOptions": {
    "EBSEnabled": true,
```

```
    "VolumeType": "gp2",
    "VolumeSize": 11
  },
  "EncryptionAtRestOptions": {
    "Enabled": true
  },
  "NodeToNodeEncryptionOptions": {
    "Enabled": true
  },
  "DomainEndpointOptions": {
    "EnforceHTTPS": true,
    "TLSSecurityPolicy": "Policy-Min-TLS-1-2-2019-07"
  },
   "AdvancedSecurityOptions": {
    "Enabled": true,
    "InternalUserDatabaseEnabled": true,
    "MasterUserOptions": {
      "MasterUserName": "master-user",
      "MasterUserPassword": "master-password"
    }
  },
  "EngineVersion": "Opensearch_1.0",
  "DomainName": "my-domain",
  "AccessPolicies": "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect\":\"Allow
\",\"Principal\":{\"AWS\":[\"123456789012\"]},\"Action\":[\"es:*\"],\"Resource\":
\"arn:aws:es:us-east-1:123456789012:domain/my-domain/*\"}]}"
}
```

For detailed information, see the Amazon OpenSearch Service API Reference.

Migrating indexes to UltraWarm storage

If you finished writing to an index and no longer need the fastest possible search performance, migrate it from hot to UltraWarm:

```
POST _ultrawarm/migration/my-index/_warm
```

Then check the status of the migration:

```
GET _ultrawarm/migration/my-index/_status

{
```

```
  "migration_status": {
    "index": "my-index",
    "state": "RUNNING_SHARD_RELOCATION",
    "migration_type": "HOT_TO_WARM",
    "shard_level_status": {
      "running": 0,
      "total": 5,
      "pending": 3,
      "failed": 0,
      "succeeded": 2
    }
  }
}
```

Index health must be green to perform a migration. If you migrate several indexes in quick succession, you can get a summary of all migrations in plaintext, similar to the `_cat` API:

```
GET _ultrawarm/migration/_status?v

index    migration_type state
my-index HOT_TO_WARM    RUNNING_SHARD_RELOCATION
```

OpenSearch Service migrates one index at a time to UltraWarm. You can have up to 200 migrations in the queue. Any request that exceeds the limit will be rejected. To check the current number of migrations in the queue, monitor the `HotToWarmMigrationQueueSize` metric. Indexes remain available throughout the migration process—no downtime.

The migration process has the following states:

```
PENDING_INCREMENTAL_SNAPSHOT
RUNNING_INCREMENTAL_SNAPSHOT
FAILED_INCREMENTAL_SNAPSHOT
PENDING_FORCE_MERGE
RUNNING_FORCE_MERGE
FAILED_FORCE_MERGE
PENDING_FULL_SNAPSHOT
RUNNING_FULL_SNAPSHOT
FAILED_FULL_SNAPSHOT
PENDING_SHARD_RELOCATION
RUNNING_SHARD_RELOCATION
FINISHED_SHARD_RELOCATION
```

As these states indicate, migrations might fail during snapshots, shard relocations, or force merges. Failures during snapshots or shard relocation are typically due to node failures or S3 connectivity issues. Lack of disk space is usually the underlying cause of force merge failures.

After a migration finishes, the same `_status` request returns an error. If you check the index at that time, you can see some settings that are unique to warm indexes:

```
GET my-index/_settings

{
  "my-index": {
    "settings": {
      "index": {
        "refresh_interval": "-1",
        "auto_expand_replicas": "false",
        "provided_name": "my-index",
        "creation_date": "1599241458998",
        "unassigned": {
          "node_left": {
            "delayed_timeout": "5m"
          }
        },
        "number_of_replicas": "1",
        "uuid": "GswyCdR0RSq0SJYmzsIpiw",
        "version": {
          "created": "7070099"
        },
        "routing": {
          "allocation": {
            "require": {
              "box_type": "warm"
            }
          }
        },
        "number_of_shards": "5",
        "merge": {
          "policy": {
            "max_merge_at_once_explicit": "50"
          }
        }
      }
    }
  }
```

```
}
```

- `number_of_replicas`, in this case, is the number of passive replicas, which don't consume disk space.
- `routing.allocation.require.box_type` specifies that the index should use warm nodes rather than standard data nodes.
- `merge.policy.max_merge_at_once_explicit` specifies the number of segments to simultaneously merge during the migration.

Indexes in warm storage are read-only unless you return them to hot storage, which makes UltraWarm best-suited to immutable data, such as logs. You can query the indexes and delete them, but you can't add, update, or delete individual documents. If you try, you might encounter the following error:

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "cluster_block_exception",
        "reason" : "index [indexname] blocked by: [TOO_MANY_REQUESTS/12/disk usage
 exceeded flood-stage watermark, index has read-only-allow-delete block];"
      }
    ],
    "type" : "cluster_block_exception",
    "reason" : "index [indexname] blocked by: [TOO_MANY_REQUESTS/12/disk usage exceeded
 flood-stage watermark, index has read-only-allow-delete block];"
  },
  "status" : 429
}
```

Automating migrations

We recommend using the section called “Index State Management” to automate the migration process after an index reaches a certain age or meets other conditions. See the sample policy that demonstrates this workflow.

Migration tuning

Index migrations to UltraWarm storage require a force merge. Each OpenSearch index is composed of some number of shards, and each shard is composed of some number of Lucene segments. The force merge operation purges documents that were marked for deletion and conserves disk space. By default, UltraWarm merges indexes into one segment, except for kNN indices, where a default value of 20 is used.

You can change this value up to 1,000 segments using the `index.ultrawarm.migration.force_merge.max_num_segments` setting. Higher values speed up the migration process, but increase query latency for the warm index after the migration finishes. To change the setting, make the following request:

```
PUT my-index/_settings
{
  "index": {
    "ultrawarm": {
      "migration": {
        "force_merge": {
          "max_num_segments": 1
        }
      }
    }
  }
}
```

To check how long this stage of the migration process takes, monitor the `HotToWarmMigrationForceMergeLatency` metric.

Cancelling migrations

UltraWarm handles migrations sequentially, in a queue. If a migration is in the queue, but has not yet started, you can remove it from the queue using the following request:

```
POST _ultrawarm/migration/_cancel/my-index
```

If your domain uses fine-grained access control, you must have the `indices:admin/ultrawarm/migration/cancel` permission to make this request.

Listing hot and warm indexes

UltraWarm adds two additional options, similar to `_all`, to help manage hot and warm indexes. For a list of all warm or hot indexes, make the following requests:

```
GET _warm
GET _hot
```

You can use these options in other requests that specify indexes, such as:

```
_cat/indices/_warm
_cluster/state/_all/_hot
```

Returning warm indexes to hot storage

If you need to write to an index again, migrate it back to hot storage:

```
POST _ultrawarm/migration/my-index/_hot
```

You can have up to 10 queued migrations from warm to hot storage at a time. OpenSearch Service processes migration requests one at a time, in the order that they were queued. To check the current number, monitor the `WarmToHotMigrationQueueSize` metric.

After the migration finishes, check the index settings to make sure they meet your needs. Indexes return to hot storage with one replica.

Restoring warm indexes from snapshots

In addition to the standard repository for automated snapshots, UltraWarm adds a second repository for warm indexes, `cs-ultrawarm`. Each snapshot in this repository contains only one index. If you delete a warm index, its snapshot remains in the `cs-ultrawarm` repository for 14 days, just like any other automated snapshot.

When you restore a snapshot from `cs-ultrawarm`, it restores to warm storage, not hot storage. Snapshots in the `cs-automated` and `cs-automated-enc` repositories restore to hot storage.

To restore an UltraWarm snapshot to warm storage

1. Identify the latest snapshot that contains the index you want to restore:

```
GET _snapshot/cs-ultrawarm/_all?verbose=false

{
  "snapshots": [{
    "snapshot": "snapshot-name",
    "version": "1.0",
    "indices": [
      "my-index"
    ]
  }]
}
```

> **Note**
>
> By default, the `GET _snapshot/<repo>` operation displays verbose data information such as start time, end time, and duration for each snapshot within a repository. The `GET _snapshot/<repo>` operation retrieves information from the files of each snapshot contained in a repository. If you do not need the start time, end time, and duration and require only the name and index information of a snapshot, we recommend using the `verbose=false` parameter when listing snapshots to minimize processing time and prevent timing out.

2. If the index already exists, delete it:

```
DELETE my-index
```

   If you don't want to delete the index, return it to hot storage and reindex it.

3. Restore the snapshot:

```
POST _snapshot/cs-ultrawarm/snapshot-name/_restore
```

   UltraWarm ignores any index settings you specify in this restore request, but you can specify options like `rename_pattern` and `rename_replacement`. For a summary of OpenSearch snapshot restore options, see the OpenSearch documentation.

Manual snapshots of warm indexes

You *can* take manual snapshots of warm indexes, but we don't recommend it. The automated `cs-ultrawarm` repository already contains a snapshot for each warm index, taken during the migration, at no additional charge.

By default, OpenSearch Service does not include warm indexes in manual snapshots. For example, the following call only includes hot indexes:

```
PUT _snapshot/my-repository/my-snapshot
```

If you choose to take manual snapshots of warm indexes, several important considerations apply.

- You can't mix hot and warm indexes. For example, the following request fails:

  ```
  PUT _snapshot/my-repository/my-snapshot
  {
    "indices": "warm-index-1,hot-index-1",
    "include_global_state": false
  }
  ```

  If they include a mix of hot and warm indexes, wildcard (*) statements fail, as well.

- You can only include one warm index per snapshot. For example, the following request fails:

  ```
  PUT _snapshot/my-repository/my-snapshot
  {
    "indices": "warm-index-1,warm-index-2,other-warm-indices-*",
    "include_global_state": false
  }
  ```

  This request succeeds:

  ```
  PUT _snapshot/my-repository/my-snapshot
  {
    "indices": "warm-index-1",
    "include_global_state": false
  }
  ```

- Manual snapshots always restore to hot storage, even if they originally included a warm index.

Migrating warm indexes to cold storage

If you have data in UltraWarm that you query infrequently, consider migrating it to cold storage. Cold storage is meant for data you only access occasionally or is no longer in active use. You can't read from or write to cold indexes, but you can migrate them back to warm storage at no cost whenever you need to query them. For instructions, see Migrating indexes to cold storage.

Best practices for KNN indexes

- Ultrawarm/Cold tier is available for all KNN index engine types. We recommed it for KNN indexes using Lucene engine and Disk-optimized vector search, which does not require to fully load the graph data in off-heap memory. While using it with native in-memory engines like FAISS and NMSLIB, you must account for the shards graph size that will be actively searched on, and provision the UltraWarm instances, preferably of the `uw.large` instance type, accordingly. For example, if customers have 2 `uw.large` instances configured, then they each will have approximately `knn.memory.circuit_breaker.limit * 61` GiB available off-heap memory. You get optimal performance if all your warm queries are targeting shards whose cumulative graph size does not exceed available off-heap memory. Latency is impacted if the available memory is lower than needed to load the graph because of evictions and waiting on off-heap memory to become available. That's why we don’t recommend using `uw.medium` instances for use cases where in-memory engines are being used or for higher search throughput cases, irrespective of engines.
- KNN indexes migrating to UltraWarm will not be force-merged to single segment. This avoids any impact on the hot and warm nodes running into OOM issues because of graph size becoming too big for in-memory engines. Due to the increase in number of segments per shard, this might result in consuming more local cache space and allowing fewer indices to migrate to the warm tier. You can choose to force-merge indexes to single segment by using the existing setting, and overriding it before migrating indexes to the warm tier. For more information, see the section called “Migration tuning”.
- If you have a use case where indexes are searched infrequently and do not serve a latency sensitive workload, you can choose to migrate those indexes to the UltraWarm tier. This will help you to scale down the hot tier compute instances and let the UltraWarm tier compute handle the query on such low priority indexes. This can also help to provide isolation of resources consumed between the queries of low and high priority indexes so they don't impact each other.

Disabling UltraWarm

The console is the simplest way to disable UltraWarm. Choose the domain, **Actions**, and **Edit cluster configuration**. Deselect **Enable UltraWarm data nodes** and choose **Save changes**. You can also use the `WarmEnabled` option in the AWS CLI and configuration API.

Before you disable UltraWarm, you must either delete all warm indexes or migrate them back to hot storage. After warm storage is empty, wait five minutes before attempting to disable UltraWarm.

Cold storage for Amazon OpenSearch Service

Cold storage lets you store any amount of infrequently accessed or historical data on your Amazon OpenSearch Service domain and analyze it on demand, at a lower cost than other storage tiers. Cold storage is appropriate if you need to do periodic research or forensic analysis on your older data. Practical examples of data suitable for cold storage include infrequently accessed logs, data that must be preserved to meet compliance requirements, or logs that have historical value.

Similar to UltraWarm storage, cold storage is backed by Amazon S3. When you need to query cold data, you can selectively attach it to existing UltraWarm nodes. You can manage the migration and lifecycle of your cold data manually or with Index State Management policies.

Topics

- Canceling migrations from cold to warm storage
- Updating cold index metadata
- Deleting cold indexes
- Disabling cold storage

Prerequisites

Cold storage has the following prerequisites:

- Cold storage requires OpenSearch or Elasticsearch version 7.9 or later.
- To enable cold storage on an OpenSearch Service domain, you must also enable UltraWarm on the same domain.
- To use cold storage, domains must have dedicated master nodes.
- If your domain uses a T2 or T3 instance type for your data nodes, you can't use cold storage.
- If your index uses approximate k-NN (`"index.knn":true`), you can move it to cold storage from version 2.17 and later. Domains on versions earlier than 2.17 can upgrade to 2.17 to use this functionality, but KNN indices creaed on versions earlier than 2.x can't migrate to Cold.
- If the domain uses fine-grained access control, non-admin users must be mapped to the `cold_manager` role in OpenSearch Dashboards in order to manage cold indexes.

> **Note**
>
> The `cold_manager` role might not exist on some preexisting OpenSearch Service domains. If you don't see the role in Dashboards, you need to manually create it.

Configure permissions

If you enable cold storage on a preexisting OpenSearch Service domain, the `cold_manager` role might not be defined on the domain. If the domain uses fine-grained access control, non-admin users must be mapped to this role in order to manage cold indexes. To manually create the `cold_manager` role, perform the following steps:

1. In OpenSearch Dashboards, go to **Security** and choose **Permissions**.
2. Choose **Create action group** and configure the following groups:

Group name	Permissions
cold_cluster	• cluster:monitor/nodes/stats • cluster:admin/ultrawarm* • cluster:admin/cold/*
cold_index	• indices:monitor/stats • indices:data/read/minmax • indices:admin/ultrawarm/migration/get • indices:admin/ultrawarm/migration/cancel

3. Choose **Roles** and **Create role**.
4. Name the role **cold_manager**.
5. For **Cluster permissions**, choose the `cold_cluster` group you created.
6. For **Index**, enter *.
7. For **Index permissions**, choose the `cold_index` group you created.
8. Choose **Create**.
9. After you create the role, map it to any user or backend role that manages cold indexes.

Cold storage requirements and performance considerations

Because cold storage uses Amazon S3, it incurs none of the overhead of hot storage, such as replicas, Linux reserved space, and OpenSearch Service reserved space. Cold storage doesn't have specific instance types because it doesn't have any compute capacity attached to it. You can store any amount of data in cold storage. Monitor the `ColdStorageSpaceUtilization` metric in Amazon CloudWatch to see how much cold storage space you're using.

Cold storage pricing

Similar to UltraWarm storage, with cold storage you only pay for data storage. There's no compute cost for cold data and you wont get billed if theres no data in cold storage.

You don't incur any transfer charges when moving data between cold and warm storage. While indexes are being migrated between warm and cold storage, you continue to pay for only one copy of the index. After the migration completes, the index is billed according to the storage tier it was

migrated to. For more information about cold storage pricing, see Amazon OpenSearch Service pricing.

Enabling cold storage

The console is the simplest way to create a domain that uses cold storage. While creating the domain, choose **Enable cold storage**. The same process works on existing domains as long as you meet the prerequisites. Even after the domain state changes from **Processing** to **Active**, cold storage might not be available for several hours.

You can also use the AWS CLI or configuration API to enable cold storage.

Sample CLI command

The following AWS CLI command creates a domain with three data nodes, three dedicated master nodes, cold storage enabled, and fine-grained access control enabled:

```
aws opensearch create-domain \
  --domain-name my-domain \
  --engine-version Opensearch_1.0 \
  --cluster-
config ColdStorageOptions={Enabled=true},WarmEnabled=true,WarmCount=4,WarmType=ultrawarm1.mediu
 \
  --ebs-options EBSEnabled=true,VolumeType=gp2,VolumeSize=11 \
  --node-to-node-encryption-options Enabled=true \
  --encryption-at-rest-options Enabled=true \
  --domain-endpoint-options EnforceHTTPS=true,TLSSecurityPolicy=Policy-Min-
TLS-1-2-2019-07 \
  --advanced-security-options
 Enabled=true,InternalUserDatabaseEnabled=true,MasterUserOptions='{MasterUserName=master-
user,MasterUserPassword=master-password}' \
  --region us-east-2
```

For detailed information, see the AWS CLI Command Reference.

Sample configuration API request

The following request to the configuration API creates a domain with three data nodes, three dedicated master nodes, cold storage enabled, and fine-grained access control enabled:

```
POST https://es.us-east-2.amazonaws.com/2021-01-01/opensearch/domain
{
```

```
  "ClusterConfig": {
    "InstanceCount": 3,
    "InstanceType": "r6g.large.search",
    "DedicatedMasterEnabled": true,
    "DedicatedMasterType": "r6g.large.search",
    "DedicatedMasterCount": 3,
    "ZoneAwarenessEnabled": true,
    "ZoneAwarenessConfig": {
      "AvailabilityZoneCount": 3
     },
    "WarmEnabled": true,
    "WarmCount": 4,
    "WarmType": "ultrawarm1.medium.search",
    "ColdStorageOptions": {
       "Enabled": true
     }
  },
  "EBSOptions": {
    "EBSEnabled": true,
    "VolumeType": "gp2",
    "VolumeSize": 11
  },
  "EncryptionAtRestOptions": {
    "Enabled": true
  },
  "NodeToNodeEncryptionOptions": {
    "Enabled": true
  },
  "DomainEndpointOptions": {
    "EnforceHTTPS": true,
    "TLSSecurityPolicy": "Policy-Min-TLS-1-2-2019-07"
  },
   "AdvancedSecurityOptions": {
    "Enabled": true,
    "InternalUserDatabaseEnabled": true,
    "MasterUserOptions": {
      "MasterUserName": "master-user",
      "MasterUserPassword": "master-password"
    }
  },
  "EngineVersion": "Opensearch_1.0",
  "DomainName": "my-domain"
}
```

For detailed information, see the Amazon OpenSearch Service API Reference.

Managing cold indexes in OpenSearch Dashboards

You can manage hot, warm and cold indexes with the existing Dashboards interface in your OpenSearch Service domain. Dashboards enables you to migrate indexes between warm and cold storage, and monitor index migration status, without using the CLI or configuration API. For more information, see Managing indexes in OpenSearch Dashboards.

Migrating indexes to cold storage

When you migrate indexes to cold storage, you provide a time range for the data to make discovery easier. You can select a timestamp field based on the data in your index, manually provide a start and end timestamp, or choose to not specify one.

Parameter	Supported value	Description
`timestamp_field`	The date/time field from the index mapping.	The minimum and maximum values of the provided field are computed and stored as the `start_time` and `end_time` metadata for the cold index.
`start_time` and `end_time`	One of the following formats: • strict_date_optional_time. For example: `yyyy-MM-dd'T'HH:mm:ss.SSSZ` or `yyyy-MM-dd` • Epoch time in milliseconds	The provided values are stored as the `start_time` and `end_time` metadata for the cold index.

If you don't want to specify a timestamp, add `?ignore=timestamp` to the request instead.

The following request migrates a warm index to cold storage and provides start and end times for the data in that index:

```
POST _ultrawarm/migration/my-index/_cold
```

```
  {
    "start_time": "2020-03-09",
    "end_time": "2020-03-09T23:00:00Z"
  }
```

Then check the status of the migration:

```
GET _ultrawarm/migration/my-index/_status

{
  "migration_status": {
    "index": "my-index",
    "state": "RUNNING_METADATA_RELOCATION",
    "migration_type": "WARM_TO_COLD"
  }
}
```

OpenSearch Service migrates one index at a time to cold storage. You can have up to 100 migrations in the queue. Any request that exceeds the limit will be rejected. To check the current number of migrations in the queue, monitor the `WarmToColdMigrationQueueSize` metric. The migration process has the following states:

```
ACCEPTED_COLD_MIGRATION - Migration request is accepted and queued.
RUNNING_METADATA_MIGRATION - The migration request was selected for execution and
 metadata is migrating to cold storage.
FAILED_METADATA_MIGRATION - The attempt to add index metadata has failed and all
 retries are exhausted.
PENDING_INDEX_DETACH - Index metadata migration to cold storage is completed. Preparing
 to detach the warm index state from the local cluster.
RUNNING_INDEX_DETACH - Local warm index state from the cluster is being removed. Upon
 success, the migration request will be completed.
FAILED_INDEX_DETACH - The index detach process failed and all retries are exhausted.
```

Automating migrations to cold storage

You can use Index State Management to automate the migration process after an index reaches a certain age or meets other conditions. See the sample policy, which demonstrates how to automatically migrate indexes from hot to UltraWarm to cold storage.

> **Note**
>
> An explicit `timestamp_field` is required in order to move indexes to cold storage using an Index State Management policy.

Canceling migrations to cold storage

If a migration to cold storage is queued or in a failed state, you can cancel the migration using the following request:

```
POST _ultrawarm/migration/_cancel/my-index

{
  "acknowledged" : true
}
```

If your domain uses fine-grained access control, you need the `indices:admin/ultrawarm/migration/cancel` permission to make this request.

Listing cold indexes

Before querying, you can list the indexes in cold storage to decide which ones to migrate to UltraWarm for further analysis. The following request lists all cold indexes, sorted by index name:

```
GET _cold/indices/_search
```

Sample response

```
{
  "pagination_id" : "je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY",
  "total_results" : 3,
  "indices" : [
    {
      "index" : "my-index-1",
      "index_cold_uuid" : "hjEoh26mRRCFxRIMdgvLmg",
      "size" : 10339,
      "creation_date" : "2021-06-28T20:23:31.206Z",
      "start_time" : "2020-03-09T00:00Z",
```

```
      "end_time" : "2020-03-09T23:00Z"
    },
    {
      "index" : "my-index-2",
      "index_cold_uuid" : "0vIS2n-oROmOWDFmwFIgdw",
      "size" : 6068,
      "creation_date" : "2021-07-15T19:41:18.046Z",
      "start_time" : "2020-03-09T00:00Z",
      "end_time" : "2020-03-09T23:00Z"
    },
    {
      "index" : "my-index-3",
      "index_cold_uuid" : "EaeXOBodTLiDYcivKsXVLQ",
      "size" : 32403,
      "creation_date" : "2021-07-08T00:12:01.523Z",
      "start_time" : "2020-03-09T00:00Z",
      "end_time" : "2020-03-09T23:00Z"
    }
  ]
}
```

Filtering

You can filter cold indexes based on a prefix-based index pattern and time range offsets.

The following request lists indexes that match the prefix pattern of `event-*`:

```
GET _cold/indices/_search
 {
   "filters":{
      "index_pattern": "event-*"
   }
 }
```

Sample response

```
{
  "pagination_id" : "je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY",
  "total_results" : 1,
  "indices" : [
    {
      "index" : "events-index",
      "index_cold_uuid" : "4eFiab7rRfSvp3slrIsIKA",
```

```
      "size" : 32263273,
      "creation_date" : "2021-08-18T18:25:31.845Z",
      "start_time" : "2020-03-09T00:00Z",
      "end_time" : "2020-03-09T23:00Z"
    }
  ]
}
```

The following request returns indexes with `start_time` and `end_time` metadata fields between `2019-03-01` and `2020-03-01`:

```
GET _cold/indices/_search
{
  "filters": {
    "time_range": {
      "start_time": "2019-03-01",
      "end_time": "2020-03-01"
    }
  }
}
```

Sample response

```
{
  "pagination_id" : "je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY",
  "total_results" : 1,
  "indices" : [
    {
      "index" : "my-index",
      "index_cold_uuid" : "4eFiab7rRfSvp3slrIsIKA",
      "size" : 32263273,
      "creation_date" : "2021-08-18T18:25:31.845Z",
      "start_time" : "2019-05-09T00:00Z",
      "end_time" : "2019-09-09T23:00Z"
    }
  ]
}
```

Sorting

You can sort cold indexes by metadata fields such as index name or size. The following request lists all indexes sorted by size in descending order:

```
GET _cold/indices/_search
 {
 "sort_key": "size:desc"
 }
```

Sample response

```
{
  "pagination_id" : "je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY",
  "total_results" : 5,
  "indices" : [
    {
      "index" : "my-index-6",
      "index_cold_uuid" : "4eFiab7rRfSvp3slrIsIKA",
      "size" : 32263273,
      "creation_date" : "2021-08-18T18:25:31.845Z",
      "start_time" : "2020-03-09T00:00Z",
      "end_time" : "2020-03-09T23:00Z"
    },
    {
      "index" : "my-index-9",
      "index_cold_uuid" : "mbD3ZRVDRI6ONqgEOsJyUA",
      "size" : 57922,
      "creation_date" : "2021-07-07T23:41:35.640Z",
      "start_time" : "2020-03-09T00:00Z",
      "end_time" : "2020-03-09T23:00Z"
    },
    {
      "index" : "my-index-5",
      "index_cold_uuid" : "EaeXOBodTLiDYcivKsXVLQ",
      "size" : 32403,
      "creation_date" : "2021-07-08T00:12:01.523Z",
      "start_time" : "2020-03-09T00:00Z",
      "end_time" : "2020-03-09T23:00Z"
    }
  ]
}
```

Other valid sort keys are `start_time:asc/desc`, `end_time:asc/desc`, and `index_name:asc/desc`.

Pagination

You can paginate a list of cold indexes. Configure the number of indexes to be returned per page with the `page_size` parameter (default is 10). Every `_search` request on your cold indexes returns a `pagination_id` which you can use for subsequent calls.

The following request paginates the results of a `_search` request of your cold indexes and displays the next 100 results:

```
GET _cold/indices/_search?page_size=100
{
"pagination_id": "je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY"
}
```

Migrating cold indexes to warm storage

After you narrow down your list of cold indexes with the filtering criteria in the previous section, migrate them back to UltraWarm where you can query the data and use it to create visualizations.

The following request migrates two cold indexes back to warm storage:

```
POST _cold/migration/_warm
 {
 "indices": "my-index1,my-index2"
 }


{
  "acknowledged" : true
}
```

To check the status of the migration and retrieve the migration ID, send the following request:

```
GET _cold/migration/_status
```

Sample response

```
{
  "cold_to_warm_migration_status" : [
    {
      "migration_id" : "tyLjXCA-S76zPQbPVHkOKA",
```

```
      "indices" : [
        "my-index1,my-index2"
      ],
      "state" : "RUNNING_INDEX_CREATION"
    }
  ]
}
```

To get index-specific migration information, include the index name:

```
GET _cold/migration/my-index/_status
```

Rather than specifying an index, you can list the indexes by their current migration status. Valid values are `_failed`, `_accepted`, and `_all`.

The following command gets the status of all indexes in a single migration request:

```
GET _cold/migration/_status?migration_id=my-migration-id
```

Retrieve the migration ID using the status request. For detailed migration information, add `&verbose=true`.

You can migrate indexes from cold to warm storage in batches of 10 or less, with a maximum of 100 indexes being migrated simultaneously. Any request that exceeds the limit will be rejected. To check the current number of migrations currently taking place, monitor the `ColdToWarmMigrationQueueSize` metric. The migration process has the following states:

```
ACCEPTED_MIGRATION_REQUEST - Migration request is accepted and queued.
RUNNING_INDEX_CREATION - Migration request is picked up for processing and will create
 warm indexes in the cluster.
PENDING_COLD_METADATA_CLEANUP - Warm index is created and the migration service will
 attempt to clean up cold metadata.
RUNNING_COLD_METADATA_CLEANUP - Cleaning up cold metadata from the indexes migrated to
 warm storage.
FAILED_COLD_METADATA_CLEANUP - Failed to clean up metadata in the cold tier.
FAILED_INDEX_CREATION - Failed to create an index in the warm tier.
```

Restoring cold indexes from snapshots

If you need to restore a deleted cold index, you can restore it back to the warm tier by following the instructions in the section called “Restoring warm indexes from snapshots” and then migrating

the index back to cold tier again. You can't restore a deleted cold index directly back to the cold tier. OpenSearch Service retains cold indexes for 14 days after they've been deleted.

Canceling migrations from cold to warm storage

If an index migration from cold to warm storage is queued or in a failed state, you can cancel it with the following request:

```
POST _cold/migration/my-index/_cancel

{
  "acknowledged" : true
}
```

To cancel migration for a batch of indexes (maximum of 10 at a time), specify the migration ID:

```
POST _cold/migration/_cancel?migration_id=my-migration-id

{
  "acknowledged" : true
}
```

Retrieve the migration ID using the status request.

Updating cold index metadata

You can update the `start_time` and `end_time` fields for a cold index:

```
PATCH _cold/my-index
 {
 "start_time": "2020-01-01",
 "end_time": "2020-02-01"
 }
```

You can't update the `timestamp_field` of an index in cold storage.

> **Note**
>
> OpenSearch Dashboards doesn't support the PATCH method. Use curl, Postman, or some other method to update cold metadata.

Deleting cold indexes

If you're not using an ISM policy you can delete cold indexes manually. The following request deletes a cold index:

```
DELETE _cold/my-index

{
  "acknowledged" : true
}
```

Disabling cold storage

The OpenSearch Service console is the simplest way to disable cold storage. Select the domain and choose **Actions**, **Edit cluster configuration**, then deselect **Enable cold storage**.

To use the AWS CLI or configuration API, under `ColdStorageOptions`, set `"Enabled"="false"`.

Before you disable cold storage, you must either delete all cold indexes or migrate them back to warm storage, otherwise the disable action fails.

OR1 storage for Amazon OpenSearch Service

OR1 is an instance family for Amazon OpenSearch Service that provides a cost-effective way to store large amounts of data. A domain with OR1 instances uses Amazon Elastic Block Store (Amazon EBS) `gp3` or `io1` volumes for primary storage, with data copied synchronously to Amazon S3 as it arrives. This storage structure provides increased indexing throughput with high durability. The OR1 instance family also supports automatic data recovery in the event of failure. For information about OR1 instance type options, see the section called “Current generation instance types”.

If you're running indexing heavy operational analytics workloads such as log analytics, observability, or security analytics, you can benefit from the improved performance and compute efficiency of OR1 instances. In addition, the automatic data recovery offered by OR1 instances improves the overall reliability of your domain.

OpenSearch Service sends storage-related OR1 metrics to Amazon CloudWatch. For a list of available metrics, see ???.

OR1 instances are available on-demand or with Reserved Instance pricing, with an hourly rate for the instances and storage provisioned in Amazon EBS and Amazon S3.

Topics

Limitations

Consider the following limitations when using OR1 instances for your domain.

- Newly created domains must be running OpenSearch version 2.11 or higher.
- Exisiting domains must be running OpenSearch version 2.15 or higher.
- Your domain must have encryption at rest enabled. For more information, see ???.
- If your domain uses dedicated master nodes, they must use Graviton instances. For more information about dedicated master nodes, see ???.
- The refresh interval for indexes on OR1 instances must be 10 seconds or higher. The default refresh interval for OR1 instances is 10 seconds.

Tuning for better ingestion throughput

To get the best indexing throughput from your OR1 instances, it is recommended you do the following:

- Use Large Bulk Sizes to improve buffer utilization. The recommended size is 10 MB.
- Use multiple clients to improve parallel processing performance.
- Set your number of active primary shards to match the number of data nodes to maximize resource utilization.

How OpenSearch optimized instances differ from non OpenSearch optimized instances

OpenSearch optimized instances differ from non OpenSearch optimized instances in the following ways:

- For OpenSearch optimized instances, indexing is only performed on primary shards.
- If OpenSearch optimized instances are configured with replicas, the indexing rate may appear lower than it actually is. For example, if there is 1 primary shard and 1 replica shard, the indexing rate may show a rate of 1000, however, the actual indexing rate is 2000.
- OpenSearch optimized instances perform buffer operations prior to sending to a remote source. This results in higher ingestion latencies.

> **Note**
>
> The `IndexingLatency` metric is not affected, as it doesn't include time to sync translog.)

- Replica shards can be a few seconds behind primary shards. The time lag can be seen from `ReplicationLagMaxTime` metric

How OR1 differs from UltraWarm storage

OpenSearch Service provides UltraWarm instances that are a cost-effective way to store large amounts of read-only data. Both OR1 and UltraWarm instances store data locally in Amazon EBS and remotely in Amazon S3. However, OR1 and UltraWarm instances differ in several important ways:

- OR1 instances keep a copy of data in *both* your local and remote store. In UltraWarm instances, data is kept primarily in remote store to reduce storage costs. Depending on your usage patterns, data can be moved to local storage.
- OR1 instances are active and can accept read and write operations, whereas the data on UltraWarm instances is read-only until you manually move it back to hot storage.
- UltraWarm relies on index snapshots for data durability. OR1 instances, by comparison, perform replication and recovery behind the scenes. In the event of a red index, OR1 instances will

automatically restore missing shards from your remote storage in Amazon S3. The recovery time varies depending on the volume of data to be recovered.

For more information about UltraWarm storage, see ???.

Using OR1 instances

You can select OR1 instances for your data nodes when you create a new domain with the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the AWS SDK. You can then index and query the data using your existing tools.

Console

1. Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/.
2. In the left navigation pane, choose **Domains**.
3. Choose **Create domain**.
4. Enter a name for your domain along with your other preferred options. Under **Instance family**, choose **OR1**. Choose **Create** to start the domain creation process.

AWS CLI

1. Navigate to your AWS CLI terminal. If you need to install the AWS CLI, see Install or update the latest version of the AWS CLI.
2. To use OR1 storage, you must provide the value of the specific OR1 instance type size in the `InstanceType` field when you create a domain. You must also enable encryption at rest.

   The following example creates a domain with OR1 instances of size `2xlarge`.

```
aws opensearch create-domain \
  --domain-name test-domain \
  --engine-version OpenSearch_2.11 \
  --cluster-config
 "InstanceType=or1.2xlarge.search,InstanceCount=3,DedicatedMasterEnabled=true,DedicatedMast
 \
  --ebs-options "EBSEnabled=true,VolumeType=gp3,VolumeSize=200" \
  --encryption-at-rest-options Enabled=true \
```

```
  --advanced-security-options
 "Enabled=true,InternalUserDatabaseEnabled=true,MasterUserOptions={MasterUserName=test-
user,MasterUserPassword=test-password}" \
  --node-to-node-encryption-options Enabled=true \
  --domain-endpoint-options EnforceHTTPS=true \
  --access-policies '{"Version":"2012-10-17","Statement":
[{"Effect":"Allow","Principal":
{"AWS":"*"},"Action":"es:*","Resource":"arn:aws:es:us-east-1:account-
id:domain/test-domain/*"}]}'
```

Index State Management in Amazon OpenSearch Service

Index State Management (ISM) in Amazon OpenSearch Service lets you define custom management policies that automate routine tasks, and apply them to indexes and index patterns. You no longer need to set up and manage external processes to run your index operations.

A policy contains a default state and a list of states for the index to transition between. Within each state, you can define a list of actions to perform and conditions that trigger these transitions. A typical use case is to periodically delete old indexes after a certain period of time. For example, you can define a policy that moves your index into a `read_only` state after 30 days and then ultimately deletes it after 90 days.

After you attach a policy to an index, ISM creates a job that runs every 5 to 8 minutes (or 30 to 48 minutes for pre-1.3 clusters) to perform policy actions, check conditions, and transition the index into different states. The base time for this job to run is every 5 minutes, plus a random 0-60% jitter is added to it to make sure you do not see a surge of activity from all your indexes at the same time. ISM doesn't run jobs if the cluster state is red.

ISM requires OpenSearch or Elasticsearch 6.8 or later.

> **Note**
>
> This documentation provides a brief overview of ISM and several sample policies. It also explains how ISM for Amazon OpenSearch Service domains differs from ISM on self-managed OpenSearch clusters. For full documentation of ISM, including a comprehensive parameter reference, descriptions of each setting, and an API reference, see Index State Management in the OpenSearch documentation.

Important

You can no longer use index templates to apply ISM policies to newly created indexes. You can continue to automatically manage newly created indexes with the ISM template field. This update introduces a breaking change that affects existing CloudFormation templates using this setting.

Create an ISM policy

To get started with Index State Management

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. Select the domain that you want to create an ISM policy for.
3. From the domain's dashboard, navigate to the OpenSearch Dashboards URL and sign in with your master username and password. The URL follows this format:

   ```
   domain-endpoint/_dashboards/
   ```

4. Open the left navigation panel within OpenSearch Dashboards and choose **Index Management**, then **Create policy**.
5. Use the visual editor or JSON editor to create policies. We recommend using the visual editor as it offers a more structured way of defining policies. For help creating policies, see the sample policies below.
6. After you create a policy, attach it to one or more indexes:

   ```
   POST _plugins/_ism/add/my-index
   {
     "policy_id": "my-policy-id"
   }
   ```

   Note

   If your domain is running a legacy Elasticsearch version, use `_opendistro` instead of `_plugins`.

Alternatively, select the index in OpenSearch Dashboards and choose **Apply policy**.

Sample policies

The following sample policies demonstrate how to automate common ISM use cases.

Hot to warm to cold storage

This sample policy moves an index from hot storage to UltraWarm, and eventually to cold storage. Then, it deletes the index.

The index is initially in the `hot` state. After ten days, ISM moves it to the `warm` state. 80 days later, after the index is 90 days old, ISM moves the index to the `cold` state. After a year, the service sends a notification to an Amazon Chime room that the index is being deleted and then permanently deletes it.

Note that cold indexes require the `cold_delete` operation rather than the normal `delete` operation. Also note that an explicit `timestamp_field` is required in your data in order to manage cold indexes with ISM.

```
{
  "policy": {
    "description": "Demonstrate a hot-warm-cold-delete workflow.",
    "default_state": "hot",
    "schema_version": 1,
    "states": [{
        "name": "hot",
        "actions": [],
        "transitions": [{
          "state_name": "warm",
          "conditions": {
            "min_index_age": "10d"
          }
        }]
      },
      {
        "name": "warm",
        "actions": [{
          "warm_migration": {},
          "retry": {
```

```
            "count": 5,
            "delay": "1h"
          }
        }],
        "transitions": [{
          "state_name": "cold",
          "conditions": {
            "min_index_age": "90d"
          }
        }]
      },
      {
        "name": "cold",
        "actions": [{
            "cold_migration": {
              "timestamp_field": "<your timestamp field>"
            }
          }
        ],
        "transitions": [{
          "state_name": "delete",
          "conditions": {
             "min_index_age": "365d"
          }
        }]
      },
      {
        "name": "delete",
        "actions": [{
          "notification": {
            "destination": {
              "chime": {
                "url": "<URL>"
              }
            },
            "message_template": {
              "source": "The index {{ctx.index}} is being deleted."
            }
          }
        },
        {
          "cold_delete": {}
        }]
      }
```

```
    ]
  }
}
```

Reduce replica count

This sample policy reduces replica count to zero after seven days to conserve disk space and then deletes the index after 21 days. This policy assumes your index is non-critical and no longer receiving write requests; having zero replicas carries some risk of data loss.

```
{
  "policy": {
    "description": "Changes replica count and deletes.",
    "schema_version": 1,
    "default_state": "current",
    "states": [{
        "name": "current",
        "actions": [],
        "transitions": [{
          "state_name": "old",
          "conditions": {
            "min_index_age": "7d"
          }
        }]
      },
      {
        "name": "old",
        "actions": [{
          "replica_count": {
            "number_of_replicas": 0
          }
        }],
        "transitions": [{
          "state_name": "delete",
          "conditions": {
            "min_index_age": "21d"
          }
        }]
      },
      {
        "name": "delete",
        "actions": [{
          "delete": {}
```

```
        }],
        "transitions": []
      }
    ]
  }
}
```

Take an index snapshot

This sample policy uses the `snapshot` operation to take a snapshot of an index as soon as it contains at least one document. `repository` is the name of the manual snapshot repository you registered in Amazon S3. `snapshot` is the name of the snapshot. For snapshot prerequisites and steps to register a repository, see the section called "Creating index snapshots".

```
{
  "policy": {
    "description": "Takes an index snapshot.",
    "schema_version": 1,
    "default_state": "empty",
    "states": [{
        "name": "empty",
        "actions": [],
        "transitions": [{
          "state_name": "occupied",
          "conditions": {
            "min_doc_count": 1
          }
        }]
      },
      {
        "name": "occupied",
        "actions": [{
          "snapshot": {
            "repository": "<my-repository>",
            "snapshot": "<my-snapshot>"
            }
          }],
          "transitions": []
      }
    ]
  }
}
```

ISM templates

You can set up an `ism_template` field in a policy so when you create an index that matches the template pattern, the policy is automatically attached to that index. In this example, any index you create with a name that begins with "log" is automatically matched to the ISM policy `my-policy-id`:

```
PUT _plugins/_ism/policies/my-policy-id
{
  "policy": {
    "description": "Example policy.",
    "default_state": "...",
    "states": [...],
    "ism_template": {
      "index_patterns": ["log*"],
      "priority": 100
    }
  }
}
```

For a more detailed example, see Sample policy with ISM template for auto rollover.

Differences

Compared to OpenSearch and Elasticsearch, ISM for Amazon OpenSearch Service has several differences.

ISM operations

- OpenSearch Service supports three unique ISM operations, `warm_migration`, `cold_migration`, and `cold_delete`:
  - If your domain has UltraWarm enabled, the `warm_migration` action transitions the index to warm storage.
  - If your domain has cold storage enabled, the `cold_migration` action transitions the index to cold storage, and the `cold_delete` action deletes the index from cold storage.

  Even if one of these actions doesn't complete within the set timeout period, the migration or deletion of indexes still continues. Setting an error_notification for one of the above actions will notify you that the action failed if it didn't complete within the timeout period, but the

notification is only for your own reference. The actual operation has no inherent timeout and continues to run until it eventually succeeds or fails.

- If your domain runs OpenSearch or Elasticsearch 7.4 or later, OpenSearch Service supports the ISM `open` and `close` operations.

- If your domain runs OpenSearch or Elasticsearch 7.7 or later, OpenSearch Service supports the ISM `snapshot` operation.

Cold storage ISM operations

For cold indexes, you must specify a `?type=_cold` parameter when you use the following ISM APIs:

- Add policy
- Remove policy
- Update policy
- Retry failed index
- Explain index

These APIs for cold indexes have the following additional differences:

- Wildcard operators are not supported except when you use it at the end. For example, `_plugins/_ism/<add, remove, change_policy, retry, explain>/logstash-*` is supported but `_plugins/_ism/<add, remove, change_policy, retry, explain>/iad-*-prod` isn't supported.
- Multiple index names and patterns are not supported. For example, `_plugins/_ism/<add, remove, change_policy, retry, explain>/app-logs` is supported but `_plugins/_ism/<add, remove, change_policy, retry, explain>/app-logs,sample-data` isn't supported.

ISM settings

OpenSearch and Elasticsearch let you change all available ISM settings using the `_cluster/settings` API. On Amazon OpenSearch Service, you can only change the following ISM settings:

- **Cluster-level settings:**
  - `plugins.index_state_management.enabled`
  - `plugins.index_state_management.history.enabled`
- **Index-level settings:**
  - `plugins.index_state_management.rollover_alias`

Tutorial: Automating Index State Management processes

This tutorial demonstrates how to implement an ISM policy that automates routine index management tasks and apply them to indexes and index patterns.

Index State Management (ISM) in Amazon OpenSearch Service lets you automate recurring index management activities, so you can avoid using additional tools to manage index lifecycles. You can create a policy that automates these operations based on index age, size, and other conditions, all from within your Amazon OpenSearch Service domain.

OpenSearch Service supports three storage tiers: the default "hot" state for active writing and low-latency analytics, UltraWarm for read-only data up to three petabytes, and cold storage for unlimited long-term archival.

This tutorial presents a sample use case of handling time-series data in daily indexes. In this tutorial, you set up a policy that takes an automated snapshot of each attached index after 24 hours. It then migrates the index from the default hot state to UltraWarm storage after two days, cold storage after 30 days, and finally deletes the index after 60 days.

Prerequisites

- Your OpenSearch Service domain must be running Elasticsearch version 6.8 or later.
- Your domain must have UltraWarm and cold storage enabled.
- You must register a manual snapshot repository for your domain.
- Your user role needs sufficient permissions to access the OpenSearch Service console. If necessary, validate and configure access to your domain.

Step 1: Configure the ISM policy

First, configure an ISM policy in OpenSearch Dashboards.

1. From your domain dashboard in the OpenSearch Service console, navigate to the OpenSearch Dashboards URL and sign in with your master username and password. The URL follows this format: *domain-endpoint*/_dashboards/.
2. In OpenSearch Dashboards, choose **Add sample data** and add one or more of the sample indexes to your domain.
3. Open the left navigation panel and choose **Index Management**, then choose **Create policy**.
4. Name the policy `ism-policy-example`.
5. Replace the default policy with the following policy:

```
{
  "policy": {
    "description": "Move indexes between storage tiers",
    "default_state": "hot",
    "states": [
      {
        "name": "hot",
        "actions": [],
        "transitions": [
          {
            "state_name": "snapshot",
            "conditions": {
              "min_index_age": "24h"
            }
          }
        ]
      },
      {
        "name": "snapshot",
        "actions": [
          {
            "retry": {
              "count": 5,
              "backoff": "exponential",
              "delay": "30m"
            },
            "snapshot": {
              "repository": "snapshot-repo",
              "snapshot": "ism-snapshot"
            }
          }
        ],
```

```
        "transitions": [
          {
            "state_name": "warm",
            "conditions": {
              "min_index_age": "2d"
            }
          }
        ]
      },
      {
        "name": "warm",
        "actions": [
          {
            "retry": {
              "count": 5,
              "backoff": "exponential",
              "delay": "1h"
            },
            "warm_migration": {}
          }
        ],
        "transitions": [
          {
            "state_name": "cold",
            "conditions": {
              "min_index_age": "30d"
            }
          }
        ]
      },
      {
        "name": "cold",
        "actions": [
          {
            "retry": {
              "count": 5,
              "backoff": "exponential",
              "delay": "1h"
            },
            "cold_migration": {
              "start_time": null,
              "end_time": null,
              "timestamp_field": "@timestamp",
              "ignore": "none"
```

```
            }
          }
        ],
        "transitions": [
          {
            "state_name": "delete",
            "conditions": {
              "min_index_age": "60d"
            }
          }
        ]
      },
      {
        "name": "delete",
        "actions": [
          {
            "cold_delete": {}
          }
        ],
        "transitions": []
      }
    ],
    "ism_template": [
      {
        "index_patterns": [
          "index-*"
        ],
        "priority": 100
      }
    ]
  }
}
```

Note

The `ism_template` field automatically attaches the policy to any newly created index that matches one of the specified `index_patterns`. In this case, all indexes that start with `index-`. You can modify this field to match an index format in your environment. For more information, see ISM templates.

6. In the `snapshot` section of the policy, replace *`snapshot-repo`* with the name of the snapshot repository that you registered for your domain. You can also optionally replace *`ism-snapshot`*, which will be the name of snapshot when it's created.
7. Choose **Create**. The policy is now visible on the **State management policies** page.

Step 2: Attach the policy to one or more indexes

Now that you created your policy, attach it to one or more indexes in your cluster.

1. Go to the **Hot indicies** tab and search for `opensearch_dashboards_sample`, which lists all of the sample indexes that you added in step 1.
2. Select all of the indexes and choose **Apply policy**, then choose the **ism-policy-example** policy that you just created.
3. Choose **Apply**.

You can monitor the indexes as they move through the various states on the **Policy managed indices** page.

Summarizing indexes in Amazon OpenSearch Service with index rollups

Index rollups in Amazon OpenSearch Service let you reduce storage costs by periodically rolling up old data into summarized indexes.

You pick the fields that interest you and use an index rollup to create a new index with only those fields aggregated into coarser time buckets. You can store months or years of historical data at a fraction of the cost with the same query performance.

Index rollups requires OpenSearch or Elasticsearch 7.9 or later.

> **Note**
>
> This documentation helps you get started with creating an index rollup job in Amazon OpenSearch Service. For comprehensive documentation, including a list of all available settings and a full API reference, see Index rollups in the OpenSearch documentation.

Creating an index rollup job

To get started, choose **Index Management** in OpenSearch Dashboards. Select **Rollup Jobs** and choose **Create rollup job**.

Step 1: Set up indexes

Set up the source and target indexes. The source index is the one that you want to roll up. The target index is where the index rollup results are saved.

After you create an index rollup job, you can't change your index selections.

Step 2: Define aggregations and metrics

Select the attributes with the aggregations (terms and histograms) and metrics (avg, sum, max, min, and value count) that you want to roll up. Make sure you don't add a lot of highly granular attributes, because you won't save much space.

Step 3: Specify schedules

Specify a schedule to roll up your indexes as it's being ingested. The index rollup job is enabled by default.

Step 4: Review and create

Review your configuration and select **Create**.

Step 5: Search the target index

You can use the standard `_search` API to search the target index. You can't access the internal structure of the data in the target index because the plugin automatically rewrites the query in the background to suit the target index. This is to make sure you can use the same query for the source and target index.

To query the target index, set `size` to 0:

```
GET target_index/_search
{
  "size": 0,
  "query": {
```

```
    "match_all": {}
  },
  "aggs": {
    "avg_cpu": {
      "avg": {
        "field": "cpu_usage"
      }
    }
  }
}
```

> **Note**
>
> OpenSearch versions 2.2 and later support searching multiple rollup indexes in one request. OpenSearch versions prior to 2.2 and legacy Elasticsearch OSS versions only support one rollup index per search.

Transforming indexes in Amazon OpenSearch Service

Whereas index rollup jobs let you reduce data granularity by rolling up old data into condensed indexes, transform jobs let you create a different, summarized view of your data centered around certain fields, so you can visualize or analyze the data in different ways.

Index transforms have an OpenSearch Dashboards user interface and a REST API. The feature requires OpenSearch 1.0 or later.

> **Note**
>
> This documentation provides a brief overview of index transforms to help you get started using it on an Amazon OpenSearch Service domain. For comprehensive documentation and a REST API reference, see Index transforms in the open source OpenSearch documentation.

Creating an index transform job

If you don't have any data in your cluster, use the sample flight data within OpenSearch Dashboards to try out transform jobs. After adding the data, launch OpenSearch Dashboards. Then choose **Index Management**, **Transform Jobs**, and **Create Transform Job**.

Step 1: Choose indexes

In the **Indices** section, select the source and target index. You can either select an existing target index or create a new one by entering a name for it.

If you want to transform just a subset of your source index, choose **Add Data Filter**, and use the OpenSearch query DSL to specify a subset of your source index.

Step 2: Choose fields

After choosing your indexes, choose the fields you want to use in your transform job, as well as whether to use groupings or aggregations.

- You can use groupings to place your data into separate buckets in your transformed index. For example, if you want to group all of the airport destinations within the sample flight data, group the `DestAirportID` field into a target field of `DestAirportID_terms` field, and you can find the grouped airport IDs in your transformed index after the transform job finishes.
- On the other hand, aggregations let you perform simple calculations. For example, you might include an aggregation in your transform job to define a new field of `sum_of_total_ticket_price` that calculates the sum of all airplane tickets. Then you can analyze the new data in your transformed index.

Step 3: Specify a schedule

Transform jobs are enabled by default and run on schedules. For **transform execution interval**, specify an interval in minutes, hours, or days.

Step 4: Review and monitor

Review your configuration and select **Create**. Then monitor the **Transform job status** column.

Step 5: Search the target index

After the job finishes, you can use the standard `_search` API to search the target index.

For example, after running a transform job that transforms the flight data based on the `DestAirportID` field, you can run the following request to return all fields that have a value of SFO:

```
GET target_index/_search
```

```
{
  "query": {
    "match": {
      "DestAirportID_terms" : "SFO"
    }
  }
}
```

Cross-cluster replication for Amazon OpenSearch Service

With cross-cluster replication in Amazon OpenSearch Service, you can replicate user indexes, mappings, and metadata from one OpenSearch Service domain to another. Using cross-cluster replication helps to ensure disaster recovery if there is an outage, and allows you to replicate data across geographically distant data centers to reduce latency. You pay standard AWS data transfer charges for the data transferred between domains.

Cross-cluster replication follows an active-passive replication model where the *local* or *follower* index pulls data from the *remote* or *leader* index. The leader index refers to the source of the data, or the index that you want to replicate data from. The follower index refers to the target for the data, or the index that you want to replicate data to.

Cross-cluster replication is available on domains running Elasticsearch 7.10 or OpenSearch 1.1 or later.

Note

This documentation describes how to set up cross-cluster replication from an Amazon OpenSearch Service perspective. This includes using the AWS Management Console to set up cross-cluster connections, which is not possible on a self-managed OpenSearch cluster. For full documentation, including a settings reference and a comprehensive API reference, see Cross-cluster replication in the OpenSearch documentation.

Topics

- Set up a cross-cluster connection
- Start replication
- Confirm replication
- Pause and resume replication
- Stop replication
- Auto-follow
- Upgrading connected domains

Limitations

Cross-cluster replication has the following limitations:

- You can't replicate data between Amazon OpenSearch Service domains and self-managed OpenSearch or Elasticsearch clusters.
- You can't replicate an index from a follower domain to another follower domain. If you want to replicate an index to multiple follower domains, you can only replicate it from the single leader domain.
- A domain can be connected, through a combination of inbound and outbound connections, to a maximum of 20 other domains.
- When you initially set up a cross-cluster connection, the leader domain must be on the same or a higher version than the follower domain.
- You can't use AWS CloudFormation to connect domains.
- You can't use cross-cluster replication on M3 or burstable (T2 and T3) instances.
- You can't replicate data between UltraWarm or cold indexes. Both indexes must be in hot storage.
- When you delete an index from the leader domain, the corresponding index on the follower domain isn't automatically deleted.

Prerequisites

Before you set up cross-cluster replication, make sure that your domains meet the following requirements:

- Elasticsearch 7.10 or OpenSearch 1.1 or later

- Fine-grained access control enabled
- Node-to-node encryption enabled

Permissions requirements

In order to start replication, you must include the `es:ESCrossClusterGet` permission on the remote (leader) domain. We recommend the following IAM policy on the remote domain. This policy also lets you perform other operations, such as indexing documents and performing standard searches:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "*"
        ]
      },
      "Action": [
        "es:ESHttp*"
      ],
      "Resource": "arn:aws:es:region:account:domain/leader-domain/*"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "es:ESCrossClusterGet",
      "Resource": "arn:aws:es:region:account:domain/leader-domain"
    }
  ]
}
```

Make sure that the `es:ESCrossClusterGet` permission is applied for `/leader-domain` and not `/leader-domain/*`.

In order for non-admin users to perform replication activities, they also need to be mapped to the appropriate permissions. Most permissions correspond to specific REST API operations. For

example, the `indices:admin/plugins/replication/index/_resume` permission lets you resume replication of an index. For a full list of permissions, see Replication permissions in the OpenSearch documentation.

> **Note**
>
> The commands to start replication and create a replication rule are special cases. Because they invoke background processes on the leader and follower domains, you must pass a `leader_cluster_role` and `follower_cluster_role` in the request. OpenSearch Service uses these roles in all backend replication tasks. For information about mapping and using these roles, see Map the leader and follower cluster roles in the OpenSearch documentation.

Set up a cross-cluster connection

To replicate indexes from one domain to another, you need to set up a cross-cluster connection between the domains. The easiest way to connect domains is through the **Connections** tab of the domain dashboard. You can also use the configuration API or the AWS CLI. Because cross-cluster replication follows a "pull" model, you initate connections from the follower domain.

> **Note**
>
> If you previously connected two domains to perform cross-cluster searches, you can't use that same connection for replication. The connection is marked as `SEARCH_ONLY` in the console. In order to perform replication between two previously connected domains, you must delete the connection and recreate it. When you've done this, the connection is available for both cross-cluster search and cross-cluster replication.

To set up a connection

1. In the Amazon OpenSearch Service console, select the follower domain, go to the **Connections** tab, and choose **Request**.
2. For **Connection alias**, enter a name for your connection.
3. Choose between connecting to a domain in your AWS account and Region or in another account or Region.

- To connect to a domain in your AWS account and Region, select the domain and choose **Request**.
- To connect to a domain in another AWS account or Region, specify the ARN of the remote domain and choose **Request**.

OpenSearch Service validates the connection request. If the domains are incompatible, the connection fails. If validation succeeds, it's sent to the destination domain for approval. When the destination domain approves the request, you can begin replication.

Cross-cluster replication supports bidirectional replication. This means that you can create an outbound connection from domain A to domain B, and another outbound connection from domain B to domain A. You can then set up replication so that domain A follows an index in domain B, and domain B follows an index in domain A.

Start replication

After you establish a cross-cluster connection, you can begin to replicate data. First, create an index on the leader domain to replicate:

```
PUT leader-01
```

To replicate that index, send this command to the follower domain:

```
PUT _plugins/_replication/follower-01/_start
{
   "leader_alias": "connection-alias",
   "leader_index": "leader-01",
   "use_roles":{
      "leader_cluster_role": "all_access",
      "follower_cluster_role": "all_access"
   }
}
```

You can find the connection alias on the **Connections** tab on your domain dashboard.

This example assumes that an admin is issuing the request and uses `all_access` for the `leader_cluster_role` and `follower_cluster_role` for simplicity. In production environments, however, we recommend that you create replication users on both the leader and

follower indexes, and map them accordingly. The usernames must be identical. For information about these roles and how to map them, see Map the leader and follower cluster roles in the OpenSearch documentation.

Confirm replication

To confirm that replication is happening, get the replication status:

```
GET _plugins/_replication/follower-01/_status

{
  "status" : "SYNCING",
  "reason" : "User initiated",
  "leader_alias" : "connection-alias",
  "leader_index" : "leader-01",
  "follower_index" : "follower-01",
  "syncing_details" : {
    "leader_checkpoint" : -5,
    "follower_checkpoint" : -5,
    "seq_no" : 0
  }
}
```

The leader and follower checkpoint values begin as negative integers and reflect the number of shards you have (-1 for one shard, -5 for five shards, and so on). The values increment to positive integers with each change that you make. If the values are the same, it means that the indexes are fully synced. You can use these checkpoint values to measure replication latency across your domains.

To further validate replication, add a document to the leader index:

```
PUT leader-01/_doc/1
{
   "Doctor Sleep":"Stephen King"
}
```

And confirm that it shows up on the follower index:

```
GET follower-01/_search

{
```

```
    ...
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "follower-01",
        "_type" : "_doc",
        "_id" : "1",
        "_score" : 1.0,
        "_source" : {
          "Doctor Sleep" : "Stephen King"
        }
      }
    ]
  }
}
```

Pause and resume replication

You can temporarily pause replication if you need to remediate issues or reduce load on the leader domain. Send this request to the follower domain. Make sure to include an empty request body:

```
POST _plugins/_replication/follower-01/_pause
{}
```

Then get the status to ensure that replication is paused:

```
GET _plugins/_replication/follower-01/_status

{
  "status" : "PAUSED",
  "reason" : "User initiated",
  "leader_alias" : "connection-alias",
  "leader_index" : "leader-01",
  "follower_index" : "follower-01"
}
```

When you're done making changes, resume replication. Send this request to the follower domain. Make sure to include an empty request body:

```
POST _plugins/_replication/follower-01/_resume
{}
```

You can't resume replication after it's been paused for more than 12 hours. You must stop replication, delete the follower index, and restart replication of the leader.

Stop replication

When you stop replication completely, the follower index unfollows the leader and becomes a standard index. You can't restart replication after you stop it.

Stop replication from the follower domain. Make sure to include an empty request body:

```
POST _plugins/_replication/follower-01/_stop
{}
```

Auto-follow

You can define a set of replication rules against a single leader domain that automatically replicate indexes that match a specified pattern. When an index on the leader domain matches one of the patterns (for example, `books*`), a matching follower index is created on the follower domain. OpenSearch Service replicates any existing indexes that match the pattern, as well as new indexes that you create. It does not replicate indexes that already exist on the follower domain.

To replicate all indexes (with the exception of system-created indexes, and those that already exist on the follower domain), use a wildcard (*) pattern.

Create a replication rule

Create a replication rule on the follower domain, and specify the name of the cross-cluster connection:

```
POST _plugins/_replication/_autofollow
{
   "leader_alias" : "connection-alias",
   "name": "rule-name",
   "pattern": "books*",
   "use_roles":{
      "leader_cluster_role": "all_access",
      "follower_cluster_role": "all_access"
   }
}
```

You can find the connection alias on the **Connections** tab on your domain dashboard.

This example assumes that an admin is issuing the request, and it uses `all_access` as the leader and follower domain roles for simplicity. In production environments, however, we recommend that you create replication users on both the leader and follower indexes and map them accordingly. The usernames must be identical. For information about these roles and how to map them, see Map the leader and follower cluster roles in the OpenSearch documentation.

To retrieve a list of existing replication rules on a domain, use the auto-follow stats API operation.

To test the rule, create an index that matches the pattern on the leader domain:

```
PUT books-are-fun
```

And check that its replica appears on the follower domain:

```
GET _cat/indices

health status index          uuid                   pri rep docs.count docs.deleted
 store.size pri.store.size
green  open   books-are-fun  ldfHO78xYYdxRMULuiTvSQ     1   1          0            0
     208b           208b
```

Delete a replication rule

When you delete a replication rule, OpenSearch Service stops replicating *new* indices that match the pattern, but continues existing replication activity until you stop replication of those indexes.

Delete replication rules from the follower domain:

```
DELETE _plugins/_replication/_autofollow
{
   "leader_alias" : "connection-alias",
   "name": "rule-name"
}
```

Upgrading connected domains

In order to upgrade the engine version of two domains that have a cross-cluster connection, upgrade the follower domain first and then the leader domain. Do not delete the connection between them, otherwise replication pauses and you won't be able to resume it.

Migrating Amazon OpenSearch Service indexes using remote reindex

Remote reindex lets you copy indexes from one Amazon OpenSearch Service domain to another. You can migrate indexes from any OpenSearch Service domains or self-managed OpenSearch and Elasticsearch clusters.

A *remote* domain and index refers to the source of the data, or the domain and index that you want to copy data from. A *local* domain and index refers to the target for the data, or the domain and index that you want to copy data to.

Remote reindexing requires OpenSearch 1.0 or later, or Elasticsearch 6.7 or later, on the local domain. The remote domain must be lower or the same major version as the local domain. Elasticsearch versions are considered to be *lower* than OpenSearch versions, meaning you can reindex data from Elasticsearch domains to OpenSearch domains. Within the same major version, the remote domain can be any minor version. For example, remote reindexing from Elasticsearch 7.10.x to 7.9 is supported, but OpenSearch 1.0 to Elasticsearch 7.10.x isn't supported.

> **Note**
>
> This documentation describes how to reindex data between Amazon OpenSearch Service domains. For full documentation for the `reindex` operation, including detailed steps and supported options, see Reindex document in the OpenSearch documentation.

Topics

Prerequisites

Remote reindex has the following requirements:

- The remote domain must be accessible from the local domain. For a remote domain that resides within a VPC, the local domain must have access to the VPC. This process varies by network configuration, but likely involves connecting to a VPN or managed network, or using the native VPC endpoint connection. To learn more, see the section called “VPC support”.
- The request must be authorized by the remote domain like any other REST request. If the remote domain has fine-grained access control enabled, you must have permission to perform reindex on the remote domain and read the index on the local domain. For more security considerations, see the section called “Fine-grained access control”.
- We recommend you create an index with the desired setting on your local domain before you start the reindex process.
- If your domain uses a T2 or T3 instance type for your data nodes, you can't use remote reindex.

Reindex data between OpenSearch Service internet domains

The most basic scenario is that the remote index is in the same AWS Region as your local domain with a publicly accessible endpoint and you have signed IAM credentials.

From the remote domain, specify the remote index to reindex from and the local index to reindex to:

```
POST _reindex
{
  "source": {
    "remote": {
      "host": "https://remote-domain-endpoint:443"
    },
    "index": "remote_index"
  },
  "dest": {
    "index": "local_index"
  }
}
```

You must add 443 at the end of the remote domain endpoint for a validation check.

To verify that the index is copied over to the local domain, send this request to the local domain:

```
GET local_index/_search
```

If the remote index is in a Region different from your local domain, pass in its Region name, such as in this sample request:

```
POST _reindex
{
  "source": {
    "remote": {
      "host": "https://remote-domain-endpoint:443",
      "region": "eu-west-1"
    },
    "index": "remote_index"
  },
  "dest": {
    "index": "local_index"
  }
}
```

In case of isolated Region like AWS GovCloud (US) or China Regions, the endpoint might not be accessible because your IAM user is not recognized in those Regions.

If the remote domain is secured with basic authentication, specify the username and password:

```
POST _reindex
{
  "source": {
    "remote": {
      "host": "https://remote-domain-endpoint:443",
      "username": "username",
      "password": "password"
    },
    "index": "remote_index"
  },
  "dest": {
    "index": "local_index"
  }
}
```

Reindex data between OpenSearch Service domains when the remote is in a VPC

Every OpenSearch Service domain is made up of its own internal virtual private cloud (VPC) infrastructure. When you create a new domain in an existing OpenSearch Service VPC, an elastic network interface is created for each data node in the VPC.

Because the remote reindex operation is performed from the remote OpenSearch Service domain, and therefore within its own private VPC, you need a way to access the local domain's VPC. You can either do this by using the built-in VPC endpoint connection feature to establish a connection through AWS PrivateLink, or by configuring a proxy.

If your local domain uses OpenSearch version 1.0 or later, you can use the console or the AWS CLI to create an AWS PrivateLink connection. An AWS PrivateLink connection allows resources in the local VPC to privately connect to resources in the remote VPC within the same AWS Region.

In order to create a VPC endpoint connection, the source domain to be reindexed must be in a local VPC, and both the source and destination domains must be in the same AWS Region.

Reindex data with the AWS Management Console

You can use remote reindex with the console to copy indexes between two domains that share a VPC endpoint connection.

1. Navigate to the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/.
2. In the left navigation pane, choose **Domains**.
3. Select the local domain, or the domain that you want to copy data to. This opens the domain details page. Choose the **Connections** tab below the general information and choose **Request**.
4. On the **Request connection** page, select **VPC Endpoint Connection** for your connection mode and enter other relevant details. These details include the remote domain, which is the domain that you want to copy data from. Then, choose **Request**.
5. Navigate to the remote domain's details page, choose the **Connections** tab, and find the **Inbound connections** table. Select the check box next to the name of the domain that you just created the connection from (the local domain). Choose **Approve**.
6. Navigate back to the local domain, choose the **Connections** tab, and find the **Outbound connections** table. After the connection between the two domains is active, an endpoint becomes available in the **Endpoint** column in the table. Copy the endpoint.

7. Open the dashboard for the local domain and choose **Dev Tools** in the left navigation. To confirm that the remote domain index doesn't exist on your local domain yet, run the following GET request. Replace *remote-domain-index-name* with your own index name.

```
GET remote-domain-index-name/_search
{
   "query":{
      "match_all":{}
   }
}
```

   In the output, you should see an error that indicates that the index wasn't found.

8. Below your GET request, create a POST request and use your endpoint as the remote host, as follows.

```
POST _reindex
{
   "source":{
      "remote":{
         "host":"connection-endpoint",
         "username":"username",
         "password":"password"
      },
      "index":"remote-domain-index-name"
   },
   "dest":{
      "index":"local-domain-index-name"
   }
}
```

   Run this request.

9. Run the GET request again. The output should now indicate that the local index exists. You can query this index to verify that OpenSearch copied all the data from the remote index.

Reindex data with OpenSearch Service API operations

You can use remote reindex with the API to copy indexes between two domains that share a VPC endpoint connection.

1. Use the CreateOutboundConnection API operation to request a new connection from your local domain to your remote domain.

```
POST https://es.region.amazonaws.com/2021-01-01/opensearch/cc/outboundConnection

{
   "ConnectionAlias": "remote-reindex-example",
   "ConnectionMode": "VPC_ENDPOINT",
   "LocalDomainInfo": {
      "AWSDomainInformation": {
         "DomainName": "local-domain-name",
         "OwnerId": "aws-account-id",
         "Region": "region"
      }
   },
   "RemoteDomainInfo": {
      "AWSDomainInformation": {
         "DomainName": "remote-domain-name",
         "OwnerId": "aws-account-id",
         "Region": "region"
      }
   }
}
```

   You receive a `ConnectionId` in the response. Save this ID to use in the next step.

2. Use the AcceptInboundConnection API operation with your connection ID to approve the request from the local domain.

```
PUT https://es.region.amazonaws.com/2021-01-01/opensearch/cc/
inboundConnection/ConnectionId/accept
```

3. Use the DescribeOutboundConnections API operation to retrieve the endpoint for your remote domain.

```
{
    "Connections": [
        {
            "ConnectionAlias": "remote-reindex-example",
            "ConnectionId": "connection-id",
            "ConnectionMode": "VPC_ENDPOINT",
            "ConnectionProperties": {
```

```
                "Endpoint": "connection-endpoint"
            },
            ...
        }
    ]
}
```

Save the *connection-endpoint* to use in Step 5.

4. To confirm that the remote domain index doesn't exist on your local domain yet, run the following GET request. Replace *remote-domain-index-name* with your own index name.

```
GET local-domain-endpoint/remote-domain-index-name/_search
{
   "query":{
      "match_all":{}
   }
}
```

In the output, you should see an error that indicates that the index wasn't found.

5. Create a POST request and use your endpoint as the remote host, as follows.

```
POST local-domain-endpoint/_reindex
{
   "source":{
      "remote":{
         "host":"connection-endpoint",
         "username":"username",
         "password":"password"
      },
      "index":"remote-domain-index-name"
   },
   "dest":{
      "index":"local-domain-index-name"
   }
}
```

Run this request.

6. Run the GET request again. The output should now indicate that the local index exists. You can query this index to verify that OpenSearch copied all the data from the remote index.

If the remote domain is hosted inside a VPC and you don't want to use the VPC endpoint connection feature, you must configure a proxy with a publicly accessible endpoint. In this case, OpenSearch Service requires a public endpoint because it doesn't have the ability to send traffic into your VPC.

When you run a domain in VPC mode, one or more endpoints are placed in your VPC. However, these endpoints are only for traffic coming into the domain within the VPC, and they don't permit traffic into the VPC itself.

The remote reindex command is run from the local domain, so the originating traffic isn't able to use those endpoints to access the remote domain. That's why a proxy is required in this use case. The proxy domain must have a certificate signed by a public certificate authority (CA). Self-signed or private CA-signed certificates are not supported.

Reindex data between non-OpenSearch Service domains

If the remote index is hosted outside of OpenSearch Service, like in a self-managed EC2 instance, set the `external` parameter to `true`:

```
POST _reindex
{
  "source": {
    "remote": {
      "host": "https://remote-domain-endpoint:443",
      "username": "username",
      "password": "password",
      "external": true
    },
    "index": "remote_index"
  },
  "dest": {
    "index": "local_index"
  }
}
```

In this case, only basic authentication with a username and password is supported. The remote domain must have a publicly accessible endpoint (even if it's in the same VPC as the local OpenSearch Service domain) and a certificate signed by a public CA. Self-signed or private CA-signed certificates aren't supported.

Reindex large datasets

Remote reindex sends a scroll request to the remote domain with the following default values:

- Search context of 5 minutes
- Socket timeout of 30 seconds
- Batch size of 1,000

We recommend tuning these parameters to accommodate your data. For large documents, consider a smaller batch size and/or longer timeout. For more information, see Scroll search.

```
POST _reindex?pretty=true&scroll=10h&wait_for_completion=false
{
  "source": {
    "remote": {
      "host": "https://remote-domain-endpoint:443",
      "socket_timeout": "60m"
    },
    "size": 100,
    "index": "remote_index"
  },
  "dest": {
    "index": "local_index"
  }
}
```

We also recommend adding the following settings to the local index for better performance:

```
PUT local_index
{
  "settings": {
    "refresh_interval": -1,
    "number_of_replicas": 0
  }
}
```

After the reindex process is complete, you can set your desired replica count and remove the refresh interval setting.

To reindex only a subset of documents that you select through a query, send this request to the local domain:

```
POST _reindex
{
  "source": {
    "remote": {
      "host": "https://remote-domain-endpoint:443"
    },
    "index": "remote_index",
    "query": {
      "match": {
        "field_name": "text"
      }
    }
  },
  "dest": {
    "index": "local_index"
  }
}
```

Remote reindex doesn't support slicing, so you can't perform multiple scroll operations for the same request in parallel.

Remote reindex settings

In addition to the standard reindexing options, OpenSearch Service supports the following options:

Options	Valid values	Description	Required
external	Boolean	If the remote domain is not an OpenSearch Service domain, or if you're reindexing between two VPC domains, specify as `true`.	No
region	String	If the remote domain is in a different	No

Options	Valid values	Description	Required
		Region, specify the Region name.	

Managing time-series data in Amazon OpenSearch Service with data streams

A typical workflow to manage time-series data involves multiple steps, such as creating a rollover index alias, defining a write index, and defining common mappings and settings for the backing indices.

Data streams in Amazon OpenSearch Service help simplify this initial setup process. Data streams work out of the box for time-based data such as application logs that are typically append-only in nature.

Data streams requires OpenSearch version 1.0 or later.

> **Note**
>
> This documentation provides basic steps to help you get started with data streams on an Amazon OpenSearch Service domain. For comprehensive documentation, see Data streams in the OpenSearch documentation.

Getting started with data streams

A data stream is internally composed of multiple backing indices. Search requests are routed to all the backing indices, while indexing requests are routed to the latest write index.

Step 1: Create an index template

To create a data stream, you first need to create an index template that configures a set of indexes as a data stream. The `data_stream` object indicates that it's a data stream and not a regular index template. The index pattern matches with the name of the data stream:

```
PUT _index_template/logs-template
```

```
{
  "index_patterns": [
    "my-data-stream",
    "logs-*"
  ],
  "data_stream": {},
  "priority": 100
}
```

In this case, each ingested document must have an `@timestamp` field. You can also define your own custom timestamp field as a property in the `data_stream` object:

```
PUT _index_template/logs-template
{
  "index_patterns": "my-data-stream",
  "data_stream": {
    "timestamp_field": {
      "name": "request_time"
    }
  }
}
```

Step 2: Create a data stream

After you create an index template, you can directly start ingesting data without creating a data stream.

Because we have a matching index template with a `data_stream` object, OpenSearch automatically creates the data stream:

```
POST logs-staging/_doc
{
  "message": "login attempt failed",
  "@timestamp": "2013-03-01T00:00:00"
}
```

Step 3: Ingest data into the data stream

To ingest data into a data stream, you can use the regular indexing APIs. Make sure every document that you index has a timestamp field. If you try to ingest a document that doesn't have a timestamp field, you get an error.

```
POST logs-redis/_doc
{
  "message": "login attempt",
  "@timestamp": "2013-03-01T00:00:00"
}
```

Step 4: Searching a data stream

You can search a data stream just like you search a regular index or an index alias. The search operation applies to all of the backing indexes (all data present in the stream).

```
GET logs-redis/_search
{
  "query": {
    "match": {
      "message": "login"
    }
  }
}
```

Step 5: Rollover a data stream

You can set up an Index State Management (ISM) policy to automate the rollover process for the data stream. The ISM policy is applied to the backing indexes at the time of their creation. When you associate a policy to a data stream, it only affects the future backing indexes of that data stream. You also don't need to provide the `rollover_alias` setting, because the ISM policy infers this information from the backing index.

> **Note**
>
> If you migrate a backing index to cold storage, OpenSearch removes this index from the data stream. Even if you move the index back to UltraWarm, the index remains independent and not part of the original data stream. After an index has been removed from the data stream, searching against the stream won't return any data from the index.

> **Warning**
>
> The write index for a data stream can't be migrated to cold storage. If you wish to migrate data in your data stream to cold storage, you must rollover the data stream before migration.

Step 6: Manage data streams in OpenSearch Dashboards

To manage data streams from OpenSearch Dashboards, open **OpenSearch Dashboards**, choose **Index Management**, select **Indices** or **Policy managed indices**.

Step 7: Delete a data stream

The delete operation first deletes the backing indexes of a data stream and then deletes the data stream itself.

To delete a data stream and all of its hidden backing indices:

```
DELETE _data_stream/name_of_data_stream
```

Monitoring data in Amazon OpenSearch Service

Proactively monitor your data in Amazon OpenSearch Service with alerting and anomaly detection. Set up alerts to receive notifications when your data exceeds certain thresholds. Anomaly detection uses machine learning to automatically detect any outliers in your streaming data. You can pair anomaly detection with alerting to ensure you're notified as soon as an anomaly is detected.

Topics

Configuring alerts in Amazon OpenSearch Service

Configure alerts in Amazon OpenSearch Service to get notified when data from one or more indexes meets certain conditions. For example, you might want to receive an email if your application logs more than five HTTP 503 errors in one hour, or you might want to page a developer if no new documents have been indexed in the last 20 minutes.

Alerting requires OpenSearch or Elasticsearch 6.2 or later.

> **Note**
>
> This documentation provides a brief overview of alerting and highlights how alerting on an Amazon OpenSearch Service domain differs from alerting on an open source OpenSearch cluster. For full alerting documentation, including a comprehensive API reference, a list of available request fields for composite monitors, and descriptions of available trigger and actions variables, see Alerting in the OpenSearch documentation.

Topics

Alerting permissions

Alerting supports fine-grained access control. For details on mixing and matching permissions to fit your use case, see Alerting security in the OpenSearch documentation.

In order to access the **Alerting** page within OpenSearch Dashboards, you must at least be mapped to the `alerting_read_access` predefined role, or be granted equivalent permissions. This role grants permissions to view alerts, destinations, and monitors, but not to acknowledge alerts or modify destinations or monitors.

Getting started with alerting

To create an alert, you configure a *monitor*, which is a job that runs on a defined schedule and queries OpenSearch indexes. You also configure one or more *triggers*, which define the conditions that generate events. Finally, you configure *actions*, which is what happens after an alert is triggered.

To get started with alerting

1. Choose **Alerting** from the OpenSearch Dashboards main menu and choose **Create monitor**.
2. Create a per-query, per-bucket, per-cluster metrics, or per-document monitor. For instructions, see Create a monitor.
3. For **Triggers**, create one or more triggers. For instructions, see Create triggers.
4. For **Actions**, set up a notification channel for the alert. Choose between Slack, Amazon Chime, a custom webhook, or Amazon SNS. As you might imagine, notifications require connectivity to the channel. For example, your OpenSearch Service domain must be able to connect to the internet to notify a Slack channel or send a custom webhook to a third-party server. The custom webhook must have a public IP address in order for an OpenSearch Service domain to send alerts to it.

> **Tip**
>
> After an action successfully sends a message, securing access to that message (for example, access to a Slack channel) is your responsibility. If your domain contains sensitive data, consider using triggers without actions and periodically checking Dashboards for alerts.

Notifications

Alerting integrates with Notifications, which is a unified system for OpenSearch notifications. Notifications let you configure which communication service you want to use and see relevant statistics and troubleshooting information. For comprehensive documentation, see Notifications in the OpenSearch documentation.

Your domain must be running OpenSearch version 2.3 or later to use notifications.

> **Note**
>
> OpenSearch notifications are separate from OpenSearch Service notifications, which provide details about service software updates, Auto-Tune enhancements, and other important domain-level information. OpenSearch notifications are plugin-specific.

Notification channels replaced alerting destinations starting with OpenSearch version 2.0. Destinations were officially deprecated, and all alerting notification will be managed through channels going forward.

When you upgrade your domains to version 2.3 or later (since OpenSearch Service support for 2.x starts with 2.3), your existing destinations are automatically migrated to notification channels. If a destination fails to migrate, the monitor will continue to use it until the monitor is migrated to a notification channel. For more inforation, see Questions about destinations in the OpenSearch documentation.

To get started with notifications, sign in to OpenSearch Dashboards and choose **Notifications**, **Channels**, and **Create channel**.

Amazon Simple Notification Service (Amazon SNS) is a supported channel type for notifications. In order to authenticate users, you either need to provide the user with full access to Amazon SNS, or let them assume an IAM role that has permissions to access Amazon SNS. For instructions, see Amazon SNS as a channel type.

Differences

Compared to the open-source version of OpenSearch, alerting in Amazon OpenSearch Service has some notable differences.

Alerting settings

OpenSearch Service lets you modify the following alerting settings:

- `plugins.scheduled_jobs.enabled`
- `plugins.alerting.alert_history_enabled`
- `plugins.alerting.alert_history_max_age`
- `plugins.alerting.alert_history_max_docs`
- `plugins.alerting.alert_history_retention_period`
- `plugins.alerting.alert_history_rollover_period`
- `plugins.alerting.filter_by_backend_roles`

All other settings use the default values which you can't change.

To disable alerting, send the following request:

```
PUT _cluster/settings
{
  "persistent" : {
    "plugins.scheduled_jobs.enabled" : false
  }
}
```

The following request configures alerting to automatically delete history indexes after seven days, rather than the default 30 days:

```
PUT _cluster/settings
{
  "persistent": {
    "plugins.alerting.alert_history_retention_period": "7d"
  }
}
```

If you previously created monitors and want to stop the creation of daily alerting indexes, delete all alert history indexes:

```
DELETE .plugins-alerting-alert-history-*
```

To reduce shard count for history indexes, create an index template. The following request sets history indexes for alerting to one shard and one replica:

```
PUT _index_template/template-name
{
  "index_patterns": [".opendistro-alerting-alert-history-*"],
  "template": {
    "settings": {
      "number_of_shards": 1,
      "number_of_replicas": 1
    }
  }
}
```

Depending on your tolerance for data loss, you might even consider using zero replicas. For more information about creating and managing index templates, see Index templates in the OpenSearch documentation.

Anomaly detection in Amazon OpenSearch Service

Anomaly detection in Amazon OpenSearch Service automatically detects anomalies in your OpenSearch data in near-real time by using the Random Cut Forest (RCF) algorithm. RCF is an unsupervised machine learning algorithm that models a sketch of your incoming data stream. The algorithm computes an `anomaly grade` and `confidence score` value for each incoming data point. Anomaly detection uses these values to differentiate an anomaly from normal variations in your data.

You can pair the anomaly detection plugin with the Alerting plugin to notify you as soon as an anomaly is detected.

Anomaly detection is available on domains running any OpenSearch version or Elasticsearch 7.4 or later. All instance types support anomaly detection except for `t2.micro` and `t2.small`.

> **Note**
>
> This documentation provides a brief overview of anomaly detection in the context of Amazon OpenSearch Service. For comprehensive documentation, including detailed steps, an API reference, a reference of all available settings, and steps to create visualizations and dashboards, see Anomaly detection in the open source OpenSearch documentation.

Prerequisites

Anomaly detection has the following prerequisites:

- Anomaly detection requires OpenSearch or Elasticsearch 7.4 or later.
- Anomaly detection only supports fine-grained access control on Elasticsearch versions 7.9 and later and all versions of OpenSearch. Prior to Elasticsearch 7.9, only admin users can create, view, and manage detectors.
- If your domain uses fine-grained access control, non-admin users must be mapped to the `anomaly_read_access` role in OpenSearch Dashboards in order to view detectors, or `anomaly_full_access` in order to create and manage detectors.

Getting started with anomaly detection

To get started, choose **Anomaly Detection** in OpenSearch Dashboards.

Step 1: Create a detector

A detector is an individual anomaly detection task. You can create multiple detectors, and all the detectors can run simultaneously, with each analyzing data from different sources.

Step 2: Add features to your detector

A feature is the field in your index that you check for anomalies. A detector can discover anomalies across one or more features. You must choose one of the following aggregations for each feature: `average()`, `sum()`, `count()`, `min()`, or `max()`.

> **Note**
>
> The `count()` aggregation method is only available in OpenSearch and Elasticsearch 7.7 or later. For Elasticsearch 7.4, use a custom expression like the following:

```
{
  "aggregation_name": {
     "value_count": {
        "field": "field_name"
     }
  }
```

```
}
```

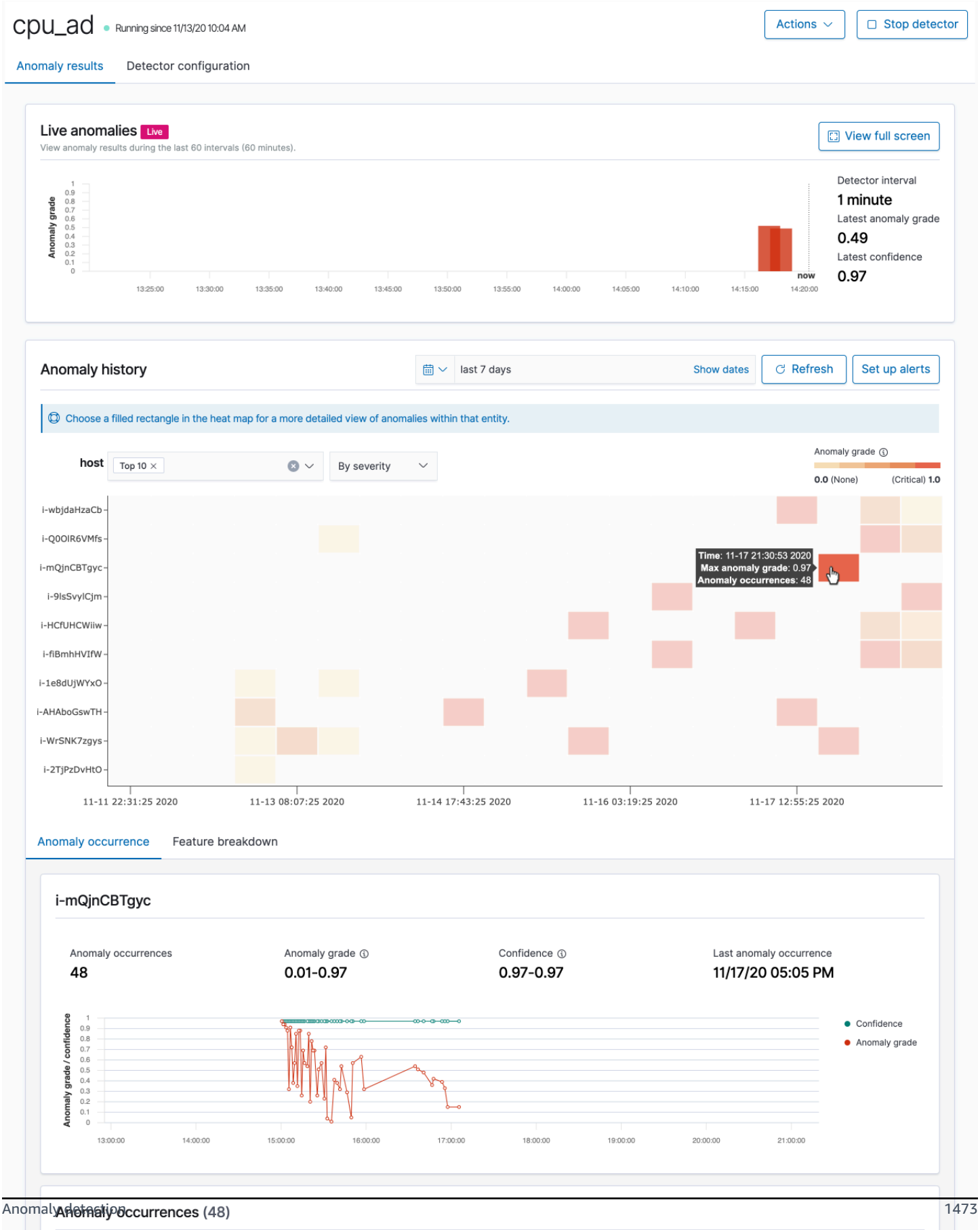
The aggregation method determines what constitutes an anomaly. For example, if you choose `min()`, the detector focuses on finding anomalies based on the minimum values of your feature. If you choose `average()`, the detector finds anomalies based on the average values of your feature. You can add a maximum of five features per detector.

You can configure the following optional settings (available in Elasticsearch 7.7 and later):

- **Category field** - Categorize or slice your data with a dimension like IP address, product ID, country code, and so on.
- **Window size** - Set the number of aggregation intervals from your data stream to consider in a detection window.

After you set up your features, preview sample anomalies and adjust the feature settings if necessary.

Step 3: Observe the results



- **Live anomalies** - displays the live anomaly results for the last 60 intervals. For example, if the interval is set to 10, it shows the results for the last 600 minutes. This chart refreshes every 30 seconds.
- **Anomaly history** - plots the anomaly grade with the corresponding measure of confidence.
- **Feature breakdown** - plots the features based on the aggregation method. You can vary the date-time range of the detector.
- **Anomaly occurrence** - shows the `Start time`, `End time`, `Data confidence`, and `Anomaly grade` for each anomaly detected.

  If you set the category field, you see an additional **Heat map** chart that correlates results for anomalous entities. Choose a filled rectangle to see a more detailed view of the anomaly.

Step 4: Set up alerts

To create a monitor to send you notifications when any anomalies are detected, choose **Set up alerts**. The plugin redirects you to the **Add monitor** page where you can configure an alert.

Tutorial: Detect high CPU usage with anomaly detection

This tutorial demonstrates how to create an anomaly detector in Amazon OpenSearch Service to detect high CPU usage. You'll use OpenSearch Dashboards to configure a detector to monitor CPU usage, and generate an alert when your CPU usage rises above a specified threshold.

> **Note**
>
> These steps apply to the latest version of OpenSearch and might differ slightly for past versions.

Prerequisites

- You must have an OpenSearch Service domain running Elasticsearch 7.4 or later, or any OpenSearch version.
- You must be ingesting application log files into your cluster that contain CPU usage data.

Step 1: Create a detector

First, create a detector that identifies anomalies in your CPU usage data.

1. Open the left panel menu in OpenSearch Dashboards and choose **Anomaly Detection**, then choose **Create detector**.
2. Name the detector **`high-cpu-usage`**.
3. For your data source, choose your index that contains CPU usage log files where you want to identify anomalies.
4. Choose the **Timestamp field** from your data. Optionally, you can add a data filter. This data filter analyzes only a subset of the data source and reduces the noise from data that's not relevant.
5. Set the **Detector interval** to **2** minutes. This interval defines the time (by minute interval) for the detector to collect the data.
6. In **Window delay**, add a **1-minute** delay. This delay adds extra processing time to ensure that all data within the window is present.
7. Choose **Next**. On the anomaly detection dashboard, under the detector name, choose **Configure model**.
8. For **Feature name**, enter **`max_cpu_usage`**. For **Feature state**, select **Enable feature**.
9. For **Find anomalies based on**, choose **Field value**.
10. For **Aggregation method**, choose **`max()`**.
11. For **Field**, select the field in your data to check for anomalies. For example, it might be called `cpu_usage_percentage`.
12. Keep all other settings as their defaults and choose **Next**.
13. Ignore the detector jobs setup and choose **Next**.
14. In the pop-up window, choose when to start the detector (automatically or manually), and then choose **Confirm**.

Now that the detector is configured, after it initializes, you will be able to see real-time results of the CPU usage in the **Real-time results** section of your detector panel. The **Live anomalies** section displays any anomalies that occur as data is being ingested in real time.

Step 2: Configure an alert

Now that you've created a detector, create a monitor that invokes an alert to send a message to Slack when it detects CPU usage that meets the conditions specified in the detector settings. You'll receive Slack notifications when data from one or more indexes meets the conditions that invoke the alert.

1. Open the left panel menu in OpenSearch Dashboards and choose **Alerting**, then choose **Create monitor**.
2. Provide a name for the monitor.
3. For **Monitor type**, choose **Per-query monitor**. A per-query monitor runs a specified query and defines the triggers.
4. For **Monitor defining method**, choose **Anomaly detector**, then select the detector that you created in the previous section from the **Detector** dropdown menu.
5. For **Schedule**, choose how often the monitor collects data and how often you receive alerts. For the purposes of this tutorial, set the schedule to run every **7** minutes.
6. In the **Triggers** section, choose **Add trigger**. For **Trigger name**, enter **`High CPU usage`**. For this tutorial, for **Severity level**, choose **1**, which is the highest level of severity.
7. For **Anomaly grade threshold**, choose **IS ABOVE**. On the menu under that, choose the grade threshold to apply. For this tutorial, set the **Anomaly grade** to **0.7**.
8. For **Anomaly confidence threshold**, choose **IS ABOVE**. On the menu under that, enter the same number as your Anomaly grade. For this tutorial, set the **Anomaly confidence threshold** to **0.7**.
9. In the **Actions** section, choose **Destination**. In the **Name** field, choose the name of the destination. On the **Type** menu, choose **Slack**. In the **Webhook URL** field, enter a webhook URL to receive alerts to. For more information, see Sending messages using incoming webhooks.
10. Choose **Create**.

Related resources

- the section called “Alerting”

- the section called “Anomaly detection”
- Anomaly detection API

Machine learning for Amazon OpenSearch Service

ML Commons is an OpenSearch plugin that provides a set of common machine learning (ML) algorithms through transport and REST API calls. Those calls choose the right nodes and resources for each ML request and monitors ML tasks to ensure uptime. This allows you to leverage existing open-source ML algorithms and reduce the effort required to develop new ML features. For more about the plugin, see Machine learning in the OpenSearch documentation. This chapter covers how to use the plugin with Amazon OpenSearch Service.

Topics

Amazon OpenSearch Service ML connectors for AWS services

When you use Amazon OpenSearch Service machine learning (ML) connectors with another AWS service, you need to set up an IAM role to securely connect OpenSearch Service to that service. AWS services that you can set up a connector to include Amazon SageMaker AI and Amazon Bedrock. In this tutorial, we cover how to create a connector from OpenSearch Service to SageMaker Runtime. For more information about connectors, see Supported connectors.

Topics

Prerequisites

To create a connector, you must have an Amazon SageMaker AI Domain endpoint and an IAM role that grants OpenSearch Service access.

Set up an Amazon SageMaker AI domain

See Deploy a Model in Amazon SageMaker AI in the *Amazon SageMaker AI Developer Guide* to deploy your machine learning model. Note the endpoint URL for your model, which you need in order to create an AI connector.

Create an IAM role

Set up an IAM role to delegate SageMaker Runtime permissions to OpenSearch Service. To create a new role, see Creating an IAM role (console) in the *IAM User Guide*. Optionally, you could use an existing role as long as it has the same set of privileges. If you do create a new role instead of using an AWS managed role, replace `opensearch-sagemaker-role` in this tutorial with the name of your own role.

1. Attach the following managed IAM policy to your new role to allow OpenSearch Service to access to your SageMaker AI endpoint. To attach a policy to a role, see Adding IAM identity permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sagemaker:InvokeEndpointAsync",
                "sagemaker:InvokeEndpoint"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

2. Follow the instructions in Modifying a role trust policy to edit the trust relationship of the role. You must specify OpenSearch Service in the `Principal` statement:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sts:AssumeRole"
```

```
            ],
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "opensearchservice.amazonaws.com"
                ]
            }
        }
    ]
}
```

We recommend that you use the `aws:SourceAccount` and `aws:SourceArn` condition keys to limit access to a specific domain. The `SourceAccount` is the AWS account ID that belongs to the owner of the domain, and the `SourceArn` is the ARN of the domain. For example, you can add the following condition block to the trust policy:

```
"Condition": {
    "StringEquals": {
        "aws:SourceAccount": "account-id"
    },
    "ArnLike": {
        "aws:SourceArn": "arn:aws:es:region:account-id:domain/domain-name"
    }
}
```

Configure permissions

In order to create the connector, you need permission to pass the IAM role to OpenSearch Service. You also need access to the `es:ESHttpPost` action. To grant both of these permissions, attach the following policy to the IAM role whose credentials are being used to sign the request:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::account-id:role/opensearch-sagemaker-role"
    },
    {
      "Effect": "Allow",
```

```
      "Action": "es:ESHttpPost",
      "Resource": "arn:aws:es:region:account-id:domain/domain-name/*"
    }
  ]
}
```

If your user or role doesn't have `iam:PassRole` permissions to pass your role, you might encounter an authorization error when you try to register a repository in the next step.

Map the ML role in OpenSearch Dashboards (if using fine-grained access control)

Fine-grained access control introduces an additional step when setting up a connector. Even if you use HTTP basic authentication for all other purposes, you need to map the `ml_full_access` role to your IAM role that has `iam:PassRole` permissions to pass `opensearch-sagemaker-role`.

1. Navigate to the OpenSearch Dashboards plugin for your OpenSearch Service domain. You can find the Dashboards endpoint on your domain dashboard on the OpenSearch Service console.
2. From the main menu choose **Security**, **Roles**, and select the **ml_full_access** role.
3. Choose **Mapped users**, **Manage mapping**.
4. Under **Backend roles**, add the ARN of the role that has permissions to pass `opensearch-sagemaker-role`.

   ```
   arn:aws:iam::account-id:role/role-name
   ```

5. Select **Map** and confirm the user or role shows up under **Mapped users**.

Create an OpenSearch Service connector

To create a connector, send a POST request to the OpenSearch Service domain endpoint. You can use curl, the sample Python client, Postman, or another method to send a signed request. Note that you can't use a POST request in the Kibana console. The request takes the following format:

```
POST domain-endpoint/_plugins/_ml/connectors/_create
{
   "name": "sagemaker: embedding",
   "description": "Test connector for Sagemaker embedding model",
   "version": 1,
   "protocol": "aws_sigv4",
   "credential": {
```

```
      "roleArn": "arn:aws:iam::account-id:role/opensearch-sagemaker-role"
   },
   "parameters": {
      "region": "region",
      "service_name": "sagemaker"
   },
   "actions": [
      {
         "action_type": "predict",
         "method": "POST",
         "headers": {
            "content-type": "application/json"
         },
         "url": "https://runtime.sagemaker.region.amazonaws.com/endpoints/endpoint-id/
invocations",
         "request_body": "{ \"inputs\": { \"question\": \"${parameters.question}\",
 \"context\": \"${parameters.context}\" } }"
      }
   ]
}
```

If your domain resides within a virtual private cloud (VPC), your computer must be connected to the VPC for the request to successfully create the AI connector. Accessing a VPC varies by network configuration, but usually involves connecting to a VPN or corporate network. To check that you can reach your OpenSearch Service domain, navigate to `https://your-vpc-domain.region.es.amazonaws.com` in a web browser and verify that you receive the default JSON response.

Sample Python client

The Python client is simpler to automate than a HTTP request and has better reusability. To create the AI connector with the Python client, save the following sample code to a Python file. The client requires the AWS SDK for Python (Boto3), `requests`, and `requests-aws4auth` packages.

```
import boto3
import requests
from requests_aws4auth import AWS4Auth

host = 'domain-endpoint/'
region = 'region'
service = 'es'
credentials = boto3.Session().get_credentials()
```

```
awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region, service,
 session_token=credentials.token)

# Register repository
path = '_plugins/_ml/connectors/_create'
url = host + path

payload = {
   "name": "sagemaker: embedding",
   "description": "Test connector for Sagemaker embedding model",
   "version": 1,
   "protocol": "aws_sigv4",
   "credential": {
      "roleArn": "arn:aws:iam::account-id:role/opensearch-sagemaker-role"
   },
   "parameters": {
      "region": "region",
      "service_name": "sagemaker"
   },
   "actions": [
      {
         "action_type": "predict",
         "method": "POST",
         "headers": {
            "content-type": "application/json"
         },
         "url": "https://runtime.sagemaker.region.amazonaws.com/endpoints/endpoint-id/
invocations",
         "request_body": "{ \"inputs\": { \"question\": \"${parameters.question}\",
 \"context\": \"${parameters.context}\" } }"
      }
   ]
}
headers = {"Content-Type": "application/json"}

r = requests.post(url, auth=awsauth, json=payload, headers=headers)
print(r.status_code)
print(r.text)
```

Amazon OpenSearch Service ML connectors for third-party platforms

In this tutorial, we cover how to create a connector from OpenSearch Service to Cohere. For more information about connectors, see Supported connectors.

When you use an Amazon OpenSearch Service machine learning (ML) connector with an external remote model, you need to store your specific authorization credentials in AWS Secrets Manager. This could be an API key, or a username and password combination. This means you also need to create an IAM role that allows OpenSearch Service access to read from Secrets Manager.

Topics

Prerequisites

To create a connector for Cohere or any external provider with OpenSearch Service, you must have an IAM role that grants OpenSearch Service access to AWS Secrets Manager, where you store your credentials. You must also store your credentials in Secrets Manager.

Create an IAM role

Set up an IAM role to delegate Secrets Manager permissions to OpenSearch Service. You can also use the existing `SecretManagerReadWrite` role. To create a new role, see Creating an IAM role (console) in the *IAM User Guide*. If you do create a new role instead of using an AWS managed role, replace `opensearch-secretmanager-role` in this tutorial with the name of your own role.

1. Attach the following managed IAM policy to your new role to allow OpenSearch Service to access to your Secrets Manager values. To attach a policy to a role, see Adding IAM Identity Permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "secretsmanager:GetSecretValue"
```

```
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

2. Follow the instructions in Modifying a role trust policy to edit the trust relationship of the role. You must specify OpenSearch Service in the `Principal` statement:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sts:AssumeRole"
            ],
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "opensearchservice.amazonaws.com"
                ]
            }
        }
    ]
}
```

We recommend that you use the `aws:SourceAccount` and `aws:SourceArn` condition keys to limit access to specific domain. The `SourceAccount` is the AWS account ID that belongs to the owner of the domain, and the `SourceArn` is the ARN of the domain. For example, you can add the following condition block to the trust policy:

```
"Condition": {
    "StringEquals": {
        "aws:SourceAccount": "account-id"
    },
    "ArnLike": {
        "aws:SourceArn": "arn:aws:es:region:account-id:domain/domain-name"
    }
}
```

Configure permissions

In order to create the connector, you need permission to pass the IAM role to OpenSearch Service. You also need access to the `es:ESHttpPost` action. To grant both of these permissions, attach the following policy to the IAM role whose credentials are being used to sign the request:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::account-id:role/opensearch-secretmanager-role"
    },
    {
      "Effect": "Allow",
      "Action": "es:ESHttpPost",
      "Resource": "arn:aws:es:region:account-id:domain/domain-name/*"
    }
  ]
}
```

If your user or role doesn't have `iam:PassRole` permissions to pass your role, you might encounter an authorization error when you try to register a repository in the next step.

Set up AWS Secrets Manager

To store your authorization credentials in Secrets Manager, see Create an AWS Secrets Manager secret in the *AWS Secrets Manager User Guide*.

After Secrets Manager accepts your key-value pair as a secret, you receive an ARN with the format: `arn:aws:secretsmanager:us-west-2:123456789012:secret:MySecret-a1b2c3`. Keep a record of this ARN, as you use it and your key when you create a connector in the next step.

Map the ML role in OpenSearch Dashboards (if using fine-grained access control)

Fine-grained access control introduces an additional step when setting up a connector. Even if you use HTTP basic authentication for all other purposes, you need to map the `ml_full_access` role to your IAM role that has `iam:PassRole` permissions to pass `opensearch-sagemaker-role`.

1. Navigate to the OpenSearch Dashboards plugin for your OpenSearch Service domain. You can find the Dashboards endpoint on your domain dashboard on the OpenSearch Service console.

2. From the main menu choose **Security**, **Roles**, and select the **ml_full_access** role.
3. Choose **Mapped users**, **Manage mapping**.
4. Under **Backend roles**, add the ARN of the role that has permissions to pass `opensearch-sagemaker-role`.

```
arn:aws:iam::account-id:role/role-name
```

5. Select **Map** and confirm the user or role shows up under **Mapped users**.

Create an OpenSearch Service connector

To create a connector, send a POST request to the OpenSearch Service domain endpoint. You can use curl, the sample Python client, Postman, or another method to send a signed request. Note that you can't use a POST request in the Kibana console. The request takes the following format:

```
POST domain-endpoint/_plugins/_ml/connectors/_create
{
    "name": "Cohere Connector: embedding",
    "description": "The connector to cohere embedding model",
    "version": 1,
    "protocol": "http",
    "credential": {
        "secretArn": "arn:aws:secretsmanager:region:account-id:secret:cohere-key-id",
        "roleArn": "arn:aws:iam::account-id:role/opensearch-secretmanager-role"
    },
    "actions": [
        {
            "action_type": "predict",
            "method": "POST",
            "url": "https://api.cohere.ai/v1/embed",
            "headers": {
                "Authorization": "Bearer ${credential.secretArn.cohere-key-used-in-
secrets-manager}"
            },
            "request_body": "{ \"texts\": ${parameters.texts}, \"truncate\": \"END\" }"
        }
    ]
}
```

The request body for this request is different than that of an open-source connector request in two ways. Inside the `credential` field, you pass the ARN for the IAM role that permits OpenSearch Service to read from Secrets Manager, along with the ARN for the what secret. In the `headers` field, you refer to the secret using the secret key and the fact its coming from an ARN.

If your domain resides within a virtual private cloud (VPC), your computer must be connected to the VPC for the request to successfully create the AI connetor. Accessing a VPC varies by network configuration, but usually involves connecting to a VPN or corporate network. To check that you can reach your OpenSearch Service domain, navigate to `https://`*`your-vpc-domain`*`.`*`region`*`.es.amazonaws.com` in a web browser and verify that you receive the default JSON response.

Sample Python client

The Python client is simpler to automate than a HTTP request and has better reusability. To create the AI connector with the Python client, save the following sample code to a Python file. The client requires the AWS SDK for Python (Boto3), `requests`, and `requests-aws4auth` packages.

```
import boto3
import requests
from requests_aws4auth import AWS4Auth

host = 'domain-endpoint/'
region = 'region'
service = 'es'
credentials = boto3.Session().get_credentials()
awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region, service,
 session_token=credentials.token)

path = '_plugins/_ml/connectors/_create'
url = host + path

payload = {
    "name": "Cohere Connector: embedding",
    "description": "The connector to cohere embedding model",
    "version": 1,
    "protocol": "http",
    "credential": {
        "secretArn": "arn:aws:secretsmanager:region:account-id:secret:cohere-key-id",
        "roleArn": "arn:aws:iam::account-id:role/opensearch-secretmanager-role"
    },
    "actions": [
```

```
        {
            "action_type": "predict",
            "method": "POST",
            "url": "https://api.cohere.ai/v1/embed",
            "headers": {
                "Authorization": "Bearer ${credential.secretArn.cohere-key-used-in-
secrets-manager}"
            },
            "request_body": "{ \"texts\": ${parameters.texts}, \"truncate\": \"END\" }"
        }
    ]
}

headers = {"Content-Type": "application/json"}

r = requests.post(url, auth=awsauth, json=payload, headers=headers)
print(r.status_code)
print(r.text)
```

Using AWS CloudFormation to set up remote inference for semantic search

Starting with OpenSearch version 2.9, you can use remote inference with semantic search to host your own machine learning (ML) models. Remote inference uses the ML Commons plugin to allow you to host your model inferences remotely on ML services, such as Amazon SageMaker AI and Amazon BedRock, and connect them to Amazon OpenSearch Service with ML connectors.

To ease the setup of remote inference, Amazon OpenSearch Service provides an AWS CloudFormation template in the console. CloudFormation is an AWS service that lets you model, provision, and manage AWS and third-party resources by treating infrastructure as code.

The OpenSearch CloudFormation template automates the model provisioning process for you, so that you can easily create a model in your OpenSearch Service domain and then use the model ID to ingest data and run neural search queries.

When you use neural sparse encoders with OpenSearch Service version 2.12 and onwards, we recommend that you use the tokenizer model locally instead of deploying remotely. For more information, see Sparse encoding models in the OpenSearch documentation.

Topics

Prerequisites

To use a CloudFormation template with OpenSearch Service, complete the following prerequisites.

Set up an OpenSearch Service domain

Before you can use a CloudFormation template, you must set up an Amazon OpenSearch Service domain with version 2.9 or later and fine-grained access control enabled. Create an OpenSearch Service backend role to give the ML Commons plugin permission to create your connector for you.

The CloudFormation template creates a Lambda IAM role for you with the default name `LambdaInvokeOpenSearchMLCommonsRole`, which you can override if you want to choose a different name. After the template creates this IAM role, you need to give the Lambda function permission to call your OpenSearch Service domain. To do so, map the role named `ml_full_access` to your OpenSearch Service backend role with the following steps:

1. Navigate to the OpenSearch Dashboards plugin for your OpenSearch Service domain. You can find the Dashboards endpoint on your domain dashboard on the OpenSearch Service console.
2. From the main menu choose **Security**, **Roles**, and select the **ml_full_access** role.
3. Choose **Mapped users**, **Manage mapping**.
4. Under **Backend roles**, add the ARN of the Lambda role that needs permission to call your domain.

   ```
   arn:aws:iam::account-id:role/role-name
   ```

5. Select **Map** and confirm the user or role shows up under **Mapped users**.

After you've mapped the role, navigate to the security configuration of your domain and add the Lambda IAM role to your OpenSearch Service access policy.

Enable permissions on your AWS account

Your AWS account must have permission to access CloudFormation and Lambda, along with whichever AWS service you choose for your template – either SageMaker Runtime or Amazon BedRock.

If you're using Amazon Bedrock, you must also register your model. See Model access in the *Amazon Bedrock User Guide* to register your model.

If you're using your own Amazon S3 bucket to provide model artifacts, you must add the CloudFormation IAM role to your S3 access policy. For more information, see Adding and removing IAM identity permissions in the *IAM User Guide*.

Amazon SageMaker AI templates

The Amazon SageMaker AI CloudFormation templates define multiple AWS resources in order to set up the neural plugin and semantic search for you.

First, use the **Integration with text embedding models through Amazon SageMaker** template to deploy a text embedding model in SageMaker Runtime as a server. If you don't provide a model endpoint, CloudFormation creates an IAM role that allows SageMaker Runtime to download model artifacts from Amazon S3 and deploy them to the server. If you provide an endpoint, CloudFormation creates an IAM role that allows the Lambda function to access the OpenSearch Service domain or, if the role already exists, updates and reuses the role. The endpoint serves the remote model that is used for the ML connector with the ML Commons plugin.

Next, use the **Integration with Sparse Encoders through Amazon Sagemaker** template to create a Lambda function that has your domain set up remote inference connectors. After the connector is created in OpenSearch Service, the remote inference can run semantic search using the remote model in SageMaker Runtime. The template returns the model ID in your domain back to you to so you can start searching.

To use the Amazon SageMaker AI CloudFormation templates

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. In the left navigation, choose **Integrations**.
3. Under each of the Amazon SageMaker AI templates, choose **Configure domain**, **Configure public domain**.
4. Follow the prompt in the CloudFormation console to provision your stack and set up a model.

> **Note**
>
> OpenSearch Service also provides a separate template to configure VPC domain. If you use this template, you need to provide the VPC ID for the Lambda function.

Amazon Bedrock templates

Similar to the Amazon SageMaker AI CloudFormation templates, the Amazon Bedrock CloudFormation template provisions the AWS resources needed to create connectors between OpenSearch Service and Amazon Bedrock.

First, the template creates an IAM role that allows the future Lambda function to access your OpenSearch Service domain. The template then creates the Lambda function, which has the domain create a connector using the ML Commons plugin. After OpenSearch Service creates the connector, the remote inference set up is finished and you can run semantic searches using the Amazon Bedrock API operations.

Note that since Amazon Bedrock hosts its own ML models, you don't need to deploy a model to SageMaker Runtime. Instead, the template uses a predetermined endpoint for Amazon Bedrock and skips the endpoint provision steps.

To use the Amazon Bedrock CloudFormation template

1. Open the Amazon OpenSearch Service console at https://console.aws.amazon.com/aos/home.
2. In the left navigation, choose **Integrations**.
3. Under **Integrate with Amazon Titan Text Embeddings model through Amazon Bedrock**, choose **Configure domain**, **Configure public domain**.
4. Follow the prompt to set up your model.

> **Note**
>
> OpenSearch Service also provides a separate template to configure VPC domain. If you use this template, you need to provide the VPC ID for the Lambda function.

In addition, OpenSearch Service provides the following Amazon Bedrock templates to connect to the Cohere model and the Amazon Titan multimodal embeddings model:

- `Integration with Cohere Embed through Amazon Bedrock`
- `Integrate with Amazon Bedrock Titan Multi-modal`

Unsupported ML Commons settings

Amazon OpenSearch Service doesn't support use of the following ML Commons settings:

- `plugins.ml_commons.allow_registering_model_via_url`
- `plugins.ml_commons.allow_registering_model_via_local_file`

For more information on ML Commons settings, see ML Commons cluster settings.

OpenSearch Service flow framework templates

Amazon OpenSearch Service flow framework templates allow you to automate complex OpenSearch Service setup and preprocessing tasks by providing templates for common use cases. For example, you can use flow framework templates to automate machine learning setup tasks. Amazon OpenSearch Service flow framework templates provide a compact description of the setup process in a JSON or YAML document. These templates describe automated workflow configurations for conversational chat or query generation, AI connectors, tools, agents, and other components that prepare OpenSearch Service for backend use for generative models.

Amazon OpenSearch Service flow framework templates can be customized to meet your specific needs. To see an example of a custom flow framework template, see flow-framework. For OpenSearch Service provided templates, see workflow-templates. For comprehensive documentation, including detailed steps, an API reference, and a reference of all available settings, see automating configuration in the open source OpenSearch documentation.

> **Note**
>
> Flow-framework does not support backend role filtering for OpenSearch Service 2.17.

Creating ML connectors in OpenSearch Service

Amazon OpenSearch Service flow framework templates allow you to configure and install ML connectors by utilizing the create connector API offered in ml-commons. You can use ML

connectors to connect OpenSearch Service to other AWS services or third party platforms. For more information on this, see Creating connectors for third-party ML platforms. The Amazon OpenSearch Service flow framework API allows you to automate OpenSearch Service setup and preprocessing tasks and can be used to create ML connectors.

Before you can create a connector in OpenSearch Service, you must do the following:

- Create an Amazon SageMaker AI domain.
- Create an IAM role.
- Configure pass role permission.
- Map the flow-framework and ml-commons roles in OpenSearch Dashboards.

For more information on how to setup ML connectors for AWS services, see Amazon OpenSearch Service ML connectors for AWS services. To learn more about using OpenSearch Service ML connectors with third-party platforms, see Amazon OpenSearch Service ML connectors for third-party platforms.

Creating a connector through a flow-framework service

To create a flow-framework template with connector, you will need to send a POST request to your OpenSearch Service domain endpoint. You can use cURL, a sample Python client, Postman, or another method to send a signed request. The POST request takes the following format:

```
POST /_plugins/_flow_framework/workflow
{
  "name": "Deploy Claude Model",
  "description": "Deploy a model using a connector to Claude",
  "use_case": "PROVISION",
  "version": {
    "template": "1.0.0",
    "compatibility": [
      "2.12.0",
      "3.0.0"
    ]
  },
  "workflows": {
    "provision": {
      "nodes": [
        {
          "id": "create_claude_connector",
```

```
          "type": "create_connector",
          "user_inputs": {
            "name": "Claude Instant Runtime Connector",
            "version": "1",
            "protocol": "aws_sigv4",
            "description": "The connector to BedRock service for Claude model",
            "actions": [
              {
                "headers": {
                  "x-amz-content-sha256": "required",
                  "content-type": "application/json"
                },
                "method": "POST",
                "request_body": "{ \"prompt\":\"${parameters.prompt}\",
 \"max_tokens_to_sample\":${parameters.max_tokens_to_sample},
 \"temperature\":${parameters.temperature},  \"anthropic_version\":
\"${parameters.anthropic_version}\" }",
                "action_type": "predict",
                "url": "https://bedrock-runtime.us-west-2.amazonaws.com/model/
anthropic.claude-instant-v1/invoke"
              }
            ],
            "credential": {
                "roleArn": "arn:aws:iam::account-id:role/opensearch-secretmanager-
role"
             },
            "parameters": {
              "endpoint": "bedrock-runtime.us-west-2.amazonaws.com",
              "content_type": "application/json",
              "auth": "Sig_V4",
              "max_tokens_to_sample": "8000",
              "service_name": "bedrock",
              "temperature": "0.0001",
              "response_filter": "$.completion",
              "region": "us-west-2",
              "anthropic_version": "bedrock-2023-05-31"
            }
          }
        }
      ]
    }
  }
}
```

If your domain resides within a virtual private cloud (Amazon VPC), you must be connected to the Amazon VPC for the request to successfully create the AI connector. Accessing an Amazon VPC varies by network configuration, but usually involves connecting to a VPN or corporate network. To check that you can reach your OpenSearch Service domain, navigate to `https://your-vpc-domain.region.es.amazonaws.com` in a web browser and verify that you receive the default JSON response.

Sample Python client

The Python client is simpler to automate than a HTTP request and has better reusability. To create the AI connector with the Python client, save the following sample code to a Python file. The client requires the AWS SDK for Python (Boto3), Requests:HTTP for Humans, and requests-aws4auth 1.2.3 packages.

```
import boto3
import requests
from requests_aws4auth import AWS4Auth

host = 'domain-endpoint/'
region = 'region'
service = 'es'
credentials = boto3.Session().get_credentials()
awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region, service,
 session_token=credentials.token)

path = '_plugins/_flow_framework/workflow'
url = host + path

payload = {
  "name": "Deploy Claude Model",
  "description": "Deploy a model using a connector to Claude",
  "use_case": "PROVISION",
  "version": {
    "template": "1.0.0",
    "compatibility": [
      "2.12.0",
      "3.0.0"
    ]
  },
  "workflows": {
    "provision": {
      "nodes": [
```

```
        {
          "id": "create_claude_connector",
          "type": "create_connector",
          "user_inputs": {
            "name": "Claude Instant Runtime Connector",
            "version": "1",
            "protocol": "aws_sigv4",
            "description": "The connector to BedRock service for Claude model",
            "actions": [
              {
                "headers": {
                  "x-amz-content-sha256": "required",
                  "content-type": "application/json"
                },
                "method": "POST",
                "request_body": "{ \"prompt\":\"${parameters.prompt}\",
 \"max_tokens_to_sample\":${parameters.max_tokens_to_sample},
 \"temperature\":${parameters.temperature},  \"anthropic_version\":
\"${parameters.anthropic_version}\" }",
                "action_type": "predict",
                "url": "https://bedrock-runtime.us-west-2.amazonaws.com/model/
anthropic.claude-instant-v1/invoke"
              }
            ],
            "credential": {
                "roleArn": "arn:aws:iam::account-id:role/opensearch-secretmanager-
role"
             },
            "parameters": {
              "endpoint": "bedrock-runtime.us-west-2.amazonaws.com",
              "content_type": "application/json",
              "auth": "Sig_V4",
              "max_tokens_to_sample": "8000",
              "service_name": "bedrock",
              "temperature": "0.0001",
              "response_filter": "$.completion",
              "region": "us-west-2",
              "anthropic_version": "bedrock-2023-05-31"
            }
          }
        }
      ]
    }
  }
```

```
}

headers = {"Content-Type": "application/json"}

r = requests.post(url, auth=awsauth, json=payload, headers=headers)
print(r.status_code)
print(r.text)
```

Pre-defined workflow templates

Amazon OpenSearch Service provides several workflow templates for some common machine learning (ML) use cases. Using a template simplifies complex setups and provides many default values for use cases like semantic or conversational search. You can specify a workflow template when you call the Create Workflow API.

- To use an OpenSearch Service provided workflow template, specify the template use case as the `use_case` query parameter.
- To use a custom workflow template, provide the complete template in the request body. For an, example of a custom template, see an example JSON template or an example YAML template.

Template Use Cases

This table provides an overview of the different templates available, a description of the templates, and the required parameters.

Template use case	Description	Required Parameters
`bedrock_titan_embedding_model_deploy`	Creates and deploys an Amazon Bedrock embedding model (by default, `titan-embed-text-v1`	`create_connector.credential.roleArn`
`bedrock_titan_embedding_model_deploy`	Creates and deploys an Amazon Bedrock multimodal embedding model (by default, `titan-embed-text-v1`	`create_connector.credential.roleArn`

Template use case	Description	Required Parameters
`cohere_embedding_model_deploy`	Creates and deploys a Cohere embedding model (by default, embed-english-v3.0).	`create_connector.credential.roleArn`, `create_connector.credential.secretArn`
`cohere_chat_model_deploy`	Creates and deploys a Cohere chat model (by default, Cohere Command).	`create_connector.credential.roleArn`, `create_connector.credential.secretArn`
`open_ai_embedding_model_deploy`	Creates and deploys an OpenAI embedding model (by default, text-embedding-ada-002).	`create_connector.credential.roleArn`, `create_connector.credential.secretArn`
`openai_chat_model_deploy`	Creates and deploys an OpenAI chat model (by default, gpt-3.5-turbo).	`create_connector.credential.roleArn`, `create_connector.credential.secretArn`
`semantic_search_with_cohere_embedding`	Configures semantic search and deploys a Cohere embedding model. You must provide the API key for the Cohere model.	`create_connector.credential.roleArn`, `create_connector.credential.secretArn`
`semantic_search_with_cohere_embedding_query_enricher`	Configures semantic search and deploys a Cohere embedding model. Adds a query_enricher search processor that sets a default model ID for neural queries. You must provide the API key for the Cohere model.	`create_connector.credential.roleArn`, `create_connector.credential.secretArn`

Template use case	Description	Required Parameters
`multimodal_search_with_bedrock_titan`	Deploys an Amazon Bedrock multimodal model and configures an ingestion pipeline with a text_image_embedding processor and a k-NN index for multimodal search. You must provide your AWS credentials.	`create_connector.credential.roleArn`

Note

For all templates that require a secret ARN, the default is to store the secret with a key name of "key" in AWS Secrets mangager.

Default templates with pretrained models

Amazon OpenSearch Service offers two additonal default workflow templates not available in the opensource OpenSearch Service.

Template use case	Description
`semantic_search_with_local_model`	Configures semantic search and deploys a pretrained model (`msmarco-distilbert-base-tas-b`). Adds a `neural_query_enricher` search processor that sets a default model ID for neural queries and creates a linked k-NN index called 'my-nlp-index'.
`hybrid_search_with_local_model`	Configures hybrid search and deploys a pretrained model (`msmarco-distilbert-base-tas-b`). Adds a `neural_query_enricher` search processor that sets a default model ID for neural queries and

Template use case	Description
	creates a linked k-NN index called 'my-nlp-index'.

Configure permissions

If you create a new domain with version 2.13 or later, permissions are already in place. If you enable flow framework on a preexisting OpenSearch Service domain with version 2.11 or earlier that you then upgrade to version 2.13 or later, you must define the `flow_framework_manager` role. Non-admin users must be mapped to this role in order to manage warm indexes on domains using fine-grained access control. To manually create the `flow_framework_manager` role, perform the following steps:

1. In OpenSearch Dashboards, go to **Security** and choose **Permissions**.
2. Choose **Create action group** and configure the following groups:

Group name	Permissions
`flow_framework_full_access`	• `cluster:admin/opensearch/flow_framework/*` • `cluster_monitor`
`flow_framework_read_accesss`	• `cluster:admin/opensearch/flow_framework/workflow/get` • `cluster:admin/opensearch/flow_framework/workflow/search` • `cluster:admin/opensearch/flow_framework/workflow_state/get` • `cluster:admin/opensearch/flow_framework/workflow_state/search`

3. Choose **Roles** and **Create role**.
4. Name the role **flow_framework_manager**.
5. For **Cluster permissions,** select `flow_framework_full_access` and `flow_framework_read_access`.

6. For **Index**, type `*`.
7. For **Index permissions**, select `indices:admin/aliases/get`, `indices:admin/mappings/get`, and `indices_monitor`.
8. Choose **Create**.
9. After you create the role, map it to any user or backend role that will manage flow framework indexes.

Security Analytics for Amazon OpenSearch Service

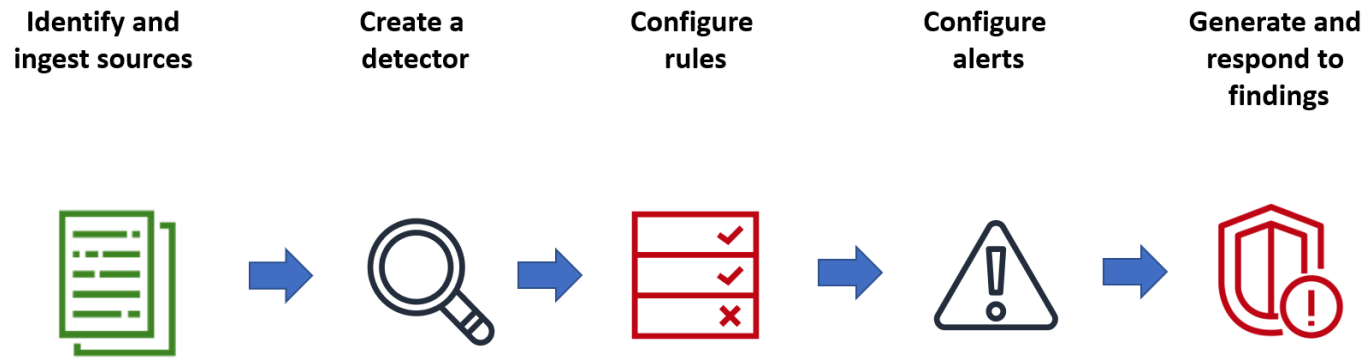
Security Analytics is an OpenSearch solution that provides visibility into your organization's infrastructure, monitors for anomalous activity, detects potential security threats in real time, and trigger alerts to pre-configured destinations. You can monitor for malicious activity from your security event logs by continuously evaluating security rules and reviewing auto-generated security findings. In addition, Security Analytics can generate automated alerts and send them to a specified notification channel, such as Slack or email.

You can use the Security Analytics plugin to detect common threats out-of-the-box and generate critical security insights from your existing security event logs, such as firewall logs, windows logs, and authentication audit logs. To use Security Analytics, your domain must be running OpenSearch version 2.5 or later.

> **Note**
>
> This documentation provides a brief overview of Security Analytics for Amazon OpenSearch Service. It defines key concepts and provides steps to configure permissions. For comprehensive documentation, including a setup guide, an API reference, and a reference of all available settings, see Security Analytics in the OpenSearch documentation..

Security analytics components and concepts

A number of tools and features provide the foundation to the operation of Security Analytics. The major components that compose the plugin include detectors, log types, rules, findings, and alerts.



Log types

OpenSearch supports several types of logs and provides out-of-the-box mappings for each type. You specify the log type and configure a time interval when you create a detector, and from there Security Analytics automatically activates a relevant set of rules that run at that interval.

Detectors

Detectors identify a range of cybersecurity threats for a log type across your data indexes. You configure your detector to use both custom rules and pre-packaged Sigma rules that evaluate events occurring in the system. The detector then generates security findings from these events. For more information about detectors, see Creating detectors in the OpenSearch documentation.

Rules

Threat detection rules define the conditions that detectors apply to ingested log data to identify a security event. Security Analytics supports importing, creating, and customizing rules to meet your requirements, and also provides prepackaged, open-source Sigma rules to detect common threats from your logs. Security Analytics maps many rules to an ever-growing knowledge base of adversary tactics and techniques maintained by the MITRE ATT&CK organization. You can use both OpenSearch Dashboards or the APIs to create and use rules. For more information about rules, see Working with rules in the OpenSearch documentation.

Findings

When a detector matches a rule with a log event, it generates a finding. Each finding includes a unique combination of select rules, a log type, and a rule severity. Findings don't necessarily point to imminent threats within the system, but they always isolate an event of interest. For more information about findings, see Working with findings in the OpenSearch documentation.

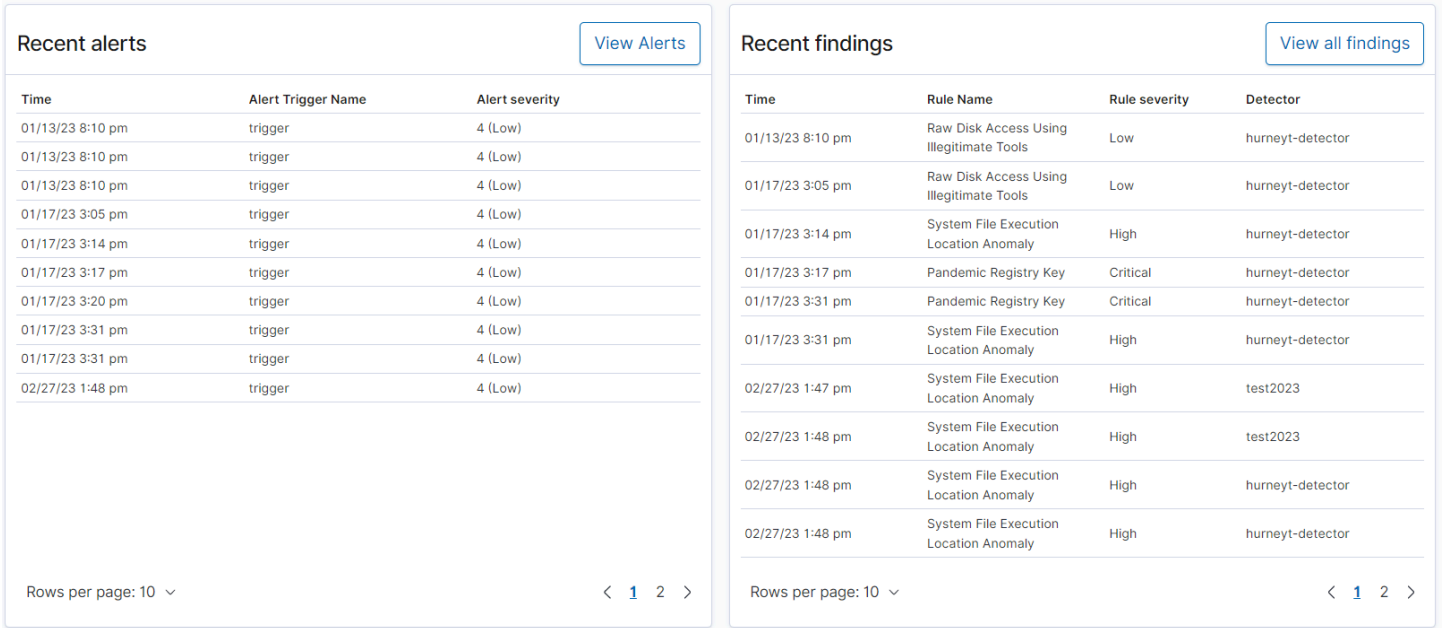
Alerts

When you create a detector, you can specify one or more conditions that trigger an alert. An alert is a notification sent to a preferred channel, such as Slack or email. You set the alert to be triggered when the detector matches one or multiple rules, and can customize the notification message. For more information about alerts, see Working with alerts in the OpenSearch documentation.

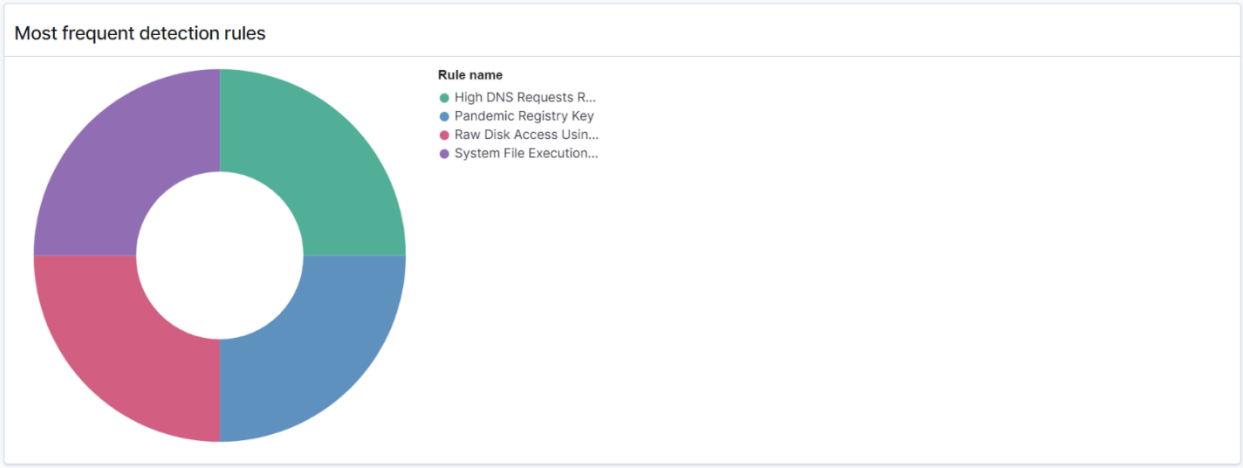
Exploring Security Analytics

You can use OpenSearch Dashboards to visualize and gain insight into your Security Analytics plugin. The **Overview** view provides information such as findings and alert counts, recent findings and alerts, frequent detection rules, and a list of your detectors. You can see a summary view comprised of multiple visualizations. The following chart, for example, shows the findings and alerts trend for various log types over a given period of time.

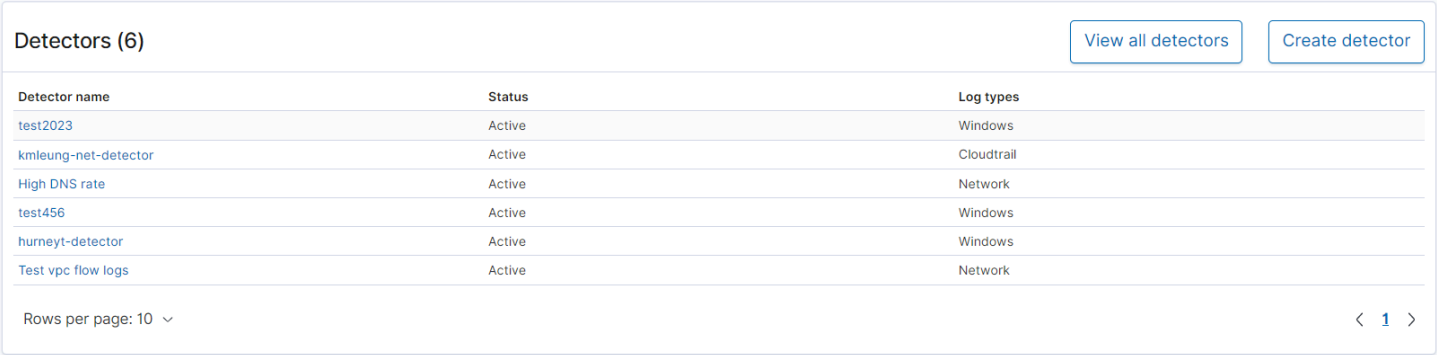
Further down the page, you can review your most recent findings and alerts.



Additionally, you can see a distribution of the most frequently triggered rules across all the active detectors. This can help you detect and investigate different types of malicious activities across log types.



Finally, you can view the status of configured detectors. From this panel, you can also navigate to the create detector workflow.



To configure your Security Analytics setup, create rules with the **Rules** page and use those rules to write detectors in the **Detectors** page. For a more focused view of your Security Analytics results, you can use the **Findings** and **Alerts** pages.

Configure permissions

If you enable Security Analytics on a preexisting OpenSearch Service domain, the `security_analytics_manager` role might not be defined on the domain. Non-admin users must be mapped to this role in order to manage warm indexes on domains using fine-grained access control. To manually create the `security_analytics_manager` role, perform the following steps:

1. In OpenSearch Dashboards, go to **Security** and choose **Permissions**.

2. Choose **Create action group** and configure the following groups:

Group name	Permissions
security_analytics_full_access	• cluster:admin/opensearch/securityanalytics/alerts/* • cluster:admin/opensearch/securityanalytics/detector/* • cluster:admin/opensearch/securityanalytics/findings/* • cluster:admin/opensearch/securityanalytics/mapping/* • cluster:admin/opensearch/securityanalytics/rule/*
security_analytics_read_access	• cluster:admin/opensearch/securityanalytics/alerts/get • cluster:admin/opensearch/securityanalytics/detector/get • cluster:admin/opensearch/securityanalytics/detector/search • cluster:admin/opensearch/securityanalytics/findings/get • cluster:admin/opensearch/securityanalytics/mapping/get • cluster:admin/opensearch/securityanalytics/mapping/view/get • cluster:admin/opensearch/securityanalytics/rule/get • cluster:admin/opensearch/securityanalytics/rule/search

3. Choose **Roles** and **Create role**.

4. Name the role **security_analytics_manager**.

5. For **Cluster permissions,** select `security_analytics_full_access` and `security_analytics_read_access`.
6. For **Index**, type `*`.
7. For **Index permissions**, select `indices:admin/mapping/put` and `indices:admin/mappings/get`.
8. Choose **Create**.
9. After you create the role, map it to any user or backend role that will manage Security Analytics indexes.

Troubleshooting

No such index error

If you have no detectors and you open the Security Analytics dashboard, you might see a notification on the bottom right that says `[index_not_found_exception] no such index [.opensearch-sap-detectors-config]`. You can disregard this notification, which disappears within a few seconds and won't appear again once you create a detector.

Observability in Amazon OpenSearch Service

The default installation of OpenSearch Dashboards for Amazon OpenSearch Service includes the Observability plugin, which you can use to visualize data-driven events using Piped Processing Language (PPL) in order to explore, discover, and query data stored in OpenSearch. The plugin requires OpenSearch 1.2 or later.

The Observability plugin provides a unified experience for collecting and monitoring metrics, logs, and traces from common data sources. Data collection and monitoring in one place enables full-stack, end-to-end observability of your entire infrastructure.

> **Note**
>
> This documentation provides a brief overview of Observability in OpenSearch Service. For comprehensive documentation of the Observability plugin, including permissions, see Observability.

Everyone's process for exploring data is different. If you're new to exploring data and creating visualizations, we recommend trying a workflow like the following.

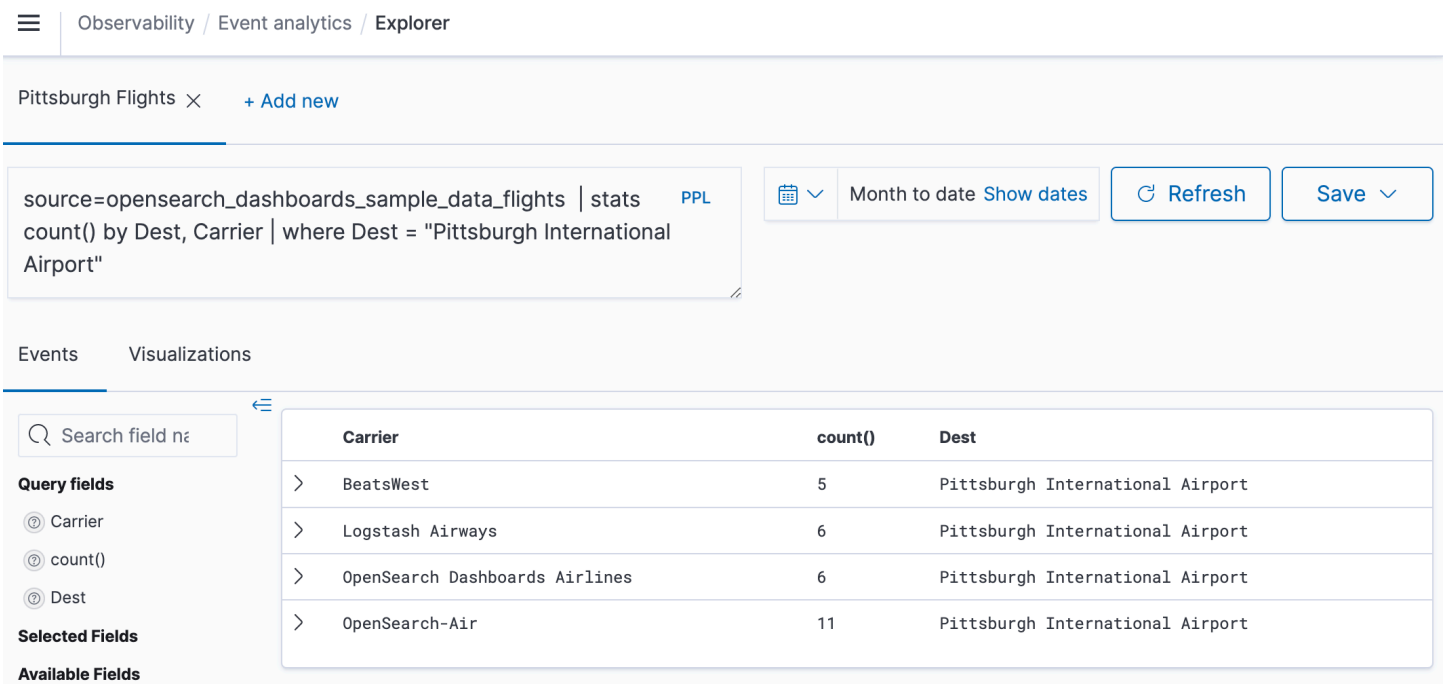
Explore your data with event analytics

To start, let's say that you're collecting flight data in your OpenSearch Service domain and you want to find out which airline had the most flights arriving in Pittsburgh International Airport last month. You write the following PPL query:

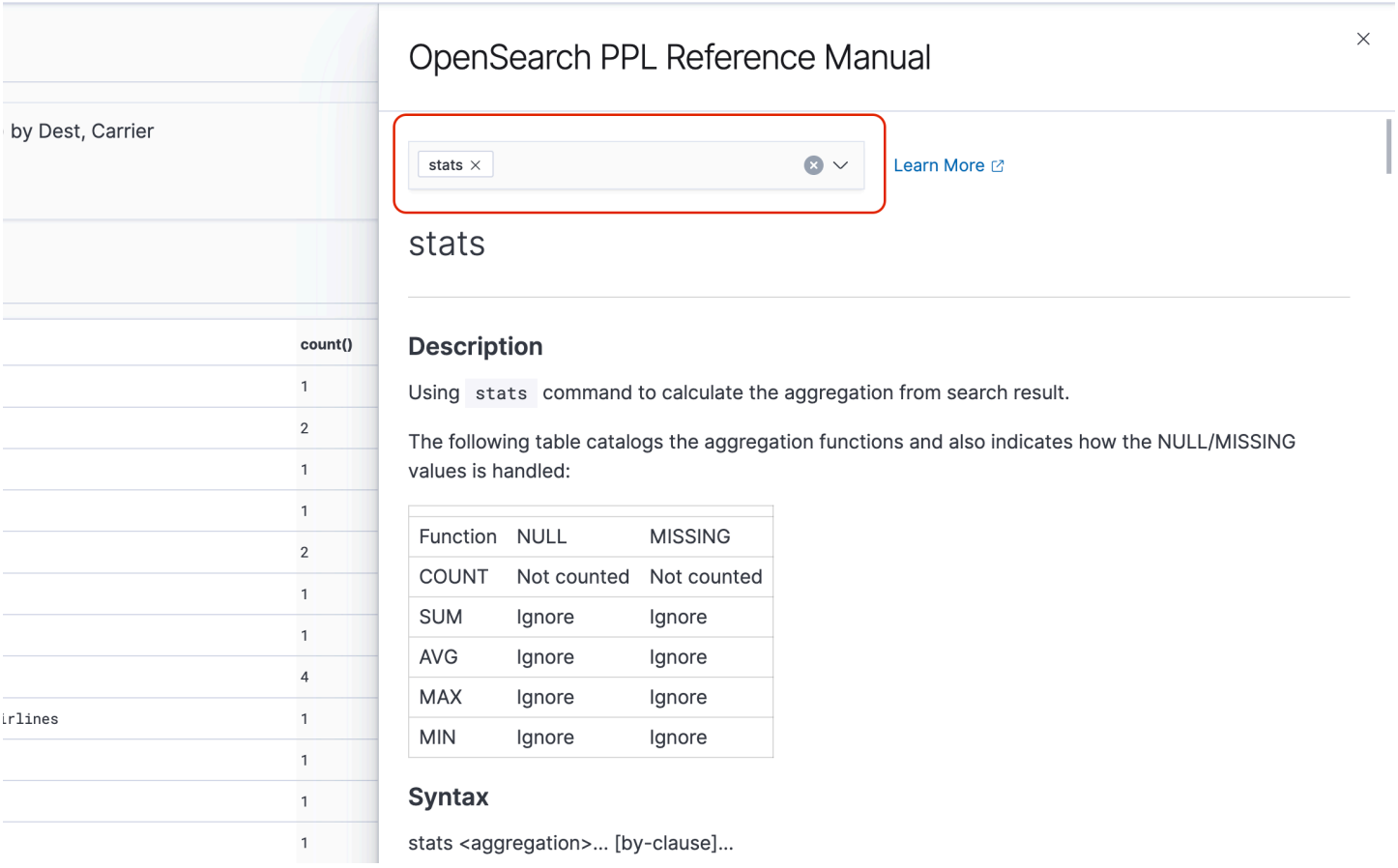
```
source=opensearch_dashboards_sample_data_flights |
    stats count() by Dest, Carrier |
    where Dest = "Pittsburgh International Airport"
```

This query pulls data from the index named `opensearch_dashboards_sample_data_flights`. It then uses the `stats` command to get a total count of flights and groups it according to destination airport and carrier. Finally, it uses the `where` clause to filter the results to flights arriving in Pittsburgh International Airport.

Here's what the data looks like when displayed over the last month:



You can choose the **PPL** button in the query editor to get usage information and examples for each PPL command:



Let's look at a more complex example, which queries for information about flight delays:

```
source=opensearch_dashboards_sample_data_flights |
    where FlightDelayMin > 0 |
    stats sum(FlightDelayMin) as minimum_delay, count() as total_delayed by Carrier,
 Dest |
    eval avg_delay=minimum_delay / total_delayed |
    sort - avg_delay
```

Each command in the query impacts the final output:

- `source=opensearch_dashboards_sample_data_flights` - pulls data from the same index as the previous example
- `where FlightDelayMin > 0` - filters the data to flights that were delayed
- `stats sum(FlightDelayMin) as minimum_delay, count() as total_delayed by Carrier` - for each carrier, gets the total minimum delay time and the total count of delayed flights
- `eval avg_delay=minimum_delay / total_delayed` - calculates the average delay time for each carrier by dividing the minimum delay time by the total number of delayed flights
- `sort - avg_delay` - sorts the results by average delay in descending order

With this query, you can determine that OpenSearch Dashboards Airlines has, on average, fewer delays.

	avg_delay	Carrier	minimum_delay	total_delayed
>	212	Logstash Airways	4470	21
>	184	OpenSearch-Air	4245	23
>	155	BeatsWest	2025	13
>	153	OpenSearch Dashboards Airlines	4305	28

You can find more sample PPL queries under **Queries and Visualizations** on the **Event analytics** page.

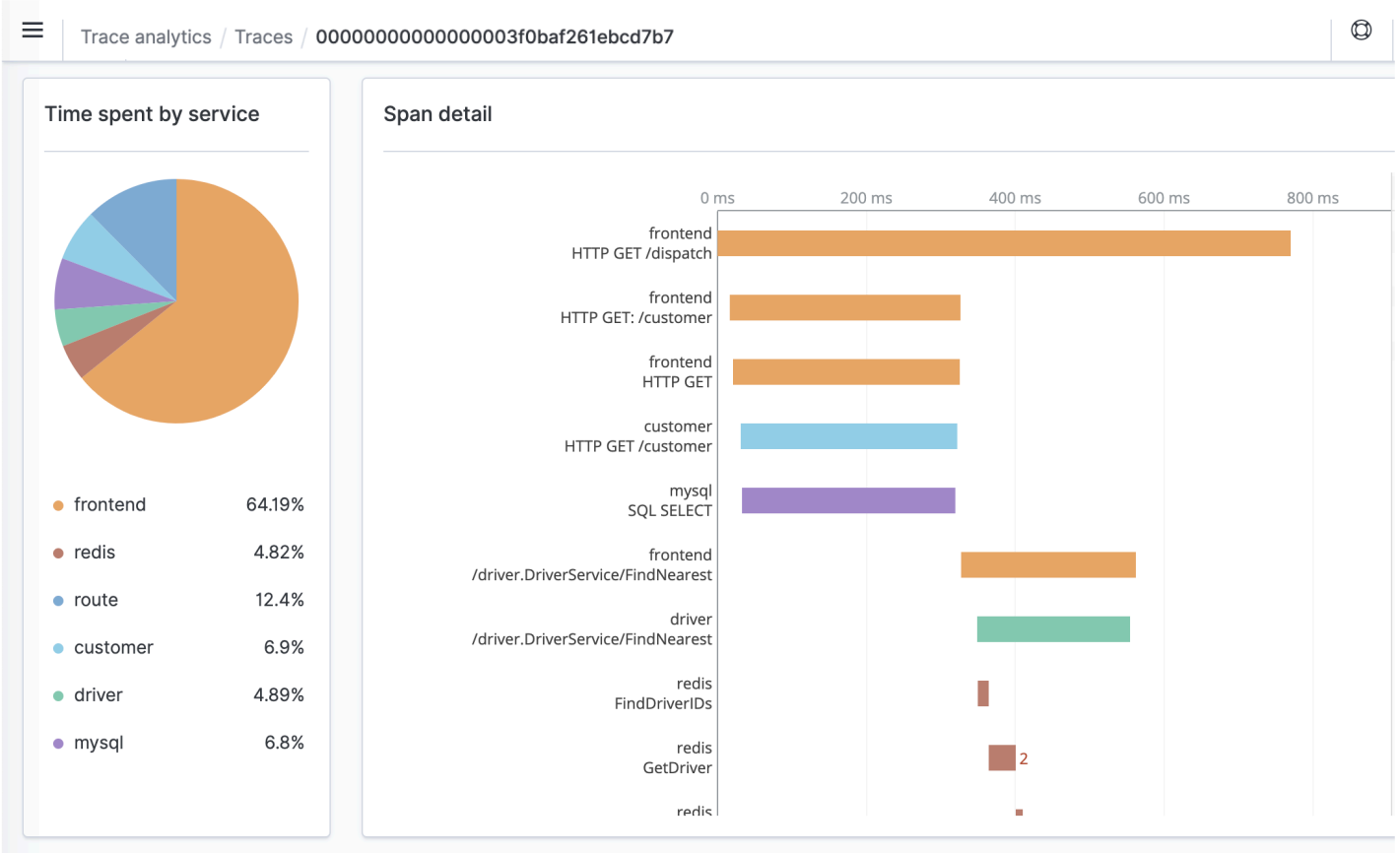
Create visualizations

Once you correctly query the data that you're interested in, you can save those queries as visualizations:

Then add those visualizations to operational panels to compare different pieces of data. Leverage notebooks to combine different visualizations and code blocks that you can share with team members.

Dive deeper with Trace Analytics

Trace Analytics provides a way to visualize the flow of events in your OpenSearch data to identify and fix performance problems in distributed applications.



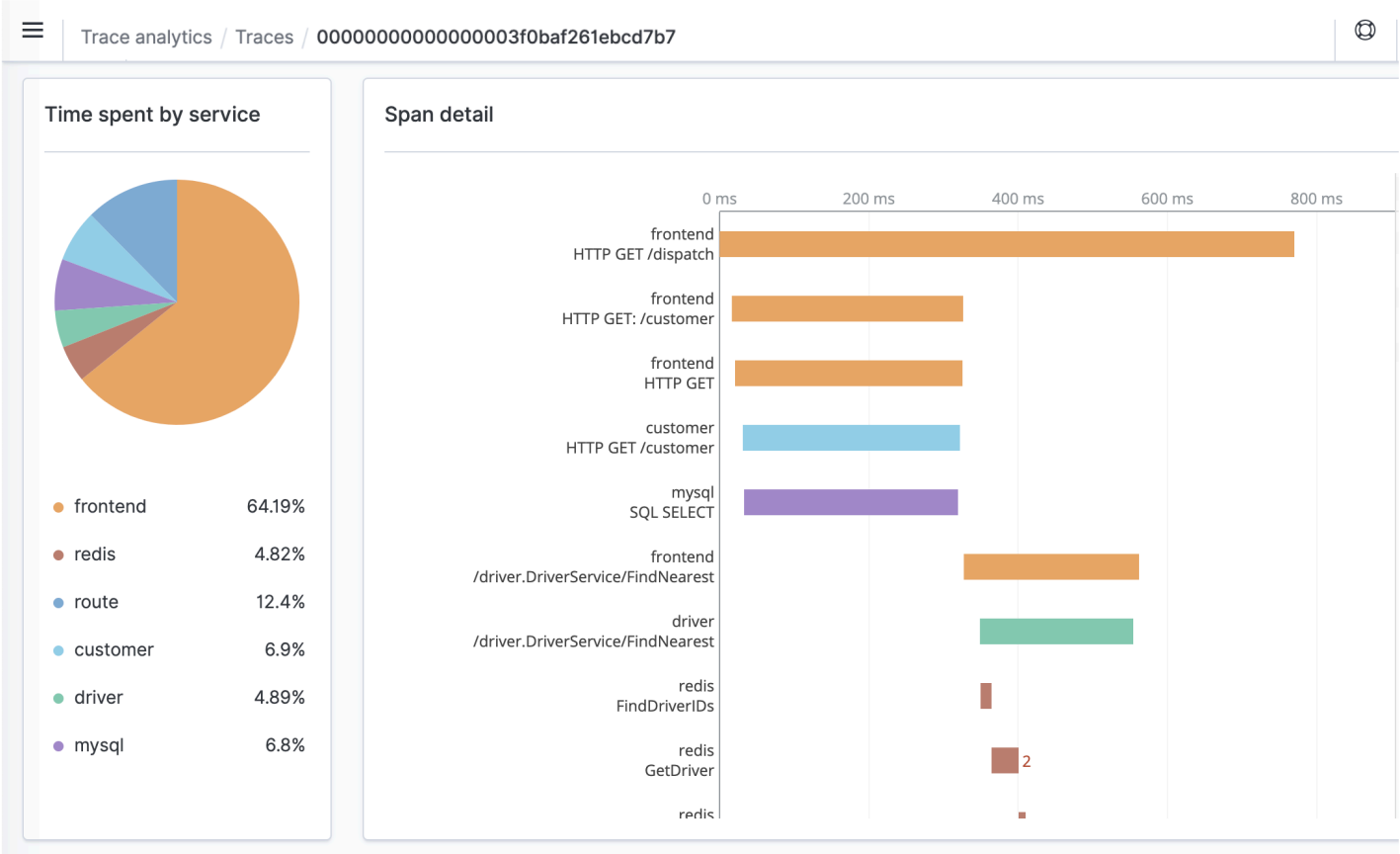
Trace Analytics for Amazon OpenSearch Service

You can use Trace Analytics, which is part of the OpenSearch Observability plugin, to analyze trace data from distributed applications. Trace Analytics requires OpenSearch or Elasticsearch 7.9 or later.

In a distributed application, a single operation, such as a user clicking a button, can trigger an extended series of events. For example, the application front end might call a backend service, which calls another service, which queries a database, which processes the query and returns a result. Then the first backend service sends a confirmation to the front end, which updates the UI.

You can use Trace Analytics to help you visualize this flow of events and identify performance problems.

> **Note**
>
> This documentation provides a brief overview of Trace Analytics. For comprehensive documentation, see Trace Analytics in the open source OpenSearch documentation.



Prerequisites

Trace Analytics requires you to add instrumentation to your application and generate trace data using an OpenTelemetry-supported library such as Jaeger or Zipkin. This step occurs entirely outside of OpenSearch Service. The AWS Distro for OpenTelemetry documentation contains example applications for many programming languages that can help you get started, including Java, Python, Go, and JavaScript.

After you add instrumentation to your application, the OpenTelemetry Collector receives data from the application and formats it into OpenTelemetry data. See the list of receivers on GitHub. AWS Distro for OpenTelemetry includes a receiver for AWS X-Ray.

Finally, you can use *Amazon OpenSearch Ingestion* to format that OpenTelemetry data for use with OpenSearch.

OpenTelemetry Collector sample configuration

To use the OpenTelemetry Collector with *Amazon OpenSearch Ingestion*, try the following sample configuration:

```
extensions:
  sigv4auth:
    region: "us-east-1"
    service: "osis"

receivers:
  jaeger:
    protocols:
      grpc:

exporters:
  otlphttp:
    traces_endpoint: "https://pipeline-endpoint.us-east-1.osis.amazonaws.com/
opentelemetry.proto.collector.trace.v1.TraceService/Export"
    auth:
      authenticator: sigv4auth
    compression: none

service:
  extensions: [sigv4auth]
  pipelines:
    traces:
      receivers: [jaeger]
      exporters: [otlphttp]
```

OpenSearch Ingestion sample configuration

To send trace data to an OpenSearch Service domain, try the following sample OpenSearch Ingestion pipeline configuration. For instructions to create a pipeline, see the section called "Creating pipelines".

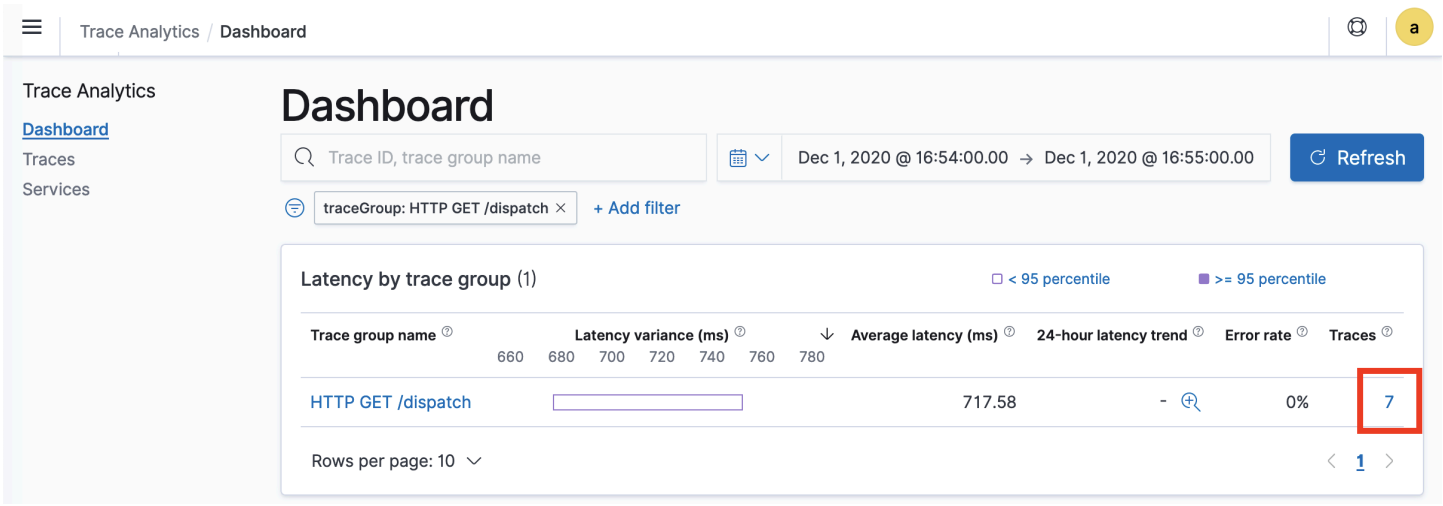
```
version: "2"
otel-trace-pipeline:
  source:
    otel_trace_source:
      "/${pipelineName}/ingest"
  processor:
    - trace_peer_forwarder:
  sink:
    - pipeline:
        name: "trace_pipeline"
    - pipeline:
```

```
        name: "service_map_pipeline"
trace-pipeline:
  source:
    pipeline:
      name: "otel-trace-pipeline"
  processor:
    - otel_traces:
  sink:
    - opensearch:
        hosts: ["https://domain-endpoint"]
        index_type: trace-analytics-raw
        aws:
          # IAM role that OpenSearch Ingestion assumes to access the domain sink
          sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
          region: "us-east-1"

service-map-pipeline:
  source:
    pipeline:
      name: "otel-trace-pipeline"
  processor:
    - service_map:
  sink:
    - opensearch:
        hosts: ["https://domain-endpoint"]
        index_type: trace-analytics-service-map
        aws:
          # IAM role that the pipeline assumes to access the domain sink
          sts_role_arn: "arn:aws:iam::{account-id}:role/pipeline-role"
          region: "us-east-1"
```

The pipeline role that you specify in the `sts_role_arn` option must have write permissions to the sink. For instructions to configure permissions for the pipeline role, see the section called “Setting up roles and users”.

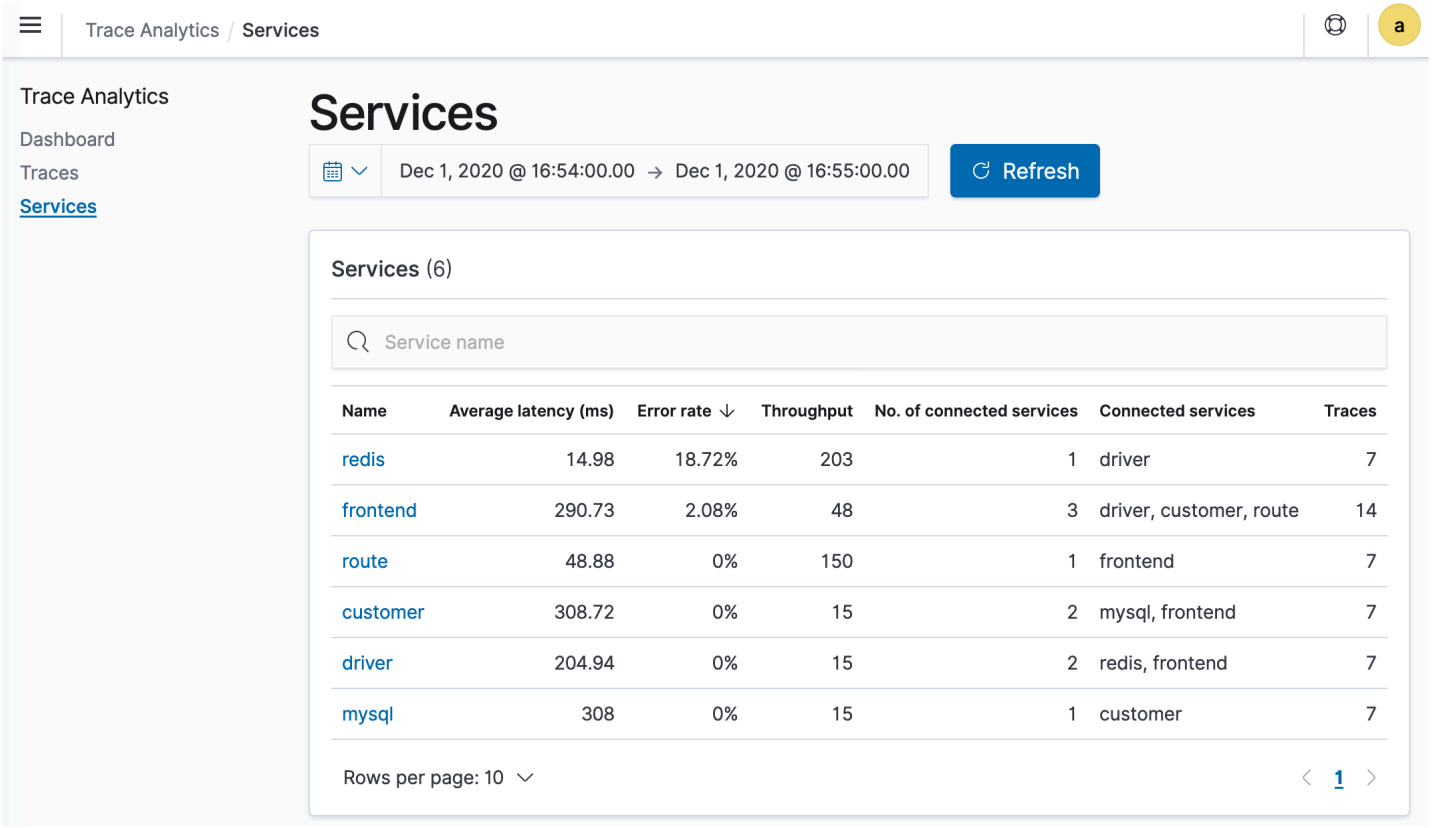
Exploring trace data

The **Dashboard** view groups traces together by HTTP method and path so that you can see the average latency, error rate, and trends associated with a particular operation. For a more focused view, try filtering by trace group name.



To drill down on the traces that make up a trace group, choose the number of traces in the right-hand column. Then choose an individual trace for a detailed summary.

The **Services** view lists all services in the application, plus an interactive map that shows how the various services connect to each other. In contrast to the dashboard (which helps identify problems by operation), the service map helps you identify problems by service. Try sorting by error rate or latency to get a sense of potential problem areas of your application.



Querying Amazon OpenSearch Service data using Piped Processing Language

Piped Processing Language (PPL) is a query language that lets you use pipe (|) syntax to query data stored in Amazon OpenSearch Service. PPL requires either OpenSearch or Elasticsearch 7.9 or later.

> **Note**
>
> This documentation provides a brief overview of PPL for Amazon OpenSearch Service. For detailed steps and a complete command reference, see PPL in the open source OpenSearch documentation.

The PPL syntax consists of commands delimited by a pipe character (|) where data flows from left to right through each pipeline. For example, the PPL syntax to find the number of hosts with HTTP 403 or 503 errors, aggregate them per host, and sort them in the order of impact is as follows:

```
source = dashboards_sample_data_logs | where response='403' or response='503' | stats
 count(request) as request_count by host, response | sort -request_count
```

To get started, choose **Query Workbench** in OpenSearch Dashboards and select **PPL**. Use the `bulk` operation to index some sample data:

```
PUT accounts/_bulk?refresh
{"index":{"_id":"1"}}
{"account_number":1,"balance":39225,"firstname":"Amber","lastname":"Duke","age":32,"gender":"M"
 Holmes
 Lane","employer":"Pyrami","email":"amberduke@pyrami.com","city":"Brogan","state":"IL"}
{"index":{"_id":"6"}}
{"account_number":6,"balance":5686,"firstname":"Hattie","lastname":"Bond","age":36,"gender":"M"
 Bristol
 Street","employer":"Netagy","email":"hattiebond@netagy.com","city":"Dante","state":"TN"}
{"index":{"_id":"13"}}
{"account_number":13,"balance":32838,"firstname":"Nanette","lastname":"Bates","age":28,"gender"
 Mady Street","employer":"Quility","city":"Nogal","state":"VA"}
{"index":{"_id":"18"}}
{"account_number":18,"balance":4180,"firstname":"Dale","lastname":"Adams","age":33,"gender":"M"
 Hutchinson Court","email":"daleadams@boink.com","city":"Orick","state":"MD"}
```

The following example returns `firstname` and `lastname` fields for documents in an accounts index with age greater than 18:

```
search source=accounts | where age > 18 | fields firstname, lastname
```

Sample Response

id	firstname	lastname
0	Amber	Duke
1	Hattie	Bond
2	Nanette	Bates
3	Dale	Adams

You can use a complete set of read-only commands like `search`, `where`, `fields`, `rename`, dedup, `stats`, `sort`, `eval`, `head`, `top`, and `rare`. The PPL plugin supports all SQL functions, including mathematical, trigonometric, date-time, string, aggregate, and advanced operators and expressions. To learn more, see the OpenSearch PPL reference manual.

Operational best practices for Amazon OpenSearch Service

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

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 and configure recommended CloudWatch alarms 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 to track user activity. For more information, see 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 for all production domains. You must also configure logging thresholds—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 for Elasticsearch and OpenSearch 1.1 to 2.15 and 4,000 for OpenSearch 2.17 and above. The 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:

- the section called "Sizing domains"
- the section called "Petabyte scale"

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:

- the section called “Node shard and storage skew”
- the section called “Index shard and storage 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 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 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 manager nodes, upgrades complete without downtime. Otherwise, the cluster might be unresponsive for several seconds post-upgrade while it elects a manager node. OpenSearch Dashboards might be unavailable during some or all of the upgrade.

There are two ways to upgrade a domain:

- In-place upgrade – This option is easier because you keep the same cluster.

- Snapshot/restore upgrade – 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.

Improve snapshot performance

To prevent your snapshot from getting stuck in processing, the instance type for the dedicated manager node should match the shard count. For more information, see the section called "Choosing instance types for dedicated manager nodes". Additionally, each node should have no more than the recommended 25 shards per GiB of Java heap memory. For more information, see the section called "Choosing the number of shards".

Enable dedicated manager nodes

Dedicated manager nodes improve cluster stability. A dedicated manager 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 to happen without downtime.

Enable and use three dedicated manager nodes for optimal domain stability across three Availability Zones. Deploying with Multi-AZ with Standby configures three dedicated manager nodes for you. For instance type recommendations, see the section called "Choosing instance types for dedicated manager nodes".

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 in the same AWS Region. Best practice is to deploy using Multi-AZ with Standby, which configures three Availability Zones, with two zones active and one acting as a standby, and with and 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.

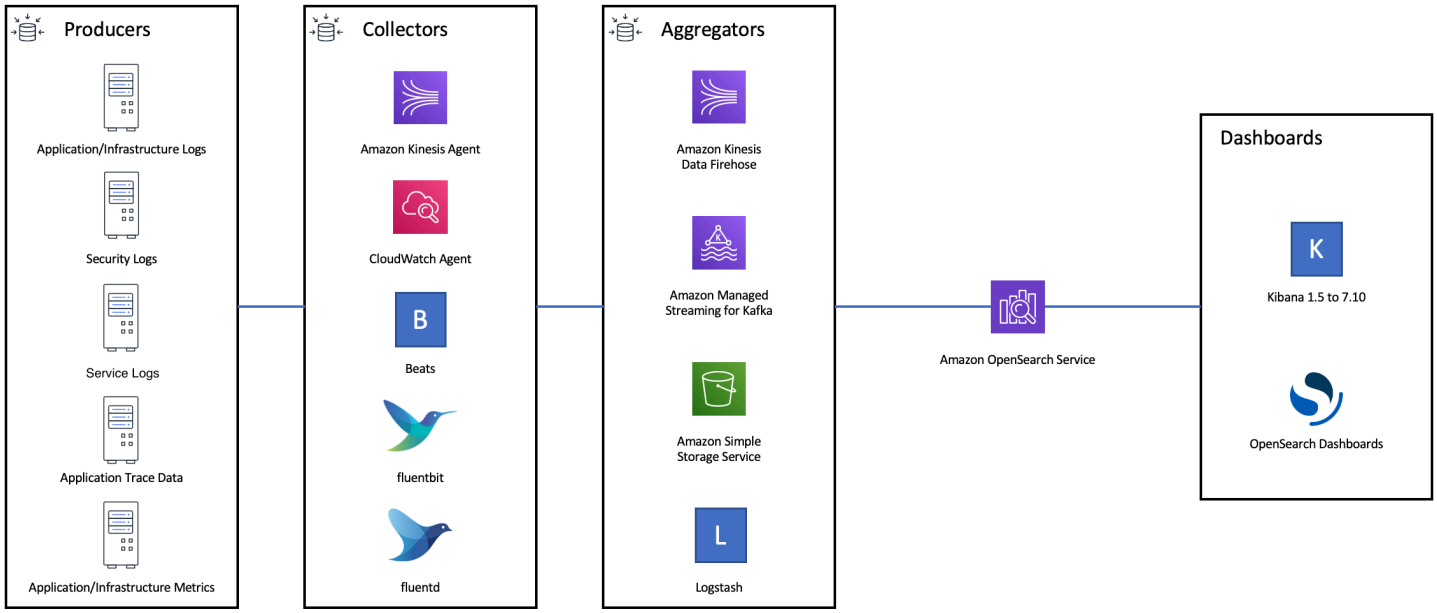
Availability Zones are isolated locations within each Region. With a two-AZ configuration, losing one Availability Zone means that you lose half of all domain capacity. Moving to three Availability Zones further reduces the impact of losing a single Availability Zone.

Control ingest flow and buffering

We recommend that you limit the overall request count using the _bulk 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 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 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 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 operation as an alternative data lifecycle management strategy to avoid shard skew.

First, set up an ISM policy. For example, see the section called “Sample policies”. Then, attach the policy to one or more indexes. If you include an 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 across multiple Regions, consider using two domains. For small or slowly changing datasets, you can set up cross-cluster replication 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 to reduce the payload size of requests and responses. You can use gzip compression with the OpenSearch Python client, or by including the following 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 the section called “Reducing response 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 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 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.

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 of your current access policies.

Apply a restrictive resource-based access policy to your domains and follow the principle of 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.

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.

Monitor with AWS Security Hub

Monitor your usage of OpenSearch Service as it relates to security best practices by using AWS Security Hub. 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 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 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. `r6g.large` instances are an option for small production workloads (both as data nodes and as dedicated manager 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 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 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 to migrate from 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 (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 in Cost Explorer. The **Amazon OpenSearch Service** heading displays specific RI purchase recommendations and projected savings.

Sizing Amazon OpenSearch Service domains

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 the primary shard, the store size of the index shows the size taken by the primary and replica shard. 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.

Other storage considerations exist:

- If your minimum storage requirement exceeds 1 PB, see the section called “Petabyte scale”.
- If you have rolling indexes and want to use a hot-warm architecture, see the section called “UltraWarm storage”.

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 up to 2.15, have a limit of *1,000* shards per node. To adjust the maximum shards per node, configure the `cluster.max_shards_per_node` setting. From OpenSearch 2.17 and above OpenSearch Service supports 1000 shards for every 16GB of data node heap up to a maximum of 4000 shards per node. For an example, see Cluster settings. For more information on shard count, see Shard count quotas.

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 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.

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 manager nodes). If you have three dedicated manager nodes, 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 the section called “Petabyte scale”.

Step 3: Perform representative testing

After configuring the cluster, you can add your indexes using the number of shards you calculated earlier, perform some representative client testing using a realistic dataset, and monitor CloudWatch metrics 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 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 manager nodes, which improve performance and cluster reliability.

Petabyte scale in Amazon OpenSearch Service

Amazon OpenSearch Service domains offer attached storage of up to 10 PB. You can configure a domain with 1000 `OR1.16xlarge.search` instance types, each with 36 TB of storage. Because of the sheer difference in scale, recommendations for domains of this size differ from our general recommendations. This section discusses considerations for creating domains, costs, storage, and shard size.

While this section frequently references the `i3.16xlarge.search` instance types, you can use several other instance types to reach 10 PB of total domain storage.

Creating domains

Domains of this size exceed the default limit of 80 instances per domain. To request a service limit increase of up to 1000 instances per domain, open a case at the AWS Support Center.

Pricing

Before creating a domain of this size, check the Amazon OpenSearch Service pricing page to ensure that the associated costs match your expectations. Examine the section called "UltraWarm storage" to see if a hot-warm architecture fits your use case.

Storage

The `i3` instance types are designed to provide fast, local non-volatile memory express (NVMe) storage. Because this local storage tends to offer performance benefits when compared to Amazon Elastic Block Store, EBS volumes are not an option when you select these instance types in OpenSearch Service. If you prefer EBS storage, use another instance type, such as `r6.12xlarge.search`.

Shard size and count

A common OpenSearch guideline is not to exceed 50 GB per shard. Given the number of shards necessary to accommodate large domains and the resources available to `i3.16xlarge.search` instances, we recommend a shard size of 100 GB.

For example, if you have 450 TB of source data and want one replica, your *minimum* storage requirement is closer to 450 TB * 2 * 1.1 / 0.95 = 1.04 PB. For an explanation of this calculation, see the section called “Calculating storage requirements”. Although 1.04 PB / 15 TB = 70 instances, you might select 90 or more `i3.16xlarge.search` instances to give yourself a storage safety net, deal with node failures, and account for some variance in the amount of data over time. Each instance adds another 20 GiB to your minimum storage requirement, but for disks of this size, those 20 GiB are almost negligible.

Controlling the number of shards is tricky. OpenSearch users often rotate indexes on a daily basis and retain data for a week or two. In this situation, you might find it useful to distinguish between "active" and "inactive" shards. Active shards are, well, actively being written to or read from. Inactive shards might service some read requests, but are largely idle. In general, you should keep the number of active shards below a few thousand. As the number of active shards approaches 10,000, considerable performance and stability risks emerge.

To calculate the number of primary shards, use this formula: 450,000 GB * 1.1 / 100 GB per shard = 4,950 shards. Doubling that number to account for replicas is 9,900 shards, which represents a major concern if all shards are active. But if you rotate indexes and only $1/7^{th}$ or $1/14^{th}$ of the shards are active on any given day (1,414 or 707 shards, respectively), the cluster might work well. As always, the most important step of sizing and configuring your domain is to perform representative client testing using a realistic dataset.

Dedicated coordinator nodes

Amazon OpenSearch Service offers the option to provision a dedicated coordinator nodes. Dedicated coordinator nodes relieve data nodes from the responsibilities of coordinating and hosting OpenSearch Dashboards and provide better resource utilization of data nodes which can improve OpenSearch domain resiliency. Dedicated coordinator nodes help in reducing the reservation of private IP addresses required for virtual private cloud (VPC) domains. Dedicated coordinator nodes enhance resource efficency, delivering up to a 15% increase in indexing througput and a 20% improvement in query performance depending on workload demands. For more information on enhanced resource efficiency, see Improve OpenSearch Service cluster resiliency and performance with dedicated coordinator nodes.

The primary function of a coordinator node role is to forward requests to the data nodes that hold the data and reduce each node's results into a single global result set. In the absence of a dedicated coordinator node, data nodes perform coordination roles alongside their core responsibilities of

search, data storage, and indexing. Data nodes require hosting Kibana and OpenSearch Dashboards and can potentially face resource (memory and CPU) strain. Provisioning a dedicated coordinator node allows a cluster to offload coordination and dashboard hosting responsibilities from data nodes and can improve a domain's resilience.

All OpenSearch versions and Elasticsearch (open source) versions 6.8 to 7.10 support provisioning of a dedicated coordinator node. Dedicated coordinator node provisioning is available for domains with dedicated cluster manager enabled.

OpenSearch VPC domains attach the elastic network interface (ENI) to dedicated coordinator nodes instead of data nodes. Dedicated coordinator nodes typically represent around 10% of total data nodes. As a result, significantly small number of private IP addresses will be reserved for VPC domains.

Best practices

This chapter provides best practices for provisioning a dedicated coordinator node 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.

- General-purpose instances are sufficient for most of the use cases.
- Keep the instance family the same for a dedicated cooordinating node and data node.
- Provision 5% to 10% of your domain's total data nodes as dedicated coordinator nodes. For example, if your domain has 90 R6g.large data nodes, you can plan for coordinator nodes to make up anywhere between 5 and 9 of the R6g.large instance type.
- As a starting point, the instance size of your dedicated coordinator nodes should be the same as your domain's data nodes. However, in certain cases, it may be advisable to use a smaller instance type with a higher count to ensure availability. For example, if you have R6g.8xlarge as the instance size for a data node, you can try 3 instances of M6g.2xlarge size instead of one M6g.8xlarge size to achieve better availability.
- Source domains typically do not perform coordination tasks. If you are using cross region search, provision a dedicated coordinator node on the destination domains.
- To avoid a single point of failure, provision a minimum of two dedicated coordinator nodes.

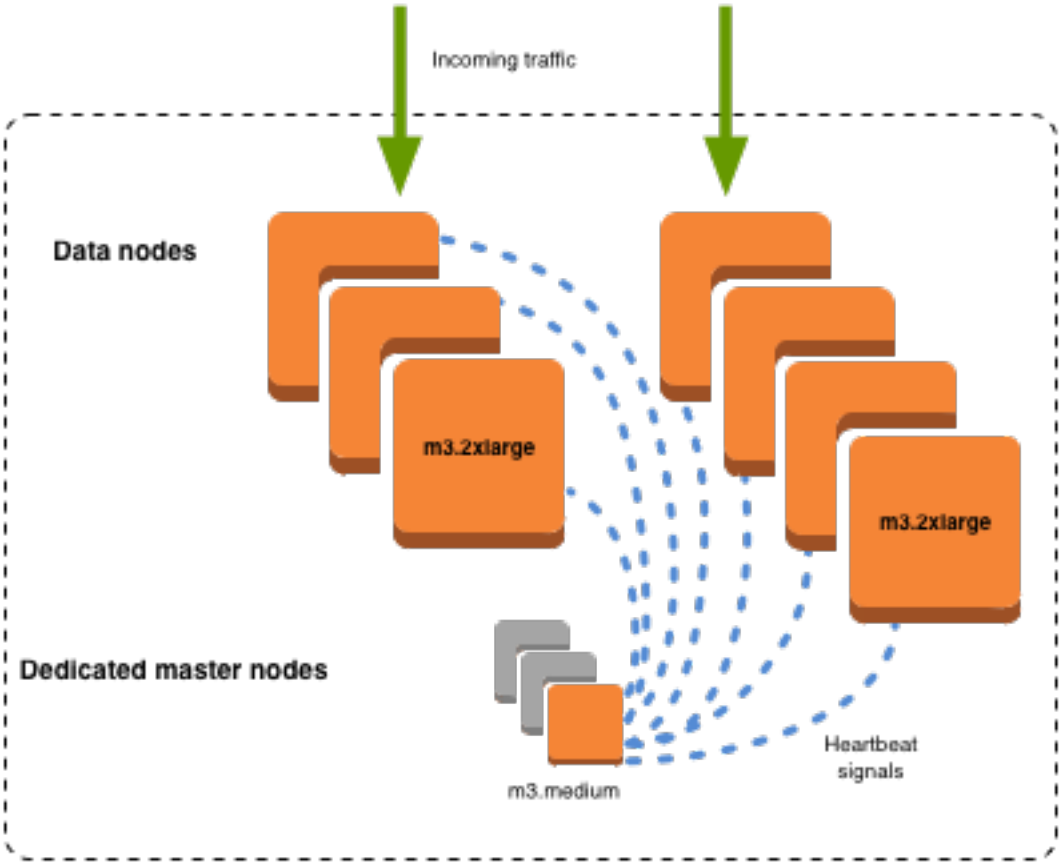
Dedicated manager nodes in Amazon OpenSearch Service

Amazon OpenSearch Service uses *dedicated manager nodes* to increase cluster stability. A dedicated manager node performs cluster management tasks, but does not hold data or respond to data upload requests. This offloading of cluster management tasks increases the stability of your domain. Just like all other node types, you pay an hourly rate for each dedicated manager node.

Dedicated manager nodes perform the following cluster management tasks:

- Track all nodes in the cluster.
- Track the number of indexes in the cluster.
- Track the number of shards belonging to each index.
- Maintain routing information for nodes in the cluster.
- Update the cluster state after state changes, such as creating an index and adding or removing nodes in the cluster.
- Replicate changes to the cluster state across all nodes in the cluster.
- Monitor the health of all cluster nodes by sending *heartbeat signals*, periodic signals that monitor the availability of the data nodes in the cluster.

The following illustration shows an OpenSearch Service domain with 10 instances. Seven of the instances are data nodes and three are dedicated manager nodes. Only one of the dedicated manager nodes is active. The two gray dedicated manager nodes wait as backup in case the active dedicated manager node fails. All data upload requests are served by the seven data nodes, and all cluster management tasks are offloaded to the active dedicated manager node.



Choosing the number of dedicated manager nodes

We recommend that you use Multi-AZ with Standby, which adds **three** dedicated manager nodes to each production OpenSearch Service domain. If you deploy with Multi-AZ without Standby or single-AZ, we still recommend three dedicated manager nodes. Never choose an even number of dedicated manager nodes. Consider the following when choosing the number of dedicated manager nodes:

- One dedicated manager node is explicitly prohibited by OpenSearch Service because you have no backup in the event of a failure. You receive a validation exception if you try to create a domain with only one dedicated manager node.
- If you have two dedicated manager nodes, your cluster doesn't have the necessary quorum of nodes to elect a new manager node in the event of a failure.

A quorum is the number of dedicated manager nodes / 2 + 1 (rounded down to the nearest whole number). In this case, 2 / 2 + 1 = 2. Because one dedicated manager node has failed and only one backup exists, the cluster doesn't have a quorum and can't elect a new manager.

- Three dedicated manager nodes, the recommended number, provides two backup nodes in the event of a manager node failure and the necessary quorum (2) to elect a new manager.
- Four dedicated manager nodes are not better than three and can cause issues if you use multiple Availability Zones.
  - If one manager node fails, you have the quorum (3) to elect a new manager. If two nodes fail, you lose that quorum, just as you do with three dedicated manager nodes.
  - In a three Availability Zone configuration, two AZs have one dedicated manager node, and one AZ has two. If that AZ experiences a disruption, the remaining two AZs don't have the necessary quorum (3) to elect a new manager.
- Having five dedicated manager nodes works as well as three and allows you to lose two nodes while maintaining a quorum. But because only one dedicated manager node is active at any given time, this configuration means that you pay for four idle nodes. Many users find this level of failover protection excessive.

If a cluster has an even number of manager-eligible nodes, OpenSearch and Elasticsearch versions 7.*x* and later ignore one node so that the voting configuration is always an odd number. In this case, four dedicated manager nodes are essentially equivalent to three (and two to one).

> **Note**
>
> If your cluster doesn't have the necessary quorum to elect a new manager node, write *and* read requests to the cluster both fail. This behavior differs from the OpenSearch default.

Choosing instance types for dedicated manager nodes

OpenSearch Service domain and instance quotas

Although dedicated manager nodes don't process search and query requests, their size is highly correlated with the instance size and number of instances, indexes, and shards that they can manage. For production clusters, we recommend, at a minimum, the following instance types for dedicated manager nodes.

These recommendations are based on typical workloads and can vary based on your needs. Clusters with many shards or field mappings can benefit from larger instance types. For more information, see Recommended CloudWatch alarms for Amazon OpenSearch Service to determine if you need to use a larger instance type.

RAM	Max Node Support for Elasticsearch and OpenSearch Service 1.x to 2.15	Max Shard Support for Elasticsearch and OpenSearch Service 2.15 and above	Max Node Support for Elasticsearch and OpenSearch Service 1.x to 2.15	Max Shard Support for Elasticsearch and OpenSearch Service 2.17 and above
2 GB	Not applicable	Not applicable	10	1K
4 GB	Not applicable	Not applicable	10	5K
8 GB	10	10K	30	15K
16 GB	30	30K	60	30K
32 GB	75	40K	120	60K
64 GB	125	75K	240	120K
128 GB	200	75K	480	240K
256 GB	Not applicable	Not applicable	1002	500K

Recommended CloudWatch alarms for Amazon OpenSearch Service

CloudWatch alarms perform an action when a CloudWatch metric exceeds a specified value for some amount of time. For example, you might want AWS to email you if your cluster health status is `red` for longer than one minute. This section includes some recommended alarms for Amazon OpenSearch Service and how to respond to them.

You can automatically deploy these alarms using AWS CloudFormation. For a sample stack, see the related GitHub repository.

Note

If you deploy the CloudFormation stack, the `KMSKeyError` and `KMSKeyInaccessible` alarms will exists in an `Insufficient Data` state because these metrics only appear if a domain encounters a problem with its encryption key.

For more information about configuring alarms, see Creating Amazon CloudWatch Alarms in the *Amazon CloudWatch User Guide*.

Alarm	Issue
`ClusterStatus.red` maximum is >= 1 for 1 minute, 1 consecutive time	At least one primary shard and its replicas are not allocated to a node. See the section called "Red cluster status".
`ClusterStatus.yellow` maximum is >= 1 for 1 minute, 5 consecutive times	At least one replica shard is not allocated to a node. See the section called "Yellow cluster status".
`FreeStorageSpace` minimum is <= 20480 for 1 minute, 1 consecutive time	A node in your cluster is down to 20 GiB of free storage space. See the section called "Lack of available storage space". This value is in MiB, so rather than 20480, we recommend setting it to 25% of the storage space for each node.
`ClusterIndexWritesBlocked` is >= 1 for 5 minutes, 1 consecutive time	Your cluster is blocking write requests. See the section called "ClusterBlockException".
`Nodes` minimum is < *x* for 1 day, 1 consecutive time	*x* is the number of nodes in your cluster. This alarm indicates that at least one node in your cluster has been unreachable for one day. See the section called "Failed cluster nodes".

Alarm	Issue

`AutomatedSnapshotFailure` maximum is >= 1 for 1 minute, 1 consecutive time	An automated snapshot failed. This failure is often the result of a red cluster health status. See the section called "Red cluster status".
	
	For a summary of all automated snapshots and some information about failures, try one of the following requests:
	
	`GET domain_endpoint /_snapshot/cs-automated/_all`
	`GET domain_endpoint /_snapshot/cs-automated-enc/_all`

`CPUUtilization` or `WarmCPUUtilization` maximum is >= 80% for 15 minutes, 3 consecutive times	100% CPU utilization might occur sometimes, but *sustained* high usage is problematic. Consider using larger instance types or adding instances.
`JVMMemoryPressure` maximum is >= 95% for 1 minute, 3 consecutive times	The cluster could encounter out of memory errors if usage increases . Consider scaling vertically. OpenSearch Service uses half of an instance's RAM for the Java heap, up to a heap size of 32 GiB. You can scale instances vertically up to 64 GiB of RAM, at which point you can scale horizontally by adding instances.
`OldGenJVMMemoryPressure` maximum is >= 80% for 1 minute, 3 consecutive times	
`ManagerCPUUtilization` maximum is >= 50% for 15 minutes, 3 consecutive times	Consider using larger instance types for your dedicated manager nodes. Because of their role in cluster stability and blue/green deployments, dedicated manager nodes should have lower CPU usage than data nodes.

Alarm	Issue
`ManagerJVMMemoryPressure` maximum is >= 95% for 1 minute, 3 consecutive times	
`ManagerOldGenJVMMemoryPressure` maximum is >= 80% for 1 minute, 3 consecutive times	
`KMSKeyError` is >= 1 for 1 minute, 1 consecutive time	The AWS KMS encryption key that is used to encrypt data at rest in your domain is disabled. Re-enable it to restore normal operations. For more information, see the section called "Encryption at rest".
`KMSKeyInaccessible` is >= 1 for 1 minute, 1 consecutive time	The AWS KMS encryption key that is used to encrypt data at rest in your domain has been deleted or has revoked its grants to OpenSearch Service. You can't recover domains that are in this state. However, if you have a manual snapshot, you can use it to migrate to a new domain. To learn more, see the section called "Encryption at rest".
`shards.active` is >= 30000 for 1 minute, 1 consecutive time	The total number of active primary and replica shards is greater than 30,000. You might be rotating your indexes too frequently. Consider using ISM to remove indexes once they reach a specific age.
5*xx* alarms >= 10% of `OpenSearchRequests`	One or more data nodes might be overloaded, or requests are failing to complete within the idle timeout period. Consider switching to larger instance types or adding more nodes to the cluster. Confirm that you're following best practices for shard and cluster architecture.

Alarm	Issue
`ManagerReachableFromNode` maximum is < 1 for 5 minutes, 1 consecutive time	This alarm indicates that the manager node stopped or is unreachable. These failures are usually the result of a network connectivity issue or an AWS dependency problem.
`ThreadpoolWriteQueue` average is >= 100 for 1 minute, 1 consecutive time	The cluster is experiencing high indexing concurrency. Review and control indexing requests, or increase cluster resources.
`ThreadpoolSearchQueue` average is >= 500 for 1 minute, 1 consecutive time	The cluster is experiencing high search concurrency. Consider scaling your cluster. You can also increase the search queue size, but increasing it excessively can cause out of memory errors.
`ThreadpoolSearchQueue` maximum is >= 5000 for 1 minute, 1 consecutive time	
Increase in `ThreadpoolSearchRejected` SUM is >=1{ math expression DIFF ( )} for 1 minute, 1 consecutive time	These alarms notify you of domain issues that might impact performance and stability.

Alarm	Issue
Increase in `Threadpool WriteRejected` SUM is >=1{ math expression DIFF ( )} for 1 minute, 1 consecutive time	

> **Note**
>
> If you just want to *view* metrics, see the section called “Monitoring cluster metrics”.

Other alarms you might consider

Consider configuring the following alarms depending on which OpenSearch Service features you regularly use.

Alarm	Issue
`WarmFreeStorageSpace` is >= 10%	You have reached 10% of your total free warm storage. `WarmFreeStorageSpace` measures the sum of your free warm storage space in MiB. UltraWarm uses Amazon S3 rather than attached disks.
`HotToWarmMigrationQueueSize` is >= 20 for 1 minute, 3 consecutive times	A high number of indexes are concurrently moving from hot to UltraWarm storage. Consider scaling your cluster.
`HotToWarmMigrationSuccessLa`	Configure this alarm so that you're notified if the `HotToWarmMigrationSuccessCount` x latency is greater than 24 hours if you’re trying to roll daily indexes.

Alarm	Issue
`tency` is >= 1 day, 1 consecutive time	
`WarmJVMMemoryPressure` maximum is >= 95% for 1 minute, 3 consecutive times	The cluster could encounter out of memory errors if usage increases . Consider scaling vertically. OpenSearch Service uses half of an instance's RAM for the Java heap, up to a heap size of 32 GiB. You can scale instances vertically up to 64 GiB of RAM, at which point you can scale horizontally by adding instances.
`WarmOldGenJVMMemoryPressure` maximum is >= 80% for 1 minute, 3 consecutive times	
`WarmToColdMigrationQueueSize` is >= 20 for 1 minute, 3 consecutive times	A high number of indexes are concurrently moving from UltraWarm to cold storage. Consider scaling your cluster.
`HotToWarmMigrationFailureCount` is >= 1 for 1 minute, 1 consecutive time	Migrations might fail during snapshots, shard relocations, or force merges. Failures during snapshots or shard relocation are typically due to node failures or S3 connectivity issues. Lack of disk space is usually the underlying cause of force merge failures.
`WarmToColdMigrationFailureCount` is >= 1 for 1 minute, 1 consecutive time	Migrations usually fail when attempts to migrate index metadata to cold storage fail. Failures can also happen when the warm index cluster state is being removed.

Alarm	Issue
`WarmToColdMigrationLatency` is >= 1 day, 1 consecutive time	Configure this alarm so that you're notified if the `WarmToColdMigrationSuccessCount` x latency is greater than 24 hours if you're trying to roll daily indexes.
`AlertingDegraded` is >= 1 for 1 minute, 1 consecutive time	Either the alerting index is red, or one or more nodes is not on schedule.
`ADPluginUnhealthy` is >= 1 for 1 minute, 1 consecutive time	The anomaly detection plugin isn't functioning properly, either because of high failure rates or because one of the indexes being used is red.
`AsynchronousSearchFailureRate` is >= 1 for 1 minute, 1 consecutive time	At least one asynchronous search failed in the last minute, which likely means the coordinator node failed. The lifecycle of an asynchronous search request is managed solely on the coordinator node, so if the coordinator goes down, the request fails.
`AsynchronousSearchStoreHealth` is >= 1 for 1 minute, 1 consecutive time	The health of the asynchronous search response store in the persisted index is red. You might be storing large asynchronous responses, which can destabilize a cluster. Try to limit your asynchronous search responses to 10 MB or less.
`SQLUnhealthy` is >= 1 for 1 minute, 3 consecutive times	The SQL plugin is returning 5*xx* response codes or passing invalid query DSL to OpenSearch. Troubleshoot the requests that your clients are making to the plugin.
`LTRStatus.red` is >= 1 for 1 minute, 1 consecutive time	At least one of the indexes needed to run the Learning to Rank plugin has missing primary shards and isn't functional.

General reference for Amazon OpenSearch Service

Amazon OpenSearch Service supports a variety of instances, operations, plugins, and other resources.

Topics

Supported instance types in Amazon OpenSearch Service

Amazon OpenSearch Service supports the following instance types. Not all Regions support all instance types. For availability details, see Amazon OpenSearch Service pricing.

For information about which instance type is appropriate for your use case, see the section called "Sizing domains", the section called "EBS volume size quotas", and the section called "Network quotas".

Current generation instance types

For the best performance, we recommend that you use the following instance types when you create new OpenSearch Service domains.

Instance type	Instances	Restrictions
i4i	i4i.large.search i4i.xlarge.search i4i.2xlarge.search	The i4i instance types require Elasticsearch 5.1 or later or any version of OpenSearch, and do not support EBS volume storage.

Instance type	Instances	Restrictions
	i4i.4xlarge.search i4i.8xlarge.search i4i.12xlarge.search i4i.16xlarge.search i4i.24xlarge.search i4i.32xlarge.search	
i4g	i4g.large.search i4g.xlarge.search i4g.2xlarge.search i4g.4xlarge.search i4g.8xlarge.search i4g.16xlarge.search	The i4g instance types require Elasticsearch 7.9 or later or any version of OpenSearch, and do not support EBS storage volumes.

Instance type	Instances	Restrictions
Graviton3	C7g.large.search C7g.xlarge.search C7g.2xlarge.search C7g.4xlarge.search C7g.8xlarge.search C7g.12xlarge.search C7g.16xlarge.search M7g.large.search M7g.xlarge.search M7g.2xlarge.search M7g.4xlarge.search M7g.8xlarge.search	Graviton3 only supports GP3.

Instance type	Instances	Restrictions
	M7g.12xlarge.search	
	M7g.16xlarge.search	
	R7g.medium.search	
	R7g.large.search	
	R7g.xlarge.search	
	R7g.2xlarge.search	
	R7g.4xlarge.search	
	R7g.8xlarge.search	
	R7g.12xlarge.search	
	R7g.16xlarge.search	
	R7gd.large.search	
	R7gd.xlarge.search	
	R7gd.2xlarge.search	

Instance type	Instances	Restrictions
	R7gd.4xlarge.search R7gd.8xlarge.search R7gd.12xlarge.search R7gd.16xlarge.search	

Instance type	Instances	Restrictions
OR1	or1.medium.search or1.large.search or1.xlarge.search or1.2xlarge.search or1.4xlarge.search or1.8xlarge.search or1.12xlarge.search or1.16xlarge.search	• The OR1 instance types require OpenSearch 2.11 or later. • OR1 instances are only compatible with other Graviton instance types master nodes (C6g, M6g, R6g).

Instance type	Instances	Restrictions

Im4gn	`im4gn.large.search`
	`im4gn.xlarge.search`
	`im4gn.2xlarge.search`
	`im4gn.4xlarge.search`
	`im4gn.8xlarge.search`
	`im4gn.16xlarge.search`
		• The Im4gn instance types require Elasticsearch 7.9 or later or any version of OpenSearch, and do not support EBS storage volumes.
		• Im4gn instances are only compatible with other Graviton instance types (C6g, M6g, R6g, R6gd). You can't combine Graviton and non-Graviton instances in the same cluster.

Instance type	Instances	Restrictions

C5	`c5.large.search`
	`c5.xlarge.search`
	`c5.2xlarge.search`
	`c5.4xlarge.search`
	`c5.9xlarge.search`
	`c5.18xlarge.search`
		The C5 instance types require Elasticsearch 5.1 or later or any version of OpenSearch.

Instance type	Instances	Restrictions

C6g	`c6g.large.search`
	`c6g.xlarge.search`
	`c6g.2xlarge.search`
	`c6g.4xlarge.search`
	`c6g.8xlarge.search`
	`c6g.12xlarge.search`
		• The C6g instance types require Elasticsearch 7.9 or later or any version of OpenSearch.
		• C6g instances are only compatible with other Graviton instance types (Im4gn, M6g, R6g, R6gd). You can't combine Graviton and non-Graviton instances in the same cluster.

Instance type	Instances	Restrictions
I3	i3.large.search i3.xlarge.search i3.2xlarge.search i3.4xlarge.search i3.8xlarge.search i3.16xlarge.search	
M5	m5.large.search m5.xlarge.search m5.2xlarge.search m5.4xlarge.search m5.12xlarge.search	The M5 instance types require Elasticsearch 5.1 or later or any version of OpenSearch.

Instance type	Instances	Restrictions

M6g	`m6g.large.search`
	`m6g.xlarge.search`
	`m6g.2xlarge.search`
	`m6g.4xlarge.search`
	`m6g.8xlarge.search`
	`m6g.12xlarge.search`
		• The M6g instance types require Elasticsearch 7.9 or later or any version of OpenSearch.
		• M6g instances are only compatible with other Graviton instance types (Im4gn, C6g, R6g, R6gd). You can't combine Graviton and non-Graviton instances in the same cluster.

Instance type	Instances	Restrictions

R5	`r5.large.search`
	`r5.xlarge.search`
	`r5.2xlarge.search`
	`r5.4xlarge.search`
	`r5.12xlarge.search`
		The R5 instance types require Elasticsearch 5.1 or later or any version of OpenSearch.

Instance type	Instances	Restrictions

R6g	`r6g.large.search`
	`r6g.xlarge.search`
	`r6g.2xlarge.search`
	`r6g.4xlarge.search`
	`r6g.8xlarge.search`
	`r6g.12xlarge.search`
		• The R6g instance types require Elasticsearch 7.9 or later or any version of OpenSearch.
		• R6g instances are only compatible with other Graviton instance types (Im4gn, C6g, M6g, R6gd). You can't combine Graviton and non-Graviton instances in the same cluster.

Instance type	Instances	Restrictions

R6gd	`r6gd.large.search`
	`r6gd.xlarge.search`
	`r6gd.2xlarge.search`
	`r6gd.4xlarge.search`
	`r6gd.8xlarge.search`
	`r6gd.12xlarge.search`
	`r6gd.16xlarge.search`
		• The R6gd instance types require Elasticsearch 7.9 or later or any version of OpenSearch, and do not support EBS storage volumes.
		• R6gd instances are only compatible with other Graviton instance types (Im4gn, C6g, M6g, R6g). You can't combine Graviton and non-Graviton instances in the same cluster.

Instance type	Instances	Restrictions
T3	`t3.small.search` `t3.medium.search`	• The T3 instance types require Elasticsearch 5.6 or later or any version of OpenSearch. • You can use T3 instance types only if your domain is provisioned without standby. For more information, see the section called “Multi-AZ without Standby”. • You can use T3 instance types only if the instance count for your domain is 10 or fewer. • The T3 instance types do not support UltraWarm storage, cold storage, or Auto-Tune.

c7i	`c7i.large.search`
	`c7i.xlarge.search`
	`c7i.2xlarge.search`
	`c7i.4xlarge.search`
	`c7i.8xlarge.search`
	`c7i.12xlarge.search`
	`c7i.16xlarge.search`
		• The c7i instance requires Elasticsearch 5.1 or later or any version of OpenSearch, and only supports GP3 storage volumes.

Instance type	Instances	Restrictions

m7i	`m7i.large.search`
	`m7i.xlarge.search`
	`m7i.2xlarge.search`
	`m7i.4xlarge.search`
	`m7i.8xlarge.search`
	`m7i.12xlarge.search`
	`m7i.16xlarge.search`
		• The m7i instance requires Elasticsearch 5.1 or later or any version of OpenSearch, and only supports GP3 storage volumes.

Instance type	Instances	Restrictions
r7i	r7i.large.search r7i.xlarge.search r7i.2xlarge.search r7i.4xlarge.search r7i.8xlarge.search r7i.12xlarge.search r7i.16xlarge.search	• The r7i instance requires Elasticsearch 5.1 or later or any version of OpenSearch, and only supports GP3 storage volumes.

Previous generation instance types

OpenSearch Service offers previous generation instance types for users who have optimized their applications around them and have yet to upgrade. We encourage you to use current generation instance types to get the best performance, but we continue to support the following previous generation instance types.

Instance type	Instances	Restrictions

C4	`c4.large.search`
	`c4.xlarge.search`
	`c4.2xlarge.search`
	`c4.4xlarge.search`
	`c4.8xlarge.search`	

I2	`i2.xlarge.search` `i2.2xlarge.search`	

M3	`m3.medium.search`
	`m3.large.search`
	`m3.xlarge.search`
	`m3.2xlarge.search`
		• The M3 instance types do not support encryption of data at rest, fine-grained access control, or cross-cluster search.
		• The M3 instance types have additional restrictions by OpenSearch version. To learn more, see the section called "Invalid M3 instance type".

M4	`m4.large.search`	

Instance type	Instances	Restrictions

	`m4.xlarge.search`
	`m4.2xlarge.search`
	`m4.4xlarge.search`
	`m4.10xlarge.search`	

R3	`r3.large.search`
	`r3.xlarge.search`
	`r3.2xlarge.search`
	`r3.4xlarge.search`
	`r3.8xlarge.search`
		The R3 instance types do not support encryption of data at rest or fine-grained access control.

Instance type	Instances	Restrictions

R4	`r4.large.search`
	`r4.xlarge.search`
	`r4.2xlarge.search`
	`r4.4xlarge.search`
	`r4.8xlarge.search`
	`r4.16xlarge.search`	

T2	`t2.micro.search`
	`t2.small.search`
	`t2.medium.search`
		• You can use the T2 instance types only if the instance count for your domain is 10 or fewer.
		• The `t2.micro.search` instance type supports only Elasticsearch 1.5 and 2.3.
		• The T2 instance types do not support encryption of data at rest, fine-grained access control, UltraWarm storage, cold storage, cross-cluster search, or Auto-Tune.

Tip

We often recommend different instance types for dedicated master nodes and data nodes.

Features by engine version in Amazon OpenSearch Service

Many OpenSearch Service features have a minimum OpenSearch version requirement or legacy Elasticsearch OSS version requirement. If you meet the minimum version for a feature, but the feature isn't available on your domain, update your domain's service software.

Feature	Minimum required OpenSearch version	Minimum required Elasticsearch version
Default concurrent segment search set to auto on hot and warm data instance type 2.xl or more	2.17	Not included
Ultrawarm /Cold tier support KNN indexes	2.17	Not included
Dedicated coordinator node	1.0	6.8
VPC support	1.0	1.0
Require HTTPS for all traffic to the domain		

Feature	Minimum required OpenSearch version	Minimum required Elasticsearch version
Multi-AZ support		
Dedicated master nodes		
Custom packages		
Custom endpoints		
Slow log publishing		
Error log publishing	1.0	5.1
Encryption of data at rest		
Cognito authentication for OpenSearch Dashboards		
In-place upgrades		
Curator support	Not included	5.1

Feature	Minimum required OpenSearch version	Minimum required Elasticsearch version
Hourly automated snapshots	1.0	5.3
Node-to-node encryption	1.0	6.0
Java high-level REST client support		
HTTP request and response compression		
Alerting	1.0	6.2
SQL	1.0	6.5
Cross-cluster search	1.0	6.7
Fine-grained access control		
SAML authentication for OpenSearch Dashboards		

Feature	Minimum required OpenSearch version	Minimum required Elasticsearch version
Auto-Tune		
Remote reindex		
UltraWarm	1.0	6.8
Index State Management		
k-NN by Euclidean distance	1.0	7.1
Anomaly Detection	1.0	7.4
k-NN by cosine similarity	1.0	7.7
Learning to Rank		
Piped processing language	1.0	7.9
OpenSearch Dashboards reports		

Feature	Minimum required OpenSearch version	Minimum required Elasticsearch version
OpenSearch Dashboards Trace Analytics		
ARM-based Graviton instances		
Cold storage		
Hamming distance, L1 Norm distance, and Painless scripting for k-NN	1.0	7.10
Asynchronous search		
Index transforms	1.0	Not included
Cross-cluster replication	1.1	7.10
ML Commons	1.3	Not included
Notifications	2.3	Not included

Feature	Minimum required OpenSearch version	Minimum required Elasticsearch version
Point in time search	2.5	Not included
Search pipelines	2.9	Not included
Machine learning connectors	2.9	Not included
Multimodal semantic search	2.11	Not included
Direct-query data sources for Amazon S3	2.11	Not included

For information about plugins, which enable some of these features and additional functionality, see the section called "Plugins by engine version". For information about the OpenSearch API for each version, see the section called "Supported operations".

Plugins by engine version in Amazon OpenSearch Service

Amazon OpenSearch Service domains come prepackaged with plugins from the OpenSearch community. The service automatically deploys and manages plugins for you, but it deploys different plugins depending on the version of OpenSearch or legacy Elasticsearch OSS you choose for your domain.

The following table lists plugins by OpenSearch version, as well as compatible versions of legacy Elasticsearch OSS. It only includes plugins that you might interact with—it's not comprehensive. OpenSearch Service uses additional plugins to enable core service functionality, such as the S3 Repository plugin for snapshots and the OpenSearch Performance Analyzer plugin for optimization

and monitoring. For a complete list of all plugins running on your domain, make the following request:

```
GET _cat/plugins?v
```

Plugin	Minimum required OpenSearch version	Minimum required Elasticsearch version
HanLP	2.11	Not supported
Hebrew Analysis	2.11	Not supported
Amazon Personalize Search Ranking	2.9	Not supported
Neural Search	2.9	Not supported
Security Analytics	2.5	Not supported
OpenSearch notifications	2.3	Not supported
ML Commons	1.3	Not supported
Sudachi Analysis (recommended for Japanese)	1.3	Not supported
STConvert	1.3	Not supported

Plugin	Minimum required OpenSearch version	Minimum required Elasticsearch version
Pinyin Analysis	1.3	Not supported
Nori Analysis	1.3	Not supported
OpenSearch observability	1.2	Not supported
OpenSearch cross-cluster replication	1.1	7.10
OpenSearch asynchronous search	1.0	7.10
IK (Chinese) Analysis	1.0	7.7
Vietnamese Analysis		
Thai analysis		
Learning to Rank		
OpenSearch anomaly detection	1.0	7.4

Plugin	Minimum required OpenSearch version	Minimum required Elasticsearch version
OpenSearch k-NN	1.0	7.1
OpenSearch Index State Management	1.0	6.8
OpenSearch security	1.0	6.7
OpenSearch SQL	1.0	6.5
OpenSearch alerting	1.0	6.2
Ukrainian Analysis	1.0	5.3
Mapper Size	1.0	5.3
Mapper Murmur3	1.0	5.1
Ingest User Agent Processor	1.0	5.1
Ingest Attachment Processor	1.0	5.1
Stempel Polish Analysis	1.0	5.1

Plugin	Minimum required OpenSearch version	Minimum required Elasticsearch version
Smart Chinese Analysis	1.0	5.1
Seunjeon Korean Analysis	1.0	5.1
Phonetic Analysis	1.0	2.3
Japanese (kuromoji) Analysis	1.0	Included on all domains
ICU Analysis	1.0	Included on all domains

Optional plugins

In addition to the default plugins that come pre-installed, Amazon OpenSearch Service supports several optional language analyzer plugins. You can use the AWS Management Console and AWS CLI to associate a plugin to a domain, disassociate a plugin from a domain, and list all plugins. An optional plugin package is compatible with a specific OpenSearch version, and can only be associated to domains with that version.

Note that for the Sudachi plugin, when you reassociate a dictionary file, it doesn't immediately reflect on the domain. The dictionary refreshes when the next blue/green deployment runs on the domain as part of a configuration change or other update. Alternatively, you can create a new package with the updated data, create a new index using this new package, reindex the existing index to the new index, and then delete the old index. If you prefer to use the reindexing approach, use an index alias so that there's no disruption to your traffic.

Optional plugins use the `ZIP-PLUGIN` package type. For more information about optional plugins, see the section called “Custom packages”.

Third party plugins

Amazon OpenSearch Service now supports third party plugins from select partners. Like optional plugins, you can use the AWS Management Console and AWS CLI to associate a plugin to a domain, disassociate a plugin from a domain, and list all third party plugins in your domain. Third party plugin packages are compatible with specific OpenSearch versions, and can only be associated to domains with that OpenSearch version.

Third party plugins are owned and provided by a third-party developer. You are responsible for obtaining and maintaining valid licences directly from the third-party developers. These third party plugins are avilable in all AWS regions where Amazon OpenSearch Service is available except **AWS GovCloud (US) Regions**.

> **Note**
>
> Some plugin providers may not enable their plugins in all AWS regions where Amazon OpenSearch Service is available, please reach out to the plugin provider on questions related to availability of the plugin in your AWS region.

For more information about third party plugins, see Custom packages for Amazon OpenSearch Service.

The following third party plugins are now available on Amazon OpenSearch Service:

- **Portal26 Encrypted Search Plugin (Titanium-lockbox) :** Portal26 encryption plugin from Portal26.ai uses NIST FIPS 140-2 certified encryption to encrypt data as its indexed by Amazon OpenSearch Service. This plugin includes a Bring Your Own Key (BYOK) capability, allowing you to setup separate encryption keys for each index.
- **Babel Street Match Plugin for OpenSearch (RNI):** This plugin accurately matches names, organizations, addresses, and dates in over 24 languages, enhancing security operations and regulatory compliance while reducing false positives and increasing operational efficiency.

The following third party plugins are available for use with Amazon OpenSearch Service:

Plugin Name	Third Party Provider	Minimum required OpenSearch Service version	Minimum required Elasticsearch version	Requires a license
Titaniu lockbo	Portal26.ai	2.15	Not supported	Y
Name Match (RNI) OpenS h plugin	babelstreet.com	2.15	Not supported	Y

The following Amazon OpenSearch Service features are unavailable for use when using third party plugins:

Plugin Name	Encryption plugin	Babel Street Match plugin
Cross cluster search	Not supported	Not supported
Cross cluster replication	Not supported	Not supported
Remote-re index	Not supported	Not supported
Auto-tune	Not supported	Not supported
Ultrawarm	Not supported	Supported

Plugin Name	Encryption plugin	Babel Street Match plugin
Multi-AZ with Standby	Not supported	Not supported

You can use the `"CreatePackage"`, `"AssociatePackage"` and `"DissociatePackage"` to upload and associate the plugin you use with your Amazon OpenSearch Service managed domain. `"PACKAGE-CONFIG"` and `"PACKAGE-LICENSE"` package types are supported for uploading the plugin configuration and license files. To obtain the license files to install Portal26, see Portal26.ai. To obtain the license files to install the Name Match (RNI) OpenSearch plugin, see Babel Street.

Prerequisites

- Ensure that you have the plugin configuration and license files for theOpenSearch version running on your Amazon OpenSearch Service domain.
- You must have the following enabled on your Amazon OpenSearch Service domain:
  - Node to node encryption
  - Encryption of data at rest
  - Set 'EnforceHTTPS' to 'true'
  - Enable support for **TLSSecurityPolicy 'Policy-Min-TLS-1-2-PFS-2023-10'** . For more information, see DomainEndpointOptions.

Installing third party plugins with AWS CLI

To enable use of third party plugins using the AWS CLI you will need to apply the following service model JSON:

1. Fetch the list of available third party plugins using the describe-packages API.

```
aws opensearch --region $REGION describe-packages --filters '[{"Name":
 "PackageType","Value": ["ZIP-PLUGIN"]}, {"Name": "PackageName","Value":
 ["<package-name>"]}]'
```

2. Create a new package for plugin license using the existing CreatePackage API.

```
aws opensearch --region $REGION create-package --package-name <package-name> --
package-type PACKAGE-LICENSE --package-source S3BucketName=<bucket>,S3Key=<key>
```

Please update the bucket and key location to point to the license file in the account's s3 bucket. The file must have a .json or .xml extension.

3. Create a new package for plugin config using the existing CreatePackage API.

```
aws opensearch --region $REGION create-package --package-name <package-name> --
package-type PACKAGE-CONFIG --package-source S3BucketName=<bucket>,S3Key=<key
```

> **Note**
>
> Please update the bucket and key location to point to the config zip file in the calling account's s3 bucket. The s3 must be in the same region where the package is created. Only zip files are supported for config type packages. The contents of the zip file must follow directory structure as expected by the plugin.

4. Use the new AssociatePackage API to associate the partner plugin along with license and configuration with a compatible Amazon OpenSearch Service domain (matching version) using the package ids of these packages.

```
aws opensearch --region $REGION associate-packages --domain-name <domain-
name> --package-list '[{"PackageID": "<plugin-package-id>"},{"PackageID":
 "<license-package-id>","PrerequisitePackageIDList":["<plugin-package-id>"]},
{"PackageID":"<config-package-id>","PrerequisitePackageIDList":["<plugin-package-
id>"]}]'
```

> **Note**
>
> Plugins are installed and uninstalled using a blue/green deployment process.

5. Use the existing ListPackagesForDomain API to see the status of the association. The association status will change as the workflow progresses from ASSOCATING to ACTIVE. The association status changes to ACTIVE once the plugin installation workflow has been completed and plugin is ready to be used.

```
aws opensearch --region $REGION list-packages-for-domain --domain-name <domain
 name>
```

6. Use the existing GetPackageVersionHistory API to see the versions for any package.
7. License/config packages can be updated using the existing UpdatePackage API. Use the following API to apply package updates to a domain.

```
aws opensearch --region $REGION update-package --package-id <package-id>
 --package-source S3BucketName=<bucket>,S3Key=<key> --package-description
 <description>
```

8. Use the existing DissociatePackage API to uninstall the plugin from any domain. You can use the existing ListPackagesForDomain API to see the status of the dissociation.

```
aws opensearch --region $REGION dissociate-package --package-id <plugin-package-id>
 --domain-name <domain name>
```

> **Note**
>
> In order to uninstall a plugin, you will have first have to disable the plugin from every index before dissociating the plugin package.

9. Use the existing ListPackagesForDomain API to see the status of the dissociation.

Supported operations in Amazon OpenSearch Service

OpenSearch Service supports many versions of OpenSearch and legacy Elasticsearch OSS. The following sections show the operations that OpenSearch Service supports for each version.

Topics

Notable API differences

New List APIs

To support large clusters with large number of indexes and shards, we have introduced new List APIs with pagination support, such as _list/indices and _list/shards. The List API retrieves statistics about indexes and shards in a paginated format. This streamlines the task of processing responses that include many indexes.

- `_list/indices`: _list/indices
- `_list/shards`: _list/shards

Changes to existing APIs

To support large clusters, we have added support in the `_cluster/stats` API to add additional metric filters to support retrieving only relevant stats responses, for example `_cluster/stats/<metric>/nodes/<node-filters>` and `_cluster/stats/<metric>/<index_metric>/nodes/<node-filters>`. For details, see _cluster/stats.

We have added support in `_cat/shards` API for task cancellation by specifying a `cancel_after_time_interval` request parameter. For details, see _cat/shards.

Limiting the response size for _cat API

To support large clusters with total instance count of more than 200 across data and warm nodes, we have a 10K limit on the number of indexes returned by the `_cat/segments API`. If the number of indexes in the response exceeds this limit, the API returns a 429 error. To avoid this, you can specify an index pattern filter in your query, such as `_cat/segments/<index-pattern>`.

Settings and statistics

OpenSearch Service only accepts PUT requests to the `_cluster/settings` API that use the "flat" settings form. It rejects requests that use the expanded settings form.

```
// Accepted
PUT _cluster/settings
{
  "persistent" : {
    "action.auto_create_index" : false
  }
```

```
}

// Rejected
PUT _cluster/settings
{
  "persistent": {
    "action": {
      "auto_create_index": false
    }
  }
}
```

The high-level Java REST client uses the expanded form, so if you need to send settings requests, use the low-level client.

Prior to Elasticsearch 5.3, the `_cluster/settings` API on OpenSearch Service domains supported only the HTTP PUT method, not the GET method. OpenSearch and later versions of Elasticsearch support the GET method, as shown in the following example:

```
GET https://domain-name.region.es.amazonaws.com/_cluster/settings?pretty
```

Here is a return example:

```
{
  "persistent": {
    "cluster": {
      "routing": {
        "allocation": {
          "cluster_concurrent_rebalance": "2",
          "node_concurrent_recoveries": "2",
          "disk": {
            "watermark": {
              "low": "1.35gb",
              "flood_stage": "0.45gb",
              "high": "0.9gb"
            }
          },
          "node_initial_primarirecoveries": "4"
        }
      }
    },
    "indices": {
```

```
      "recovery": {
        "max_bytper_sec": "40mb"
      }
    }
  }
}
```

If you compare responses from an open source OpenSearch cluster and OpenSearch Service for certain settings and statistics APIs, you might notice missing fields. OpenSearch Service redacts certain information that exposes service internals, such as the file system data path from `_nodes/stats` or the operating system name and version from `_nodes`.

Shrink

The `_shrink` API can cause upgrades, configuration changes, and domain deletions to fail. We don't recommend using it on domains that run Elasticsearch versions 5.3 or 5.1. These versions have a bug that can cause snapshot restoration of shrunken indices to fail.

If you use the `_shrink` API on other Elasticsearch or OpenSearch versions, make the following request before starting the shrink operation:

```
PUT https://domain-name.region.es.amazonaws.com/source-index/_settings
{
  "settings": {
    "index.routing.allocation.require._name": "name-of-the-node-to-shrink-to",
    "index.blocks.read_only": true
  }
}
```

Then make the following requests after completing the shrink operation:

```
PUT https://domain-name.region.es.amazonaws.com/source-index/_settings
{
  "settings": {
    "index.routing.allocation.require._name": null,
    "index.blocks.read_only": false
  }
}

PUT https://domain-name.region.es.amazonaws.com/shrunken-index/_settings
{
  "settings": {
```

```
    "index.routing.allocation.require._name": null,
    "index.blocks.read_only": false
  }
}
```

New list APIs

To support large clusters with huge number of indexes and shards, we have introduced new list APIs with pagination support i.e. `_list/indices` and `_list/shards`. The List API retrieves statistics about indexes and shards in a paginated format. This streamlines the task of processing responses that include many indexes. For more information on `_list/indices`, see List indices. For more information on `_list/shards`, see List shards.

Changes to existing APIs

To support large clusters, we have added support in `_cluster/stats/<metric>/nodes/<node-filters>` and `_cluster/stats/<metric>/<index_metric>/nodes/<node-filters>`. For more information on `_cluster/stats`, see Cluster stats.

Limiting the response size for _cat APIs

To support large clusters with total instance count more than 200 across data and warm nodes, we have a 10,000 limit on the number of indexes returned by _cat/segments API. If the number of indexes in the response exceeds this limit, the API returns a 429 error. To avoid this, you can specify an index pattern filter in your query (for example, `_cat/segments/<index-pattern>`).

Additionally, support for task cancellation has is now available for `_cat/shards` API for task cancellation by specifying `cancel_after_time_interval` request parameter. For more information on this, see CAT shards.

Choosing the instance types for dedicated master nodes

The following table provides recommendations for choosing the appropriate instance types for dedicdated master nodes:

RAM	Maximum node supported	Maximum shard supported	
2 GB	10	1,000	

RAM	Maximum node supported	Maximum shard supported
4 GB	10	5,000
8 GB	30	15,000
16 GB	60	30,000
32 GB	120	60,000
64 GB	240	120,000
128 GB	480	240,000
256 GB	1002	500,000

OpenSearch version 2.17

For OpenSearch 2.17, OpenSearch Service supports the following operations. For information about most of the operations, see the OpenSearch REST API reference, or the API reference for the specific plugin.

- All operations in the index path (such as */index-name* / _forcemerge , */index-name* /update/*id*, and */index-name* /_close)
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_list
- /_ltr
- /_mapping
- /_mget
- /_msearch
- /_refresh
- /_reindex [1]
- /_render
- /_resolve/index
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search/pipeline
- /_search/point_in_time
- /_search profile
- /_shard_stores

- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
  - cluster.max_shards_per_node
  - cluster.search.request.slowlog.level
  - cluster.search.request.slowlog.threshold.warn
  - cluster.search.request.slowlog.threshold.info
  - cluster.search.request.slowlog.threshold.debug
  - cluster.search.request.slowlog.threshold.trace
- /_mtermvectors
- /_nodes
- /_plugins/_asynchronous_search
- /_plugins/_alerting
- /_plugins/_anomaly_detection
- /_plugins/_ism
- /_plugins/_ml
- /_plugins/_notifications
- /_plugins/_ppl
- /_plugins/_security
- /_plugins/_security_analytics
- /_plugins/_sm
- /_plugins/_sql
- /_percolate
- /_rank_eval
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

  - `search.phase_took_enabled`
- `/_cluster/state`
- `/_cluster/stats`
- `/_count`
- `/_dashboards`

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the `/_tasks` operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refer to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic OpenSearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and others.
5. See the section called “Shrink”.

> **Note**
>
> Currently, changing the `cluster.max_shards_per_node` setting functionality is not enabled for customers with Multi-AZ (Availability Zone) with standby.

OpenSearch version 2.15

For OpenSearch 2.15, OpenSearch Service supports the following operations. For information about most of the operations, see the OpenSearch REST API reference, or the API reference for the specific plugin.

- All operations in the index path (such as */index-name* /_forcemerge , */index-name* /update/*id*, and */index-name* /_close)
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
  - cluster.max_shards_per_node
  - cluster.search.request.slowlog.level
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_ltr
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_plugins/_asynchronous_search
- /_plugins/_alerting
- /_plugins/_anomaly_detection
- /_plugins/_ism
- /_plugins/_ml
- /_plugins/_notifications
- /_plugins/_ppl
- /_plugins/_security
- /_plugins/_security_analytics
- /_plugins/_sm
- /_plugins/_sql
- /_percolate
- /_rank_eval
- /_refresh
- /_reindex [1]
- /_render
- /_resolve/index
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search/pipeline
- /_search/point_in_time
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

    - cluster.search.request.slowlog.threshold.warn
    - cluster.search.request.slowlog.threshold.info
    - cluster.search.request.slowlog.threshold.debug
    - cluster.search.request.slowlog.threshold.trace
    - search.phase_took_enabled
- /_cluster/state
- /_cluster/stats
- /_count
- /_dashboards

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to /_search/scroll with a message body must specify "Content-Length" in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in scroll_id values, use the request body, not the query string, to pass scroll_id values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic OpenSearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

OpenSearch version 2.13

For OpenSearch 2.13, OpenSearch Service supports the following operations. For information about most of the operations, see the OpenSearch REST API reference, or the API reference for the specific plugin.

- All operations in the index path (such as */index-name* /_forcemerge , */index-name* /update/*id*, and */index-name* /_close)
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_ltr
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_plugins/_asynchronous_search
- /_plugins/_alerting
- /_plugins/_anomaly_detection
- /_plugins/_ism
- /_plugins/_ml
- /_plugins/_notifications
- /_plugins/_ppl
- /_plugins/_security
- /_refresh
- /_reindex [1]
- /_render
- /_resolve/index
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search/pipeline
- /_search/point_in_time
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

  - indices.breaker.total.limit
  - cluster.max_shards_per_node
  - cluster.search.request.slowlog.level
  - cluster.search.request.slowlog.threshold.warn
  - cluster.search.request.slowlog.threshold.info
  - cluster.search.request.slowlog.threshold.debug
  - cluster.search.request.slowlog.threshold.trace
  - search.phase_took_enabled
- /_cluster/state
- /_cluster/stats
- /_count
- /_dashboards
- /_plugins/_security_analytics
- /_plugins/_sm
- /_plugins/_sql
- /_percolate
- /_rank_eval

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to /_search/scroll with a message body must specify "Content-Length" in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in scroll_id values, use the request body, not the query string, to pass scroll_id values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.

4. Refers to the PUT method. For information about the GET method, see the section called "Notable API differences". This list only refers to the generic OpenSearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called "Shrink".

OpenSearch version 2.11

For OpenSearch 2.11, OpenSearch Service supports the following operations. For information about most of the operations, see the OpenSearch REST API reference, or the API reference for the specific plugin.

- All operations in the index path (such as */index-name* /_forcemerge , */index-name* /update/*id*, and */index-name* /_close)
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_ltr
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_plugins/_asynchronous_search
- /_plugins/_alerting
- /_plugins/_anomaly_detection
- /_plugins/_ism
- /_plugins/_ml
- /_refresh
- /_reindex [1]
- /_render
- /_resolve/index
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search/pipeline
- /_search/point_in_time
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]

- action.search.shard_count.limit
- indices.breaker.fielddata.limit
- indices.breaker.request.limit
- indices.breaker.total.limit
- cluster.max_shards_per_node
- /_cluster/state
- /_cluster/stats
- /_count
- /_dashboards
- /_plugins/_notifications
- /_plugins/_ppl
- /_plugins/_security
- /_plugins/_security_analytics
- /_plugins/_sm
- /_plugins/_sql
- /_percolate
- /_rank_eval
- /_validate

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the `/_tasks` operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic OpenSearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

OpenSearch version 2.9

For OpenSearch 2.9, OpenSearch Service supports the following operations. For information about most of the operations, see the OpenSearch REST API reference, or the API reference for the specific plugin.

- All operations in the index path (such as */index-name* /_forcemerge , */index-name* /update/*id*, and */index-name* /_close)
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_ltr
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_plugins/_asynchronous_search
- /_plugins/_alerting
- /_plugins/_anomaly_detection
- /_plugins/_ism
- /_plugins/_ml
- /_plugins/_notifications
- /_plugins/_ppl
- /_plugins/_security
- /_refresh
- /_reindex [1]
- /_render
- /_resolve/index
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search/pipeline
- /_search/point_in_time
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

- indices.breaker.total.limit
- cluster.max_shards_per_node
- /_cluster/state
- /_cluster/stats
- /_count
- /_dashboards
- /_plugins/_security_analytics
- /_plugins/_sm
- /_plugins/_sql
- /_percolate
- /_rank_eval

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to /_search/scroll with a message body must specify "Content-Length" in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in scroll_id values, use the request body, not the query string, to pass scroll_id values to OpenSearch Service.
3. For considerations about using scripts, see the section called "Other supported resources".
4. Refers to the PUT method. For information about the GET method, see the section called "Notable API differences". This list only refers to the generic OpenSearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called "Shrink".

OpenSearch version 2.7

For OpenSearch 2.7, OpenSearch Service supports the following operations. For information about most of the operations, see the OpenSearch REST API reference, or the API reference for the specific plugin.

- All operations in the index path (such as */index-name* /_forcemerge ,*/index-nam*
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_refresh
- /_reindex [1]
- /_render
- /_resolve/index

- *e* /update/*id*, and /*index-name* /_close)
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
  - cluster.max_shards_per_node
- /_cluster/state
- /_cluster/stats
- /_count
- /_dashboards
- /_flush
- /_ingest/pipeline
- /_ltr
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_plugins/_asynchronous_search
- /_plugins/_alerting
- /_plugins/_anomaly_detection
- /_plugins/_ism
- /_plugins/_ml
- /_plugins/_notifications
- /_plugins/_ppl
- /_plugins/_security
- /_plugins/_security_analytics
- /_plugins/_sm
- /_plugins/_sql
- /_percolate
- /_rank_eval
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search/point_in_time
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the `/_tasks` operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called "Other supported resources".
4. Refers to the PUT method. For information about the GET method, see the section called "Notable API differences". This list only refers to the generic OpenSearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called "Shrink".

OpenSearch version 2.5

For OpenSearch 2.5, OpenSearch Service supports the following operations. For information about most of the operations, see the OpenSearch REST API reference, or the API reference for the specific plugin.

- All operations in the index path (such as */index-name* `/_forcemerge` , */index-name* `/update/`*id*, and */index-name* `/_close`)
- `/_alias`
- `/_aliases`
- `/_all`
- `/_analyze`
- `/_bulk`
- `/_cat` (except `/_cat/nodeattrs`)
- `/_cluster/allocation/explain`
- `/_delete_by_query` [1]
- `/_explain`
- `/_field_caps`
- `/_field_stats`
- `/_flush`
- `/_ingest/pipeline`
- `/_ltr`
- `/_mapping`
- `/_mget`
- `/_msearch`
- `/_mtermvectors`
- `/_nodes`
- `/_refresh`
- `/_reindex` [1]
- `/_render`
- `/_resolve/index`
- `/_rollover`
- `/_scripts` [3]
- `/_search`[2]
- `/_search/point_in_time`
- `/_search profile`
- `/_shard_stores`
- `/_shrink`[5]
- `/_snapshot`

- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
  - cluster.max_shards_per_node
- /_cluster/state
- /_cluster/stats
- /_count
- /_dashboards
- /_plugins/_asynchronous_search
- /_plugins/_alerting
- /_plugins/_anomaly_detection
- /_plugins/_ism
- /_plugins/_ml
- /_plugins/_notifications
- /_plugins/_ppl
- /_plugins/_security
- /_plugins/_security_analytics
- /_plugins/_sm
- /_plugins/_sql
- /_percolate
- /_rank_eval
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to /_search/scroll with a message body must specify "Content-Length" in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in scroll_id values, use the request body, not the query string, to pass scroll_id values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic OpenSearch operations that

OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.

5. See the section called "Shrink".

OpenSearch version 2.3

For OpenSearch 2.3, OpenSearch Service supports the following operations. For information about most of the operations, see the OpenSearch REST API reference, or the API reference for the specific plugin.

- All operations in the index path (such as */index-name* /_forcemerge , */index-name* /update/*id*, and */index-name* /_close)
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_ltr
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_plugins/_asynchronous_search
- /_plugins/_alerting
- /_plugins/_anomaly_detection
- /_plugins/_ism
- /_plugins/_ml
- _plugins/_notifications
- /_refresh
- /_reindex [1]
- /_render
- /_resolve/index
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
  - cluster.max_shards_per_node
- /_cluster/state
- /_cluster/stats
- /_count
- /_dashboards
- /_plugins/_ppl
- /_plugins/_security
- /_plugins/_sql
- /_percolate
- /_rank_eval

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to /_search/scroll with a message body must specify "Content-Length" in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in scroll_id values, use the request body, not the query string, to pass scroll_id values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic OpenSearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

OpenSearch version 1.3

For OpenSearch 1.3, OpenSearch Service supports the following operations. For information about most of the operations, see the OpenSearch REST API reference, or the API reference for the specific plugin.

- All operations in the index path (such as */index-name* / _forcemerge , */index-name* /update/*id*, and */index-name* /_close)
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
  - cluster.max_shards_per_node
- /_cluster/state
- /_cluster/stats
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_ltr
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_plugins/_asynchronous_search
- /_plugins/_alerting
- /_plugins/_anomaly_detection
- /_plugins/_ism
- /_plugins/_ml
- /_plugins/_ppl
- /_plugins/_security
- /_plugins/_sql
- /_percolate
- /_rank_eval
- /_refresh
- /_reindex [1]
- /_render
- /_resolve/index
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

- /_count
- /_dashboards

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to /_search/scroll with a message body must specify "Content-Length" in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in scroll_id values, use the request body, not the query string, to pass scroll_id values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic OpenSearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

OpenSearch version 1.2

For OpenSearch 1.2, OpenSearch Service supports the following operations. For information about most of the operations, see the OpenSearch REST API reference, or the API reference for the specific plugin.

- All operations in the index path (such as */index-name* /_forcemerge , */index-name* /update/*id*, and */index-name* /_close)
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_ltr
- /_mapping
- /_mget
- /_refresh
- /_reindex [1]
- /_render
- /_resolve/index
- /_rollover
- /_scripts [3]
- /_search [2]
- /_search profile
- /_shard_stores

- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
  - cluster.max_shards_per_node
- /_cluster/state
- /_cluster/stats
- /_count
- /_dashboards
- /_msearch
- /_mtermvectors
- /_nodes
- /_plugins/_asynchronous_search
- /_plugins/_alerting
- /_plugins/_anomaly_detection
- /_plugins/_ism
- /_plugins/_ppl
- /_plugins/_security
- /_plugins/_sql
- /_percolate
- /_rank_eval
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to /_search/scroll with a message body must specify "Content-Length" in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in scroll_id values, use the request body, not the query string, to pass scroll_id values to OpenSearch Service.

3. For considerations about using scripts, see the section called "Other supported resources".
4. Refers to the PUT method. For information about the GET method, see the section called "Notable API differences". This list only refers to the generic OpenSearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called "Shrink".

OpenSearch version 1.1

For OpenSearch 1.1, OpenSearch Service supports the following operations. For information about most of the operations, see the OpenSearch REST API reference, or the API reference for the specific plugin.

- All operations in the index path (such as */index-name* /_forcemerge , */index-name* /update/*id*, and */index-name* /_close)
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_ltr
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_plugins/_asynchronous_search
- /_plugins/_alerting
- /_plugins/_anomaly_detection
- /_plugins/_ism
- /_refresh
- /_reindex [1]
- /_render
- /_resolve/index
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
  - cluster.max_shards_per_node
- /_cluster/state
- /_cluster/stats
- /_count
- /_dashboards
- /_plugins/_ppl
- /_plugins/_security
- /_plugins/_sql
- /_plugins/_transforms
- /_percolate
- /_rank_eval

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to /_search/scroll with a message body must specify "Content-Length" in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in scroll_id values, use the request body, not the query string, to pass scroll_id values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic OpenSearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

OpenSearch version 1.0

For OpenSearch 1.0, OpenSearch Service supports the following operations. For information about most of the operations, see the OpenSearch REST API reference, or the API reference for the specific plugin.

- All operations in the index path (such as */index-name* / _forcemerge , */index-name* /update/*id*, and */index-name* /_close)
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_ltr
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_plugins/_asynchronous_search
- /_plugins/_alerting
- /_plugins/_anomaly_detection
- /_plugins/_ism
- /_plugins/_ppl
- /_plugins/_security
- /_plugins/_sql
- /_plugins/_transforms
- /_percolate
- /_refresh
- /_reindex [1]
- /_render
- /_resolve/index
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

- indices.breaker.total.limit
- cluster.max_shards_per_node
- /_cluster/state
- /_cluster/stats
- /_count
- /_dashboards
- /_rank_eval

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to /_search/scroll with a message body must specify "Content-Length" in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in scroll_id values, use the request body, not the query string, to pass scroll_id values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic OpenSearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

Elasticsearch version 7.10

For Elasticsearch 7.10, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* /_forcemerge ,*/index-name* /update/*id*, and */index-name* /_close)
- /_alias
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_refresh
- /_reindex [1]
- /_render
- /_resolve/index
- /_rollover

- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
  - cluster.max_shards_per_node
- /_cluster/state
- /_cluster/stats
- /_count
- /_index_template [6]
- /_ingest/pipeline
- /_index_template
- /_ltr
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_opendistro/_alerting
- /_opendistro/_asynchronous_search
- /_opendistro/_anomaly_detection
- /_opendistro/_ism
- /_opendistro/_ppl
- /_opendistro/_security
- /_opendistro/_sql
- /_percolate
- /_plugin/kibana
- /_plugins/_replication
- /_rank_eval
- /_scripts [3]
- /_search[2]
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template [6]
- /_update_by_query [1]
- /_validate

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.

2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called "Other supported resources".
4. Refers to the PUT method. For information about the GET method, see the section called "Notable API differences". This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called "Shrink".
6. Legacy index templates (`_template`) were replaced by composable templates (`_index_template`) starting with Elasticsearch 7.8. Composable templates take precedence over legacy templates. If no composable template matches a given index, a legacy template can still match and be applied. The `_template` operation still works on OpenSearch and later versions of Elasticsearch OSS, but GET calls to the two template types return different results.

Elasticsearch version 7.9

For Elasticsearch 7.9, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* `/_forcemerge` , */index-name* `/update/`*id*, and */index-name* `/_close`)
- `/_alias`
- `/_aliases`
- `/_all`
- `/_analyze`
- `/_bulk`
- `/_cat` (except `/_cat/nodeattrs`)
- `/_cluster/allocation/explain`
- `/_delete_by_query` [1]
- `/_explain`
- `/_field_caps`
- `/_field_stats`
- `/_flush`
- `/_index_template` [6]
- `/_ingest/pipeline`
- `/_ltr`
- `/_mapping`
- `/_mget`
- `/_msearch`
- `/_mtermvectors`
- `/_nodes`
- `/_refresh`
- `/_reindex` [1]
- `/_render`
- `/_resolve/index`
- `/_rollover`
- `/_scripts` [3]
- `/_search`[2]
- `/_search profile`
- `/_shard_stores`
- `/_shrink`[5]
- `/_snapshot`
- `/_split`
- `/_stats`

- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
  - cluster.max_shards_per_node
- /_cluster/state
- /_cluster/stats
- /_count
- /_opendistro/_alerting
- /_opendistro/_anomaly_detection
- /_opendistro/_ism
- /_opendistro/_ppl
- /_opendistro/_security
- /_opendistro/_sql
- /_percolate
- /_plugin/kibana
- /_rank_eval
- /_status
- /_tasks
- /_template [6]
- /_update_by_query [1]
- /_validate

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to /_search/scroll with a message body must specify "Content-Length" in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in scroll_id values, use the request body, not the query string, to pass scroll_id values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic OpenSearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

6. Legacy index templates (`_template`) were replaced by composable templates (`_index_template`) starting with Elasticsearch 7.8. Composable templates take precedence over legacy templates. If no composable template matches a given index, a legacy template can still match and be applied. The `_template` operation still works on OpenSearch and later versions of Elasticsearch OSS, but GET calls to the two template types return different results.

Elasticsearch version 7.8

For Elasticsearch 7.8, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* / `_forcemerge` , */index-name* /`update/`*id*, and */index-name* /`_close`)
- `/_alias`
- `/_aliases`
- `/_all`
- `/_analyze`
- `/_bulk`
- `/_cat` (except `/_cat/nodeattrs`)
- `/_cluster/allocation/explain`
- `/_cluster/health`
- `/_cluster/pending_tasks`
- `/_cluster/settings` for several properties[4]:
  - `action.auto_create_index`
  - `action.search.shard_count.limit`
- `/_cluster/state`
- `/_cluster/stats`
- `/_count`
- `/_delete_by_query` [1]
- `/_explain`
- `/_field_caps`
- `/_field_stats`
- `/_flush`
- `/_index_template` [6]
- `/_ingest/pipeline`
- `/_ltr`
- `/_mapping`
- `/_mget`
- `/_msearch`
- `/_mtermvectors`
- `/_nodes`
- `/_opendistro/_alerting`
- `/_opendistro/_anomaly_detection`
- `/_opendistro/_ism`
- `/_refresh`
- `/_reindex` [1]
- `/_render`
- `/_rollover`
- `/_scripts` [3]
- `/_search`[2]
- `/_search profile`
- `/_shard_stores`
- `/_shrink`[5]
- `/_snapshot`
- `/_split`
- `/_stats`
- `/_status`
- `/_tasks`
- `/_template` [6]
- `/_update_by_query` [1]
- `/_validate`

- `indices.breaker.fielddata.limit`
- `indices.breaker.request.limit`
- `indices.breaker.total.limit`
- `cluster.max_shards_per_node`

- `/_opendistro/_security`
- `/_opendistro/_sql`
- `/_percolate`
- `/_plugin/kibana`
- `/_rank_eval`

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the `/_tasks` operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.
6. Legacy index templates (`_template`) were replaced by composable templates (`_index_template`) starting with Elasticsearch 7.8. Composable templates take precedence over legacy templates. If no composable template matches a given index, a legacy template can still match and be applied. The `_template` operation still works on OpenSearch and later versions of Elasticsearch OSS, but GET calls to the two template types return different results.

Elasticsearch version 7.7

For Elasticsearch 7.7, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* /

- `/_cluster/state`

- `/_refresh`

_forcemerge ,/*index-name* /update/*id*, and /*index-name* /_close)
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
  - cluster.max_shards_per_node
- /_cluster/stats
- /_count
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_ltr
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_opendistro/_alerting
- /_opendistro/_anomaly_detection
- /_opendistro/_ism
- /_opendistro/_security
- /_opendistro/_sql
- /_percolate
- /_plugin/kibana
- /_rank_eval
- /_reindex [1]
- /_render
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.

2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

Elasticsearch version 7.4

For Elasticsearch 7.4, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* `/_forcemerge` , */index-name* `/update/`*id*, and */index-name* `/_close`)
- `/_alias`
- `/_aliases`
- `/_all`
- `/_analyze`
- `/_bulk`
- `/_cat` (except `/_cat/nodeattrs`)
- `/_cluster/allocation/explain`
- `/_cluster/health`
- `/_cluster/pending_tasks`
- `/_cluster/settings` for several properties[4]:
- `/_cluster/state`
- `/_cluster/stats`
- `/_count`
- `/_delete_by_query` [1]
- `/_explain`
- `/_field_caps`
- `/_field_stats`
- `/_flush`
- `/_ingest/pipeline`
- `/_mapping`
- `/_mget`
- `/_msearch`
- `/_mtermvectors`
- `/_nodes`
- `/_opendistro/_alerting`
- `/_refresh`
- `/_reindex` [1]
- `/_render`
- `/_rollover`
- `/_scripts` [3]
- `/_search`[2]
- `/_search profile`
- `/_shard_stores`
- `/_shrink`[5]
- `/_snapshot`
- `/_split`
- `/_stats`
- `/_status`
- `/_tasks`
- `/_template`
- `/_update_by_query` [1]
- `/_validate`

- `action.auto_create_index`
- `action.search.shard_count.limit`
- `indices.breaker.fielddata.limit`
- `indices.breaker.request.limit`
- `indices.breaker.total.limit`
- `cluster.max_shards_per_node`

- `/_opendistro/_anomaly_detection`
- `/_opendistro/_ism`
- `/_opendistro/_security`
- `/_opendistro/_sql`
- `/_percolate`
- `/_plugin/kibana`
- `/_rank_eval`

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the `/_tasks` operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

Elasticsearch version 7.1

For Elasticsearch 7.1, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* /

- `/_cluster/state`
- `/_cluster/stats`

- `/_refresh`
- `/_reindex` [1]

_forcemerge and */index-name* /update/*id*) **except** */index-name* /_close
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
  - cluster.max_shards_per_node
- /_count
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_opendistro/_alerting
- /_opendistro/_ism
- /_opendistro/_security
- /_opendistro/_sql
- /_percolate
- /_plugin/kibana
- /_rank_eval
- /_render
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.

2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called "Other supported resources".
4. Refers to the PUT method. For information about the GET method, see the section called "Notable API differences". This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called "Shrink".

Elasticsearch version 6.8

For Elasticsearch 6.8, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* `/_forcemerge` and */index-name* `/update/`*id*) **except** */index-name* `/_close`
- `/_alias`
- `/_aliases`
- `/_all`
- `/_analyze`
- `/_bulk`
- `/_cat` (except `/_cat/nodeattrs`)
- `/_cluster/allocation/explain`
- `/_cluster/health`
- `/_cluster/pending_tasks`
- `/_cluster/settings` for several properties[4]:
- `/_cluster/state`
- `/_cluster/stats`
- `/_count`
- `/_delete_by_query` [1]
- `/_explain`
- `/_field_caps`
- `/_field_stats`
- `/_flush`
- `/_ingest/pipeline`
- `/_mapping`
- `/_mget`
- `/_msearch`
- `/_mtermvectors`
- `/_nodes`
- `/_opendistro/_alerting`
- `/_opendistro/_ism`
- `/_refresh`
- `/_reindex` [1]
- `/_render`
- `/_rollover`
- `/_scripts` [3]
- `/_search`[2]
- `/_search profile`
- `/_shard_stores`
- `/_shrink`[5]
- `/_snapshot`
- `/_split`
- `/_stats`
- `/_status`
- `/_tasks`
- `/_template`
- `/_update_by_query` [1]
- `/_validate`

- `action.auto_create_index`
- `action.search.shard_count.limit`
- `indices.breaker.fielddata.limit`
- `indices.breaker.request.limit`
- `indices.breaker.total.limit`
- `cluster.max_shards_per_node`
- `cluster.blocks.read_only`
- `/_opendistro/_security`
- `/_opendistro/_sql`
- `/_percolate`
- `/_plugin/kibana`
- `/_rank_eval`

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the `/_tasks` operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

Elasticsearch version 6.7

For Elasticsearch 6.7, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* / _forcemerge and */index-name* /update/*id*) **except** */index-name* /_close
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
  - cluster.max_shards_per_node
- /_cluster/state
- /_cluster/stats
- /_count
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_opendistro/_alerting
- /_opendistro/_security
- /_opendistro/_sql
- /_percolate
- /_plugin/kibana
- /_rank_eval
- /_refresh
- /_reindex [1]
- /_render
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the `/_tasks` operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the `PUT` method. For information about the `GET` method, see the section called “Notable API differences”. This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

Elasticsearch version 6.5

For Elasticsearch 6.5, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* `/_forcemerge` and */index-name* `/update/`*id*) **except** */index-name* `/_close`
- `/_alias`
- `/_aliases`
- `/_all`
- `/_analyze`
- `/_bulk`
- `/_cat` (except `/_cat/nodeattrs`)
- `/_cluster/allocation/explain`
- `/_cluster/health`
- `/_cluster/state`
- `/_cluster/stats`
- `/_count`
- `/_delete_by_query` [1]
- `/_explain`
- `/_field_caps`
- `/_field_stats`
- `/_flush`
- `/_ingest/pipeline`
- `/_mapping`
- `/_mget`
- `/_msearch`
- `/_mtermvectors`
- `/_nodes`
- `/_refresh`
- `/_reindex` [1]
- `/_render`
- `/_rollover`
- `/_scripts` [3]
- `/_search`[2]
- `/_search profile`
- `/_shard_stores`
- `/_shrink`[5]
- `/_snapshot`
- `/_split`
- `/_stats`
- `/_status`
- `/_tasks`

- `/_cluster/pending_tasks`
- `/_cluster/settings` for several properties[4]:
  - `action.auto_create_index`
  - `action.search.shard_count.limit`
  - `indices.breaker.fielddata.limit`
  - `indices.breaker.request.limit`
  - `indices.breaker.total.limit`
- `/_opendistro/_alerting`
- `/_opendistro/_sql`
- `/_percolate`
- `/_plugin/kibana`
- `/_rank_eval`
- `/_template`
- `/_update_by_query` [1]
- `/_validate`

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the `/_tasks` operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

Elasticsearch version 6.4

For Elasticsearch 6.4, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* / _forcemerge and */index-name* /update/*id*) **except** */index-name* /_close
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
- /_cluster/state
- /_cluster/stats
- /_count
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_opendistro/_alerting
- /_percolate
- /_plugin/kibana
- /_rank_eval
- /_refresh
- /_reindex [1]
- /_render
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.

2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called "Other supported resources".
4. Refers to the PUT method. For information about the GET method, see the section called "Notable API differences". This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called "Shrink".

Elasticsearch version 6.3

For Elasticsearch 6.3, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* `/_forcemerge` and */index-name* `/update/`*id*) **except** */index-name* `/_close`
- `/_alias`
- `/_aliases`
- `/_all`
- `/_analyze`
- `/_bulk`
- `/_cat` (except `/_cat/nodeattrs`)
- `/_cluster/allocation/explain`
- `/_cluster/health`
- `/_cluster/pending_tasks`
- `/_cluster/settings` for several properties[4]:
- `/_cluster/state`
- `/_cluster/stats`
- `/_count`
- `/_delete_by_query` [1]
- `/_explain`
- `/_field_caps`
- `/_field_stats`
- `/_flush`
- `/_ingest/pipeline`
- `/_mapping`
- `/_mget`
- `/_msearch`
- `/_mtermvectors`
- `/_nodes`
- `/_opendistro/_alerting`
- `/_percolate`
- `/_refresh`
- `/_reindex` [1]
- `/_render`
- `/_rollover`
- `/_scripts` [3]
- `/_search`[2]
- `/_search profile`
- `/_shard_stores`
- `/_shrink`[5]
- `/_snapshot`
- `/_split`
- `/_stats`
- `/_status`
- `/_tasks`
- `/_template`
- `/_update_by_query` [1]
- `/_validate`

- `action.auto_create_index`
- `action.search.shard_count.limit`
- `indices.breaker.fielddata.limit`
- `indices.breaker.request.limit`
- `indices.breaker.total.limit`

- `/_plugin/kibana`
- `/_rank_eval`

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the `/_tasks` operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

Elasticsearch version 6.2

For Elasticsearch 6.2, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* `/_forcemerge` and */index-name* `/update/`*id*) **except** */index-name* `/_close`

- `/_cluster/state`
- `/_cluster/stats`
- `/_count`
- `/_delete_by_query` [1]

- `/_refresh`
- `/_reindex` [1]
- `/_render`
- `/_rollover`

- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_opendistro/_alerting
- /_percolate
- /_plugin/kibana
- /_rank_eval
- /_scripts [3]
- /_search[2]
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_split
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to /_search/scroll with a message body must specify "Content-Length" in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in scroll_id values, use the request body, not the query string, to pass scroll_id values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.

4. Refers to the PUT method. For information about the GET method, see the section called "Notable API differences". This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called "Shrink".

Elasticsearch version 6.0

For Elasticsearch 6.0, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* /_forcemerge and */index-name* /update/*id*) **except** */index-name* /_close
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
- /_cluster/state
- /_cluster/stats
- /_count
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_percolate
- /_plugin/kibana
- /_refresh
- /_reindex [1]
- /_render
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

- `indices.breaker.fielddata.limit`
- `indices.breaker.request.limit`
- `indices.breaker.total.limit`

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the `/_tasks` operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the `PUT` method. For information about the `GET` method, see the section called “Notable API differences”. This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

Elasticsearch version 5.6

For Elasticsearch 5.6, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* `/_forcemerge` and */index-name* `/update/`*id*) **except** */index-name* `/_close`
- `/_alias`
- `/_aliases`
- `/_all`
- `/_cluster/state`
- `/_cluster/stats`
- `/_count`
- `/_delete_by_query` [1]
- `/_explain`
- `/_field_caps`
- `/_field_stats`
- `/_render`
- `/_rollover`
- `/_scripts` [3]
- `/_search` [2]
- `/_search profile`
- `/_shard_stores`
- `/_shrink` [5]

- `/_analyze`
- `/_bulk`
- `/_cat` (except `/_cat/nodeattrs`)
- `/_cluster/allocation/explain`
- `/_cluster/health`
- `/_cluster/pending_tasks`
- `/_cluster/settings` for several properties[4]:
  - `action.auto_create_index`
  - `action.search.shard_count.limit`
  - `indices.breaker.fielddata.limit`
  - `indices.breaker.request.limit`
  - `indices.breaker.total.limit`
- `/_flush`
- `/_ingest/pipeline`
- `/_mapping`
- `/_mget`
- `/_msearch`
- `/_mtermvectors`
- `/_nodes`
- `/_percolate`
- `/_plugin/kibana`
- `/_refresh`
- `/_reindex` [1]
- `/_snapshot`
- `/_stats`
- `/_status`
- `/_tasks`
- `/_template`
- `/_update_by_query` [1]
- `/_validate`

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the `/_tasks` operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.

5. See the section called “Shrink”.

Elasticsearch version 5.5

For Elasticsearch 5.5, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* /_forcemerge and */index-name* /update/*id*) **except** */index-name* /_close
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties[4]:
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
- /_cluster/state
- /_cluster/stats
- /_count
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_percolate
- /_plugin/kibana
- /_refresh
- /_reindex [1]
- /_render
- /_rollover
- /_scripts [3]
- /_search[2]
- /_search profile
- /_shard_stores
- /_shrink[5]
- /_snapshot
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

- indices.breaker.total.limit

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to /_search/scroll with a message body must specify "Content-Length" in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in scroll_id values, use the request body, not the query string, to pass scroll_id values to OpenSearch Service.
3. For considerations about using scripts, see the section called “Other supported resources”.
4. Refers to the PUT method. For information about the GET method, see the section called “Notable API differences”. This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
5. See the section called “Shrink”.

Elasticsearch version 5.3

For Elasticsearch 5.3, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* /_forcemerge and */index-name* /update/*id*) **except** */index-name* /_close
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/state
- /_cluster/stats
- /_count
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_mapping
- /_mget
- /_render
- /_rollover
- /_search[2]
- /_search profile
- /_shard_stores
- /_shrink[4]
- /_snapshot
- /_stats
- /_status
- /_tasks
- /_template

- `/_cluster/allocation/explain`
- `/_cluster/health`
- `/_cluster/pending_tasks`
- `/_cluster/settings` for several properties[3]:
  - `action.auto_create_index`
  - `action.search.shard_count.limit`
  - `indices.breaker.fielddata.limit`
  - `indices.breaker.request.limit`
  - `indices.breaker.total.limit`
- `/_msearch`
- `/_mtermvectors`
- `/_nodes`
- `/_percolate`
- `/_plugin/kibana`
- `/_refresh`
- `/_reindex` [1]
- `/_update_by_query` [1]
- `/_validate`

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the `/_tasks` operation along with these operations to verify that the requests completed successfully.
2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. Refers to the PUT method. For information about the GET method, see the section called "Notable API differences". This list only refers to the generic Elasticsearch operations that OpenSearch Service supports and does not include plugin-specific supported operations for anomaly detection, ISM, and so on.
4. See the section called "Shrink".

Elasticsearch version 5.1

For Elasticsearch 5.1, OpenSearch Service supports the following operations.

- All operations in the index path (such as */index-name* /_forcemerge and */index-name* /update/*id*) **except** */index-name* /_close
- /_alias
- /_aliases
- /_all
- /_analyze
- /_bulk
- /_cat (except /_cat/nodeattrs)
- /_cluster/allocation/explain
- /_cluster/health
- /_cluster/pending_tasks
- /_cluster/settings for several properties (PUT only):
  - action.auto_create_index
  - action.search.shard_count.limit
  - indices.breaker.fielddata.limit
  - indices.breaker.request.limit
  - indices.breaker.total.limit
- /_cluster/state
- /_cluster/stats
- /_count
- /_delete_by_query [1]
- /_explain
- /_field_caps
- /_field_stats
- /_flush
- /_ingest/pipeline
- /_mapping
- /_mget
- /_msearch
- /_mtermvectors
- /_nodes
- /_percolate
- /_plugin/kibana
- /_refresh
- /_reindex [1]
- /_render
- /_rollover
- /_search[2]
- /_search profile
- /_shard_stores
- /_shrink[3]
- /_snapshot
- /_stats
- /_status
- /_tasks
- /_template
- /_update_by_query [1]
- /_validate

1. Cluster configuration changes might interrupt these operations before completion. We recommend that you use the /_tasks operation along with these operations to verify that the requests completed successfully.

2. DELETE requests to `/_search/scroll` with a message body must specify `"Content-Length"` in the HTTP header. Most clients add this header by default. To avoid a problem with = characters in `scroll_id` values, use the request body, not the query string, to pass `scroll_id` values to OpenSearch Service.
3. See the section called “Shrink”.

Elasticsearch version 2.3

For Elasticsearch 2.3, OpenSearch Service supports the following operations.

- All operations in the index path (such as *`/index-name`* `/_forcemerge` and *`/index-name`* `/_recovery`) **except** *`/index-name`* `/_close`
- `/_alias`
- `/_aliases`
- `/_all`
- `/_analyze`
- `/_bulk`
- `/_cache/clear` (index only)
- `/_cat` (except `/_cat/nodeattrs`)
- `/_cluster/health`
- `/_cluster/settings` for several properties (PUT only):
  - `indices.breaker.fielddata.limit`
  - `indices.breaker.request.limit`
  - `indices.breaker.total.limit`
  - `threadpool.get.queue_size`
  - `threadpool.bulk.queue_size`
  - `threadpool.index.queue_size`
  - `threadpool.percolate.queue_size`
- `/_cluster/stats`
- `/_count`
- `/_flush`
- `/_mapping`
- `/_mget`
- `/_msearch`
- `/_nodes`
- `/_percolate`
- `/_plugin/kibana`
- `/_refresh`
- `/_render`
- `/_search`
- `/_snapshot`
- `/_stats`
- `/_status`
- `/_template`

- `threadpool.search.queue_size`
- `threadpool.suggest.queue_size`

Elasticsearch version 1.5

For Elasticsearch 1.5, OpenSearch Service supports the following operations.

- All operations in the index path, such as `/index-name /_optimize` and `/index-name /_warmer`, **except** `/index-name /_close`
- `/_alias`
- `/_aliases`
- `/_all`
- `/_analyze`
- `/_bulk`
- `/_cat`
- `/_cluster/health`
- `/_cluster/settings` for several properties (PUT only):
  - `indices.breaker.fielddata.limit`
  - `indices.breaker.request.limit`
  - `indices.breaker.total.limit`
  - `threadpool.get.queue_size`
  - `threadpool.bulk.queue_size`
  - `threadpool.index.queue_size`
  - `threadpool.percolate.queue_size`
  - `threadpool.search.queue_size`
  - `threadpool.suggest.queue_size`
- `/_cluster/stats`
- `/_count`
- `/_flush`
- `/_mapping`
- `/_mget`
- `/_msearch`
- `/_nodes`
- `/_percolate`
- `/_plugin/kibana`
- `/_plugin/kibana3`
- `/_plugin/migration`
- `/_refresh`
- `/_search`
- `/_snapshot`
- `/_stats`
- `/_status`
- `/_template`

Amazon OpenSearch Service quotas

Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Unless otherwise noted, each quota is Region-specific.

To view the quotas for OpenSearch Service domains and instances, Amazon OpenSearch Serverless, and Amazon OpenSearch Ingestion, see Amazon OpenSearch Service quotas in the *AWS General Reference*.

To view the quotas for OpenSearch Service in the AWS Management Console, open the Service Quotas console. In the navigation pane, choose **AWS services** and select **Amazon OpenSearch Service**. To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*.

UltraWarm storage quotas

The following table lists the UltraWarm instance types and the maximum amount of storage that each type can use. For more information about UltraWarm, see the section called "UltraWarm storage".

Instance type	Maximum storage
`ultrawarm1.medium.search`	1.5 TiB
`ultrawarm1.large.search`	20 TiB

Number of data nodes per AZ

The following table lists the total number of data nodes for AZ deployment is below, the overall limit signifies the number of data nodes per limit including both the hot and warm node count. storage that each type can use.

AZ Configuration	Hot Node Count Limit	Warm Node Count Limit	Overall Limit (Hot + Warm)
1 - AZ	334	250	334
2 - AZ	668	500	668

AZ Configuration	Hot Node Count Limit	Warm Node Count Limit	Overall Limit (Hot + Warm)
3 - AZ	1002	750	1002

Total node limit by instance family

The following table lists the total node limit by instance family.

Instance family	ElasticSearch OpenSearch up to 2.15	OpenSearch 2.17 and above	Default limit
T2	10	10	10
T3	10	10	10
M3, C4, M4, R4, C5, M5, R5, I2, I3	10	200	80
Graviton 2, Gravtion 3	200	400	80
C7, R7i, M7i, i4i	200	400	80
OR1.medium.search and OR1.large.search	200	400	80
OR1.xlarge.search and above	200	1002	80
Ultrawarm1	150	750	150

EBS volume size quotas

The following table shows the minimum and maximum sizes for EBS volumes for each instance type that OpenSearch Service supports. For information about which instance types include instance storage and additional hardware details, see Amazon OpenSearch Service pricing.

- If you choose magnetic storage under **EBS volume type** when creating your domain, the maximum volume size is 100 GiB for all instance types except t2.small and t2.medium, and

all Graviton instances (M6g, C6g, R6g, and R6gd), which don't support magnetic storage. For the maximum sizes listed in the following table, choose one of the SSD options.

- Some older-generation instance types include instance storage, but also support EBS storage. If you choose EBS storage for one of these instance types, the storage volumes are *not* additive. You can use either an EBS volume or the instance storage, not both.

Instance type	Minimum EBS size	Maximum EBS size (gp2)	Maximum EBS size (gp3)
t2.micro.search	10 GiB	35 GiB	N/A
t2.small.search	10 GiB	35 GiB	N/A
t2.medium.search	10 GiB	35 GiB	N/A
t3.small.search	10 GiB	100 GiB	100 GiB
t3.medium.search	10 GiB	200 GiB	200 GiB
m3.medium.search	10 GiB	100 GiB	N/A
m3.large.search	10 GiB	512 GiB	N/A
m3.xlarge.search	10 GiB	512 GiB	N/A
m3.2xlarge.search	10 GiB	512 GiB	N/A
m4.large.search	10 GiB	512 GiB	N/A
m4.xlarge.search	10 GiB	1 TiB	N/A
m4.2xlarge.search	10 GiB	1.5 TiB	N/A
m4.4xlarge.search	10 GiB	1.5 TiB	N/A
m4.10xlarge.search	10 GiB	1.5 TiB	N/A
m5.large.search	10 GiB	512 GiB	1 TiB

Instance type	Minimum EBS size	Maximum EBS size (gp2)	Maximum EBS size (gp3)
m5.xlarge.search	10 GiB	1 TiB	2 TiB
m5.2xlarge.search	10 GiB	1.5 TiB	3 TiB
m5.4xlarge.search	10 GiB	3 TiB	6 TiB
m5.12xlarge.search	10 GiB	9 TiB	18 TiB
m6g.large.search	10 GiB	512 GiB	1 TiB
m6g.xlarge.search	10 GiB	1 TiB	2 TiB
m6g.2xlarge.search	10 GiB	1.5 TiB	3 TiB
m6g.4xlarge.search	10 GiB	3 TiB	6 TiB
m6g.8xlarge.search	10 GiB	6 TiB	12 TiB
m6g.12xlarge.search	10 GiB	9 TiB	18 TiB
c4.large.search	10 GiB	100 GiB	N/A
c4.xlarge.search	10 GiB	512 GiB	N/A
c4.2xlarge.search	10 GiB	1 TiB	N/A
c4.4xlarge.search	10 GiB	1.5 TiB	N/A
c4.8xlarge.search	10 GiB	1.5 TiB	N/A
c5.large.search	10 GiB	256 GiB	256 GiB
c5.xlarge.search	10 GiB	512 GiB	512 GiB
c5.2xlarge.search	10 GiB	1 TiB	1 TiB
c5.4xlarge.search	10 GiB	1.5 TiB	1.5 TiB

Instance type	Minimum EBS size	Maximum EBS size (gp2)	Maximum EBS size (gp3)
c5.9xlarge.search	10 GiB	3.5 TiB	3.5 TiB
c5.18xlarge.search	10 GiB	7 TiB	7 TiB
c6g.large.search	10 GiB	256 GiB	256 GiB
c6g.xlarge.search	10 GiB	512 GiB	512 GiB
c6g.2xlarge.search	10 GiB	1 TiB	1 TiB
c6g.4xlarge.search	10 GiB	1.5 TiB	1.5 TiB
c6g.8xlarge.search	10 GiB	3 TiB	3 TiB
c6g.12xlarge.search	10 GiB	4.5 TiB	4.5 TiB
r3.large.search	10 GiB	512 GiB	N/A
r3.xlarge.search	10 GiB	512 GiB	N/A
r3.2xlarge.search	10 GiB	512 GiB	N/A
r3.4xlarge.search	10 GiB	512 GiB	N/A
r3.8xlarge.search	10 GiB	512 GiB	N/A
r4.large.search	10 GiB	1 TiB	N/A
r4.xlarge.search	10 GiB	1.5 TiB	N/A
r4.2xlarge.search	10 GiB	1.5 TiB	N/A
r4.4xlarge.search	10 GiB	1.5 TiB	N/A
r4.8xlarge.search	10 GiB	1.5 TiB	N/A
r4.16xlarge.search	10 GiB	1.5 TiB	N/A

Instance type	Minimum EBS size	Maximum EBS size (gp2)	Maximum EBS size (gp3)
r5.large.search	10 GiB	1 TiB	2 TiB
r5.xlarge.search	10 GiB	1.5 TiB	3 TiB
r5.2xlarge.search	10 GiB	3 TiB	6 TiB
r5.4xlarge.search	10 GiB	6 TiB	12 TiB
r5.12xlarge.search	10 GiB	12 TiB	24 TiB
r6g.large.search	10 GiB	1 TiB	2 TiB
r6g.xlarge.search	10 GiB	1.5 TiB	3 TiB
r6g.2xlarge.search	10 GiB	3 TiB	6 TiB
r6g.4xlarge.search	10 GiB	6 TiB	12 TiB
r6g.8xlarge.search	10 GiB	8 TiB	16 TiB
r6g.12xlarge.search	10 GiB	12 TiB	24 TiB
r6gd.large.search	N/A	N/A	N/A
r6gd.xlarge.search	N/A	N/A	N/A
r6gd.2xlarge.search	N/A	N/A	N/A
r6gd.4xlarge.search	N/A	N/A	N/A
r6gd.8xlarge.search	N/A	N/A	N/A
r6gd.12xlarge.search	N/A	N/A	N/A
r6gd.16xlarge.search	N/A	N/A	N/A
i2.xlarge.search	10 GiB	512 GiB	N/A

Instance type	Minimum EBS size	Maximum EBS size (gp2)	Maximum EBS size (gp3)
i2.2xlarge.search	10 GiB	512 GiB	N/A
i3.large.search	N/A	N/A	N/A
i3.xlarge.search	N/A	N/A	N/A
i3.2xlarge.search	N/A	N/A	N/A
i3.4xlarge.search	N/A	N/A	N/A
i3.8xlarge.search	N/A	N/A	N/A
i3.16xlarge.search	N/A	N/A	N/A
or1.medium.search	20 GiB	N/A	768 GiB
or1.large.search	20 GiB	N/A	1532 GiB
or1.xlarge.search	20 GiB	N/A	3 TiB
or1.2xlarge.search	20 GiB	N/A	6 TiB
or1.4xlarge.search	20 GiB	N/A	12 TiB
or1.8xlarge.search	20 GiB	N/A	16 TiB
or1.12xlarge.search	20 GiB	N/A	24 TiB
or1.16xlarge.search	20 GiB	N/A	36 TiB
im4gn.large.search	N/A	N/A	N/A
im4gn.xlarge.search	N/A	N/A	N/A
im4gn.2xlarge.search	N/A	N/A	N/A
im4gn.4xlarge.search	N/A	N/A	N/A

Instance type	Minimum EBS size	Maximum EBS size (gp2)	Maximum EBS size (gp3)
im4gn.8xlarge.search	N/A	N/A	N/A
im4gn.16xlarge.search	N/A	N/A	N/A
C7g.large.search	10 GiB	N/A	256 GiB
C7g.xlarge.search	10 GiB	N/A	512 GiB
C7g.2xlarge.search	10 GiB	N/A	1 TiB
C7g.4xlarge.search	10 GiB	N/A	1.5 TiB
C7g.8xlarge.search	10 GiB	N/A	3 TiB
C7g.12xlarge.search	10 GiB	N/A	4.5 TiB
C7g.16xlarge.search	10 GiB	N/A	6 TiB
M7g.medium.search	10 GiB	N/A	4 GiB
M7g.large.search	10 GiB	N/A	768 GiB
M7g.xlarge.search	10 GiB	N/A	2 TiB
M7g.2xlarge.search	10 GiB	N/A	3 TiB
M7g.4xlarge.search	10 GiB	N/A	6 TiB
M7g.8xlarge.search	10 GiB	N/A	12 TiB
M7g.12xlarge.search	10 GiB	N/A	18 TiB
M7g.16xlarge.search	10 GiB	N/A	24 TiB
R7g.medium.search	10 GiB	N/A	768 GiB
R7g.large.search	10 GiB	N/A	1.5 TiB

Instance type	Minimum EBS size	Maximum EBS size (gp2)	Maximum EBS size (gp3)
R7g.xlarge.search	10 GiB	N/A	3 TiB
R7g.2xlarge.search	10 GiB	N/A	6 TiB
R7g.4xlarge.search	10 GiB	N/A	12 TiB
R7g.8xlarge.search	10 GiB	N/A	16 TiB
R7g.12xlarge.search	10 GiB	N/A	24 TiB
R7g.16xlarge.search	10 GiB	N/A	36 TiB
R7gd.large.search	N/A	N/A	N/A
R7gd.xlarge.search	N/A	N/A	N/A
R7gd.2xlarge.search	N/A	N/A	N/A
R7gd.4xlarge.search	N/A	N/A	N/A
R7gd.8xlarge.search	N/A	N/A	N/A
R7gd.12xlarge.search	N/A	N/A	N/A
R7gd.16xlarge.search	N/A	N/A	N/A
i4i.large.search	10 GiB	N/A	N/A
i4i.xlarge.search	10 GiB	N/A	N/A
i4i.2xlarge.search	10 GiB	N/A	N/A
i4i.4xlarge.search	10 GiB	N/A	N/A
i4i.8xlarge.search	10 GiB	N/A	N/A
i4i.12xlarge.search	10 GiB	N/A	N/A

Instance type	Minimum EBS size	Maximum EBS size (gp2)	Maximum EBS size (gp3)
i4i.16xlarge.search	10 GiB	N/A	N/A
i4i.24xlarge.search	10 GiB	N/A	N/A
i4i.32xlarge.search	10 GiB	N/A	N/A
i4g.large.search	10 GiB	N/A	N/A
i4g.xlarge.search	10 GiB	N/A	N/A
i4g.2xlarge.search	10 GiB	N/A	N/A
i4g.4xlarge.search	10 GiB	N/A	N/A
i4g.8xlarge.search	10 GiB	N/A	N/A
i4g.16xlarge.search	10 GiB	N/A	N/A
c7i.large.search	10 GiB	N/A	256 GiB
c7i.xlarge.search	10 GiB	N/A	512 GiB
c7i.2xlarge.search	10 GiB	N/A	1 TiB
c7i.4xlarge.search	10 GiB	N/A	1.5 TiB
c7i.8xlarge.search	10 GiB	N/A	3 TiB
c7i.12xlarge.search	10 GiB	N/A	4.5 TiB
c7i.16xlarge.search	10 GiB	N/A	6 TiB
m7i.large.search	10 GiB	N/A	768 GiB
m7i.xlarge.search	10 GiB	N/A	2 TiB
m7i.2xlarge.search	10 GiB	N/A	3 TiB

Instance type	Minimum EBS size	Maximum EBS size (gp2)	Maximum EBS size (gp3)
m7i.4xlarge.search	10 GiB	N/A	6 TiB
m7i.8xlarge.search	10 GiB	N/A	12 TiB
m7i.12xlarge.search	10 GiB	N/A	18 TiB
m7i.16xlarge.search	10 GiB	N/A	24 TiB
r7i.large.search	10 GiB	N/A	1.5 TiB
r7i.xlarge.search	10 GiB	N/A	3 TiB
r7i.2xlarge.search	10 GiB	N/A	6 TiB
r7i.4xlarge.search	10 GiB	N/A	12 TiB
r7i.8xlarge.search	10 GiB	N/A	16 TiB
r7i.12xlarge.search	10 GiB	N/A	24 TiB
r7i.12xlarge.search	10 GiB	N/A	36 TiB

Network quotas

The following table shows the maximum size of HTTP request payloads.

Instance type	Maximum size of HTTP request payloads
`t2.micro.search`	10 MiB
`t2.small.search`	10 MiB
`t2.medium.search`	10 MiB
`t3.small.search`	10 MiB

Instance type	Maximum size of HTTP request payloads
t3.medium.search	10 MiB
m3.medium.search	10 MiB
m3.large.search	10 MiB
m3.xlarge.search	100 MiB
m3.2xlarge.search	100 MiB
m4.large.search	10 MiB
m4.xlarge.search	100 MiB
m4.2xlarge.search	100 MiB
m4.4xlarge.search	100 MiB
m4.10xlarge.search	100 MiB
m5.large.search	10 MiB
m5.xlarge.search	100 MiB
m5.2xlarge.search	100 MiB
m5.4xlarge.search	100 MiB
m5.12xlarge.search	100 MiB
m6g.large.search	10 MiB
m6g.xlarge.search	100 MiB
m6g.2xlarge.search	100 MiB

Instance type	Maximum size of HTTP request payloads
m6g.4xlarge.search	100 MiB
m6g.8xlarge.search	100 MiB
m6g.12xlarge.search	100 MiB
c4.large.search	10 MiB
c4.xlarge.search	100 MiB
c4.2xlarge.search	100 MiB
c4.4xlarge.search	100 MiB
c4.8xlarge.search	100 MiB
c5.large.search	10 MiB
c5.xlarge.search	100 MiB
c5.2xlarge.search	100 MiB
c5.4xlarge.search	100 MiB
c5.9xlarge.search	100 MiB
c5.18xlarge.search	100 MiB
c6g.large.search	10 MiB
c6g.xlarge.search	100 MiB
c6g.2xlarge.search	100 MiB

Instance type	Maximum size of HTTP request payloads
c6g.4xlarge.search	100 MiB
c6g.8xlarge.search	100 MiB
c6g.12xlarge.search	100 MiB
r3.large.search	10 MiB
r3.xlarge.search	100 MiB
r3.2xlarge.search	100 MiB
r3.4xlarge.search	100 MiB
r3.8xlarge.search	100 MiB
r4.large.search	100 MiB
r4.xlarge.search	100 MiB
r4.2xlarge.search	100 MiB
r4.4xlarge.search	100 MiB
r4.8xlarge.search	100 MiB
r4.16xlarge.search	100 MiB
r5.large.search	100 MiB
r5.xlarge.search	100 MiB
r5.2xlarge.search	100 MiB
r5.4xlarge.search	100 MiB

Instance type	Maximum size of HTTP request payloads
r5.12xlarge.search	100 MiB
r6g.large.search	100 MiB
r6g.xlarge.search	100 MiB
r6g.2xlarge.search	100 MiB
r6g.4xlarge.search	100 MiB
r6g.8xlarge.search	100 MiB
r6g.12xlarge.search	100 MiB
r6gd.large.search	100 MiB
r6gd.xlarge.search	100 MiB
r6gd.2xlarge.search	100 MiB
r6gd.4xlarge.search	100 MiB
r6gd.8xlarge.search	100 MiB
r6gd.12xlarge.search	100 MiB
r6gd.16xlarge.search	100 MiB

Instance type	Maximum size of HTTP request payloads
i2.xlarge.search	100 MiB
i2.2xlarge.search	100 MiB
i3.large.search	100 MiB
i3.xlarge.search	100 MiB
i3.2xlarge.search	100 MiB
i3.4xlarge.search	100 MiB
i3.8xlarge.search	100 MiB
i3.16xlarge.search	100 MiB
or1.medium.search	10 MiB
or1.large.search	100 MiB
or1.xlarge.search	100 MiB
or1.2xlarge.search	100 MiB
or1.4xlarge.search	100 MiB
or1.8xlarge.search	100 MiB
or1.12xlarge.search	100 MiB
or1.16xlarge.search	100 MiB

Instance type	Maximum size of HTTP request payloads
im4gn.large.search	100 MiB
im4gn.xlarge.search	100 MiB
im4gn.2xlarge.search	100 MiB
im4gn.4xlarge.search	100 MiB
im4gn.8xlarge.search	100 MiB
im4gn.16xlarge.search	100 MiB
i4i.large.search	100 MiB
i4i.xlarge.search	100 MiB
i4i.2xlarge.search	100 MiB
i4i.4xlarge.search	100 MiB
i4i.8xlarge.search	100 MiB
i4i.12xlarge.search	100 MiB
i4i.16xlarge.search	100 MiB

Instance type	Maximum size of HTTP request payloads
i4i.24xlarge.search	100 MiB
i4i.32xlarge.search	100 MiB
i4g.large.search	100 MiB
i4g.xlarge.search	100 MiB
i4g.2xlarge.search	100 MiB
i4g.4xlarge.search	100 MiB
i4g.8xlarge.search	100 MiB
i4g.16xlarge.search	100 MiB
c7i.large.search	100 MiB
c7i.xlarge.search	100 MiB
c7i.2xlarge.search	100 MiB
c7i.4xlarge.search	100 MiB
c7i.8xlarge.search	100 MiB
c7i.12xlarge.search	100 MiB

Instance type	Maximum size of HTTP request payloads
c7i.16xlarge.search	100 MiB
m7i.large.search	100 MiB
m7i.xlarge.search	100 MiB
m7i.2xlarge.search	100 MiB
m7i.4xlarge.search	100 MiB
m7i.8xlarge.search	100 MiB
m7i.12xlarge.search	100 MiB
m7i.16xlarge.search	100 MiB
r7i.large.search	100 MiB
r7i.xlarge.search	100 MiB
r7i.2xlarge.search	100 MiB
r7i.4xlarge.search	100 MiB
r7i.8xlarge.search	100 MiB
r7i.12xlarge.search	100 MiB

Instance type	Maximum size of HTTP request payloads
r7i.16xlarge.search	100 MiB

Shard size quotas

The following section lists the maximum shard sizes for various instance families.

Instance type	Multi-AZ without Standby	Multi-AZ with Standby
R5, C5, M5	N/A	65 GiB
I3	N/A	65 GiB
R6g, C6g, M6g, R6gd	N/A	65 GiB
OR1	100 GiB	65 GiB
Im4gn	N/A	65 GiB

To request a quota increase, contact AWS Support.

Shard count quotas

The following section lists the maximum shard count for OpenSearch versions.

Engine Version	Limit	Notes
Elasticsearch 1.5 to 6.x	No default limit	
Elasticsearch 7.x	1000	Default limit can be changed via cluster.max_shards_per_node setting.
OpenSearch 1.x to 2.15	1000	Default limit can be changed via cluster.max_shards_per_node setting.

Engine Version	Limit	Notes
OpenSearch 2.17 and above	1000 per every 16 GB of heap to a max of 4000	The default limit can't be changed.

Java process quota

OpenSearch Service limits Java processes to a heap size of 32 GiB. Advanced users can specify the percentage of the heap used for field data. For more information, see the section called "Advanced cluster settings" and the section called "JVM OutOfMemoryError".

Domain policy quota

OpenSearch Service limits access policies on domains to 100 KiB.

Reserved Instances in Amazon OpenSearch Service

Reserved Instances (RIs) in Amazon OpenSearch Service offer significant discounts compared to standard On-Demand Instances. The instances themselves are identical; RIs are just a billing discount applied to On-Demand Instances in your account. For long-lived applications with predictable usage, RIs can provide considerable savings over time.

OpenSearch Service RIs require one- or three-year terms and have three payment options that affect the discount rate:

- **No Upfront** – You pay nothing upfront. You pay a discounted hourly rate for every hour within the term.
- **Partial Upfront** – You pay a portion of the cost upfront, and you pay a discounted hourly rate for every hour within the term.
- **All Upfront** – You pay the entirety of the cost upfront. You don't pay an hourly rate for the term.

Generally speaking, a larger upfront payment means a larger discount. You can't cancel Reserved Instances—when you reserve them, you commit to paying for the entire term—and upfront payments are nonrefundable.

RIs are not flexible; they only apply to the exact instance type that you reserve. For example, a reservation for eight `c5.2xlarge.search` instances does not apply to sixteen

`c5.xlarge.search` instances or four `c5.4xlarge.search` instances. For full details, see Amazon OpenSearch Service pricing and FAQ.

Purchasing Reserved Instances (console)

The console lets you view your existing Reserved Instances and purchase new ones.

To purchase a reservation

1. Go to https://aws.amazon.com, and then choose **Sign In to the Console**.
2. Under **Analytics**, choose **Amazon OpenSearch Service**.
3. Choose **Reserved Instance Leases** from the navigation pane.

   On this page, you can view your existing reservations. If you have many reservations, you can filter them to more easily identify and view a particular reservation.

   > **Tip**
   >
   > If you don't see the **Reserved Instance Leases** link, create a domain in the AWS Region.

4. Choose **Order Reserved Instance**.
5. Provide a unique and descriptive name.
6. Choose an instance type and the number of instances. For guidance, see the section called "Sizing domains".
7. Choose a term length and payment option. Review the payment details carefully.
8. Choose **Next**.
9. Review the purchase summary carefully. Purchases of Reserved Instances are non-refundable.
10. Choose **Order**.

Purchasing Reserved Instances (AWS CLI)

The AWS CLI has commands for viewing offerings, purchasing a reservation, and viewing your reservations. The following command and sample response show the offerings for a given AWS Region:

```
aws opensearch describe-reserved-instance-offerings --region us-east-1
{
```

```
  "ReservedInstanceOfferings": [
    {
      "FixedPrice": x,
      "ReservedInstanceOfferingId": "1a2a3a4a5-1a2a-3a4a-5a6a-1a2a3a4a5a6a",
      "RecurringCharges": [
        {
          "RecurringChargeAmount": y,
          "RecurringChargeFrequency": "Hourly"
        }
      ],
      "UsagePrice": 0.0,
      "PaymentOption": "PARTIAL_UPFRONT",
      "Duration": 31536000,
      "InstanceType": "m4.2xlarge.search",
      "CurrencyCode": "USD"
    }
  ]
}
```

For an explanation of each return value, see the following table.

Field	Description
`FixedPrice`	The upfront cost of the reservation.
`ReservedInstanceOfferingId`	The offering ID. Make note of this value if you want to reserve the offering.
`RecurringCharges`	The hourly rate for the reservation.
`UsagePrice`	A legacy field. For OpenSearch Service, this value is always 0.
`PaymentOption`	No Upfront, Partial Upfront, or All Upfront.
`Duration`	Length of the term in seconds: • 31536000 seconds is one year. • 94608000 seconds is three years.
`InstanceType`	The instance type for the reservation. For information about the hardware resources

Field	Description
	that are allocated to each instance type, see Amazon OpenSearch Service pricing.
`CurrencyCode`	The currency for `FixedPrice` and `RecurringChargeAmount`.

This next example purchases a reservation:

```
aws opensearch purchase-reserved-instance-offering --reserved-instance-offering-
id 1a2a3a4a5-1a2a-3a4a-5a6a-1a2a3a4a5a6a --reservation-name my-reservation --instance-
count 3 --region us-east-1
{
  "ReservationName": "my-reservation",
  "ReservedInstanceId": "9a8a7a6a-5a4a-3a2a-1a0a-9a8a7a6a5a4a"
}
```

Finally, you can list your reservations for a given Region using the following example:

```
aws opensearch describe-reserved-instances --region us-east-1
{
  "ReservedInstances": [
    {
      "FixedPrice": x,
      "ReservedInstanceOfferingId": "1a2a3a4a5-1a2a-3a4a-5a6a-1a2a3a4a5a6a",
      "ReservationName": "my-reservation",
      "PaymentOption": "PARTIAL_UPFRONT",
      "UsagePrice": 0.0,
      "ReservedInstanceId": "9a8a7a6a-5a4a-3a2a-1a0a-9a8a7a6a5a4a",
      "RecurringCharges": [
        {
          "RecurringChargeAmount": y,
          "RecurringChargeFrequency": "Hourly"
        }
      ],
      "State": "payment-pending",
      "StartTime": 1522872571.229,
      "InstanceCount": 3,
      "Duration": 31536000,
      "InstanceType": "m4.2xlarge.search",
```

```
      "CurrencyCode": "USD"
    }
  ]
}
```

> **Note**
>
> `StartTime` is Unix epoch time, which is the number of seconds that have passed since midnight UTC of 1 January 1970. For example, 1522872571 epoch time is 20:09:31 UTC of 4 April 2018. You can use online converters.

To learn more about the commands used in the preceding examples, see the AWS CLI Command Reference.

Purchasing Reserved Instances (AWS SDKs)

The AWS SDKs (except the Android and iOS SDKs) support all the operations that are defined in the Amazon OpenSearch Service API Reference, including the following:

- `DescribeReservedInstanceOfferings`
- `PurchaseReservedInstanceOffering`
- `DescribeReservedInstances`

This sample script uses the OpenSearchService low-level Python client from the AWS SDK for Python (Boto3) to purchase Reserved Instances. You must provide a value for `instance_type`.

```
import boto3
from botocore.config import Config

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default region.

my_config = Config(
    # Optionally lets you specify a region other than your default.
    region_name='us-east-1'
)

client = boto3.client('opensearch', config=my_config)
```

```
instance_type = '' # e.g. m4.2xlarge.search


def describe_RI_offerings(client):
    """Gets the Reserved Instance offerings for this account"""

    response = client.describe_reserved_instance_offerings()
    offerings = (response['ReservedInstanceOfferings'])
    return offerings


def check_instance(offering):
    """Returns True if instance type is the one you specified above"""

    if offering['InstanceType'] == instance_type:
        return True

    return False


def get_instance_id():
    """Iterates through the available offerings to find the ID of the one you
 specified"""

    instance_type_iterator = filter(
        check_instance, describe_RI_offerings(client))
    offering = list(instance_type_iterator)
    id = offering[0]['ReservedInstanceOfferingId']
    return id


def purchase_RI_offering(client):
    """Purchase Reserved Instances"""

    response = client.purchase_reserved_instance_offering(
        ReservedInstanceOfferingId = get_instance_id(),
        ReservationName = 'my-reservation',
        InstanceCount = 1
    )
    print('Purchased reserved instance offering of type ' + instance_type)
    print(response)
```

```
def main():
    """Purchase Reserved Instances"""
    purchase_RI_offering(client)
```

For more information about installing and using the AWS SDKs, see AWS Software Development Kits.

Examining costs

Cost Explorer is a free tool that you can use to view your spending data for the past 13 months. Analyzing this data helps you identify trends and understand if RIs fit your use case. If you already have RIs, you can group by **Purchase Option** and show amortized costs to compare that spending to your spending for On-Demand Instances. You can also set usage budgets to make sure you are taking full advantage of your reservations. For more information, see Analyzing Your Costs with Cost Explorer in the *AWS Billing User Guide*.

Other supported resources in Amazon OpenSearch Service

This topic describes additional resources that Amazon OpenSearch Service supports.

bootstrap.memory_lock

OpenSearch Service enables `bootstrap.memory_lock` in `opensearch.yml`, which locks JVM memory and prevents the operating system from swapping it to disk. This applies to all supported instance types except for the following:

- `t2.micro.search`
- `t2.small.search`
- `t2.medium.search`
- `t3.small.search`
- `t3.medium.search`

Scripting module

OpenSearch Service supports scripting for Elasticsearch 5.*x* and later domains. It does not support scripting for 1.5 or 2.3.

Supported scripting options include the following:

- Painless

- Lucene Expressions
- Mustache

For Elasticsearch 5.5 and later domains, and all OpenSearch domains, OpenSearch Service supports stored scripts using the `_scripts` endpoint. Elasticsearch 5.3 and 5.1 domains support inline scripts only.

TLS transport

OpenSearch Service supports HTTP on port 80 and HTTPS over port 443, but does not support TLS transport.

Custom plugins

Custom plugins for Amazon OpenSearch Service is a new plugin management option that extends OpenSearch functionality in areas like language analysis, custom filtering, and ranking, enabling you to craft personalized search experiences. Custom plugins for OpenSearch can be developed by extending the `org.opensearch.plugins.Plugin` class and then packaged as a .zip file. The following plugin extensions are currently supported by Amazon OpenSearch Service:

- Analysis plugin: Extends analysis functionality by adding custom analyzers, character tokenizers, or filters for text processing.
- Search plugin: Enhances search capabilities with custom query types, similarity algorithms, suggest options, and aggregations.

You can use the Amazon OpenSearch Service console or existing APIs for custom packages to upload and associate a plugin with your Amazon OpenSearch Service domain, for more information on custom packages, see Custom packages for Amazon OpenSearch Service. Additonally, you can use `DescribePackages` to describe all the packages in your account to view details such as what OpenSearch version is currently in use or error details. Amazon OpenSearch Service validates plugin package for version compatibility, security vulnerabilities, and permitted plugin operations.

Custom plugins are supported on OpenSearch Service domains running OpenSearch version 2.15 or later, and are available in 14 regions globally: US West (Oregon), US East (Ohio), US East (N. Virginia), South America (Sao Paulo), Europe(Paris), Europe (London), Europe (Ireland), Europe

(Frankfurt), Canada (Central), Asia Pacific (Tokyo), Asia Pacific (Sydney), Asia Pacific (Singapore), Asia Pacific (Seoul) and Asia Pacific (Mumbai).

> **Note**
>
> Custom plugins contain user developed code. Any issues, including SLA breaches, caused by user developed code will not be eligible for SLA credits. For more information, see Amazon OpenSearch Service SLA exclusions under Amazon OpenSearch Service - Service Level Agreement.

Plugin limits

You can create up to 25 custom plugins per account. Maximum number of plugins that can be associated with a single domain is 20, this number is inclusive of all plugin types i.e. optional, third party or custom. Maximum allowed uncompressed size for a plugin is 1 GB.

The following table lists the features that are not available when using custom plugins:

Amazon OpenSearch Service features	Custom plugins
Cross cluster search	Not supported.
Cross cluster replication	Not supported
Remote-reindex	Not supported
Auto-tune	Not supported
Multi-AZ with Standby	Not supported
Centralized OpenSearch user Interface	Not supported

Using custom plugins with OpenSearch Service

Prerequisites for using custom plugins with OpenSearch Service

Before you can use custom plugins with Amazon OpenSearch Service you will need to be sure you have the following set up:

- Node to node encryption
- Encryption of data at rest
- EnforceHTTPS set to `true`
- Clients must support TLSSecurityPolicy '`Policy-Min-TLS-1-2-PFS-2023-10`' , you can set this up with following command:

```
aws opensearch update-domain-config –domain-name domain-name –domain-endpoint-options
 '{"TLSSecurityPolicy":"Policy-Min-TLS-1-2-PFS-2023-10" }'
```

  For more information, see DomainEndpointOptions.
- The descriptor.properties file for your plugin's supported engine version should be similar to 2.15.0 or the 2.x.0.i.e patch version should be zero.

Installing custom plugins with AWS CLI

You can install custom plugins using the AWS CLI. Before you can associate a custom plugin with your domain, you must upload it to an Amazon S3 bucket. You must create the Amazon S3 bucket in the same region you intend to use the plugin. For instructions on how to do this, see Uploading objects in the What is Amazon S3 guide. If your plugin contains sensitive information, select server-side encryption with S3-managed keys when you upload it. After you upload the file, make note of its Amazon S3 path. See the following example Amazon S3 path format:

```
s3://bucket-name/file-path/file-name
```

You will need to create a new package for your custom plugin. You can do this using the existing CreatePackage API. When creating your new package, please update the bucket and key location to point to your custom plugin's .zip file in the calling account's Amazon S3 bucket. Note that, your Amazon S3 bucket must be in the same region as the package being created. Only .zip files are supported for `ZIP-PLUGIN` packages. The contents of the .zip file must follow directory structure as expected by the plugin. To create a package, see the following example:

```
aws opensearch --region $REGION create-package --package-name <package-name> --package-
type ZIP-PLUGIN --package-source S3BucketName=<bucket>,S3Key=<key> --engine-version
 OpenSearch_2.15
```

You can view the status of the create package operation, including any validation and security vulnerability findings errors by using the describe-packages. To do this, see the following example:

```
aws opensearch --region $REGION describe-packages --filters '[{"Name":
 "PackageType","Value": ["ZIP-PLUGIN"]}, {"Name": "PackageName","Value": ["<package-
name>"]}]'
```

The following is a sample response of the describe-packages API:

```
{ "PackageDetailsList": [ {
"PackageID": "pkg-identifier",
"PackageName": "custom-plugin-test",
"PackageType": "ZIP-PLUGIN",
"PackageStatus": "VALIDATION_FAILED",
"CreatedAt": "2024-11-11T13:07:18.297000-08:00",
"LastUpdatedAt": "2024-11-11T13:10:13.843000-08:00",
"ErrorDetails":
{ "ErrorType": "", "ErrorMessage":
"PluginValidationFailureReason : Dependency Scan reported 3 vulnerabilities for the
 plugin: CVE-2022-23307, CVE-2019-17571, CVE-2022-23305" },
"EngineVersion": "OpenSearch_2.15",
"AllowListedUserList": [],
"PackageOwner": "OWNER-XXXX" } ] }
```

Note

During the create package operation, Amazon OpenSearch Service checks the `ZIP-PLUGIN` for version compatibility, supported plugin extensions and security vulnerabilities. In particular the security vulnerabilities are scanned using the Amazon Inspector service. The results of these checks are shown in the `ErrorDetails` field of the API response.

Use the AssociatePackage API to associate the plugin with your Amazon OpenSearch Service domain using the package id of package created in the previous step. If you have multiple plugins, you can use the AssociatePackages API to associate multiple packages to a domain in a single operation. To do this, see the following example:

```
aws opensearch --region $REGION associate-package --domain-name <domain-name> --
package-id <package-id>
```

> **Note**
>
> Plugins are installed and uninstalled using a blue/green deployment process.

You can use the ListPackagesForDomain API to see the status of the association. The association status will change as the workflow progresses from `ASSOCIATING` to `ACTIVE`. The association status changes to `ACTIVE` once the plugin installation workflow is complete and your plugin is ready to be used. To do this, see the following example:

```
aws opensearch --region $REGION list-packages-for-domain
      --domain-name <domain-name>
```

Updating custom plugins

You can update custom plugins using the existing UpdatePackage API. You can use the following associate-packages API example to apply package updates to a domain. To do this, see the following example:

```
aws opensearch --region $REGION update-package --package-id <package-id>  --package-
source S3BucketName=<bucket>,S3Key=<key> --package-description <description>
```

> **Note**
>
> You can audit create, update, associate and disassociate operations on your plugin using AWS CloudTrail. For more information refer to the documentation Monitoring Amazon OpenSearch Service API calls with AWS CloudTrail.

Upgrading your domain with custom plugins

To upgrade your Amazon OpenSearch Service domain that has associated custom plugins to a later version of OpenSearch, you can use the CreatePackage API to create a new package for your plugin.

> **Note**
>
> Please ensure that the package name is the same for the plugin for all engine versions. Changing the package name will cause the upgrade domain process to fail during the blue/green deployment.

For instructions on upgrading your Amazon OpenSearch Service domain, see Upgrading Amazon OpenSearch Service . Amazon OpenSearch Service will disassociate the previous version of your plugin package and install the new version of the plugin through a blue/green deployment.

Encrypting custom plugins

When using the CreatePackage API , you can set the `PackageEncryptionOptions` to `true` and pass the KMS key ARN that you would like to use for encryption. To do this, see the following example:

```
aws opensearch --region $REGION create-package --package-name <package-name> --package-
type ZIP-PLUGIN --package-source S3BucketName=<bucket>,S3Key=<key> --engine-version
 OpenSearch_2.15
"PackageConfigOptions": {
     ...
  }
  "PackageEncryptionOptions": {
    "Enabled": true,
    "KmsKeyId":"kms_key_arn"
  }
```

You can enable the same option while updating the package by using the UpdatePackage API.

> **Note**
>
> If a KMS key key is disabled or deleted, it can leave the cluster in-operational.

Uninstalling custom plugins

You can uninstall custom plugins using the existing DissociatePackage API to uninstall a plugin from a domain. This step also removes any related configuration and/or license packages

associated with the plugin. You can use the existing ListPackagesForDomain API to see the status of the dissociation. Additionally, you can also use the DissociatePackages API to uninstall multiple plugins from a domain in a single operation.

You can use the following disassociate-packages API example to apply package updates to a domain. To do this, see the following example:

```
aws opensearch --region $REGION dissociate-package --package-id <plugin-package-id> --
domain-name <domain-name>
```

Note

In order to uninstall a plugin, you will first need to disable the plugin from every index before dissociating the plugin package. If you try to uninstall a plugin without disabling it from every index the blue/green deployment process will get stuck in the processing state.

Amazon OpenSearch Service tutorials

This chapter includes several start-to-finish tutorials for working with Amazon OpenSearch Service, including how to migrate to the service, build a simple search application, and create a visualization in OpenSearch Dashboards.

Topics

Tutorial: Creating and searching for documents in Amazon OpenSearch Service

In this tutorial, you learn how to create and search for a document in Amazon OpenSearch Service. You add data to an index in the form of a JSON document. OpenSearch Service creates an index around the first document that you add.

This tutorial explains how to make HTTP requests to create documents, automatically generate an ID for a document, and perform basic and advanced searches on your documents.

> **Note**
>
> This tutorial uses a domain with open access. For the highest level of security, we recommend that you put your domain inside a virtual private cloud (VPC).

Prerequisites

This tutorial has the following prerequisites:

- You must have an AWS account.
- You must have an active OpenSearch Service domain.

Adding a document to an index

To add a document to an index, you can use any HTTP tool, such as Postman, cURL, or the OpenSearch Dashboards console. These examples assume that you're using the developer console in OpenSearch Dashboards. If you're using a different tool, adjust accordingly by providing the full URL and credentials, if necessary.

To add a document to an index

1. Navigate to the OpenSearch Dashboards URL for your domain. You can find the URL on the domain's dashboard in the OpenSearch Service console. The URL follows this format:

   ```
   domain-endpoint/_dashboards/
   ```

2. Sign in using your primary username and password.
3. Open the left navigation panel and choose **Dev Tools**.
4. The HTTP verb for creating a new resource is PUT, which is what you use to create a new document and index. Enter the following command in the console:

   ```
   PUT fruit/_doc/1
   {
     "name":"strawberry",
     "color":"red"
   }
   ```

   The PUT request creates an index named *fruit* and adds a single document to the index with an ID of 1. It produces the following response:

   ```
   {
     "_index" : "fruit",
     "_type" : "_doc",
     "_id" : "1",
     "_version" : 1,
     "result" : "created",
     "_shards" : {
       "total" : 2,
       "successful" : 2,
       "failed" : 0
     },
     "_seq_no" : 0,
   ```

```
  "_primary_term" : 1
}
```

Creating automatically generated IDs

OpenSearch Service can automatically generate an ID for your documents. The command to generate IDs uses a POST request instead of a PUT request, and it requires no document ID (in comparison to the previous request).

Enter the following request in the developer console:

```
POST veggies/_doc
{
  "name":"beet",
  "color":"red",
  "classification":"root"
}
```

This request creates an index named *veggies* and adds the document to the index. It produces the following response:

```
{
  "_index" : "veggies",
  "_type" : "_doc",
  "_id" : "3WgyS4IB5DLqbRIvLxtF",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 2,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

Note that addditional `_id` field in the response, which indicates that an ID was automatically created.

> **Note**
>
> You don't provide anything after _doc in the URL, where the ID normally goes. Because you're creating a document with a generated ID, you don't provide one yet. That's reserved for updates.

Updating a document with a POST command

To update a document, you use an HTTP POST command with the ID number.

First, create a document with an ID of 42:

```
POST fruits/_doc/42
{
  "name":"banana",
  "color":"yellow"
}
```

Then use that ID to update the document:

```
POST fruits/_doc/42
{
  "name":"banana",
  "color":"yellow",
  "classification":"berries"
}
```

This command updates the document with the new field `classification`. It produces the following response:

```
{
  "_index" : "fruits",
  "_type" : "_doc",
  "_id" : "42",
  "_version" : 2,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 2,
```

```
    "failed" : 0
  },
  "_seq_no" : 1,
  "_primary_term" : 1
}
```

> **Note**
>
> If you try to update a document that does not exist, OpenSearch Service creates the document.

Performing bulk actions

You can use the `POST _bulk` API operation to perform multiple actions on one or more indexes in one request. Bulk action commands take the following format:

```
POST /_bulk
<action_meta>\n
<action_data>\n
<action_meta>\n
<action_data>\n
```

Each action requires two lines of JSON. First, you provide the action description or metadata. On the next line, you provide the data. Each part is separated by a newline (\n). An action description for an insert might look like this:

```
{ "create" : { "_index" : "veggies", "_type" : "_doc", "_id" : "7" } }
```

And the next line containing the data might look like this:

```
{ "name":"kale", "color":"green", "classification":"leafy-green" }
```

Taken together, the metadata and the data represent a single action in a bulk operation. You can perform many operations in one request, like this:

```
POST /_bulk
{ "create" : { "_index" : "veggies", "_id" : "35" } }
```

```
{ "name":"kale", "color":"green", "classification":"leafy-green" }
{ "create" : { "_index" : "veggies", "_id" : "36" } }
{ "name":"spinach", "color":"green", "classification":"leafy-green" }
{ "create" : { "_index" : "veggies", "_id" : "37" } }
{ "name":"arugula", "color":"green", "classification":"leafy-green" }
{ "create" : { "_index" : "veggies", "_id" : "38" } }
{ "name":"endive", "color":"green", "classification":"leafy-green" }
{ "create" : { "_index" : "veggies", "_id" : "39" } }
{ "name":"lettuce", "color":"green", "classification":"leafy-green" }
{ "delete" : { "_index" : "vegetables", "_id" : "1" } }
```

Notice that the last action is a `delete`. There's no data following the `delete` action.

Searching for documents

Now that data exists in your cluster, you can search for it. For example, you might want to search for all root vegetables, or get a count of all leafy greens, or find the number of errors logged per hour.

Basic searches

A basic search looks something like this:

```
GET veggies/_search?q=name:l*
```

The request produces a JSON response that contains the lettuce document.

Advanced searches

You can perform more advanced searches by providing the query options as JSON in the request body:

```
GET veggies/_search
{
  "query": {
    "term": {
      "name": "lettuce"
    }
  }
}
```

This example also produces a JSON response with the lettuce document.

Sorting

You can perform more of this type of query using sorting. First, you need to recreate the index, because the automatic field mapping chose types that can't be sorted by default. Send the following requests to delete and recreate the index:

```
DELETE /veggies

PUT /veggies
{
   "mappings":{
      "properties":{
         "name":{
            "type":"keyword"
         },
         "color":{
            "type":"keyword"
         },
         "classification":{
            "type":"keyword"
         }
      }
   }
}
```

Then repopulate the index with data:

```
POST /_bulk
{ "create" : { "_index" : "veggies", "_id" : "7"  } }
{ "name":"kale", "color":"green", "classification":"leafy-green" }
{ "create" : { "_index" : "veggies", "_id" : "8" } }
{ "name":"spinach", "color":"green", "classification":"leafy-green" }
{ "create" : { "_index" : "veggies", "_id" : "9" } }
{ "name":"arugula", "color":"green", "classification":"leafy-green" }
{ "create" : { "_index" : "veggies", "_id" : "10" } }
{ "name":"endive", "color":"green", "classification":"leafy-green" }
{ "create" : { "_index" : "veggies", "_id" : "11" } }
{ "name":"lettuce", "color":"green", "classification":"leafy-green" }
```

Now you can search with a sort. This request adds an ascending sort by the classification:

```
GET veggies/_search
{
  "query" : {
    "term": { "color": "green" }
  },
  "sort" : [
      "classification"
  ]
}
```

Related resources

For more information, see the following resources:

- *Getting started*
- *Indexing data*
- *Searching data*

Tutorial: Migrating to Amazon OpenSearch Service

Index snapshots are a popular way to migrate from a self-managed OpenSearch or legacy Elasticsearch cluster to Amazon OpenSearch Service. Broadly, the process consists of the following steps:

1. Take a snapshot of the existing cluster, and upload the snapshot to an Amazon S3 bucket.
2. Create an OpenSearch Service domain.
3. Give OpenSearch Service permissions to access the bucket, and ensure you have permissions to work with snapshots.
4. Restore the snapshot on the OpenSearch Service domain.

This walkthrough provides more detailed steps and alternate options, where applicable.

Take and upload the snapshot

Although you can use the repository-s3 plugin to take snapshots directly to S3, you have to install the plugin on every node, tweak `opensearch.yml` (or `elasticsearch.yml` if using an

Elasticsearch cluster), restart each node, add your AWS credentials, and finally take the snapshot. The plugin is a great option for ongoing use or for migrating larger clusters.

For smaller clusters, a one-time approach is to take a shared file system snapshot and then use the AWS CLI to upload it to S3. If you already have a snapshot, skip to step 4.

To take a snapshot and upload it to Amazon S3

1. Add the `path.repo` setting to `opensearch.yml` (or `Elasticsearch.yml`) on all nodes, and then restart each node.

```
path.repo: ["/my/shared/directory/snapshots"]
```

2. Register a snapshot repository, which is required before you take a snapshot. A repository is just a storage location: a shared file system, Amazon S3, Hadoop Distributed File System (HDFS), etc. In this case, we'll use a shared file system ("fs"):

```
PUT _snapshot/my-snapshot-repo-name
{
  "type": "fs",
  "settings": {
    "location": "/my/shared/directory/snapshots"
  }
}
```

3. Take the snapshot:

```
PUT _snapshot/my-snapshot-repo-name/my-snapshot-name
{
  "indices": "migration-index1,migration-index2,other-indices-*",
  "include_global_state": false
}
```

4. Install the AWS CLI, and run `aws configure` to add your credentials.

5. Navigate to the snapshot directory. Then run the following commands to create a new S3 bucket and upload the contents of the snapshot directory to that bucket:

```
aws s3 mb s3://bucket-name --region us-west-2
aws s3 sync . s3://bucket-name --sse AES256
```

Depending on the size of the snapshot and the speed of your internet connection, this operation can take a while.

Create a domain

Although the console is the easiest way to create a domain, in this case, you already have the terminal open and the AWS CLI installed. Modify the following command to create a domain that fits your needs:

```
aws opensearch create-domain \
  --domain-name migration-domain \
  --engine-version OpenSearch_1.0 \
  --cluster-config InstanceType=c5.large.search,InstanceCount=2 \
  --ebs-options EBSEnabled=true,VolumeType=gp2,VolumeSize=100 \
  --node-to-node-encryption-options Enabled=true \
  --encryption-at-rest-options Enabled=true \
  --domain-endpoint-options EnforceHTTPS=true,TLSSecurityPolicy=Policy-Min-
TLS-1-2-2019-07 \
  --advanced-security-options
 Enabled=true,InternalUserDatabaseEnabled=true,MasterUserOptions='{MasterUserName=master-
user,MasterUserPassword=master-user-password}' \
  --access-policies '{"Version":"2012-10-17","Statement":
[{"Effect":"Allow","Principal":{"AWS":["*"]},"Action":
["es:ESHttp*"],"Resource":"arn:aws:es:us-west-2:123456789012:domain/migration-domain/
*"}]}' \
  --region us-west-2
```

As is, the command creates an internet-accessible domain with two data nodes, each with 100 GiB of storage. It also enables fine-grained access control with HTTP basic authentication and all encryption settings. Use the OpenSearch Service console if you need a more advanced security configuration, such as a VPC.

Before issuing the command, change the domain name, master user credentials, and account number. Specify the same AWS Region that you used for the S3 bucket and an OpenSearch/Elasticsearch version that is compatible with your snapshot.

Important

Snapshots are only forward-compatible, and only by one major version. For example, you can't restore a snapshot from an OpenSearch 1.*x* cluster on an Elasticsearch 7.*x* cluster, only

an OpenSearch 1.*x* or 2.*x* cluster. Minor version matters, too. You can't restore a snapshot from a self-managed 5.3.3 cluster on a 5.3.2 OpenSearch Service domain. We recommend choosing the most recent version of OpenSearch or Elasticsearch that your snapshot supports. For a table of compatible versions, see the section called "Using a snapshot to migrate data".

Provide permissions to the S3 bucket

In the AWS Identity and Access Management (IAM) console, create a role with the following permissions and trust relationship. When creating the role, choose **S3** as the **AWS Service**. Name the role `OpenSearchSnapshotRole` so it's easy to find.

Permissions

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Action": [
        "s3:ListBucket"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::bucket-name"
      ]
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::bucket-name/*"
      ]
    }
  ]
}
```

Trust relationship

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Principal": {
        "Service": "es.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Then give your personal IAM role permissions to assume `OpenSearchSnapshotRole`. Create the following policy and attach it to your identity:

Permissions

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::123456789012:role/OpenSearchSnapshotRole"
    }
  ]
}
```

Map the snapshot role in OpenSearch Dashboards (if using fine-grained access control)

If you enabled fine-grained access control, even if you use HTTP basic authentication for all other purposes, you need to map the `manage_snapshots` role to your IAM role so you can work with snapshots.

To give your identity permissions to work with snapshots

1. Log in to Dashboards using the master user credentials you specified when you created the OpenSearch Service domain. You can find the Dashboards URL in the OpenSearch Service console. It takes the form of `https://domain-endpoint/_dashboards/`.
2. From the main menu choose **Security**, **Roles**, and select the **manage_snapshots** role.

3. Choose **Mapped users**, **Manage mapping**.
4. Add the domain ARN of your personal IAM role in the appropriate field. The ARN takes one of the following formats:

```
arn:aws:iam::123456789123:user/user-name
```

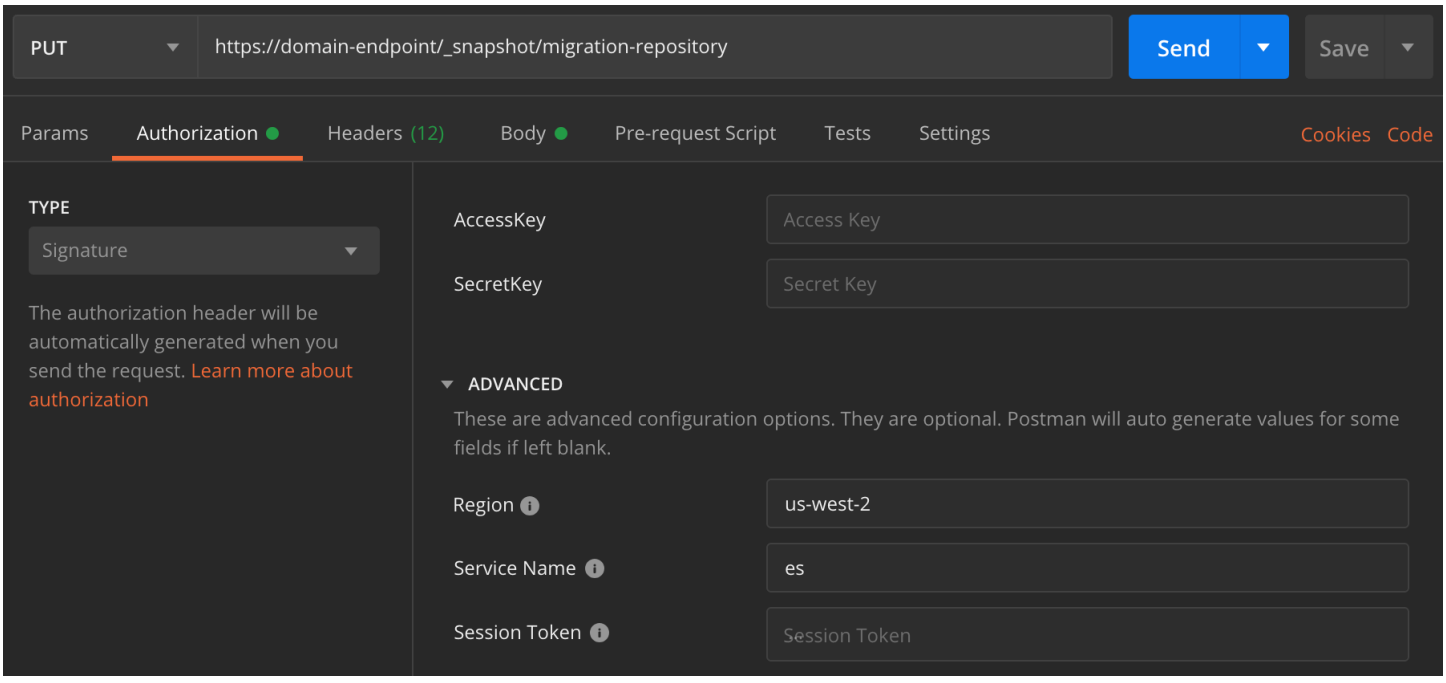
```
arn:aws:iam::123456789123:role/role-name
```

5. Select **Map** and confirm role shows up under **Mapped users**.

Restore the snapshot

At this point, you have two ways to access your OpenSearch Service domain: HTTP basic authentication with your master user credentials or AWS authentication using your IAM credentials. Because snapshots use Amazon S3, which has no concept of the master user, you must use your IAM credentials to register the snapshot repository with your OpenSearch Service domain.

Most programming languages have libraries to assist with signing requests, but the simpler approach is to use a tool like Postman and put your IAM credentials into the **Authorization** section.



To restore the snapshot

1. Regardless of how you choose to sign your requests, the first step is to register the repository:

```
PUT _snapshot/my-snapshot-repo-name
{
  "type": "s3",
  "settings": {
    "bucket": "bucket-name",
    "region": "us-west-2",
    "role_arn": "arn:aws:iam::123456789012:role/OpenSearchSnapshotRole"
  }
}
```

2. Then list the snapshots in the repository, and find the one you want to restore. At this point, you can continue using Postman or switch to a tool like curl.

   Shorthand

   ```
   GET _snapshot/my-snapshot-repo-name/_all
   ```

   curl

   ```
   curl -XGET -u 'master-user:master-user-password' https://domain-endpoint/
   _snapshot/my-snapshot-repo-name/_all
   ```

3. Restore the snapshot.

   Shorthand

   ```
   POST _snapshot/my-snapshot-repo-name/my-snapshot-name/_restore
   {
     "indices": "migration-index1,migration-index2,other-indices-*",
     "include_global_state": false
   }
   ```

   curl

   ```
   curl -XPOST -u 'master-user:master-user-password' https://domain-endpoint/
   _snapshot/my-snapshot-repo-name/my-snapshot-name/_restore \
     -H 'Content-Type: application/json' \
     -d '{"indices":"migration-index1,migration-index2,other-indices-
   *","include_global_state":false}'
   ```

4. Finally, verify that your indexes restored as expected.

Shorthand

```
GET _cat/indices?v
```

curl

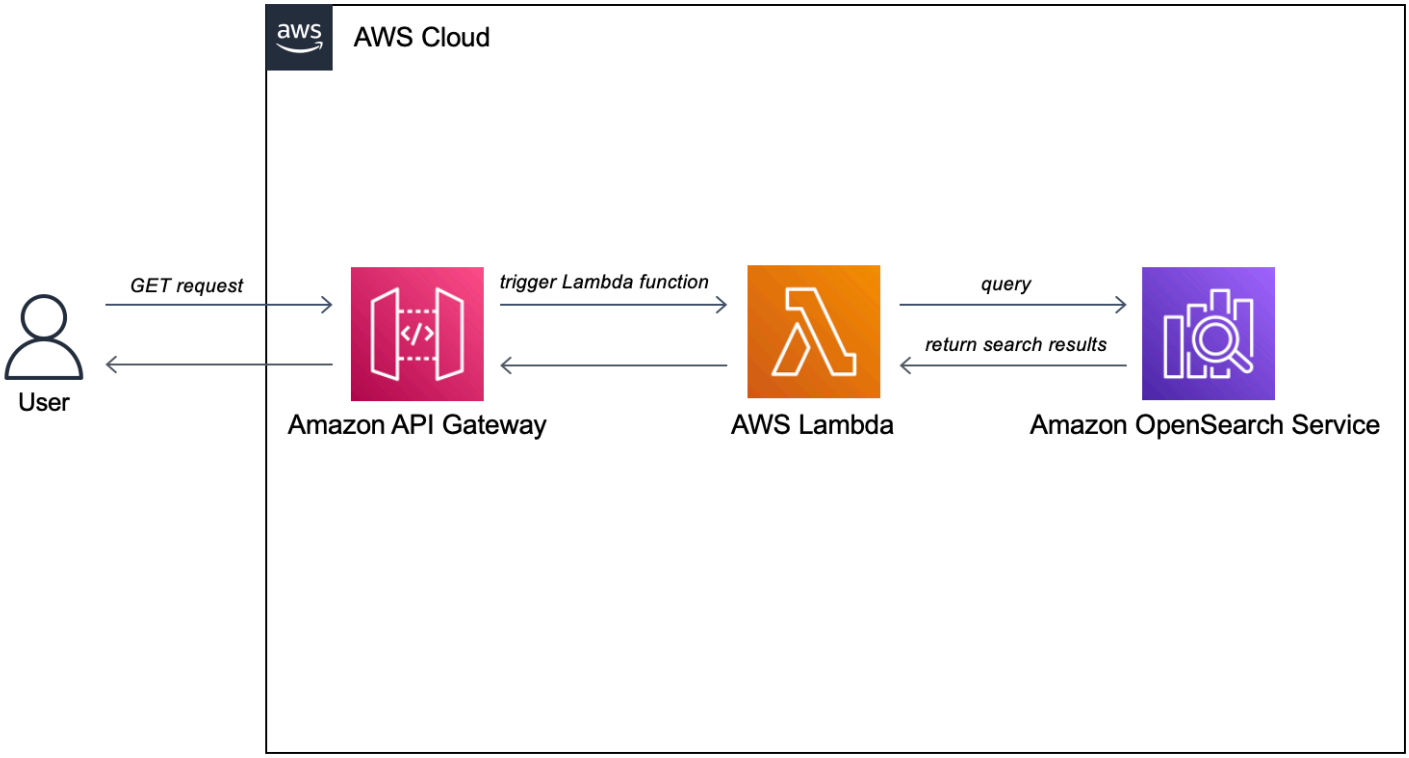
```
curl -XGET -u 'master-user:master-user-password' https://domain-endpoint/_cat/
indices?v
```

At this point, the migration is complete. You might configure your clients to use the new OpenSearch Service endpoint, resize the domain to suit your workload, check the shard count for your indexes, switch to an IAM master user, or start building visualizations in OpenSearch Dashboards.

Tutorial: Creating a search application with Amazon OpenSearch Service

A common way to create a search application with Amazon OpenSearch Service is to use web forms to send user queries to a server. Then you can authorize the server to call the OpenSearch APIs directly and have the server send requests to OpenSearch Service. However, if you want to write client-side code that doesn't rely on a server, you should compensate for the security and performance risks. Allowing unsigned, public access to the OpenSearch APIs is inadvisable. Users might access unsecured endpoints or impact cluster performance through overly broad queries (or too many queries).

This chapter presents a solution: use Amazon API Gateway to restrict users to a subset of the OpenSearch APIs and AWS Lambda to sign requests from API Gateway to OpenSearch Service.



Note

Standard API Gateway and Lambda pricing applies, but within the limited usage of this tutorial, costs should be negligible.

Prerequisites

A prerequisite for this tutorial is an OpenSearch Service domain. If you don't already have one, follow the steps in Create an OpenSearch Service domain to create one.

Step 1: Index sample data

Download sample-movies.zip, unzip it, and then use the _bulk API operation to add the 5,000 documents to the `movies` index:

```
POST https://search-my-domain.us-west-1.es.amazonaws.com/_bulk
{ "index": { "_index": "movies", "_id": "tt1979320" } }
{"directors":["Ron
 Howard"],"release_date":"2013-09-02T00:00:00Z","rating":8.3,"genres":
["Action","Biography","Drama","Sport"],"image_url":"http://ia.media-imdb.com/images/
```

```
M/MV5BMTQyMDE0MTY0OV5BMl5BanBnXkFtZTcwMjI2OTI0OQ@@._V1_SX400_.jpg","plot":"A re-
creation of the merciless 1970s rivalry between Formula One rivals James Hunt and
 Niki Lauda.","title":"Rush","rank":2,"running_time_secs":7380,"actors":["Daniel
 Brühl","Chris Hemsworth","Olivia Wilde"],"year":2013,"id":"tt1979320","type":"add"}
{ "index": { "_index": "movies", "_id": "tt1951264" } }
{"directors":["Francis Lawrence"],"release_date":"2013-11-11T00:00:00Z","genres":
["Action","Adventure","Sci-Fi","Thriller"],"image_url":"http://ia.media-imdb.com/
images/M/
MV5BMTAyMjQ3OTAxMzNeQTJeQWpwZ15BbWU4MDU0NzA1MzAx._V1_SX400_.jpg","plot":"Katniss
 Everdeen and Peeta Mellark become targets of the Capitol after
 their victory in the 74th Hunger Games sparks a rebellion in
 the Districts of Panem.","title":"The Hunger Games: Catching
 Fire","rank":4,"running_time_secs":8760,"actors":["Jennifer Lawrence","Josh
 Hutcherson","Liam Hemsworth"],"year":2013,"id":"tt1951264","type":"add"}
...
```

Note that the above is an example command with a small subset of the available data. To perform the `_bulk` operation, you need to copy and paste the entire contents of the `sample-movies` file. For futher instructions, see the section called “Option 2: Upload multiple documents”.

You can also use the following curl command to achieve the same result:

```
curl -XPOST -u 'master-user:master-user-password' 'domain-endpoint/_bulk' --data-binary
 @bulk_movies.json -H 'Content-Type: application/json'
```

Step 2: Create and deploy the Lambda function

Before you create your API in API Gateway, create the Lambda function that it passes requests to.

Create the Lambda function

In this solution, API Gateway passes requests to a Lambda function, which queries OpenSearch Service and returns results. Because this sample function uses external libraries, you need to create a deployment package and upload it to Lambda.

To create the deployment package

1. Open a command prompt and create a `my-opensearch-function` project directory. For example, on macOS:

   ```
   mkdir my-opensearch-function
   ```

2. Navigate to the `my-sourcecode-function` project directory.

```
cd my-opensearch-function
```

3. Copy the contents of the following sample Python code and save it in a new file named `opensearch-lambda.py`. Add your Region and host endpoint to the file.

```
import boto3
import json
import requests
from requests_aws4auth import AWS4Auth

region = '' # For example, us-west-1
service = 'es'
credentials = boto3.Session().get_credentials()
awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region, service,
 session_token=credentials.token)

host = '' # The OpenSearch domain endpoint with https:// and without a trailing
 slash
index = 'movies'
url = host + '/' + index + '/_search'

# Lambda execution starts here
def lambda_handler(event, context):

    # Put the user query into the query DSL for more accurate search results.
    # Note that certain fields are boosted (^).
    query = {
        "size": 25,
        "query": {
            "multi_match": {
                "query": event['queryStringParameters']['q'],
                "fields": ["title^4", "plot^2", "actors", "directors"]
            }
        }
    }

    # Elasticsearch 6.x requires an explicit Content-Type header
    headers = { "Content-Type": "application/json" }

    # Make the signed HTTP request
    r = requests.get(url, auth=awsauth, headers=headers, data=json.dumps(query))
```

```
    # Create the response and add some extra content to support CORS
    response = {
        "statusCode": 200,
        "headers": {
            "Access-Control-Allow-Origin": '*'
        },
        "isBase64Encoded": False
    }

    # Add the search results to the response
    response['body'] = r.text
    return response
```

4. Install the external libraries to a new `package` directory.

```
pip3 install --target ./package boto3
pip3 install --target ./package requests
pip3 install --target ./package requests_aws4auth
```

5. Create a deployment package with the installed library at the root. The following command generates a `my-deployment-package.zip` file in your project directory.

```
cd package
zip -r ../my-deployment-package.zip .
```

6. Add the `opensearch-lambda.py` file to the root of the zip file.

```
cd ..
zip my-deployment-package.zip opensearch-lambda.py
```

For more information about creating Lambda functions and deployment packages, see Deploy Python Lambda functions with .zip file archives in the *AWS Lambda Developer Guide* and the section called "Create the Lambda deployment package" in this guide.

To create your function using the Lambda console

1. Navigate to the Lambda console at https://console.aws.amazon.com/lambda/home. On the left navigation pane, choose **Functions**.
2. Select **Create function**.

3. Configure the following fields:

   - Function name: opensearch-function
   - Runtime: Python 3.9
   - Architecture: x86_64

   Keep all other default options and choose **Create function**.
4. In the **Code source** section of the function summary page, choose the **Upload from** dropdown and select **.zip file**. Locate the `my-deployment-package.zip` file that you created and choose **Save**.
5. The *handler* is the method in your function code that processes events. Under **Runtime settings**, choose **Edit** and change the handler name according to the name of the file in your deployment package where the Lambda function is located. Since your file is named `opensearch-lambda.py`, rename the handler to *`opensearch-lambda`*`.lambda_handler`. For more information, see Lambda function handler in Python.

Step 3: Create the API in API Gateway

Using API Gateway lets you create a more limited API and simplifies the process of interacting with the OpenSearch `_search` API. API Gateway lets you enable security features like Amazon Cognito authentication and request throttling. Perform the following steps to create and deploy an API:

Create and configure the API

To create your API using the API Gateway console

1. Navigate to the API Gateway console at https://console.aws.amazon.com/apigateway/home. On the left navigation pane, choose **APIs**.
2. Locate **REST API** (not private) and choose **Build**.
3. On the following page, locate the **Create new API** section and make sure **New API** is selected.
4. Configure the following fields:

   - API name: **opensearch-api**
   - Description: **Public API for searching an Amazon OpenSearch Service domain**
   - Endpoint Type: **Regional**

5. Choose **Create API**.
6. Choose **Actions** and **Create Method**.
7. Select **GET** in the dropdown and click the checkmark to confirm.
8. Configure the following settings, then choose **Save**:

Setting	Value
Integration type	Lambda function
Use Lambda proxy integration	Yes
Lambda region	*us-west-1*
Lambda function	opensearch-lambda
Use default timeout	Yes

Configure the method request

Choose **Method Request** and configure the following settings:

Setting	Value
Authorization	NONE
Request Validator	Validate query string parameters and headers
API Key Required	false

Under **URL Query String Parameters**, choose **Add query string** and configure the following parameter:

Setting	Value
Name	q

Setting	Value
Required	Yes

Deploy the API and configure a stage

The API Gateway console lets you deploy an API by creating a deployment and associating it with a new or existing stage.

1. Choose **Actions** and **Deploy API**.
2. For **Deployment stage** choose **New Stage** and name the stage `opensearch-api-test`.
3. Choose **Deploy.**
4. Configure the following settings in the stage editor, then choose **Save Changes**:

Setting	Value
Enable throttling	Yes
Rate	1000
Burst	500

These settings configure an API that has only one method: a `GET` request to the endpoint root (`https://`*`some-id`*`.execute-api.`*`us-west-1`*`.amazonaws.com/search-es-api-test`). The request requires a single parameter (q), the query string to search for. When called, the method passes the request to Lambda, which runs the `opensearch-lambda` function. For more information, see Creating an API in Amazon API Gateway and Deploying a REST API in Amazon API Gateway.

Step 4: (Optional) Modify the domain access policy

Your OpenSearch Service domain must allow the Lambda function to make `GET` requests to the `movies` index. If your domain has an open access policy with fine-grained access control enabled, you can leave it as-is:

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "es:*",
      "Resource": "arn:aws:es:us-west-1:123456789012:domain/domain-name/*"
    }
  ]
}
```

Alternatively, you can choose to make your domain access policy more granular. For example, the following minimum policy provides `opensearch-lambda-role` (created through Lambda) read access to the `movies` index. To get the exact name of the role that Lambda automatically creates, go to the AWS Identity and Access Management (IAM) console, choose **Roles**, and search for "lambda".

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/service-role/opensearch-lambda-
role-1abcdefg"
      },
      "Action": "es:ESHttpGet",
      "Resource": "arn:aws:es:us-west-1:123456789012:domain/domain-name/movies/_search"
    }
  ]
}
```

> **Important**
>
> If you have fine-grained access control enabled for the domain, you also need to map the role to a user in OpenSearch Dashboards, otherwise you'll see permissions errors.

For more information about access policies, see the section called “Configuring access policies”.

Map the Lambda role (if using fine-grained access control)

Fine-grained access control introduces an additional step before you can test the application. Even if you use HTTP basic authentication for all other purposes, you need to map the Lambda role to a user, otherwise you'll see permissions errors.

1. Navigate to the OpenSearch Dashboards URL for the domain.
2. From the main menu, choose **Security**, **Roles**, and select the link to `all_access`, the role you need to map the Lambda role to.
3. Choose **Mapped users**, **Manage mapping**.
4. Under **Backend roles**, add the Amazon Resource Name (ARN) of the Lambda role. The ARN should take the form of `arn:aws:iam::`*`123456789123`*`:role/service-role/`*`opensearch-lambda-role-1abcdefg`*.
5. Select **Map** and confirm the user or role shows up under **Mapped users**.

Step 5: Test the web application

To test the web application

1. Download sample-site.zip, unzip it, and open `scripts/search.js` in your favorite text editor.
2. Update the `apigatewayendpoint` variable to point to your API Gateway endpoint and add a backslash to the end of the given path. You can quickly find the endpoint in API Gateway by choosing **Stages** and selecting the name of the API. The `apigatewayendpoint` variable should take the form of `https://`*`some-id`*`.execute-api.`*`us-west-1`*`.amazonaws.com/opensearch-api-test/`.
3. Open `index.html` and try running searches for *thor*, *house*, and a few other terms.

Movie Search

thor

Found 7 results.

Thor

2011 — The powerful but arrogant god Thor is cast out of Asgard to live amongst humans in Midgard (Earth), where he soon becomes one of their finest defenders.

Thor: The Dark World

2013 — Faced with an enemy that even Odin and Asgard cannot withstand, Thor must embark on his most perilous and personal journey yet, one that will reunite him with Jane Foster and force him to sacrifice everything to save us all.

Vikingdom

2013 — A forgotten king, Eirick, is tasked with the impossible odds to defeat Thor, the God of Thunder.

Troubleshoot CORS errors

Even though the Lambda function includes content in the response to support CORS, you still might see the following error:

```
Access to XMLHttpRequest at '<api-gateway-endpoint>' from origin 'null' has been
 blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present in the
 requested resource.
```

If this happens, try the following:

1. Enable CORS on the GET resource. Under **Advanced**, set **Access-Control-Allow-Credentials** to `'true'`.
2. Redeploy your API in API Gateway (**Actions**, **Deploy API**).
3. Delete and re-add your Lambda function trigger. Add re-add it, choose **Add trigger** and create the HTTP endpoint that invokes your function. The trigger must have the following configuration:

Trigger	API	Deployment Stage	Security
API Gateway	opensearch-api	opensearch-api-test	Open

Next steps

This chapter is just a starting point to demonstrate a concept. You might consider the following modifications:

- Add your own data to the OpenSearch Service domain.
- Add methods to your API.
- In the Lambda function, modify the search query or boost different fields.
- Style the results differently or modify `search.js` to display different fields to the user.

Tutorial: Visualizing customer support calls with OpenSearch Service and OpenSearch Dashboards

This chapter is a full walkthrough of the following situation: a business receives some number of customer support calls and wants to analyze them. What is the subject of each call? How many were positive? How many were negative? How can managers search or review the the transcripts of these calls?

A manual workflow might involve employees listening to recordings, noting the subject of each call, and deciding whether or not the customer interaction was positive.

Such a process would be extremely labor-intensive. Assuming an average time of 10 minutes per call, each employee could listen to only 48 calls per day. Barring human bias, the data they generate would be highly accurate, but the *amount* of data would be minimal: just the subject of the call and a boolean for whether or not the customer was satisfied. Anything more involved, such as a full transcript, would take a huge amount of time.

Using Amazon S3, Amazon Transcribe, Amazon Comprehend, and Amazon OpenSearch Service, you can automate a similar process with very little code and end up with much more data. For example, you can get a full transcript of the call, keywords from the transcript, and an overall "sentiment" of the call (positive, negative, neutral, or mixed). Then you can use OpenSearch and OpenSearch Dashboards to search and visualize the data.

While you can use this walkthrough as-is, the intent is to spark ideas about how to enrich your JSON documents before you index them in OpenSearch Service.

Estimated Costs

In general, performing the steps in this walkthrough should cost less than $2. The walkthrough uses the following resources:

- S3 bucket with less than 100 MB transferred and stored

  To learn more, see Amazon S3 Pricing.
- OpenSearch Service domain with one `t2.medium` instance and 10 GiB of EBS storage for several hours

  To learn more, see Amazon OpenSearch Service Pricing.
- Several calls to Amazon Transcribe

To learn more, see Amazon Transcribe Pricing.

- Several natural language processing calls to Amazon Comprehend

  To learn more, see Amazon Comprehend Pricing.

Topics

Step 1: Configure prerequisites

Before proceeding, you must have the following resources.

Prerequisite	Description
Amazon S3 bucket	For more information, see Creating a Bucket in the *Amazon Simple Storage Service User Guide*.
OpenSearch Service domain	The destination for data. For more information, see Creating OpenSearch Service domains.

If you don't already have these resources, you can create them using the following AWS CLI commands:

```
aws s3 mb s3://my-transcribe-test --region us-west-2
```

```
aws opensearch create-domain --domain-name my-transcribe-test --engine-version
 OpenSearch_1.0 --cluster-config  InstanceType=t2.medium.search,InstanceCount=1
 --ebs-options EBSEnabled=true,VolumeType=standard,VolumeSize=10 --access-
policies '{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Principal":
{"AWS":"arn:aws:iam::123456789012:root"},"Action":"es:*","Resource":"arn:aws:es:us-
west-2:123456789012:domain/my-transcribe-test/*"}]}' --region us-west-2
```

Note

These commands use the `us-west-2` Region, but you can use any Region that Amazon Comprehend supports. To learn more, see the AWS General Reference.

Step 2: Copy sample code

1. Copy and paste the following Python 3 sample code into a new file named `call-center.py`:

```
import boto3
import datetime
import json
import requests
from requests_aws4auth import AWS4Auth
import time
import urllib.request

# Variables to update
audio_file_name = '' # For example, 000001.mp3
bucket_name = '' # For example, my-transcribe-test
domain = '' # For example, https://search-my-transcribe-test-12345.us-
west-2.es.amazonaws.com
index = 'support-calls'
type = '_doc'
region = 'us-west-2'

# Upload audio file to S3.
s3_client = boto3.client('s3')

audio_file = open(audio_file_name, 'rb')

print('Uploading ' + audio_file_name + '...')
response = s3_client.put_object(
    Body=audio_file,
    Bucket=bucket_name,
    Key=audio_file_name
)

# # Build the URL to the audio file on S3.
# # Only for the us-east-1 region.
# mp3_uri = 'https://' + bucket_name + '.s3.amazonaws.com/' + audio_file_name
```

```
# Get the necessary details and build the URL to the audio file on S3.
# For all other regions.
response = s3_client.get_bucket_location(
    Bucket=bucket_name
)
bucket_region = response['LocationConstraint']
mp3_uri = 'https://' + bucket_name + '.s3-' + bucket_region + '.amazonaws.com/' +
 audio_file_name

# Start transcription job.
transcribe_client = boto3.client('transcribe')

print('Starting transcription job...')
response = transcribe_client.start_transcription_job(
    TranscriptionJobName=audio_file_name,
    LanguageCode='en-US',
    MediaFormat='mp3',
    Media={
        'MediaFileUri': mp3_uri
    },
    Settings={
        'ShowSpeakerLabels': True,
        'MaxSpeakerLabels': 2 # assumes two people on a phone call
    }
)

# Wait for the transcription job to finish.
print('Waiting for job to complete...')
while True:
    response =
 transcribe_client.get_transcription_job(TranscriptionJobName=audio_file_name)
    if response['TranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED',
 'FAILED']:
        break
    else:
        print('Still waiting...')
    time.sleep(10)

transcript_uri = response['TranscriptionJob']['Transcript']['TranscriptFileUri']

# Open the JSON file, read it, and get the transcript.
response = urllib.request.urlopen(transcript_uri)
raw_json = response.read()
```

```
loaded_json = json.loads(raw_json)
transcript = loaded_json['results']['transcripts'][0]['transcript']

# Send transcript to Comprehend for key phrases and sentiment.
comprehend_client = boto3.client('comprehend')

# If necessary, trim the transcript.
# If the transcript is more than 5 KB, the Comprehend calls fail.
if len(transcript) > 5000:
    trimmed_transcript = transcript[:5000]
else:
    trimmed_transcript = transcript

print('Detecting key phrases...')
response = comprehend_client.detect_key_phrases(
    Text=trimmed_transcript,
    LanguageCode='en'
)

keywords = []
for keyword in response['KeyPhrases']:
    keywords.append(keyword['Text'])

print('Detecting sentiment...')
response = comprehend_client.detect_sentiment(
    Text=trimmed_transcript,
    LanguageCode='en'
)

sentiment = response['Sentiment']

# Build the Amazon OpenSearch Service URL.
id = audio_file_name.strip('.mp3')
url = domain + '/' + index + '/' + type + '/' + id

# Create the JSON document.
json_document = {'transcript': transcript, 'keywords': keywords, 'sentiment':
 sentiment, 'timestamp': datetime.datetime.now().isoformat()}

# Provide all details necessary to sign the indexing request.
credentials = boto3.Session().get_credentials()
awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region,
 'opensearchservice', session_token=credentials.token)
```

```
# Index the document.
print('Indexing document...')
response = requests.put(url, auth=awsauth, json=json_document, headers=headers)

print(response)
print(response.json())
```

2. Update the initial six variables.
3. Install the required packages using the following commands:

```
pip install boto3
pip install requests
pip install requests_aws4auth
```

4. Place your MP3 in the same directory as `call-center.py` and run the script. A sample output follows:

```
$ python call-center.py
Uploading 000001.mp3...
Starting transcription job...
Waiting for job to complete...
Still waiting...
Still waiting...
Still waiting...
Still waiting...
Still waiting...
Still waiting...
Still waiting...
Detecting key phrases...
Detecting sentiment...
Indexing document...
<Response [201]>
{u'_type': u'call', u'_seq_no': 0, u'_shards': {u'successful': 1, u'failed': 0,
 u'total': 2}, u'_index': u'support-calls4', u'_version': 1, u'_primary_term': 1,
 u'result': u'created', u'_id': u'000001'}
```

`call-center.py` performs a number of operations:

1. The script uploads an audio file (in this case, an MP3, but Amazon Transcribe supports several formats) to your S3 bucket.
2. It sends the audio file's URL to Amazon Transcribe and waits for the transcription job to finish.

The time to finish the transcription job depends on the length of the audio file. Assume minutes, not seconds.

> **Tip**
>
> To improve the quality of the transcription, you can configure a custom vocabulary for Amazon Transcribe.

3. After the transcription job finishes, the script extracts the transcript, trims it to 5,000 characters, and sends it to Amazon Comprehend for keyword and sentiment analysis.
4. Finally, the script adds the full transcript, keywords, sentiment, and current time stamp to a JSON document and indexes it in OpenSearch Service.

> **Tip**
>
> LibriVox has public domain audiobooks that you can use for testing.

(Optional) Step 3: Index sample data

If you don't have a bunch of call recordings handy—and who does?—you can index the sample documents in sample-calls.zip, which are comparable to what `call-center.py` produces.

1. Create a file named `bulk-helper.py`:

```
import boto3
from opensearchpy import OpenSearch, RequestsHttpConnection
import json
from requests_aws4auth import AWS4Auth

host = '' # For example, my-test-domain.us-west-2.es.amazonaws.com
region = '' # For example, us-west-2
service = 'es'

bulk_file = open('sample-calls.bulk', 'r').read()

credentials = boto3.Session().get_credentials()
awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region, service,
 session_token=credentials.token)
```

```
search = OpenSearch(
    hosts = [{'host': host, 'port': 443}],
    http_auth = awsauth,
    use_ssl = True,
    verify_certs = True,
    connection_class = RequestsHttpConnection
)

response = search.bulk(bulk_file)
print(json.dumps(response, indent=2, sort_keys=True))
```

2. Update the initial two variables for `host` and `region`.
3. Install the required package using the following command:

```
pip install opensearch-py
```

4. Download and unzip sample-calls.zip.
5. Place `sample-calls.bulk` in the same directory as `bulk-helper.py` and run the helper. A sample output follows:

```
$ python bulk-helper.py
{
  "errors": false,
  "items": [
    {
      "index": {
        "_id": "1",
        "_index": "support-calls",
        "_primary_term": 1,
        "_seq_no": 42,
        "_shards": {
          "failed": 0,
          "successful": 1,
          "total": 2
        },
        "_type": "_doc",
        "_version": 9,
        "result": "updated",
        "status": 200
      }
    },
```

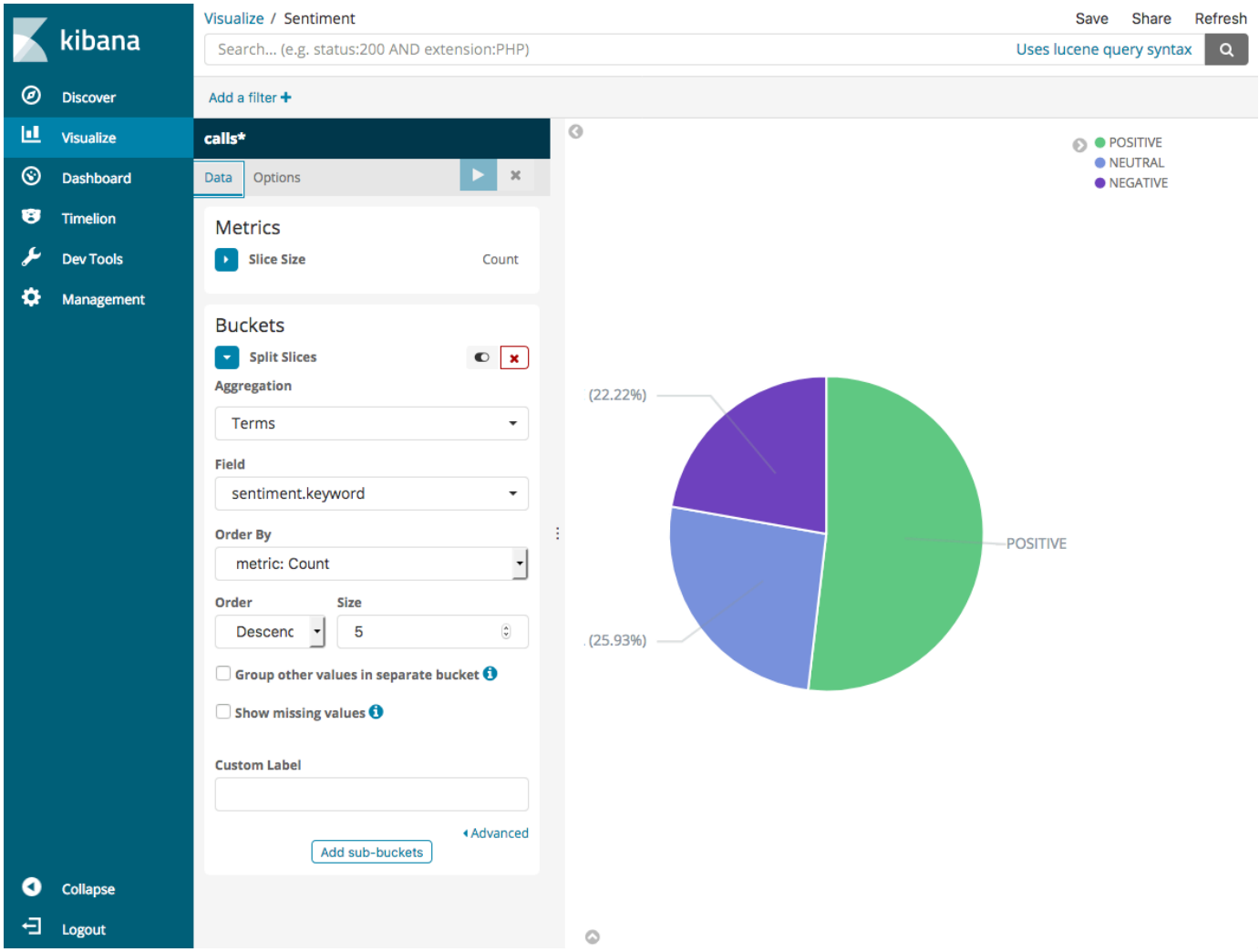
```
    ...
  ],
  "took": 27
}
```

Step 4: Analyze and visualize your data

Now that you have some data in OpenSearch Service, you can visualize it using OpenSearch Dashboards.

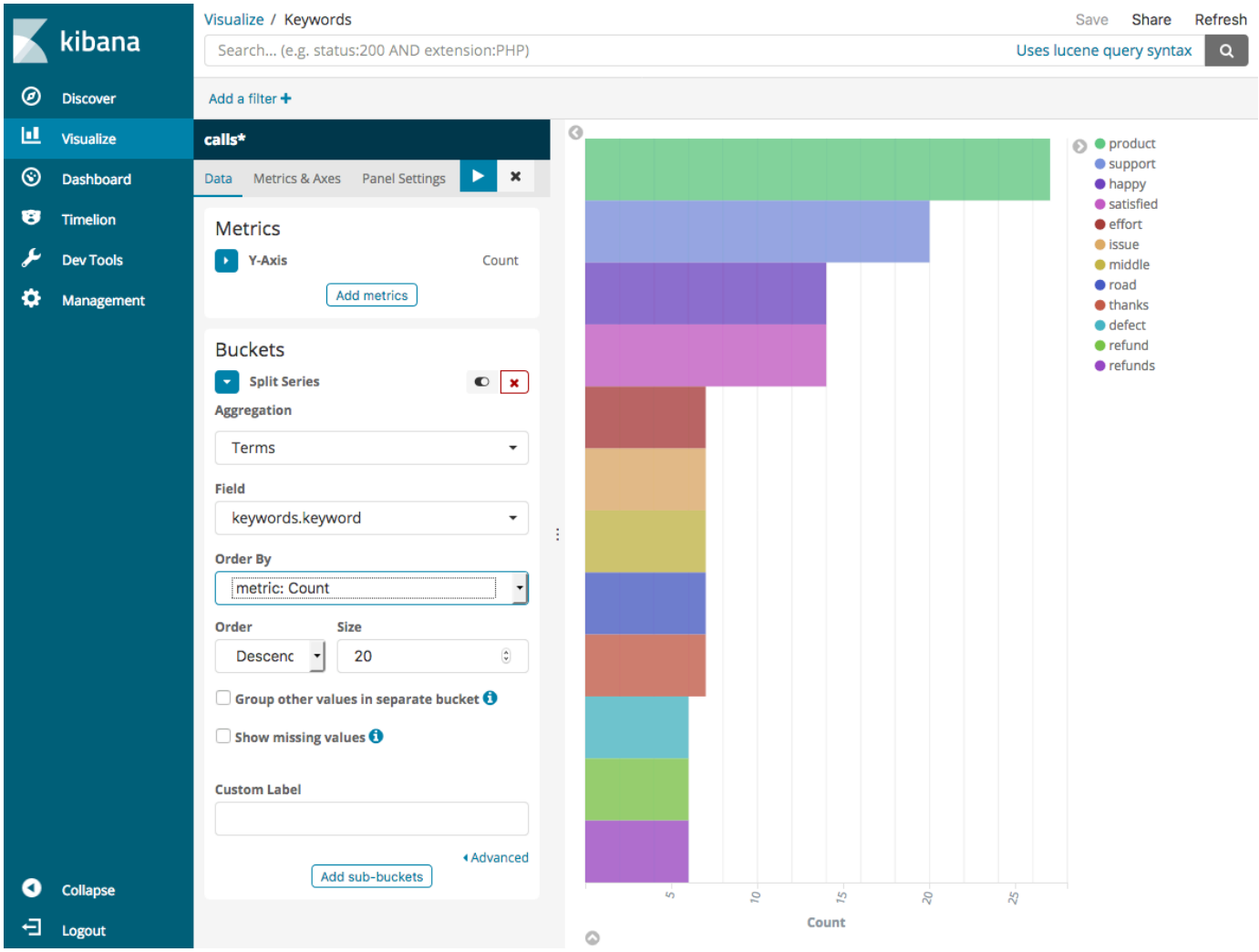
1. Navigate to `https://search-domain.region.es.amazonaws.com/_dashboards`.
2. Before you can use OpenSearch Dashboards, you need an index pattern. Dashboards uses index patterns to narrow your analysis to one or more indices. To match the `support-calls` index that `call-center.py` created, go to **Stack Management**, **Index Patterns**, and define an index pattern of `support*`, and then choose **Next step**.
3. For **Time Filter field name**, choose **timestamp**.
4. Now you can start creating visualizations. Choose **Visualize**, and then add a new visualization.
5. Choose the pie chart and the `support*` index pattern.
6. The default visualization is basic, so choose **Split Slices** to create a more interesting visualization.

   For **Aggregation**, choose **Terms**. For **Field**, choose **sentiment.keyword**. Then choose **Apply changes** and **Save**.

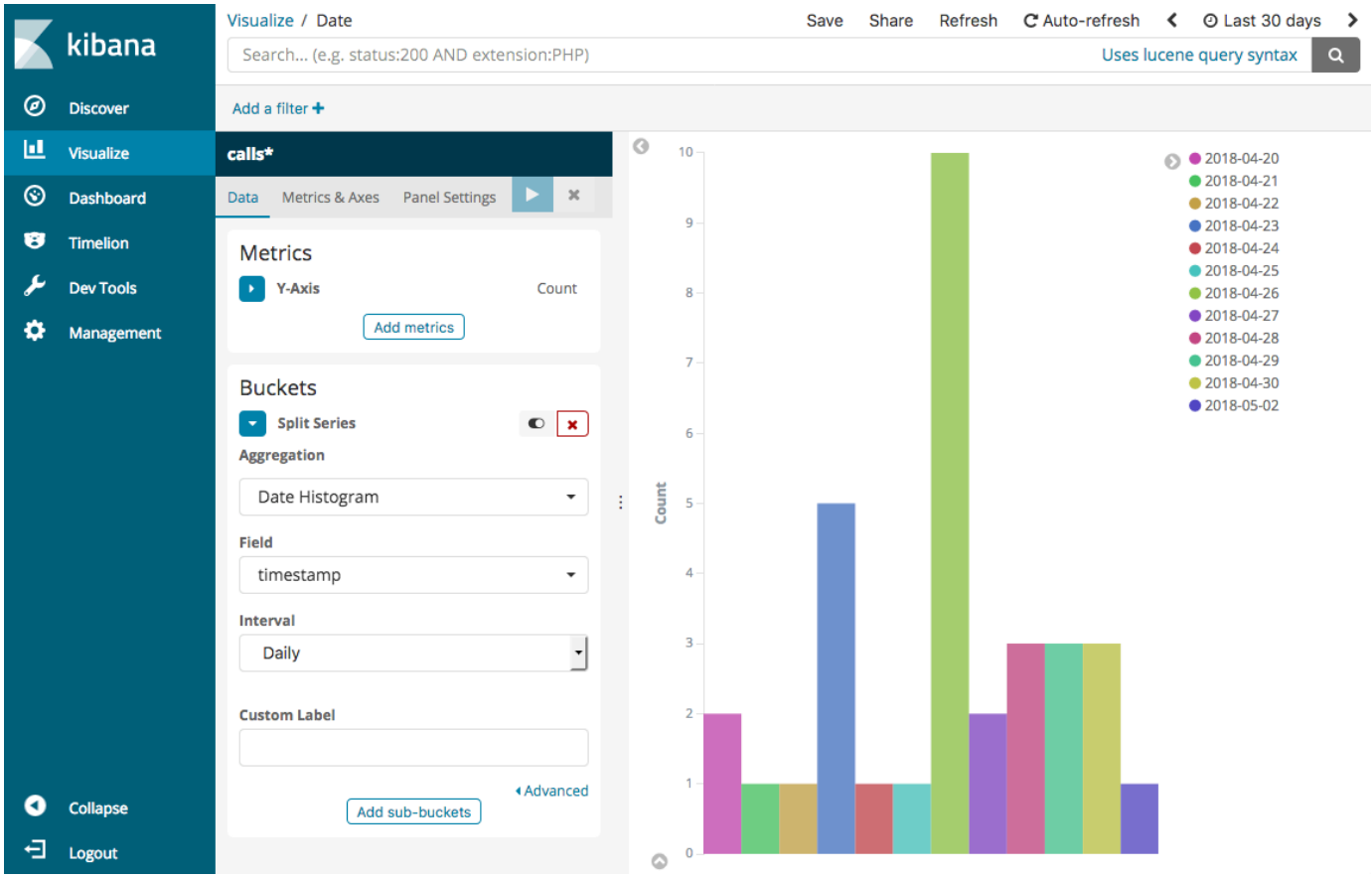


7. Return to the **Visualize** page, and add another visualization. This time, choose the horizontal bar chart.
8. Choose **Split Series**.

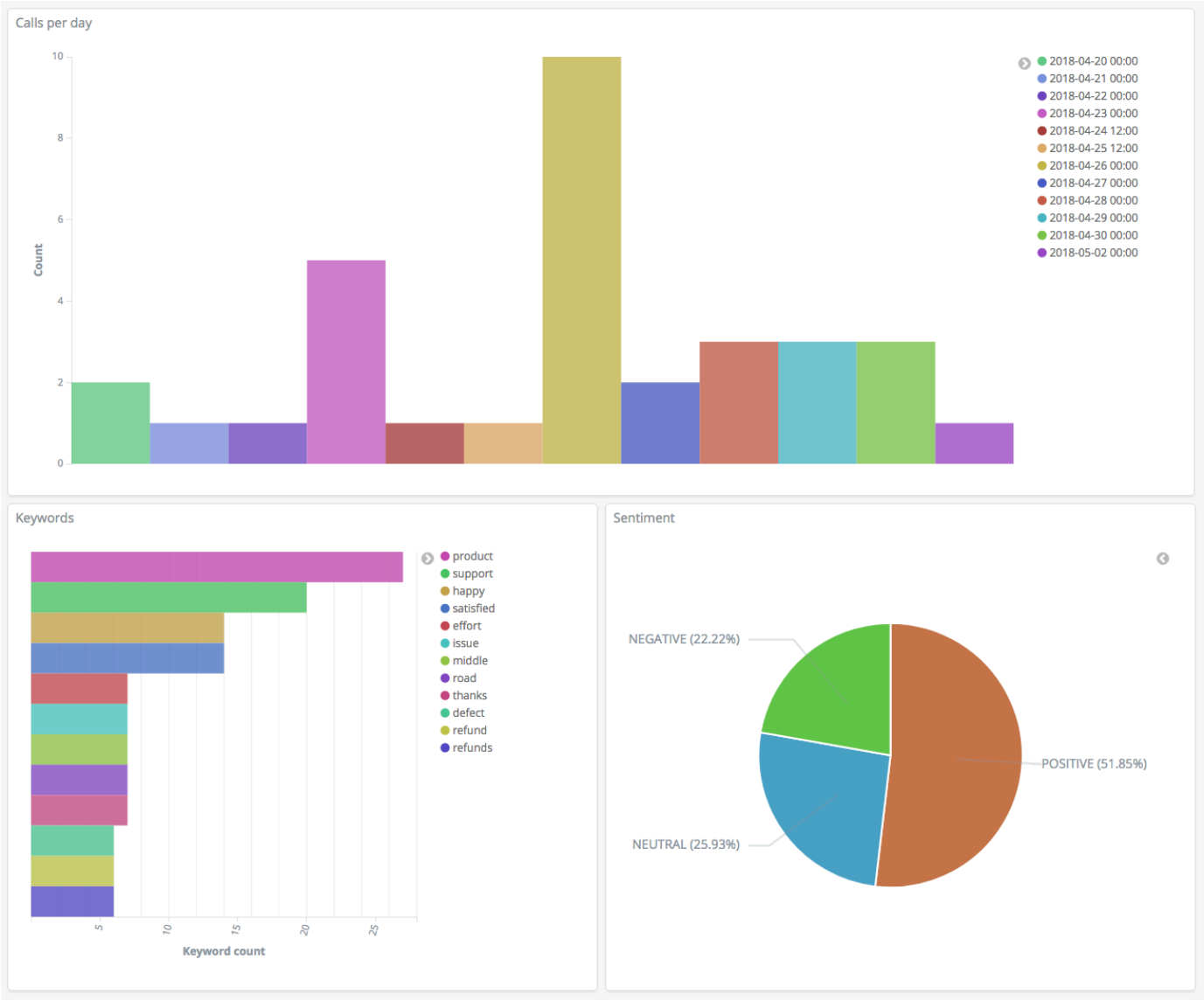
   For **Aggregation**, choose **Terms**. For **Field**, choose **keywords.keyword** and change **Size** to 20. Then choose **Apply Changes** and **Save**.



9. Return to the **Visualize** page and add one final visualization, a vertical bar chart.
10. Choose **Split Series**. For **Aggregation**, choose **Date Histogram**. For **Field**, choose **timestamp** and change **Interval** to **Daily**.
11. Choose **Metrics & Axes** and change **Mode** to **normal**.
12. Choose **Apply Changes** and **Save**.



13. Now that you have three visualizations, you can add them to a Dashboards visualization. Choose **Dashboard**, create a dashboard, and add your visualizations.



Step 5: Clean up resources and next steps

To avoid unnecessary charges, delete the S3 bucket and OpenSearch Service domain. To learn more, see Delete a Bucket in the *Amazon Simple Storage Service User Guide* and Delete an OpenSearch Service domain in this guide.

Transcripts require much less disk space than MP3 files. You might be able to shorten your MP3 retention window—for example, from three months of call recordings to one month—retain years of transcripts, and still save on storage costs.

You could also automate the transcription process using AWS Step Functions and Lambda, add additional metadata before indexing, or craft more complex visualizations to fit your exact use case.

Amazon OpenSearch Service rename - Summary of changes

On September 8, 2021, our search and analytics suite was renamed to Amazon OpenSearch Service. OpenSearch Service supports OpenSearch as well as legacy Elasticsearch OSS. The following sections describe the different parts of the service that changed with the rename, and what actions you need to take to ensure that your domains continue to function properly.

Some of these changes only apply when you upgrade your domains from Elasticsearch to OpenSearch. In other cases, such as in the Billing and Cost Management console, the experience changes immediately.

Note that this list is not exhaustive. While other parts of the product also changed, these updates are the most relevant.

New API version

The new version of the OpenSearch Service configuration API (2021-01-01) works with OpenSearch as well as legacy Elasticsearch OSS. 21 API operations were replaced with more concise and engine-agnostic names (for example, `CreateElasticsearchDomain` changed to `CreateDomain`), but OpenSearch Service continues to support both API versions.

We recommend that you use the new API operations to create and manage domains going forward. Note that when you use the new API operations to create a domain, you need to specify the `EngineVersion` parameter in the format `Elasticsearch_X.Y` or `OpenSearch_X.Y`, rather than just the version number. If you don't specify a version, it defaults to the latest version of OpenSearch.

Upgrade your AWS CLI to version 1.20.40 or later in order to use `aws opensearch ...` to create and manage your domains. For the new CLI format, see the OpenSearch CLI reference.

Renamed instance types

Instance types in Amazon OpenSearch Service are now in the format `<type>.<size>.search`—for example, `m6g.large.`*`search`* rather than `m6g.large.`*`elasticsearch`*. You don't need to

take any action. Existing domains will start automatically referring to the new instance types within the API and in the Billing and Cost Management console.

If you have Reserved Instances (RIs), your contract won't be impacted by the change. The old configuration API version is still compatible with the old naming format, but if you want to use the new API version, you need to use the new format.

Access policy changes

The following sections describe what actions you need to take to update your access policies.

IAM policies

We recommend that you update your IAM policies to use the renamed API operations. However, OpenSearch Service will continue to respect existing policies by internally replicating the old API permissions. For example, if you currently have permission to perform the `CreateElasticsearchDomain` operation, you can now make calls to both `CreateElasticsearchDomain` (old API operation) and `CreateDomain` (new API operation). The same applies to explicit denies. For a list of updated API operations, see the policy element reference.

SCP policies

Service control policies (SCPs) introduce an additional layer of complexity compared to standard IAM. To prevent your SCP policies from breaking, you need to add both the old *and* the new API operations to each of your SCP policies. For example, if a user currently has allow permissions for `CreateElasticsearchDomain`, you also need to grant them allow permissions for `CreateDomain` so they can retain the ability to create domains. The same applies to explicit denies.

For example:

```
"Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "es:CreateElasticsearchDomain",
        "es:CreateDomain"
         ...
```

```
      ],
    },
      "Effect": "Deny",
      "Action:" [
        "es:DeleteElasticsearchDomain",
        "es:DeleteDomain"
         ...
```

New resource types

OpenSearch Service introduces the following new resource types:

Resource	Description
`AWS::OpenSearchService::Domain`	Represents an Amazon OpenSearch Service domain. This resource exists at the service level and isn't specific to the software running on the domain. It applies to services like AWS CloudFormation and AWS Resource Groups, in which you create and manage resources for the service as a whole. For instructions to upgrade domains defined within CloudFormation from Elasticsearch to OpenSearch, see Remarks in the CloudFormation User Guide.
`AWS::OpenSearch::Domain`	Represents OpenSearch/Elasticsearch software running on a domain. This resource applies to services like AWS CloudTrail and AWS Config, which reference the software running *on* the domain rather than OpenSearch Service as a whole. These services now contain separate resource types for domains running Elasticsearch (`AWS::Elasticsearch::Domain`) versus domains running OpenSearch (`AWS::OpenSearch::Domain`).

Note

In AWS Config, you'll continue to see your data under the existing `AWS::Elasticsearch::Domain` resource type for several weeks, even if you upgrade one or more domains to OpenSearch.

Kibana renamed to OpenSearch Dashboards

OpenSearch Dashboards, the AWS alternative to Kibana, is an open-source visualization tool designed to work with OpenSearch. After you upgrade a domain from Elasticsearch to OpenSearch, the `/_plugin/kibana` endpoint changes to `/_dashboards`. OpenSearch Service will redirect all requests to the new endpoint, but if you use the Kibana endpoint in any of your IAM policies, update those policies to include the new `/_dashboards` endpoint as well.

If you're using the section called "SAML authentication for OpenSearch Dashboards", before you upgrade your domain to OpenSearch, you need to change all Kibana URLs configured in your identity provider (IdP) from `/_plugin/kibana` to `/_dashboards`. The most common URLs are assertion consumer service (ACS) URLs and recipient URLs.

The default `kibana_read_only` role for OpenSearch Dashboards was renamed to `opensearch_dashboards_read_only`, and the `kibana_user` role was renamed to `opensearch_dashboards_user`. The change applies to all *newly-created* OpenSearch 1.*x* domains running service software R20211203 or later. If you upgrade an existing domain to service software R20211203, the role names remain the same.

Renamed CloudWatch metrics

Several CloudWatch metrics change for domains running OpenSearch. When you upgrade a domain to OpenSearch, the metrics change automatically and your current CloudWatch alarms will break. Before upgrading your cluster from an Elasticsearch version to an OpenSearch version, make sure to update your CloudWatch alarms to use the new metrics.

The following metrics changed:

Original metric name	New name
`KibanaHealthyNodes`	`OpenSearchDashboardsHealthyNodes`

Original metric name	New name
KibanaConcurrentConnections	OpenSearchDashboardsConcurrentConnections
KibanaHeapTotal	OpenSearchDashboardsHeapTotal
KibanaHeapUsed	OpenSearchDashboardsHeapUsed
KibanaHeapUtilization	OpenSearchDashboardsHeapUtilization
KibanaOS1MinuteLoad	OpenSearchDashboardsOS1MinuteLoad
KibanaRequestTotal	OpenSearchDashboardsRequestTotal
KibanaResponseTimesMaxInMillis	OpenSearchDashboardsResponseTimesMaxInMillis
ESReportingFailedRequestSysErrCount	KibanaReportingFailedRequestSysErrCount
ESReportingRequestCount	KibanaReportingRequestCount
ESReportingFailedRequestUserErrCount	KibanaReportingFailedRequestUserErrCount
ESReportingSuccessCount	KibanaReportingSuccessCount
ElasticsearchRequests	OpenSearchRequests

For a full list of metrics that OpenSearch Service sends to Amazon CloudWatch, see the section called “Monitoring cluster metrics”.

Billing and Cost Management console changes

Historic data in the Billing and Cost Management console and in Cost and Usage Reports will continue to use the old service name, so you need to start using filters for both **Amazon**

OpenSearch Service and the legacy Elasticsearch name when searching for data. If you have existing saved reports, update the filters to make sure they also include OpenSearch Service. You might initially receive an alert when your usage decreases for Elasticsearch and increases for OpenSearch, but it disappears within several days.

In addition to the service name, the following fields will change for all reports, bills, and price list API operations:

Field	Old format	New format
Instance type	`m5.large.elasticsearch`	`m5.large.search`
Product family	Elasticsearch Instance Elasticsearch Volume	Amazon OpenSearch Service Instance Amazon OpenSearch Service Volume
Pricing description	$5.098 per c5.18xlarge.elasticsearch instance hour (or partial hour) - EU	$5.098 per c5.18xlarge.search instance hour (or partial hour) - EU
Instance family	`ultrawarm.elasticsearch`	`ultrawarm.search`

New event format

The format of events that OpenSearch Service sends to Amazon EventBridge and Amazon CloudWatch has changed, specifically the `detail-type` field. The source field (`aws.es`) remains the same. For the complete format for each event type, see the section called “Monitoring events”. If you have existing event rules that depend on the old format, make sure to update them to conform to the new format.

What's staying the same?

The following features and functionality, among others not listed, will remain the same:

- Service principal (`es.amazonaws.com`)
- Vendor code
- Domain ARNs
- Domain endpoints

Get started: Upgrade your domains to OpenSearch 1.x

OpenSearch 1.*x* supports upgrades from Elasticsearch versions 6.8 and 7.*x*. For instructions to upgrade your domain, see the section called "Upgrading a domain (console)". If you're using the AWS CLI or configuration API to upgrade your domain, you need to specify the `TargetVersion` as `OpenSearch_1.x`.

OpenSearch 1.*x* introduces an additional domain setting called **Enable compatibility mode**. Because certain Elasticsearch OSS clients and plugins check the cluster version before connecting, compatibility mode sets OpenSearch to report its version as 7.10 so these clients continue to work.

You can enable compatibility mode when you create OpenSearch domains for the first time, or when you upgrade to OpenSearch from an Elasticsearch version. If it's not set, the parameter defaults to `false` when you create a domain, and `true` when you upgrade a domain.

To enable compatibility mode using the configuration API, set `override_main_response_version` to `true`:

```
POST https://es.us-east-1.amazonaws.com/2021-01-01/opensearch/upgradeDomain
{
  "DomainName": "domain-name",
  "TargetVersion": "OpenSearch_1.0",
  "AdvancedOptions": {
    "override_main_response_version": "true"
   }
}
```

To enable or disable compatibility mode on *existing* OpenSearch domains, you need to use the OpenSearch _cluster/settings API operation:

```
PUT /_cluster/settings
{
  "persistent" : {
```

```
    "compatibility.override_main_response_version" : true
  }
}
```

Troubleshooting Amazon OpenSearch Service

This topic describes how to identify and solve common Amazon OpenSearch Service issues. Consult the information in this section before contacting AWS Support.

Can't access OpenSearch Dashboards

The OpenSearch Dashboards endpoint doesn't support signed requests. If the access control policy for your domain only grants access to certain IAM roles and you haven't configured Amazon Cognito authentication, you might receive the following error when you attempt to access Dashboards:

```
"User: anonymous is not authorized to perform: es:ESHttpGet"
```

If your OpenSearch Service domain uses VPC access, you might not receive this error, but the request might time out. To learn more about correcting this issue and the various configuration options available to you, see the section called “Controlling access to Dashboards ”, the section called “About access policies on VPC domains”, and the section called “Identity and Access Management”.

Can't access VPC domain

See the section called “About access policies on VPC domains” and the section called “Testing VPC domains”.

Cluster in read-only state

Compared to earlier Elasticsearch versions, OpenSearch and Elasticsearch 7.*x* use a different system for cluster coordination. In this new system, when the cluster loses quorum, the cluster is unavailable until you take action. Loss of quorum can take two forms:

- If your cluster uses dedicated master nodes, quorum loss occurs when half or more are unavailable.
- If your cluster does not use dedicated master nodes, quorum loss occurs when half or more of your data nodes are unavailable.

If quorum loss occurs and your cluster has more than one node, OpenSearch Service restores quorum and places the cluster into a read-only state. You have two options:

- Remove the read-only state and use the cluster as-is.
- Restore the cluster or individual indexes from a snapshot.

If you prefer to use the cluster as-is, verify that cluster health is green using the following request:

```
GET _cat/health?v
```

If cluster health is red, we recommend restoring the cluster from a snapshot. You can also see the section called "Red cluster status" for troubleshooting steps. If cluster health is green, check that all expected indexes are present using the following request:

```
GET _cat/indices?v
```

Then run some searches to verify that the expected data is present. If it is, you can remove the read-only state using the following request:

```
PUT _cluster/settings
{
  "persistent": {
    "cluster.blocks.read_only": false
  }
}
```

If quorum loss occurs and your cluster has only one node, OpenSearch Service replaces the node and does *not* place the cluster into a read-only state. Otherwise, your options are the same: use the cluster as-is or restore from a snapshot.

In both situations, OpenSearch Service sends two events to your AWS Health Dashboard. The first informs you of the loss of quorum. The second occurs after OpenSearch Service successfully restores quorum. For more information about using the AWS Health Dashboard, see the AWS Health User Guide.

Red cluster status

A red cluster status means that at least one primary shard and its replicas are not allocated to a node. OpenSearch Service keeps trying to take automated snapshots of all indexes regardless of their status, but the snapshots fail while the red cluster status persists.

The most common causes of a red cluster status are failed cluster nodes and the OpenSearch process crashing due to a continuous heavy processing load.

> **Note**
>
> OpenSearch Service stores automated snapshots for 14 days regardless of the cluster status. Therefore, if the red cluster status persists for more than two weeks, the last healthy automated snapshot will be deleted and you could permanently lose your cluster's data. If your OpenSearch Service domain enters a red cluster status, Support might contact you to ask whether you want to address the problem yourself or you want the support team to assist. You can set a CloudWatch alarm to notify you when a red cluster status occurs.

Ultimately, red shards cause red clusters, and red indexes cause red shards. To identify the indexes causing the red cluster status, OpenSearch has some helpful APIs.

- `GET /_cluster/allocation/explain` chooses the first unassigned shard that it finds and explains why it cannot be allocated to a node:

```
{
    "index": "test4",
    "shard": 0,
    "primary": true,
    "current_state": "unassigned",
    "can_allocate": "no",
    "allocate_explanation": "cannot allocate because allocation is not permitted to
 any of the nodes"
}
```

- `GET /_cat/indices?v` shows the health status, number of documents, and disk usage for each index:

```
health status index           uuid                   pri rep docs.count docs.deleted
 store.size pri.store.size
```

```
green  open   test1           30h1EiMvS5uAFr2t5CEVoQ   5   0        820            0
       14mb          14mb
green  open   test2           sdIxs_WDT56afFGu5KPbFQ   1   0          0            0
       233b          233b
green  open   test3           GGRZp_TBRZuSaZpAGk2pmw   1   1          2            0
     14.7kb         7.3kb
red    open   test4           BJxfAErbTtu5HBjIXJV_7A   1   0
green  open   test5           _8C6MIXOSxCqVYicH3jsEA   1   0          7            0
     24.3kb        24.3kb
```

Deleting red indexes is the fastest way to fix a red cluster status. Depending on the reason for the red cluster status, you might then scale your OpenSearch Service domain to use larger instance types, more instances, or more EBS-based storage and try to recreate the problematic indexes.

If deleting a problematic index isn't feasible, you can restore a snapshot, delete documents from the index, change the index settings, reduce the number of replicas, or delete other indexes to free up disk space. The important step is to resolve the red cluster status *before* reconfiguring your OpenSearch Service domain. Reconfiguring a domain with a red cluster status can compound the problem and lead to the domain being stuck in a configuration state of **Processing** until you resolve the status.

Automatic remediation of red clusters

If your cluster's status is continuously red for more than an hour, OpenSearch Service attempts to automatically fix it by rerouting unallocated shards or restoring from past snapshots.

If it fails to fix one or more red indexes and the cluster status remains red for a total of 14 days, OpenSearch Service takes further action only if the cluster meets *at least one* of the following criteria:

- Has only one availability zone
- Has no dedicated master nodes
- Contains burstable instance types (T2 or T3)

At this time, if your cluster meets one of these criteria, OpenSearch Service sends you daily notifications over the next 7 days explaining that if you don't fix these indexes, all unassigned shards will be deleted. If your cluster status is still red after 21 days, OpenSearch Service deletes the unassigned shards (storage and compute) on all red indexes. You receive notifications in

the **Notifications** panel of the OpenSearch Service console for each of these events. For more information, see the section called "Cluster health events".

Recovering from a continuous heavy processing load

To determine if a red cluster status is due to a continuous heavy processing load on a data node, monitor the following cluster metrics.

Relevant metric	Description	Recovery
JVMMemoryPressure	Specifies the percentage of the Java heap used for all data nodes in a cluster. View the **Maximum** statistic for this metric, and look for smaller and smaller drops in memory pressure as the Java garbage collector fails to reclaim sufficient memory. This pattern likely is due to complex queries or large data fields. x86 instance types use the Concurrent Mark Sweep (CMS) garbage collector, which runs alongside application threads to keep pauses short. If CMS is unable to reclaim enough memory during its normal collections, it triggers a full garbage collection, which can lead to long application pauses and impact cluster stability. ARM-based Graviton instance types use the Garbage-First (G1) garbage collector, which is similar to CMS, but uses additional short pauses and heap defragmentation to further	Set memory circuit breakers for the JVM. For more information, see the section called "JVM OutOfMemoryError". If the problem persists, delete unnecessary indexes, reduce the number or complexity of requests to the domain, add instances, or use larger instance types.

Relevant metric	Description	Recovery
	reduce the need for full garbage collections. In either case, if memory usage continues to grow beyond what the garbage collector can reclaim during full garbage collections, OpenSearch crashes with an out of memory error. On all instance types, a good rule of thumb is to keep usage below 80%. The `_nodes/stats/jvm` API offers a useful summary of JVM statistics, memory pool usage, and garbage collection information: `GET domain-endpoint /_nodes/stats/jvm?pretty`	
CPUUtilization	Specifies the percentage of CPU resources used for data nodes in a cluster. View the **Maximum** statistic for this metric, and look for a continuous pattern of high usage.	Add data nodes or increase the size of the instance types of existing data nodes.
Nodes	Specifies the number of nodes in a cluster. View the **Minimum** statistic for this metric. This value fluctuates when the service deploys a new fleet of instances for a cluster.	Add data nodes.

Yellow cluster status

A yellow cluster status means the primary shards for all indexes are allocated to nodes in a cluster, but the replica shards for at least one index aren't. Single-node clusters always initialize with a yellow cluster status because there's no other node to which OpenSearch Service can assign a replica. To achieve green cluster status, increase your node count. For more information, see the section called "Sizing domains".

Multi-node clusters might briefly have a yellow cluster status after creating a new index or after a node failure. This status self-resolves as OpenSearch replicates data across the cluster. Lack of disk space can also cause yellow cluster status; the cluster can only distribute replica shards if nodes have the disk space to accommodate them.

ClusterBlockException

You might receive a `ClusterBlockException` error for the following reasons.

Lack of available storage space

If one or more nodes in your cluster has storage space less than the minimum value of 1) 20% of available storage space, or 2) 20 GiB of storage space, basic write operations like adding documents and creating indexes can start to fail. the section called "Calculating storage requirements" provides a summary of how OpenSearch Service uses disk space.

To avoid issues, monitor the `FreeStorageSpace` metric in the OpenSearch Service console and create CloudWatch alarms to trigger when `FreeStorageSpace` drops below a certain threshold. `GET /_cat/allocation?v` also provides a useful summary of shard allocation and disk usage. To resolve issues associated with a lack of storage space, scale your OpenSearch Service domain to use larger instance types, more instances, or more EBS-based storage.

High JVM memory pressure

When the **JVMMemoryPressure** metric exceeds 92% for 30 minutes, OpenSearch Service triggers a protection mechanism and blocks all write operations to prevent the cluster from reaching red status. When the protection is on, write operations fail with a `ClusterBlockException` error, new indexes can't be created, and the `IndexCreateBlockException` error is thrown.

When the **JVMMemoryPressure** metric returns to 88% or lower for five minutes, the protection is disabled, and write operations to the cluster are unblocked.

High JVM memory pressure can be caused by spikes in the number of requests to the cluster, unbalanced shard allocations across nodes, too many shards in a cluster, field data or index mapping explosions, or instance types that can't handle incoming loads. It can also be caused by using aggregations, wildcards, or wide time ranges in queries.

To reduce traffic to the cluster and resolve high JVM memory pressure issues, try one or more of the following:

- Scale the domain so that the maximum heap size per node is 32 GB.
- Reduce the number of shards by deleting old or unused indexes.
- Clear the data cache with the `POST` *`index-name`*`/_cache/clear?fielddata=true` API operation. Note that clearing the cache can disrupt in-progress queries.

In general, to avoid high JVM memory pressure in the future, follow these best practices:

- Avoid aggregating on text fields, or change the mapping type for your indexes to `keyword`.
- Optimize search and indexing requests by choosing the correct number of shards.
- Set up Index State Management (ISM) policies to regularly remove unused indexes.

Error migrating to Multi-AZ with Standby

The following issues might occur when you migrate an existing domain to Multi-AZ with standby.

Creating an index, index template, or ISM policy during migration from domains without standby to domains with standby

If you create an index while migrating a domain from Multi-AZ without Standby to with Standby, and the index template or ISM policy doesn't follow the recommended data copy guidelines, this can cause a data inconsistency and the migration may fail. To avoid this situation, create the new index with a data copy count (including both primary nodes and replicas) that is multiple of three. You can check the migratation progress using the `DescribeDomainChangeProgress` API. If you encounter a replica count error, fix the error and then contact AWS Support to retry the migration.

Incorrect number of data copies

If you don't have the right number of data copies in your domain, the migrating to Multi-AZ with Standby will fail.

JVM OutOfMemoryError

A JVM `OutOfMemoryError` typically means that one of the following JVM circuit breakers was reached.

Circuit breaker	Description	Cluster setting property
Parent Breaker	Total percentage of JVM heap memory allowed for all circuit breakers. The default value is 95%.	`indices.breaker.total.limit`
Field Data Breaker	Percentage of JVM heap memory allowed to load a single data field into memory. The default value is 40%. If you upload data with large fields, you might need to raise this limit.	`indices.breaker.fielddata.limit`
Request Breaker	Percentage of JVM heap memory allowed for data structures used to respond to a service request. The default value is 60%. If your service requests involve calculating aggregations, you might need to raise this limit.	`indices.breaker.request.limit`

Failed cluster nodes

Amazon EC2 instances might experience unexpected terminations and restarts. Typically, OpenSearch Service restarts the nodes for you. However, it's possible for one or more nodes in an OpenSearch cluster to remain in a failed condition.

To check for this condition, open your domain dashboard on the OpenSearch Service console. Go to the **Cluster health** tab and find the **Total nodes** metric. See if the reported number of nodes is fewer than the number that you configured for your cluster. If the metric shows that one or more nodes is down for more than one day, contact AWS Support.

You can also set a CloudWatch alarm to notify you when this issue occurs.

> **Note**
>
> The **Total nodes** metric is not accurate during changes to your cluster configuration and during routine maintenance for the service. This behavior is expected. The metric will report the correct number of cluster nodes soon. To learn more, see the section called "Configuration changes".

To protect your clusters from unexpected node terminations and restarts, create at least one replica for each index in your OpenSearch Service domain.

Exceeded maximum shard limit

OpenSearch as well as 7.*x* versions of Elasticsearch have a default setting of no more than 1,000 shards per node. OpenSearch/Elasticsearch throw an error if a request, such as creating a new index, would cause you to exceed this limit. If you encounter this error, you have several options:

- Add more data nodes to the cluster.
- Increase the `_cluster/settings/cluster.max_shards_per_node` setting.
- Use the _shrink API to reduce the number of shards on the node.

Domain stuck in processing state

Your OpenSearch Service domain enters the "Processing" state when it's in the middle of a configuration change. When you initiate a configuration change, the domain status changes to

"Processing" while OpenSearch Service creates a new environment. In the new environment, OpenSearch Service launches a new set of applicable nodes (such as data, master, or UltraWarm). After the migration completes, the older nodes are terminated.

The cluster can get stuck in the "Processing" state if either of these situations occurs:

- A new set of data nodes fails to launch.

- Shard migration to the new set of data nodes is unsuccessful.
- Validation check has failed with errors.

For detailed resolution steps in each of these situations, see Why is my Amazon OpenSearch Service domain stuck in the "Processing" state?.

Low EBS burst balance

OpenSearch Service sends you a console notification when the EBS burst balance on one of your General Purpose (SSD) volumes is below 70%, and a follow-up notification if the balance falls below 20%. To fix this issue, you can either scale up your cluster, or reduce the read and write IOPS so that the burst balance can be credited. The burst balance stays at 0 for domains with gp3 volumes types, and domains with gp2 volumes that have a volume size above 1000 GiB. For more information, see General Purpose SSD volumes (gp2). You can monitor EBS burst balance with the `BurstBalance` CloudWatch metric.

Can't enable audit logs

You might encounter the following error when you try to enable audit log publishing using the OpenSearch Service console:

The Resource Access Policy specified for the CloudWatch Logs log group does not grant sufficient permissions for Amazon OpenSearch Service to create a log stream. Please check the Resource Access Policy.

If you encounter this error, verify that the `resource` element of your policy includes the correct log group ARN. If it does, take the following steps:

1. Wait several minutes.

2. Refresh the page in your web browser.
3. Choose **Select existing group**.
4. For **Existing log group**, choose the log group that you created before receiving the error message.
5. In the access policy section, choose **Select existing policy**.
6. For **Existing policy**, choose the policy that you created before receiving the error message.
7. Choose **Enable**.

If the error persists after repeating the process several times, contact AWS Support.

Can't close index

OpenSearch Service supports the `_close` API only for OpenSearch and Elasticsearch versions 7.4 and later. If you're using an older version and are restoring an index from a snapshot, you can delete the existing index (before or after reindexing it).

Client license checks

The default distributions of Logstash and Beats include a proprietary license check and fail to connect to the open source version of OpenSearch. Make sure you use the Apache 2.0 (OSS) distributions of these clients with OpenSearch Service.

Request throttling

If you receive persistent `403 Request throttled due to too many requests` or `429 Too Many Requests` errors, consider scaling vertically. Amazon OpenSearch Service throttles requests if the payload would cause memory usage to exceed the maximum size of the Java heap.

Can't SSH into node

You can't use SSH to access any of the nodes in your OpenSearch cluster, and you can't directly modify `opensearch.yml`. Instead, use the console, AWS CLI, or SDKs to configure your domain. You can specify a few cluster-level settings using the OpenSearch REST APIs, as well. To learn

more, see the Amazon OpenSearch Service API Reference and the section called "Supported operations".

If you need more insight into the performance of the cluster, you can publish error logs and slow logs to CloudWatch.

"Not Valid for the Object's Storage Class" snapshot error

OpenSearch Service snapshots do not support the S3 Glacier storage class. You might encounter this error when you attempt to list snapshots if your S3 bucket includes a lifecycle rule that transitions objects to the S3 Glacier storage class.

If you need to restore a snapshot from the bucket, restore the objects from S3 Glacier, copy the objects to a new bucket, and register the new bucket as a snapshot repository.

Invalid host header

OpenSearch Service requires that clients specify `Host` in the request headers. A valid `Host` value is the domain endpoint without `https://`, such as:

```
Host: search-my-sample-domain-ih2lhn2ew2scurji.us-west-2.es.amazonaws.com
```

If you receive an `Invalid Host Header` error when making a request, check that your client or proxy includes the OpenSearch Service domain endpoint (and not, for example, its IP address) in the `Host` header.

Invalid M3 instance type

OpenSearch Service doesn't support adding or modifying M3 instances to existing domains running OpenSearch or Elasticsearch versions 6.7 and later. You can continue to use M3 instances with Elasticsearch 6.5 and earlier.

We recommend choosing a newer instance type. For domains running OpenSearch or Elasticsearch 6.7 or later, the following restriction apply:

- If your existing domain does not use M3 instances, you can no longer change to them.
- If you change an existing domain from an M3 instance type to another instance type, you can't switch back.

Hot queries stop working after enabling UltraWarm

When you enable UltraWarm on a domain, if there are no preexisting overrides to the `search.max_buckets` setting, OpenSearch Service automatically sets the value to `10000` to prevent memory-heavy queries from saturating warm nodes. If your hot queries are using more than 10,000 buckets, they might stop working when you enable UltraWarm.

Because you can't modify this setting due to the managed nature of Amazon OpenSearch Service, you need to open a support case to increase the limit. Limit increases don't require a premium support subscription.

Can't downgrade after upgrade

In-place upgrades are irreversible, but if you contact AWS Support, they can help you restore the automatic, pre-upgrade snapshot on a new domain. For example, if you upgrade a domain from Elasticsearch 5.6 to 6.4, AWS Support can help you restore the pre-upgrade snapshot on a new Elasticsearch 5.6 domain. If you took a manual snapshot of the original domain, you can perform that step yourself.

Need summary of domains for all AWS Regions

The following script uses the Amazon EC2 describe-regions AWS CLI command to create a list of all Regions in which OpenSearch Service could be available. Then it calls list-domain-names for each Region:

```
for region in `aws ec2 describe-regions --output text | cut -f4`
do
    echo "\nListing domains in region '$region':"
    aws opensearch list-domain-names --region $region --query 'DomainNames'
done
```

You receive the following output for each Region:

```
Listing domains in region:'us-west-2'...
[
  {
    "DomainName": "sample-domain"
  }
]
```

Regions in which OpenSearch Service is not available return "Could not connect to the endpoint URL."

Browser error when using OpenSearch Dashboards

Your browser wraps service error messages in HTTP response objects when you use Dashboards to view data in your OpenSearch Service domain. You can use developer tools commonly available in web browsers, such as Developer Mode in Chrome, to view the underlying service errors and assist your debugging efforts.

To view service errors in Chrome

1. From the Chrome top menu bar, choose **View**, **Developer**, **Developer Tools**.
2. Choose the **Network** tab.
3. In the **Status** column, choose any HTTP session with a status of 500.

To view service errors in Firefox

1. From the menu, choose **Tools**, **Web Developer**, **Network**.
2. Choose any HTTP session with a status of 500.
3. Choose the **Response** tab to view the service response.

Node shard and storage skew

Node *shard skew* is when one or more nodes within a cluster has significantly more shards than the other nodes. Node *storage skew* is when one or more nodes within a cluster has significantly more storage (`disk.indices`) than the other nodes. While both of these conditions can occur temporarily, like when a domain has replaced a node and is still allocating shards to it, you should address them if they persist.

To identify both types of skew, run the _cat/allocation API operation and compare the `shards` and `disk.indices` entries in the response:

```
shards    | disk.indices  | disk.used  | disk.avail   | disk.total   | disk.percent |
host      | ip        | node
   264    |      465.3mb  |   229.9mb  |      1.4tb   |      1.5tb   |            0 |
 x.x.x.x  | x.x.x.x   | node1
```

```
   115     |        7.9mb   |    83.7mb   |     49.1gb    |     49.2gb    |            0 |
x.x.x.x   | x.x.x.x   | node2
   264     |      465.3mb   |   235.3mb   |      1.4tb    |      1.5tb    |            0 |
 x.x.x.x   | x.x.x.x   | node3
   116     |        7.9mb   |    82.8mb   |     49.1gb    |     49.2gb    |            0 |
x.x.x.x   | x.x.x.x   | node4
   115     |        8.4mb   |      85mb   |     49.1gb    |     49.2gb    |            0 |
x.x.x.x   | x.x.x.x   | node5
```

While some storage skew is normal, anything over 10% from the average is significant. When shard distribution is skewed, CPU, network, and disk bandwidth usage can also become skewed. Because more data generally means more indexing and search operations, the heaviest nodes also tend to be the most resource-strained nodes, while the lighter nodes represent underutilized capacity.

Remediation: Use shard counts that are multiples of the data node count to ensure that each index is distributed evenly across data nodes.

Index shard and storage skew

Index *shard skew* is when one or more nodes hold more of an index's shards than the other nodes. Index *storage skew* is when one or more nodes hold a disproportionately large amount of an index's total storage.

Index skew is harder to identify than node skew because it requires some manipulation of the _cat/shards API output. Investigate index skew if there's some indication of skew in the cluster or node metrics. The following are common indications of index skew:

- HTTP 429 errors occurring on a subset of data nodes
- Uneven index or search operation queueing across data nodes
- Uneven JVM heap and/or CPU utilization across data nodes

Remediation: Use shard counts that are multiples of the data node count to ensure that each index is distributed evenly across data nodes. If you still see index storage or shard skew, you might need to force a shard reallocation, which occurs with every blue/green deployment of your OpenSearch Service domain.

Unauthorized operation after selecting VPC access

When you create a new domain using the OpenSearch Service console, you have the option to select VPC or public access. If you select **VPC access**, OpenSearch Service queries for VPC information and fails if you don't have the proper permissions:

```
You are not authorized to perform this operation. (Service: AmazonEC2; Status Code:
 403; Error Code: UnauthorizedOperation
```

To enable this query, you must have access to the `ec2:DescribeVpcs`, `ec2:DescribeSubnets`, and `ec2:DescribeSecurityGroups` operations. This requirement is only for the console. If you use the AWS CLI to create and configure a domain with a VPC endpoint, you don't need access to those operations.

Stuck at loading after creating VPC domain

After creating a new domain that uses VPC access, the domain's **Configuration state** might never progress beyond **Loading**. If this issue occurs, you likely have AWS Security Token Service (AWS STS) *disabled* for your Region.

To add VPC endpoints to your VPC, OpenSearch Service needs to assume the `AWSServiceRoleForAmazonOpenSearchService` role. Thus, AWS STS must be enabled to create new domains that use VPC access in a given Region. To learn more about enabling and disabling AWS STS, see the IAM User Guide.

Denied requests to the OpenSearch API

With the introduction of tag-based access control for the OpenSearch API, you might start seeing access denied errors where you didn't before. This might be because one or more of your access policies contains `Deny` using the `ResourceTag` condition, and those conditions are now being honored.

For example, the following policy used to only deny access to the `CreateDomain` action from the configuration API, if the domain had the tag `environment=production`. Even though the action list also includes `ESHttpPut`, the deny statement didn't apply to that action or any other `ESHttp*` actions.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Action": [
      "es:CreateDomain",
      "es:ESHttpPut"
    ],
    "Effect": "Deny",
    "Resource": "*",
    "Condition": {
      "ForAnyValue:StringEquals": {
        "aws:ResourceTag/environment": [
          "production"
        ]
      }
    }
  }]
}
```

With the added support of tags for OpenSearch HTTP methods, an IAM identity-based policy like the above will result in the attached user being denied access to the `ESHttpPut` action. Previously, in the absence of tags validation, the attached user would have still been able to send PUT requests.

If you start seeing access denied errors after updating your domains to service software R20220323 or later, check your identity-based access policies to see if this is the case and update them if necessary to allow access.

Can't connect from Alpine Linux

Alpine Linux limits DNS response size to 512 bytes. If you try to connect to your OpenSearch Service domain from Alpine Linux version 3.18.0 or lower, DNS resolution can fail if the domain is in a VPC and has more than 20 nodes. If you use an Alpine Linux version higher than 3.18.0, you should to be able to resolve more than 20 hosts. For more information, see the Alpine Linux 3.18.0 release notes.

If your domain is in a VPC, we recommend using other Linux distributions, such as Debian, Ubuntu, CentOS, Red Hat Enterprise Linux, or Amazon Linux 2, to connect to it.

Too many requests for Search Backpressure

CPU-based admission control is a gatekeeping mechanism that proactively limits the number of requests to a node based on its current capacity, both for organic increases and spikes in traffic. Excessive requests return an HTTP 429 “Too Many Requests” status code upon rejection. This errors indicates either insufficient cluster resources, resource-intensive search requests, or an unintended spike in the workload.

Search Backpressure provides the reason for rejection, which can help fine-tune resource-intensive search requests. For traffic spikes, we recommend client-side retries with exponential backoff and jitter.

Certificate error when using SDK

Because AWS SDKs use the CA certificates from your computer, changes to the certificates on the AWS servers can cause connection failures when you attempt to use an SDK. Error messages vary, but typically contain the following text:

```
Failed to query OpenSearch
...
SSL3_GET_SERVER_CERTIFICATE:certificate verify failed
```

You can prevent these failures by keeping your computer's CA certificates and operating system up-to-date. If you encounter this issue in a corporate environment and do not manage your own computer, you might need to ask an administrator to assist with the update process.

The following list shows minimum operating system and Java versions:

- Microsoft Windows versions that have updates from January 2005 or later installed contain at least one of the required CAs in their trust list.
- Mac OS X 10.4 with Java for Mac OS X 10.4 Release 5 (February 2007), Mac OS X 10.5 (October 2007), and later versions contain at least one of the required CAs in their trust list.
- Red Hat Enterprise Linux 5 (March 2007), 6, and 7 and CentOS 5, 6, and 7 all contain at least one of the required CAs in their default trusted CA list.
- Java 1.4.2_12 (May 2006), 5 Update 2 (March 2005), and all later versions, including Java 6 (December 2006), 7, and 8, contain at least one of the required CAs in their default trusted CA list.

The three certificate authorities are:

- Amazon Root CA 1
- Starfield Services Root Certificate Authority - G2
- Starfield Class 2 Certification Authority

Root certificates from the first two authorities are available from Amazon Trust Services, but keeping your computer up-to-date is the more straightforward solution. To learn more about ACM-provided certificates, see AWS Certificate Manager FAQs.

> **Note**
>
> Currently, OpenSearch Service domains in the us-east-1 Region use certificates from a different authority. We plan to update the Region to use these new certificate authorities in the near future.

Document history for Amazon OpenSearch Service

This topic describes important changes to Amazon OpenSearch Service. Service software updates add support for new features, security patches, bug fixes, and other improvements. To use new features, you might need to update the service software on your domain. For more information, see the section called “Service software updates”.

Service features are rolled out incrementally to the AWS Regions where a service is available. We update this documentation for the first release only. We don't provide information about Region availability or announce subsequent Region rollouts. For information about Region availability of service features, and to subscribe to notifications about updates, see What's New with AWS?

Relevant dates to this history:

- **Current product version—**2021-01-01
- **Latest product release—**December 9, 2024
- **Latest documentation update—**December 9, 2024

For notifications about updates, you can subscribe to the RSS feed.

> **Note**
>
> **Patch releases**: Service software versions that end in "-P" and a number, such as R20211203-P4, are patch releases. Patches are likely to include performance improvements, minor bug fixes, and security fixes or posture improvements. Since patches do not include new features or breaking changes, they generally do not have direct user or documentation impact, which is why the specifics of each patch are not included in this document history.

Change	Description	Date
Amazon OpenSearch Service zero-ETL integration with CloudWatch Logs and Security Lake	Amazon OpenSearch Service now supports *direct queries* to query data in CloudWatch Logs and Security Lake.	December 1, 2024

kNN indexes	From version 2.17, kNN indexes are eligible to be migrated to UltraWarm and Cold tiers.	November 13, 2024
OpenSearch concurrent segment search	Concurrent segment search is now set to default **auto** mode for domains on version 2.17. Also, there is now support to enable/disable concurrent search at an index level in the managed service. For more information, see concurrent segment search and concurrent segment search at the index or cluster level.	November 13, 2024
OpenSearch 2.17 support	Amazon OpenSearch Service now supports OpenSearch version 2.17. This version includes all features that were part of versions 2.12 and 2.13. For more information, see the 2.17 and 2.17 release notes.	November 13, 2024
Updated `AmazonOpenSearchServiceRolePolicyData`	Added AWS/OpenSearch as a new namespace type to the `cloudwatch:PutMetricData` action for `AmazonOpenSearchServiceRolePolicy` .	October 30, 2024

Added CloudWatch namespaces	Added AWS/OpenSearch as a new namespace type to the `cloudwatch:PutMetricData` action on the the service-linked role page.	October 30, 2024
OpenSearch 2.15 support	Amazon OpenSearch Service now supports OpenSearch version 2.15. This version includes all features that were part of versions 2.12 and 2.13. For more information, see the 2.12 and 2.13 release notes.	October 11, 2024
Update to service-linked role policy	Adds the Sid `AllowAOSSCloudwatchMetrics` to the the service-linked role policy `AmazonOpenSearchServerlessServiceRolePolicy` .	July 12, 2024
New service-linked role	Amazon OpenSearch Service adds a service-linked role called `AWSServiceRoleForOpensearchIngestionSelfManagedVpce` , which allows Amazon OpenSearch Ingestion to send metric data to Amazon CloudWatch for pipelines with self-managed VPC endpoints.	June 12, 2024
Amazon OpenSearch Service zero-ETL integration with Amazon S3	Amazon OpenSearch Service now supports *direct queries* to query data in Amazon S3.	May 22, 2024

OpenSearch 2.13 support	Amazon OpenSearch Service now supports OpenSearch version 2.13. This version includes all features that were part of versions 2.12 and 2.13. For more information, see the 2.12 and 2.13 release notes.	May 21, 2024
Amazon OpenSearch Ingestion support for Data Prepper version 2.7	Amazon OpenSearch Ingestion adds support for Data Prepper version 2.7. For more information, see the 2.7 release notes.	April 4, 2024
AWS service private access for OpenSearch Serverless collections	You can now grant specific AWS services, such as Amazon Bedrock, access to your OpenSearch Serverless collections within a network access policy.	March 28, 2024
In-place EBS updates	You can now make some EBS changes to your domains without a blue/green deployment in Amazon OpenSearch Service.	February 14, 2024
Configuration change visibility	You can now track domain configuration changes in the Amazon OpenSearch Service console and using the configuration API.	February 6, 2024

Vector search collections general availability	Amazon OpenSearch Serverless *vector search* collections are now generally available. The following notable improvements were made during the preview phase: • Vector search collections now supports workloads with billion of vectors, each with up to 128 dimensions. • OpenSearch Dashboards now supports vector search collections.	November 29, 2023
OR1 instances	Amazon OpenSearch Service now supports OR1 instance types.	November 29, 2023
Direct queries with Amazon S3 (preview)	Direct queries provide a fully managed solution for making transactional data availabl e in Amazon OpenSearch Service within seconds of it being written to an Amazon S3 bucket.	November 29, 2023

10 TiB capacity for time series collections	Amazon OpenSearch Serverless adds support for up to 10 TiB of index data for *time series* collections. This release also supports a maximum allowed capacity of 200 OCUs for all types of collections and the ability to disable standby replicas when you create a collection.	November 29, 2023
OpenSearch 2.11 support	Amazon OpenSearch Service now supports OpenSearch version 2.11. This version includes all features that were part of versions 2.10 and 2.11. For more information, see the 2.10 and 2.11 release notes.	November 17, 2023
Amazon OpenSearch Ingestion support for Data Prepper version 2.6	Amazon OpenSearch Ingestion adds support for Data Prepper version 2.6. For more information, see the 2.6 release notes. In addition, you can specify Amazon DynamoDB as a pipeline source. For more information, see Using an OpenSearch Ingestion pipeline with Amazon DynamoDB.	November 17, 2023

Amazon OpenSearch Ingestion support for Data Prepper version 2.5	Amazon OpenSearch Ingestion adds support for Data Prepper version 2.5. For more information, see the 2.5 release notes. In addition, you can now specify an OpenSearch Service domain or OpenSearch Serverless collection as a pipeline source. For more information, see the OpenSearch source plugin in the Data Prepper documentation.	November 17, 2023
CloudFormation template for remote inference	To ease the setup of remote inference for semantic search, Amazon OpenSearch Service provides an AWS CloudFormation template in the console that automates the model provisioning process for you.	November 7, 2023
Update to service-linked role policy	Adds the permissions necessary for the service-linked role policy `AmazonOpenSearchServiceRolePolicy` to assign and unassign IPv6 addresses. The deprecated Elasticsearch policy `AmazonElasticsearchServiceRolePolicy` has also been updated to ensure backwards compatibility.	October 26, 2023

Amazon OpenSearch Serverless lifecycle policies	Amazon OpenSearch Serverless introduces index lifecycle policies to streamline the management of data retention and deletion. You can now use APIs or a configuration interface in the console to set data retention polices for *time series* collections, eliminating the need for creating daily indexes or scripts to delete old data.	October 25, 2023
Im4gn instance support	Amazon OpenSearch Service now supports Im4gn instance types. Im4gn instances are optimized for workloads that manage large datasets and need high storage density per vCPU.	October 20, 2023
Administrative options	Amazon OpenSearch Service now offers several administrative options that provide granular control if you need to troubleshoot issues with your domain. These options include the ability to restart the OpenSearch process on a data node and the ability to restart a data node.	October 17, 2023

Optional plugins	Amazon OpenSearch Service adds support for four new language analyzer plugins: Nori (Korean), Sudachi (Japanese), Pinyin (Chinese) , and STConvert Analysis (Chinese), as well as the Amazon Personalize Search Ranking plugin.	October 16, 2023
OpenSearch 2.9 support	Amazon OpenSearch Service now supports OpenSearc h version 2.9. This version includes all features that were part of versions 2.8 and 2.9. For more information, see the 2.8 and 2.9 release notes.	October 2, 2023
ML connectors	Amazon OpenSearch Service adds support for machine learning (ML) connectors. Connectors facilitate access to ML models hosted on other AWS services, or on third-party machine learning (ML) platforms.	September 6, 2023
Amazon OpenSearch Ingestion adds support for Data Prepper version 2.4	Amazon OpenSearch Ingestion adds support for Data Prepper version 2.4. For more information, see the 2.4 release notes. In addition, you can now specify Amazo n Managed Streaming for Apache Kafka (Amazon MSK) as a pipeline source.	August 31, 2023

6 TiB capacity for time series collections	Amazon OpenSearch Serverless adds support for up to 6 TiB of index data for *time series* collections. This release also supports a maximum allowed capacity of 100 OCUs for both *search* and *time series* collections.	August 15, 2023
Vector search collections	Amazon OpenSearch Serverless adds the option to create a *vector search* collection, which you can use to store vector embeddings to power similiarity and semantic searches.	July 26, 2023
OpenSearch 2.7 support	Amazon OpenSearch Service now supports OpenSearch version 2.7. This version includes all features that were part of versions 2.6 and 2.7. For more information, see the 2.6 and 2.7 release notes.	July 10, 2023
Data Prepper 2.3 support	Amazon OpenSearch Ingestion adds support Data Prepper version 2.3. For more information, see the 2.3 release notes. In addition, you can now specify Amazon Security Lake as a pipeline source.	June 26, 2023

Multi-AZ with Standby	Amazon OpenSearch Service adds the option to deploy a domain across three Availability Zones (AZs), with each AZ containing a complete copy of data and with nodes in one of these AZs acting as a standby. The Multi-AZ with Standby deployment option provides 99.99% availability and consistent performance in the event of an infrastructure failure.	May 3, 2023
New service-linked role	Amazon OpenSearch Service adds a service-linked role called `AWSServiceRoleForAmazonOpenSearchIngestionService`, which allows Amazon OpenSearch Ingestion to send metric data to Amazon CloudWatch.	April 26, 2023

Amazon OpenSearch Ingestion	Amazon OpenSearch Ingestion is a fully managed data collector that delivers real-time log and trace data to OpenSearch Service domains and OpenSearch Serverless collections. OpenSearch Ingestion eliminates the need for you to use third-party solutions like Logstash or Jaeger to ingest data into your domains and collections.	April 26, 2023
OpenSearch 2.5 support	Amazon OpenSearch Service now supports OpenSearch version 2.5. This version includes all features that were part of versions 2.4 and 2.5. For more information, see the 2.4 and 2.5 release notes.	March 13, 2023

Off-peak maintenance windows	Amazon OpenSearch Service adds off-peak windows, which are daily 10-hour, low-traffic time blocks during which it can schedule service software updates and Auto-Tune optimizations that require a blue/green deployment. Off-peak updates help to minimize strain on a cluster's dedicated master nodes during higher traffic periods. For new domains created after February 16, the off-peak window is automatically configured for between 10:00 P.M. and 8:00 A.M. local time. For existing domains, you need to explicitly enable the window.	February 16, 2023
Configure SAML authentication during domain creation	Amazon OpenSearch Service now supports configuring SAML authentication during domain creation. Previously, you had to configure SAML options after the domain was already created.	February 1, 2023

Remote reindex for VPC domains	Amazon OpenSearch Service adds the option for a VPC endpoint connection between two domains. You can now use remote reindex to copy indexes from one VPC domain to another without a reverse proxy. Your VPC domains must be running service software R20221114 or later to use this feature.	January 31, 2023

Amazon OpenSearch Serverless general availability	Amazon OpenSearch Serverless is now generally available. The following notable improvements were made during the preview phase: • Capacity can now scale *down* to the minimum configured OCUs when there's a decrease in traffic on the collection endpoint. • The maximum allowed OCUs for both indexing and search was increased from 20 to 50. Each OCU includes enough hot ephemeral storage for 120 GiB of index data. • You can now configure data access settings while creating collections, rather than having to configure them in a separate workflow.	January 25, 2023
Async dry run	Amazon OpenSearch Service now supports async dry run, which allows you to perform a validation check prior to making a configuration change, and notifies you if your changes will cause a blue/green deployment.	January 19, 2023

New service-linked role	Amazon OpenSearch Service adds a service-linked role called `AWSServiceRoleForAmazonOpenSearchServerless` , which allows OpenSearch Serverless to send metric data to Amazon CloudWatch.	November 29, 2022
Amazon OpenSearch Serverless preview	Amazon OpenSearch Serverless is an on-demand, auto scaling, serverless configuration for Amazon OpenSearch Service. Serverless removes the operational complexities of provisioning, configuring, and tuning your OpenSearch clusters.	November 29, 2022
OpenSearch 2.3 support	Amazon OpenSearch Service now supports OpenSearch version 2.3. This version includes all features that were part versions 2.0, 2.1, and 2.2. For more information, see the 2.0, 2.1, 2.2, and 2.3 release notes. Version 2.3 contains a **breaking change**. For more information, see Supported upgrade paths.	November 15, 2022

Notifications plugin support	Amazon OpenSearch Service now supports the Notifications plugin, which offers a central location for all of your notifications from OpenSearch plugins. Starting with version 2.0, alerting destinations were deprecated and replaced with notification *channels*.	November 15, 2022
Kibana 7.1.1 support	Amazon OpenSearch Service domains running Elasticsearch 7.1 now support the latest patch release for Kibana 7.1.1, which adds bug fixes and improves security. When you update your 7.1 domains to service software R20221114, OpenSearch Service will automatically upgrade them to this patch release.	November 15, 2022
Kibana 6.8.13 support	Amazon OpenSearch Service domains running Elasticsearch 6.8 now support the latest patch release for Kibana 6.8.13, which adds bug fixes and improves security. When you update your 6.8 domains to service software R20221114, OpenSearch Service will automatically upgrade them to this patch release.	November 15, 2022

Kibana 6.3.2 support	Amazon OpenSearch Service domains running Elasticsearch 6.3 now support the latest patch release for Kibana 6.3.2, which adds bug fixes and improves security. When you update your 6.3 domains to service software R20221114, OpenSearch Service will automatically upgrade them to this patch release.	November 15, 2022
AWS PrivateLink	With Amazon OpenSearch Service-managed VPC endpoints, you can connect directly to OpenSearch Service VPC domains by using an interface VPC endpoint instead of connecting over the internet. An OpenSearch Service-managed VPC endpoint is accessible only within the VPC where the endpoint is provisioned, or from any VPCs peered with the VPC where the endpoint is provisioned, as permitted by the route tables and security groups. Your VPC domain must be running service software R20220928 or later to connect to an interface VPC endpoint.	November 7, 2022

Bug fixes and performance improvements	Service software R20220928 includes bug fixes and performance enhancements, including improved SAML logging. The update also changes the default tenant to `Global` rather than `Private`.	October 3, 2022
Improved API reference	Amazon OpenSearch Service offers an improved, all-encompassing configuration API reference. The new references contains all available actions and data types, sample request and response syntax, and links to the corresponding SDK references for all supported languages.	September 13, 2022
Blue/green validation	Amazon OpenSearch Service now performs a validation check prior to blue/green deployments, and surfaces validation errors if your domain is not eligible for an update.	August 16, 2022
OpenSearch 1.3 support	Amazon OpenSearch Service now supports OpenSearch version 1.3. For more information, see the 1.3 release notes.	July 27, 2022

ML Commons plugin support	Amazon OpenSearch Service adds support for the ML Commons plugin, which provides a set of common machine learning algorithms through transport and REST API calls. You can also interact with the ML Commons plugin through PPL commands.	July 27, 2022
gp3 volume support	Amazon OpenSearch Service adds support for the gp3 EBS General Purpose SSD volume type. You can specify additional provisioned IOPS and throughput when you create or modify the domain.	July 26, 2022
Enhanced best practices documentation	The Amazon OpenSearch Service documentation provides improved operational best practices and general recommendations for creating and operating OpenSearch Service domains.	July 6, 2022
Integration with Service Quotas	You can now view quotas for Amazon OpenSearch Service, and request quota increases, from the Service Quotas console.	June 29, 2022

Tag-based access control for the OpenSearch API	You can now use tags to control access to the OpenSearch APIs. Previously, you could only use tags to control access to the configuration API.	June 16, 2022
Cross-cluster search across Regions	Cross-cluster search is now supported across AWS Regions as long as both domains are running Elasticsearch version 7.10 or later, or any version of OpenSearch.	June 14, 2022
Single Kibana 5.6 support	Amazon OpenSearch Service adds support for single Kibana 5.6.16. With single Kibana 5.6.16, you can use Kibana 5.6 as your front end while connecting to Elasticsearch versions 5.1, 5.3, 5.5, and 5.6. You must be on service software R20220323 or later to use single Kibana 5.6.	April 4, 2022
R20220323-P1	Amazon OpenSearch Service recently released service software update R20220323, but the update was subsequently rolled back because of an issue. We recommend that you update your domains to patch release R20220323-P1 or later, which fixes the issue.	April 4, 2022

OpenSearch 1.2 support	Amazon OpenSearch Service now supports OpenSearch version 1.2. For more information, see the 1.2 release notes.	April 4, 2022
Observability	The default installation of OpenSearch Dashboards for Amazon OpenSearch Service includes the Observability plugin, which you can use to visualize data-driven events using Piped Processing Language (PPL) to explore and query your data. The plugin requires OpenSearch 1.2 or later and service software R20220323 or later.	April 4, 2022
Kibana 7.7.1 support	Amazon OpenSearch Service domains running Elasticsearch 7.7 now support the latest patch release for Kibana 7.7, which adds bug fixes and improves security. When you update your 7.7 domains to service software R20220323 or later, OpenSearch Service will automatically upgrade them to this patch release.	April 4, 2022

JVM memory pressure metric changes	Amazon OpenSearch Service changed the logic for the `JVMMemoryPressure` CloudWatch metrics to more accurately reflect memory utilization. Previously, the metrics only considered the old generation memory pool of JVM heap. With this change, the metric also considers the young generation memory pool. After you update your domain to service software R20220323, you might see an increase in the `JVMMemoryPressure`, `MasterJVMMemoryPressure`, and/or `WarmJVMMemoryPressure` metrics.	April 4, 2022
Custom dictionaries with the IK (Chinese) Analysis plugin	Amazon OpenSearch Service now supports using custom dictionaries with the IK (Chinese) Analysis plugin.	April 4, 2022

Cross-cluster replication on existing domains	Amazon OpenSearch Service removed the limitation that you can only implement cross-cluster search and cross-cluster replication on domains created on or after June 3rd, 2020. You can now enable these features on all domains regardless of when they were created. Both domains must be on service software R20220323 or later.	April 4, 2022
Blue/green deployment visibility	Amazon OpenSearch Service now offers more visibility into the progress of blue/green deployments. You can monitor these details in the console or using the configuration API.	January 27, 2022
Fine-grained access control on existing domains	You can now enable fine-grained access control on existing domains. You can enable a temporary migration period for open/IP-based access policies to ensure that users can continue to access your domain while you create and map roles. Enabling fine-grained access control on existing domains requires service software R20211203 or later.	January 6, 2022

Renamed OpenSearch Dashboards roles	With service software R20211203, the `kibana_user` role was renamed to `opensearch_dashboards_user`, and `kibana_read_only` was renamed to `opensearch_dashboards_read_only`. This change applies to all *newly-created* OpenSearch 1.*x* domains. For existing OpenSearch domains that you upgrade to service software R20211203, the roles remain the same.	January 4, 2022
OpenSearch 1.1 support	Amazon OpenSearch Service now supports OpenSearch version 1.1. For more information, see the 1.1 release notes.	January 4, 2022
ISM visual editor	The default installation of OpenSearch Dashboards for Amazon OpenSearch Service now supports the visual editor for ISM policies. This feature requires OpenSearch 1.1 or later.	January 4, 2022

Cross-service confused deputy prevention update	Amazon OpenSearch Service supports using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in IAM resource policies to prevent the confused deputy problem. You must be on service software R20211203 or later to use these condition keys.	January 4, 2022

Log4j patch	Service software R20211203-P2 updates the version of Log4j used in OpenSearch Service as recommended by the advisories in CVE-2021-44228 and CVE-2021-45046. The patch applies to domains running all versions of OpenSearch and Elasticsearch. OpenSearch Service will continue to update various Log4j versions internally, and they will not necessarily be restricted to the latest version of Log4j. The Log4j version on your domain depends on the software version that the domain is running. However, irrespective of the Log4j version, as long as you're running R20211203-P2 or later, your domains contain the Log4j update required to address CVE-2021-44228 and CVE-2021-45046.	December 15, 2021
Cross-cluster replication	Cross-cluster replication lets you replicate indices, mappings, and metadata from one OpenSearch Service domain to another. Cross-cluster replication requires a domain running Elasticsearch 7.10 or OpenSearch 1.1 or later.	October 5, 2021

New AWS-managed policies	The launch of Amazon OpenSearch Service includes new AWS-managed policies and the deprecation of old policies.	September 8, 2021
Kibana 6.4.3 support	Amazon OpenSearch Service domains running legacy Elasticsearch version 6.4 now support the latest patch release for Kibana 6.4, which adds bug fixes and improves security. OpenSearch Service will automatically upgrade domains to this patch release.	September 8, 2021
Data streams	Amazon OpenSearch Service adds support for data streams, which simplify the process of managing time-series data. Your domain must be running OpenSearch 1.0 or later to use data streams.	September 8, 2021

Amazon OpenSearch Service	AWS renames Amazon OpenSearch Service to remove the legacy "Elastics earch" branding. Amazon OpenSearch Service suppo rts OpenSearch and legacy Elasticsearch OSS. When you create a cluster, you have the option of which search engine to use. OpenSearch Service offers broad compatibility with Elasticsearch OSS 7.10, the final open source version of the software.	September 8, 2021
Cold storage	Cold storage is a new storage tier for infrequently accessed or historical data. Cold indices only occupy S3 storage and have no compute attached to them. Cold storage requires a domain running Elasticsearch 7.9 or later and service softw are R20210426 or later.	May 13, 2021
ARM-based Graviton instances	Amazon OpenSearch Service now supports ARM-based Graviton instance types (M6G, C6G, R6G, and R6GD). Graviton instance types are available on new and existing domains running Elasticsearch 7.9 or later and service software R20210331 or later.	May 4, 2021

ISM templates	Amazon OpenSearch Service adds support for ISM templates, which let you automatically attach an ISM policy to an index if the index matches a pattern defined in the policy. ISM templates require service software R20210426 or later. This update also deprecates the `policy_id` setting, meaning you can no longer use index templates to apply ISM policies to newly created indices. The update introduces a breaking change for existing CloudFormation templates using this setting.	April 27, 2021
Elasticsearch 7.10 support	Amazon OpenSearch Service now supports Elasticsearch version 7.10. For more information, see 7.10 release notes.	April 21, 2021
Asynchronous search	Amazon OpenSearch Service now supports asynchronous search, which lets you run search requests in the background. Asynchronous search requires a domain running Elasticsearch 7.10 or later and service software R20210331 or later.	April 21, 2021

Tag-based access control for the configuration API	You can now use AWS tags to control access to the Amazon ES configuration API.	March 2, 2021
Auto-Tune	Amazon OpenSearch Service adds Auto-Tune, which uses performance and usage metrics from your cluster to suggest changes to the JVM settings on your nodes. Auto-Tune requires a domain running Elasticsearch 6.7 or later and service software R20201117 or later.	February 24, 2021
Trace Analytics	The default installation of Kibana for Amazon OpenSearch Service now includes the trace analyti cs plugin, which lets you monitor trace data from your distributed applications. The plugin requires a domain running Elasticsearch 7.9 or later and service software R20210201 or later.	February 17, 2021

Shard metrics	Amazon OpenSearch Service adds the following CloudWatch metrics for tracking shard status: `Shards.active` , `Shards.unassigned` , `Shards.delayedUnassigned` `Shards.activePrimary` , `Shards.initializing` , `Shards.relocating` . The metrics are available on domains running service software R20210201 or later.	February 17, 2021
Kibana reports	The default installation of Kibana for Amazon OpenSearch Service now supports on-demand reports for the Discover, Visualize , and Dashboard pages. This feature requires Elasticsearch 7.9 or later and service software R20210201 or later.	February 17, 2021
Kibana 5.6.16 support	Amazon OpenSearch Service domains running Elasticsearch 5.6 now support the latest patch release for Kibana 5.6, which adds bug fixes and improves security. Amazon ES will automatically upgrade domains to this patch release.	February 17, 2021

Encryption for existing domains	Amazon OpenSearch Service now supports enabling encryption of data at rest and node-to-node encryption on existing domains running Elasticsearch 6.7 or later. After you enable these settings, you can't disable them.	January 27, 2021
Remote reindex	Amazon OpenSearch Service now supports remote reindex, which lets you migrate indices from remote domains. This feature requires service software R20201117 or later.	November 24, 2020
Piped Processing Language	Amazon OpenSearch Service now supports Piped Processin g Language (PPL), a query language that lets you use pipe (|) syntax to query data stored in Elasticsearch. This feature requires service software R20201117 or later. To learn more, see .	November 24, 2020
Kibana notebooks	Amazon OpenSearch Service adds support for Kibana notebooks, which lets you combine live visualizations and narrative text in a single interface. This feature requires service software R20201117 or later.	November 24, 2020

Gantt charts	The default installation of Kibana for Amazon OpenSearch Service now supports a new visualization type, Gantt charts. This feature requires service software R20201117 or later.	November 24, 2020
Elasticsearch 7.9 support	Amazon OpenSearch Service now supports Elasticsearch version 7.9. For more information, see 7.9 release notes.	November 24, 2020
Anomaly detection updates	Anomaly detection for Amazon OpenSearch Service adds support for high cardinality, which lets you categorize anomalies with a dimension like IP address, product ID, country code, and so on. This feature requires service software R20201117 or later.	November 24, 2020
Dynamic dictionary updates	Amazon OpenSearch Service now lets you update your search analyzers without reindexing. You can update the dictionary files on some or all of your domains, and Amazon ES tracks package versions over time so that you have a history of what changed and when. This feature requires service software R20201019 or later.	November 17, 2020

Custom endpoints	Amazon OpenSearch Service now supports custom endpoints, which let you give your Amazon ES domain a new URL. If you ever swap domains, you can maintain the same URL. This feature requires service software R20201019 or later.	November 5, 2020
New language plugins	Amazon OpenSearch Service now supports IK (Chinese) Analysis, Vietnamese Analysis, and Thai Analysis plugins on domains running Elasticsearch 7.7 or later with service softw are R20201019 or later.	October 28, 2020
Elasticsearch 7.8 support	Amazon OpenSearch Service now supports Elasticse arch version 7.8. For more information, see 7.8 release notes.	October 28, 2020
SAML authentication for Kibana	Amazon OpenSearch Service now supports SAML authentic ation for Kibana, which lets you use third-party identity providers to log in to Kibana, manage fine-grained access control, search your data, and build visualizations. This feature requires service software R20201019 or later.	October 27, 2020

T3 instances	Amazon OpenSearch Service now supports the `t3.small` and `t3.medium` instance types.	September 23, 2020
Audit logs	Amazon OpenSearch Service now supports audit logs for your data, which lets you track failed login attempts, user access to indices, documents, and fields, and much more. This feature requires service software R20200910 or later.	September 16, 2020
UltraWarm updates	UltraWarm for Amazon OpenSearch Service adds new metrics, new settings, a larger migration queue, and a cancellation API. These updates require service software R20200910 or later. For more information, see .	September 14, 2020
Learning to Rank	Amazon OpenSearch Service now supports the open source Learning to Rank plugin, which lets you use machine learning technologies to improve search relevance. This feature requires service software R20200721 or later.	July 27, 2020

k-NN cosine similarity	k-Nearest Neighbor (k-NN) now lets you search for "nearest neighbors" by c osine similarity in addition to Euclidean distance. This feature requires service software R20200721 or later.	July 23, 2020
gzip compression	Amazon OpenSearch Service now supports gzip compressi on for most HTTP requests and responses, which can reduce latency and conserve bandwidth. This feature requires service software R20200721 or later.	July 23, 2020
Elasticsearch 7.7 support	Amazon OpenSearch Service now supports Elasticse arch version 7.7. For more information, see 7.7 release notes.	July 23, 2020
Kibana map service	The default installation of Kibana for Amazon OpenSearch Service now includes a WMS map server, except for domains in the India and China Regions.	June 18, 2020
SQL improvements	SQL support for Amazon OpenSearch Service now supports many new operation s, a dedicated Kibana user interface for data explorati on, and an interactive CLI. For more information, see .	June 3, 2020

Cross-cluster search	Amazon OpenSearch Service lets you perform cross-clu ster queries and aggregations across multiple connected domains.	June 3, 2020
Anomaly detection	Amazon OpenSearch Service lets you automatically detect anomalies in near-real time.	June 3, 2020
UltraWarm	UltraWarm storage for Amazon OpenSearch Service has left public preview and is now generally available. The feature now supports a wider range of versions and AWS Regions. For more informati on, see .	May 5, 2020
Custom dictionaries	Amazon OpenSearch Service lets you upload custom dictionary files for use with your cluster. These files improve your search results by telling Elasticsearch to ignore certain high-frequency words or to treat terms as equivalent.	April 21, 2020
Elasticsearch 7.4 Support	Amazon OpenSearch Service now supports Elasticse arch version 7.4. For more information, see Supported versions.	March 12, 2020

k-NN	Amazon OpenSearch Service adds support for k-Nearest Neighbor (k-NN) search. k-NN requires service software R20200302 or later.	March 3, 2020
Index State Management	Amazon OpenSearch Service adds Index State Management (ISM), which lets you automate routine tasks, such as deleting indices when they reach a certain age. This feature requires service software R20200302 or later.	March 3, 2020
Elasticsearch 5.6.16 support	Amazon OpenSearch Service now supports the latest patch release for version 5.6, which adds bug fixes and improves security. Amazon ES will automatically upgrade existing 5.6 domains to this release. Note that this Elasticsearch release incorrectly reports its version as 5.6.17.	March 2, 2020
Fine-grained access control	Amazon OpenSearch Service now supports fine-grained access control, which offers security at the index, document, and field level, Kibana multi-tenancy, and optional HTTP basic authentication for your cluster.	February 11, 2020

UltraWarm storage (preview)	Amazon OpenSearch Service adds UltraWarm, a new warm storage tier that uses Amazon S3 and a sophisticated caching solution to improve performance. For indices that you are not actively writing to and query less frequentl y, UltraWarm storage offers significantly lower costs per GiB.	December 3, 2019
Encryption features for China Regions	Encryption of data at rest and node-to-node encryptio n are now available in the `cn-north-1` China (Beijing) Region and `cn-northw est-1` China (Ningxia) Region.	November 20, 2019
Require HTTPS	You can now require that all traffic to your Amazon ES domains arrive over HTTPS. When configuring your domain, check the **Require HTTPS** box. This feature requires service software R20190808 or later.	October 3, 2019
Elasticsearch 7.1 and 6.8 support	Amazon OpenSearch Service now supports Elasticsearch version 7.1 and 6.8. For more information, see Supported versions.	August 13, 2019

Hourly snapshots	Rather than daily snapshots, Amazon OpenSearch Service now takes hourly snapshots of domains running Elasticsearch 5.3 and later so that you have more frequent backups from which to restore your data.	July 8, 2019
Elasticsearch 6.7 support	Amazon OpenSearch Service now supports Elasticsearch version 6.7. For more information, see Supported versions.	May 29, 2019
SQL support	Amazon OpenSearch Service now lets you query your data using SQL. SQL support requires service software R20190418 or later.	May 15, 2019
5-series instance types	Amazon OpenSearch Service now supports M5, C5, and R5 instance types. Compared to previous-generation instance types, these new types offer better performance at lower prices. For more information, see Limits.	April 24, 2019
Elasticsearch 6.5 support	Amazon OpenSearch Service now supports Elasticsearch version 6.5.	April 8, 2019

Alerting	Alerting for Amazon OpenSearch Service notifies you when data from one or more Amazon ES indices meets certain conditions. Alerting requires service software R20190221 or later.	March 25, 2019
Three Availability Zone support	Amazon OpenSearch Service now supports three Availabil ity Zones in many Regions. This release also includes a streamlined console exper ience. This multi-AZ requires service software R20181023 or later.	February 7, 2019
Elasticsearch 6.4 support	Amazon OpenSearch Service now supports Elasticsearch version 6.4.	January 23, 2019
200-node clusters	Amazon ES now lets you create clusters with up to 200 data nodes for a total of 3 PB of storage.	January 22, 2019
Service software updates	Amazon ES now lets you manually update the service software for your domain in order to benefit from new features more quickly or update at a low traffic time. To learn more, see .	November 20, 2018

New CloudWatch metrics	Amazon ES now offers node-level metrics and new **Cluster health** and **Instance health** tabs in the Amazon ES console.	November 20, 2018
China (Beijing) support	Amazon OpenSearch Service is now available in the cn-north-1 Region, where it s upports the M4, C4, and R4 instance types.	October 17, 2018
Node-to-node encryption	Amazon OpenSearch Service now supports node-to-node encryption, which keeps your data encrypted as Amazon ES distributes it throughout your cluster.	September 18, 2018
In-place version upgrades	Amazon OpenSearch Service now supports in-place version upgrades.	August 14, 2018
Elasticsearch 6.3 and 5.6 support	Amazon OpenSearch Service now supports Elasticsearch version 6.3 and 5.6.	August 14, 2018
Error logs	Amazon ES now lets you publish Elasticsearch error logs to Amazon CloudWatch.	July 31, 2018
China (Ningxia) Reserved Instances	Amazon ES now offers Reserved Instances in the China (Ningxia) Region.	May 29, 2018
Reserved Instances	Amazon ES now offers support for Reserved Instances.	May 7, 2018

Earlier updates

The following table describes important changes Amazon ES before May 2018.

Change	Description	Date
Amazon Cognito Authentication for Kibana	Amazon ES now offers login page protection for Kibana. To learn more, see the section called "Amazon Cognito authentication for OpenSearch Dashboards".	April 2, 2018
Elasticsearch 6.2 Support	Amazon OpenSearch Service now supports Elasticsearch version 6.2.	March 14, 2018
Korean Analysis Plugin	Amazon ES now supports a memory-optimized version of the Seunjeon Korean analysis plugin.	March 13, 2018
Instant Access Control Updates	Changes to the access control policies on Amazon ES domains now take effect instantly.	March 7, 2018
Petabyte Scale	Amazon ES now supports I3 instance types and total domain storage of up to 1.5 PB. To learn more, see the section called "Petabyte scale".	19 December 2017
Encryption of Data at Rest	Amazon ES now supports encryption of data at rest. To learn more, see the section called "Encryption at rest".	December 7, 2017
Elasticsearch 6.0 Support	Amazon ES now supports Elasticsearch version 6.0. For migration considerations and instructions, see the section called "Upgrading domains".	December 6, 2017
VPC Support	Amazon ES now lets you launch domains within an Amazon Virtual Private Cloud. VPC support provides an additional layer of security and simplifies communic ations between Amazon ES and other services within a VPC. To learn more, see the section called "VPC support".	October 17, 2017
Slow Logs Publishin g	Amazon ES now supports the publishing of slow logs to CloudWatch Logs. To learn more, see the section called "Monitoring logs".	October 16, 2017

Change	Description	Date
Elasticsearch 5.5 Support	Amazon ES now supports Elasticsearch version 5.5. You can now restore automated snapshots without contacting Support and store scripts using the `_scripts` API.	September 7, 2017
Elasticsearch 5.3 Support	Amazon ES added support for Elasticsearch version 5.3.	June 1, 2017
More Instances and EBS Capacity per Cluster	Amazon ES now supports up to 100 nodes and 150 TB EBS capacity per cluster.	April 5, 2017
Canada (Central) and EU (London) Support	Amazon ES added support for the following Regions: Canada (Central), ca-central-1, and EU (London), eu-west-2.	March 20, 2017
More Instances and Larger EBS Volumes	Amazon ES added support for more instances and larger EBS volumes.	February 21, 2017
Elasticsearch 5.1 Support	Amazon ES added support for Elasticsearch version 5.1.	January 30, 2017
Support for the Phonetic Analysis Plugin	Amazon ES now provides built-in integration with the Phonetic Analysis plugin, which allows you to run “sounds-like” queries on your data.	December 22, 2016
US East (Ohio) Support	Amazon ES added support for the following Region: US East (Ohio), us-east-2.	October 17, 2016
New Performance Metric	Amazon ES added a performance metric, `ClusterUsedSpace`.	July 29, 2016
Elasticsearch 2.3 Support	Amazon ES added support for Elasticsearch version 2.3.	July 27, 2016

Change	Description	Date
Asia Pacific (Mumbai) Support	Amazon ES added support for the following Region: Asia Pacific (Mumbai), ap-south-1.	June 27, 2016
More Instances per Cluster	Amazon ES increased the maximum number of instances (instance count) per cluster from 10 to 20.	May 18, 2016
Asia Pacific (Seoul) Support	Amazon ES added support for the following Region: Asia Pacific (Seoul), ap-northeast-2.	January 28, 2016
Amazon ES	Initial release.	October 1, 2015

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.