Implementation Guide

Migration Assistant for Amazon OpenSearch Service

Migration Assistant for Amazon OpenSearch Service: Implementation Guide

Table of Contents

Build an environment to upgrade, migrate, and compare OpenSearch clusters

Publication date: *November 2023*

OpenSearch is widely adopted for log analytics and search functionalities. However, self-managing OpenSearch can be operationally demanding. Amazon OpenSearch Service and Amazon OpenSearch Service Serverless offer more manageable alternatives, but transitioning to these services or updating to the latest OpenSearch version has historically been complex. Also, it can be difficult for a customer to predict the outcome of a migration. The Migration Assistant for Amazon OpenSearch Service solution addresses these challenges - it simplifies the migration process, ensures integrity, and validates performance post-migration.

The Migration Assistant for Amazon OpenSearch Service solution is a toolkit designed to ease the transition to OpenSearch, facilitate upgrades to the latest OpenSearch versions, and refine cluster configurations based on observed traffic patterns. Whether you're looking to set up a proof-of-concept in AWS, transition production workloads with confidence, or enhance your current OpenSearch clusters, this guide provides references to step-by-step instructions, best practices, and insights to leverage the full potential of the OpenSearch migrations package.

Benefits of using this solution:

- Migrate historical data from legacy clusters to OpenSearch clusters, including Amazon OpenSearch Service (AOS) Domains and Amazon OpenSearch Service Serverless (AOSS) collections.
- Intercept and redirect live traffic from self-managed Elasticsearch or OpenSearch clusters to Amazon OpenSearch Service domains and Amazon OpenSearch Service Serverless collections with minimal latency.
- Replicate production traffic on target clusters to validate and ensure accuracy.
- Simulate real-world traffic by capturing and replaying request patterns to fine-tune system performance.
- Deploy across the most common AWS Regions for global reach and scalability.
- Provides a recommended path for migration while continuing to maintain service availability.

Note

Currently, the Migration Assistant for Amazon OpenSearch Service solution supports migrating from Elasticsearch versions 7.0 to 7.10.2 to OpenSearch and Serverless versions 1.x and 2.x.

This implementation guide provides an overview of the Migration Assistant for Amazon OpenSearch Service solution, its reference architecture and components, considerations for planning the deployment, and configuration steps for deploying the solution to the Amazon Web Services (AWS) Cloud. It also references the solution's open-source documentation on GitHub, which includes a User guide, developer documentation, and tips to enhance and contribute to the solution.

The intended audience for using this solution's features and capabilities in their environment includes solution architects, business decision makers, DevOps engineers, data scientists, and cloud professionals.

Use this navigation table to quickly find answers to these questions:

If you want to . . .	Read . . .
Know the cost for running this solution. The estimated cost for running this solution in the US East (N. Virginia) Region is USD $2754 for a 1 TB migration in 1 week.	Cost
Understand the security considerations for this solution.	Security
Know how to plan for quotas for this solution.	Quotas
Know which AWS Regions support this solution.	Supported AWS Regions
View or download the AWS CloudForm ation template included in this solution	AWS CloudFormation template

If you want to . . .	Read . . .
to automatically deploy the infrastructure resources (the "stack") for this solution.	

Features and benefits

The solution provides the following features:

Historical backfill with capture and restore

This solution guides users through the process of transferring data from an originating (source) cluster to a designated (target) cluster.

Live traffic capture and replay

The solution offers guidance and tools to intercept traffic intended for an original cluster and archive it for future replay on a destination cluster. Typically, the replay occurs at the same rate and concurrency as the original traffic to precisely mimic the workload experienced by the source cluster. Users can choose to replay the recorded traffic subsequently or adjust the replay speed. This flexibility enables users to fine-tune the target cluster, enhancing its performance to suit their requirements.

Traffic verification

The solution records requests and responses between the source and destination clusters for comparison. It then forwards the latency metrics and response codes to an analytics platform, enabling users to analyze the data essential for transitioning their traffic from a legacy system to a new Amazon OpenSearch Service destination.

Integration with AWS Service Catalog AppRegistry and Application Manager, a capability of AWS Systems Manager

This solution includes a Service Catalog AppRegistry resource to register the solution's CloudFormation template and its underlying resources as an application in both Service Catalog AppRegistry and Application Manager. With this integration, you can centrally manage the solution's resources and enable application search, reporting, and management actions.

Use cases

Migrating historical data

Migration Assistant for Amazon OpenSearch Service offers various options for migrating historical data, including detailed guidance on running a historical migration applicable across all supported migration routes, such as from Elasticsearch 7.10.2 to OpenSearch 1.0.

Near real-time migration of HTTP traffic between clusters

The solution offers you the option to capture data destined for a source cluster and store this data for reuse. A user can replay this data to a target cluster in near real-time to migrate as soon as possible, or replay at a later time.

Replay traffic to multiple targets

The solution allows you to capture traffic for replay through multiple instances or in sequential runs, facilitating the validation of diverse cluster workloads and configurations.

Precise simulation of your cluster workloads

The solution allows users to capture and replay traffic either simultaneously with multiple instances, or in separate sequential runs. This feature aids in validating different cluster workloads and configurations. By default, the Replayer preserves the original concurrency and request rate to accurately simulate production loads, ensuring a fair like-for-like comparison.

Verify target cluster results

The solution facilitates user comparisons of source and target traffic in terms of accuracy and performance. It captures metrics and logs for analysis, providing users with the necessary confidence to migrate their production traffic to a new target.

Concepts and definitions

This section describes key concepts and defines terminology specific to this solution:

source cluster

The originating cluster on a specific version of Elasticsearch or OpenSearch that the user is attempting to either upgrade or decommission.

target cluster

The destination cluster that the user is trying upgrade, migrate to, or optimize.

capture proxy

A pass-through HTTP proxy designed to capture and log all of the request and response traffic to a durable source for later reuse.

replayer

A tool designed to simulate original traffic workloads by retrieving recorded request traffic and sending it to a target cluster. The Replayer correlates the request and response traffic of the originating request with the request and response traffic to the target, and stores the traffic persistently.

historical data

Documents that were on the source cluster before the Capture Proxy received traffic.

live/continuous data

Data intercepted by the Capture Proxy and subsequently processed through a Replayer. Initially, this information is transmitted from clients to the source cluster, where it is intercepted by the Capture Proxy. Subsequently, the data is relayed back to the designated target cluster.

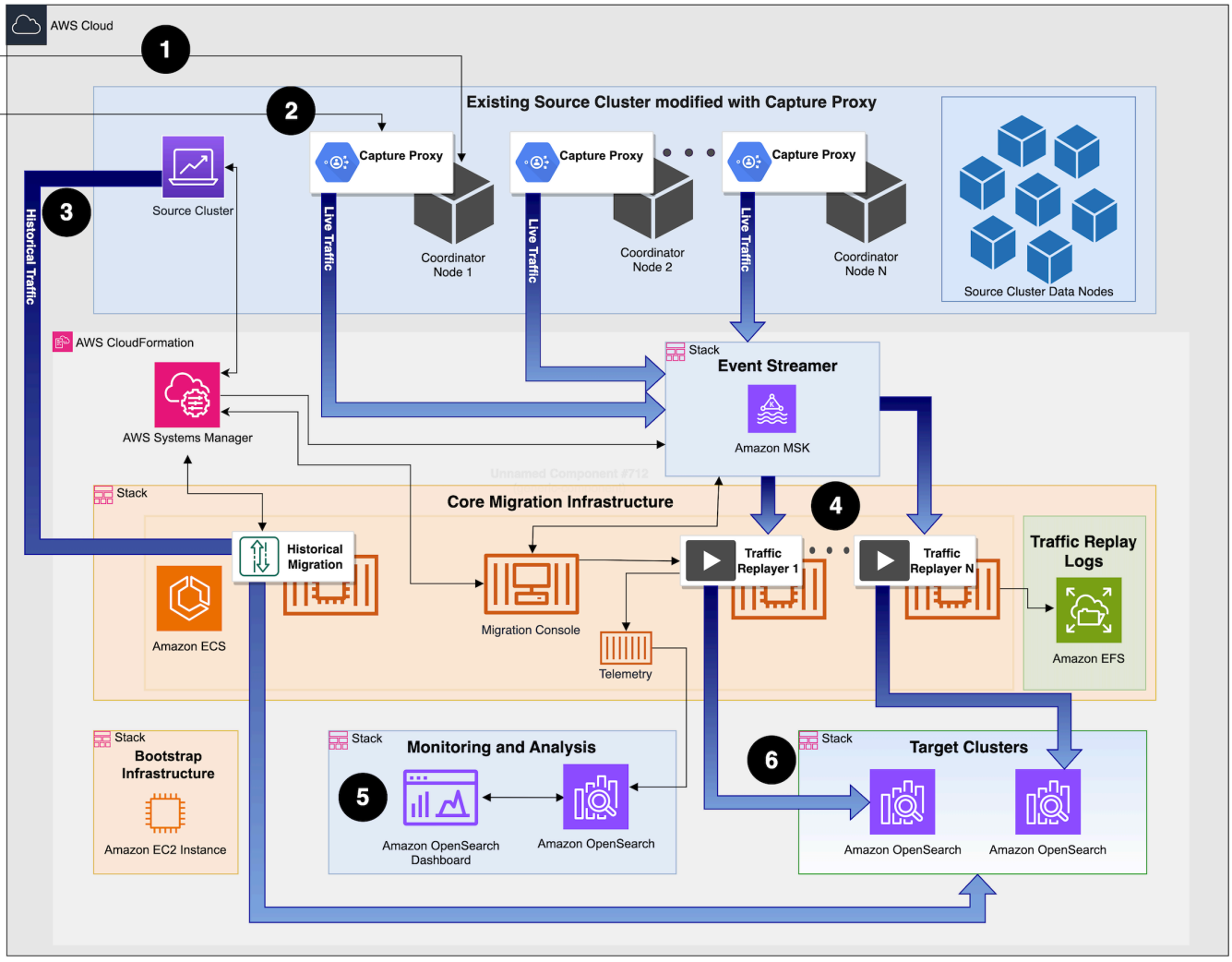
For a general reference of AWS terms, see the AWS glossary.

Architecture overview

This section provides a reference implementation architecture diagram for the components deployed with this solution.

Architecture diagram

Deploying this solution with the default parameters deploys the following components in your AWS account.



Migration Assistant for Amazon OpenSearch Service architecture on AWS

> **Note**
>
> AWS CloudFormation resources are created from AWS Cloud Development Kit (AWS CDK) constructs.

The high-level process flow for the solution components deployed with the AWS CloudFormation template is as follows:

1. Traffic is directed to the existing cluster, reaching each coordinator node.
2. A Capture Proxy is added before each coordinator node in the cluster, allowing for traffic capture and storage in Amazon MSK.
3. Once continuous traffic capture is in place, the user initiates a historical backfill.
4. Following the backfill, the user replays the captured traffic using a Traffic Replayer.
5. The user evaluates the outcomes from routing traffic to both the original and the new cluster.
6. After confirming the new cluster's functionality meets expectations, the user dismantles all related stacks, retaining only the new cluster's setup. Additionally, the user can retire and discard the old cluster's legacy infrastructure and all of the solution's stacks, keeping only the new cluster.

AWS Well-Architected design considerations

This solution uses the best practices from the AWS Well-Architected Framework, which helps customers design and operate reliable, secure, efficient, and cost-effective workloads in the cloud.

This section describes how the design principles and best practices of the Well-Architected Framework benefit this solution.

Operational excellence

This section describes how we architected this solution using the principles and best practices of the operational excellence pillar.

- The solution enables users to transition from legacy, self-managed OpenSearch and OpenSearch clusters to AWS OpenSearch, a managed service that significantly reduces the operational burden associated with maintaining outdated systems.

Security

This section describes how we architected this solution using the principles and best practices of the security pillar.

- The solution facilitates user migration to Amazon OpenSearch Service, which is regularly updated to uphold a strong security posture.

Reliability

This section describes how we architected this solution using the principles and best practices of the reliability pillar.

- The solution introduces a lightweight proxy layer between the client and the source cluster to capture data reliably using Kafka. This ensures that requests can be replayed on the target cluster, with the flexibility to modify or resend any failed requests.

Performance efficiency

This section describes how we architected this solution using the principles and best practices of the performance efficiency pillar.

- The tools within the solution have been optimized for efficiently capturing and replaying traffic, mitigating the need for horizontal scaling.

Cost optimization

This section describes how we architected this solution using the principles and best practices of the cost optimization pillar.

- The solution provisions CloudFormation stacks designed to operate specifically during migration processes, which can be safely decommissioned post-migration.
- The solution operates on AWS Fargate within Elastic Container Service, ensuring cost-effectiveness as charges are incurred only for the duration that the containers are active.
- For those interested in a preliminary local trial, a local version of the solution can be built and deployed in a user's own environment before cloud implementation. Detailed instructions

and resources for this are available at the OpenSearch Migrations GitHub repository: https://github.com/opensearch-project/opensearch-migrations.

Sustainability

This section describes how we architected this solution using the principles and best practices of the sustainability pillar.

- The solution allows customers to migrate their workloads to more environmentally friendly hardware, for example Graviton instance types.

Architecture details

This section describes the components and AWS services that make up this solution and the architecture details on how these components work together.

Self-service Elasticsearch/OpenSearch source cluster

The source cluster for this solution is based on Elasticsearch or OpenSearch, operating on EC2 instances or alternative computing infrastructure. Configure a proxy to interface with the source cluster, positioning the proxy in front of, or on each of the cluster coordinating nodes.

Capture Proxy

Capture Proxy is designed for HTTP RESTful traffic. It functions by relaying traffic to a source cluster and concurrently dividing the traffic, channeling it towards a robust stream-processing service for subsequent playback.

Traffic Replayer

Traffic Replayer functions as a traffic simulation utility, replicating real-world workloads by retrieving recorded request traffic and dispatching it to a designated target cluster. It systematically associates the original requests and their responses with those directed to the target cluster, ensuring that this correlated data is available for further analysis.

OpenTelemetry Collector container

OpenTelemetry Collector is a component of the OpenTelemetry project, which is an observability framework built for the cloud and is a Cloud Native Computing Foundation (CNCF) sandbox project. It provides a unified way to receive, process, and export telemetry data like metrics, logs, and traces.

Historical Data Migration container

The Historical Data Migration container runs as a one-time task to copy index metadata and historical data from the source cluster to the target cluster. It compares the indices between the

two clusters to identify non-conflicting indices that must be migrated, and then leverages the open-source Data Prepper data collector to duplicate index data to the target cluster.

Migration Management Console

The Migration Management Console is a containerized portal that operates on Fargate within the Elastic Container Service (ECS) ecosystem. Its primary role is to facilitate the deployment of the Migration Assistant for Amazon OpenSearch Service solution, along with providing a suite of tools designed to aid in the migration process.

AWS services in this solution

AWS service	Description
AWS CloudFormation	**Core.** Infrastructure as Code (IaC) templates used to deploy and configure Migration Assistant.
Amazon OpenSearch Service (AOS)	**Core.** A Search, Logging, and Analytics Engine that users can upgrade to, migrate to, and use to compare the results of a source and target cluster.
Amazon Managed Streaming Service for Apache Kafka (MSK)	**Core.** Stream-processor that is fully managed. It is used a a durable way to store and reuse HTTP traffic.
Amazon Elastic Container Service (ECS)	**Core.** Runs highly secure, reliable, and scalable containers. The Migration Console, Replayer, and OpenTelemetry Collector run in ECS.
Amazon Elastic File System	**Core.** Scalable persistent storage utilized for retaining the request and response data from both the source and target clusters.
Amazon S3	**Core.** Storage allocated for Historical Backfill tasks, which involves exporting a snapshot

AWS service	Description
	from the source to be restored by the target cluster. S3 is also used to store IaC content.
AWS Systems Manager	**Supporting.** Provides you visibility and control of your infrastructure on AWS. Systems Manager provides a unified user interface so you can view operational data from multiple AWS services and enables you to automate operational tasks across your AWS resources.
AWS Secrets Manager	**Supporting.** A secure way for storing sensitive data, such as cluster credentials, that is required for Migration Assistant.
Amazon EC2	**Supporting.** Provides networking and security infrastructure for Migration Assitant including securing groups, and Virtual Private Networks.
AWS Lambda	**Supporting.** Lambda facilitates the execution of serverless functions and is employed by Migration Assistant to operate its suite of tools.
Amazon CloudWatch	**Optional.** Observe and monitor resources and applications on AWS or in the local Docker solution.

Additionally, the following tools are used in this solution:

Service	Description
Amazon OpenSearch Serverless	**Supporting.** A serverless version of OpenSearch, which functions as a search, logging, and analytics engine, offers users the flexibility to migrate to and compare as the target cluster.

Service	Description
	Configuration of this target is at the user's discretion.
Amazon OpenSearch Dashboards	**Optional.** A visualization tool designed for analyzing the status and outcomes of Traffic Replay by comparing data between source and target clusters.

How this solution works

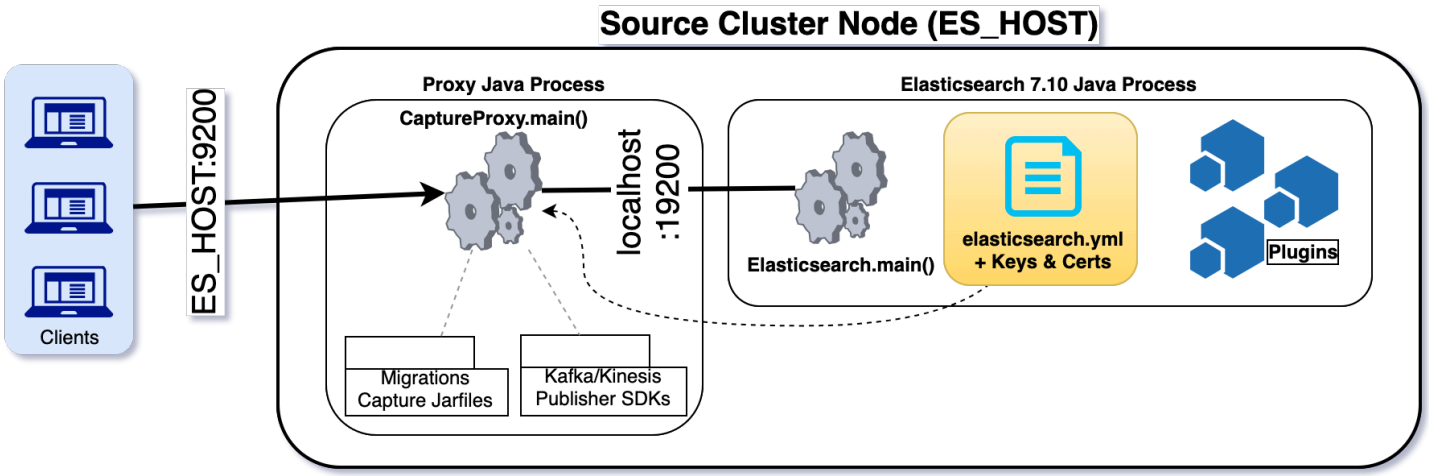
The Migration Assistant for Amazon OpenSearch Service solution combines custom services and tools with established ones. Those custom services include a Capture Proxy and Traffic Replayer that can replay HTTP/1.1 traffic to additional target servers. The Capture Proxy intercepts production traffic and offloads it to a durable store so that the replayer can reconstruct and replay requests to a target web service.

Request and responses are captured for the proxy and are preserved from the target interactions so that users can compare the differences between the two clusters. Since Capture Proxy is handling data on the critical path, the proxy is designed to offload data as simply as possible to minimize the proxy's impact on overall performance (latency, load, and so on). Independently, you can start a Historical Data Migration container via the Migration Console to backfill data already present in the source cluster.

Capture Proxy

The Capture Proxy is designed to be able to supplant the original source cluster port so that clients can continue to operate without any changes. A simple way to accomplish that is to install a proxy alongside the source cluster web services that service client traffic. Assign the port of the Capture Proxy to the port that clients were already using and rebind the existing web service to a new port. The new port can be bound to the loopback address to restrict connections from outside the host, requiring all traffic to flow through the Capture Proxy. The Capture Proxy is then configured to connect to the source cluster via localhost on an unused network port.

Capture Proxy uses the OpenSearch Security Plugin to parse its own TLS configuration, making it familiar and convenient to pull existing TLS settings from an Elasticsearch or OpenSearch cluster configuration.



Capture Proxy installation workflow (loopback bound port is set to 19200)

The Capture Proxy can also be deployed on standalone hardware with the following two caveats:

1. The Capture Proxy is only designed to proxy traffic for one destination. If that destination is a large number of nodes with a load balancer, any number of proxies that are necessary to support the traffic load can be setup to send traffic though the nodes that the load balancer is using.
2. The second caveat to installing the Capture Proxy on separate hardware is that the infrastructure will need to change. The clients might also need to change if it isn't feasible for the Proxy to replace the original web services (for example, fixed IP addresses are being used by clients and servers). In addition to the increased complexity and induced latency, it's harder to scope down access to the source cluster such that all traffic must flow through Capture Proxy. In the co-located scenario, binding the source web service to the loopback address ensures that all traffic services by the source will be captured. If there are ingress points other than through Capture Proxy, some requests may not be logged and will therefore, not be migrated to the target cluster.

For all the issues with a standalone proxy installation, it can still be quite manageable if the client traffic is routed through Load Balancer via a hostname that can provide a level of indirection. In that case, the hostname can be remapped to a new Capture Proxy or, more likely, a new Load Balancer that is situated in front of a fleet of Capture Proxies. Enabling that without disruption would require adding a second DNS entry for the existing Load Balancer that the new fleet of Capture Proxies would connect to.

TLS

In order for the proxy to write data that can be later replayed, the request and response data must be readable as HTTP traffic streams. If an existing client and server (cluster) are using TLS to transfer data, that data must be decrypted before being offloaded. The data is transmitted to Kafka using TLS, AWS IAM authentication, and that data is stored using encryption at rest. That encrypted data is able to be read in unencrypted form by the authenticated Traffic Replayer with the MSK client, even though the data is stored and transferred with some form of encryption throughout the process.

Terminating TLS is a necessary operation for any service, including the Capture Proxy. When the Capture Proxy is deployed on the same hardware as the source clusters (web services) AND users that are confident that there are no other ingress points to the source cluster other than the Capture Proxy, the source web service can bind only to the loopback address and forgo TLS. That configuration will reduce the number of TLS operations so that the webtraffic is decrypted only once and the Kafka traffic is encrypted once. When TLS must be used to connect to a source cluster, additional measures, such as the `–destinationConnectionPoolSize` argument are being developed to reduce the cost of negotiating new connections.

Additional impact

In addition to the impact incurred from TLS decrypting and encrypting, there will be a significant impact to network load. This solution assumes that the network has enough spare capacity to send out up to double the amount of network traffic, albeit to different destinations.

> **Note**
>
> Both requests AND responses are captured, so the amount of traffic offloaded will be proportional to their sum. The proxy doesn't compress traffic so that the computational burden can be minimized. Many requests and responses may already be compressed, diminishing the impacts of such efforts.

Migration Assistant for Amazon OpenSearch Service has two related, but distinct tasks concerning data migration:

1. Provide a consistent environment on the target that mimics the source at each point in time, including the end of the migration, when the pair should be equivalent.
2. Compare the responses for every request that went through the source.

These tasks have different degrees of importance for the overall migration. If a PUT (or any other mutating call) is dropped from the replay, it could have a long-lasting and irreversible impact on all future results. Because of that, the Capture Proxy parses HTTP messages as they are received. With one exception, all read data is immediately forwarded to the source cluster while data is asynchronously offloaded to Kafka.

The Capture Proxy ensures that all mutating requests have been committed to Kafka so that there are strong guarantees about the durability of those records. The Capture Proxy does that before allowing the entire request to be sent to the source service. This behavior means that GET traffic must flow through the system without being impacted by the latency of calls to Kafka. However, mutating requests (`PUT`, `POST`, `DELETE`, `PATCH`) will be impacted. Clients that have made those requests will not receive a response or will not be able to make another request until all of the prior offloaded traffic has been committed (which could include prior requests and responses from the same connection). That ensures that no mutating request was sent to the source without first being committed to Kafka. However, it also means that a request could be committed to Kafka without ever being sent and handled by the source web service.

Requests that are suspected of not being processed (or fully processed) by the source clusters are detectable by the Capture Proxy. Those requests will be missing a response. Currently, the Capture Proxy doesn't reconcile which of these requests have likely succeeded or failed. However, in practice, many real-world examples would have retried the failed request, resulting in a received response.

Design

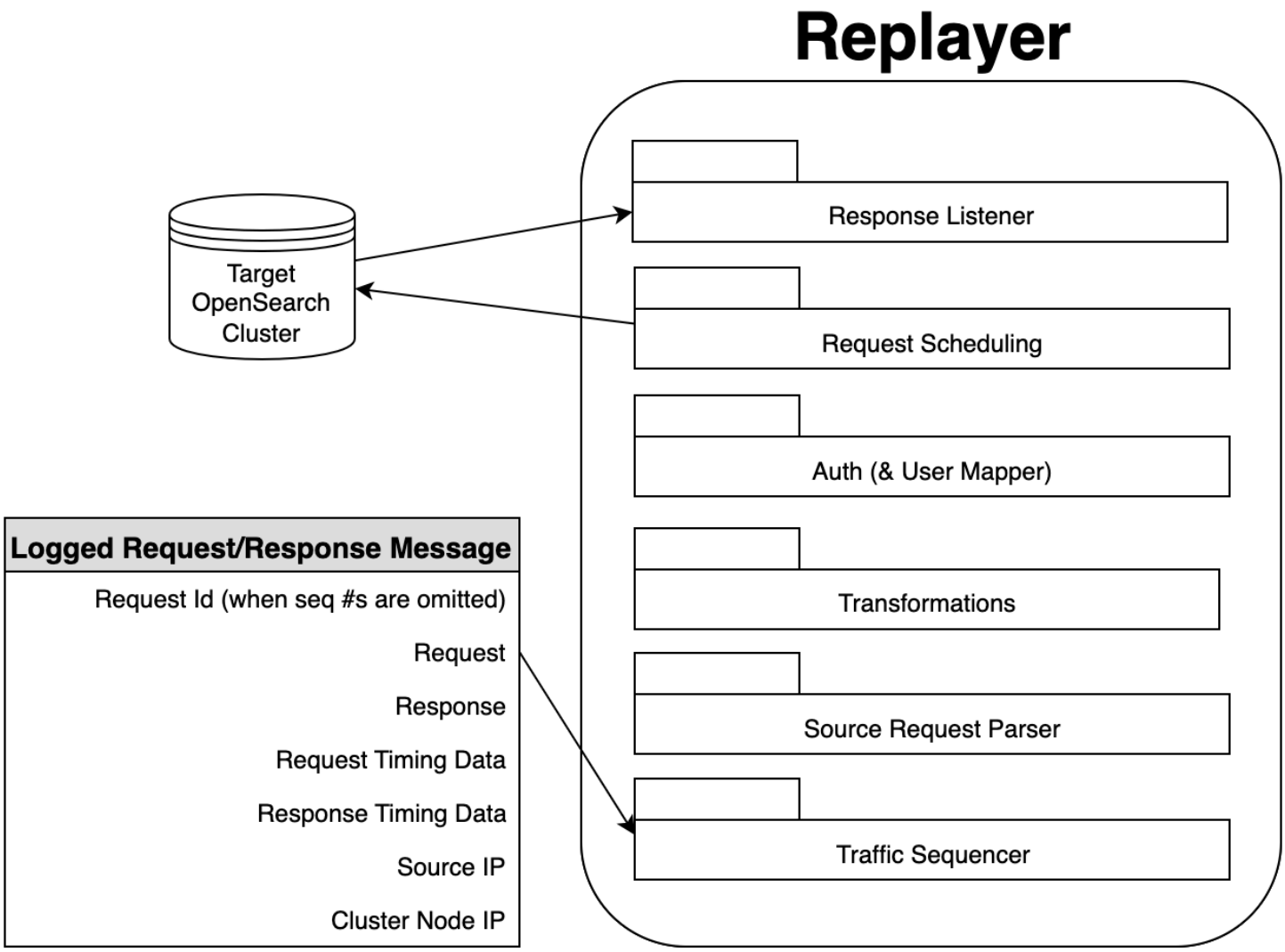
Data is organized into `TrafficObservations` (`Read`, `Write`, `Close`, and so on) that have timestamps and are organized into larger `TrafficStream` objects. These objects observations are Protobuf wrappers to the raw bytes as they were received by the Proxy after TLS decryption or raw bytes to be sent just before TLS encryption. Those `TrafficStream` objects are organized by connection. Each socket connection will have a stream of `TrafficObservations`, which are flushed to Kafka as `TrafficStream` objects for the next sequence of observations for a given connection. Concurrent connections will have concurrent `TrafficStream` objects, each with their respective **connectionId**. The **connectionId** is a globally unique id for that connection, though a unique **nodeId** is also included for diagnostics and future partitioning.

All network activity is asynchronously data-driven, using the same framework (Netty) that Elasticsearch, OpenSearch, and numerous other projects use. Using the same framework also mitigates some risk that HTTP could be parsed differently by the source and the proxy. While this is always possible, it is greatly diminished by both using the same mature HTTP framework.

Traffic Replayer

Traffic Replayer consumes network traffic that has been packaged into a stream of Protobuf encoded objects; reconstructs that stream into HTTP requests; sends the stream to a target cluster; and records the responses alongside the request and the traffic from the original source interaction.

Traffic Replayer is a long-running Kafka stream processor. While it leverages Kafka's ordering guarantees for sequencing `TrafficObservations`, it must still group `TrafficStreams` by their **connectionIds**. As `TrafficStream` objects are reassembled by connection, individual requests and their responses are sent through a pipeline. That pipeline includes rewriting the request as necessary, scheduling, and sending the requests to match the source time intervals, and aggregating the response along with all of the context for the current request. After that information has been logged for future analysis, the `TrafficStream` objects that have been fully handled are committed from Kafka so that they are not processed again if the Replayer is restarted.



Traffic Replayer pipeline overview

Message transformation

The reassembly process is careful to preserve timestamps of individual pieces (or packets) of the traffic stream and the unique id from the `TrafficCaptureSource`. Once all of the bytes for a request have been accumulated, the bytes are sent through a netty pipeline for further processing. This processing includes rewriting headers, such as the host value, changing user-authentication, and transforming the contents of the payload.

The Traffic Replayer provides a wrapper to a Map object that represents the headers and payload in a nested JSON-like key-value structure. That Map object is passed through an `IJsonTransformer` object that may rewrite the request by altering headers or the body. To minimize unnecessary and expensive operations, the netty pipeline parses the HTTP headers first and runs the transformation

before the pipeline has been fully configured. If the transformation did not attempt to access the payload, the pipeline won't be configured to parse the JSON from the body from the message.

Likewise, the pipeline attempts to setup as few handlers as possible to eliminate unnecessary (de)compression and repackaging. The configuration of what processing must be done is determined after the initial transformation. When the entire message must be transformed, it is done so with netty handlers that generally work with minimal data as is necessary. At the end of this pipeline, a new sequence of buffers is ready to be sent to the target server. The shape of the sequence of buffers attempts to match that of the original sequence to more closely match the traffic patterns that the source was receiving. For example, if the source got 120 bytes with one byte per second, the target request will also get 120 bytes over 120 seconds.

This message transformation also includes rewriting authorization headers. In the Basic-Auth case, that rewrite only involves the headers. If there were no other transformations, the body of the content does not need to be parsed. However, if the authorization scheme being used is AWS Auth (SigV4), a handler to parse the body will be added to the pipeline alongside mechanisms to fully consume the contents so that the signature can be accurately computed.

In some cases, the pipeline may be unable to parse a message, or the message might not require any rewrite. In those cases, the parsing of the current request is unwound and the request is sent exactly as sent to the source to the target cluster. The response will be handled like any response for a fully transformed message, though the final metadata will show whether the request had transformation skipped or if it was due to an error.

There are a number of optimizations in the pipeline that may seem to make processing more complicated. Rather, some paths will require a considerable amount of complexity. Some of the complexity is inherent to HTTP (for example, SigV4 signing) and other complexity is to provide a simple API for developers and users to provide their own transformations. Managing these complexities efficiently is done via `RequestPipelineOrchestrator` and `NettyDecodedHttpRequestPreliminaryConvertHandler`. These preserve as much performance as possible by reducing the amount of work that needs to be done and reducing the amount of logs to describe that work.

User transformations

Users may specify what Transformation to run by providing a `.jar` file that can load an implementation of the `IJsonTransformer` class via Java's `ServiceLoader`. As described in the Message transformation section, most of the complexities in parsing HTTP messages are abstracted away. The transformer itself can determine paths, headers, and run sophisticated JSON

remapping by pulling in libraries such as Jackson, GSON, or JSON manipulation packages like Jolt or JMESPath.

Sending requests and timing

The Traffic Replayer manages its own sense of time. Every observation that is received has a timestamp that it was recorded by the proxy. This timestamp could be weeks or milliseconds in the past depending on the gap between when the replay has begun and when capture recording started. Currently, a replay always starts at the beginning of the captured stream and it fixes the current time to the time of the first interaction. For users that would like to catch-up or stress test the system, the Replayer's time mapping function can include a speedup factor (F) so that something that happened (N) seconds after the initially recorded interaction will happen N/F seconds after the transformed version of the initial interaction was sent to the target service. This functionality is managed by a `TimeShifter` class that is effectively just a function that maps scalar values after some initialization. Timing values can be controlled via command line parameters.

That timing drives much of the rest of the Traffic Replayer. When a request is fully reconstructed, the message transformation work is *scheduled* to be done just before it would be scheduled to be sent. That's to guarantee that *temporally sensitive* values like SigV4 signatures won't go stale. It also keeps more data within one thread, making for less contention (cache invalidations) and allows for simpler code.

Like the Capture Proxy and the transformation pipelines described above, requests are sent via netty. Netty's architecture allows for a very large number of requests to be handled concurrently through cooperative, rather than preemptive, multitasking. While that requires code to be data-driven and to never block, it also affords the use of very simple data structures that must be threadsafe across multiple threads. Several classes are designed to run from one netty thread. Those will be initialized for each worker thread that netty spins up (which can be specified on the command line).

Netty manages each connection, as defined initially by the clients that were sending traffic to the Capture Proxy, within its own `EventLoop`. The `EventLoop` within the Traffic Replayer will have a connection to the target service, which may break and must be reestablished. However, that same `EventLoop` (and its Thread) will be affiliated with its connection for the lifetime of the connection. That lifetime will be shut down either when the connection's `TrafficStream` has encountered a *close* observation OR if no observations have been encountered in a period of time and they are expired by the accumulation phase. That means that everything happening within processing, sending, and receiving the messages for a given connection will be thread-safe.

Each connection's `EventLoop` and `Channel` are grouped within a `ConnectionReplaySession`. This session also includes data to schedule the interactions for transforming requests, sending them, and closing connections. The schedules are maintained as a key-value map from the time that an operation should happen, post time shifting (so in real-time, not source time). As work items are completed, the next item is pulled from the schedule if it is ready to run or a timer is set on the `EventLoop` to rerun it when appropriate. Because there are many interactions scheduled within one connection's session, the actual times that interactions occur could drift. For example, assume the source service took 5 seconds to service six requests sequentially on one socket connection that was kept alive. If the target service takes 10 seconds to service each request, the target will take 60 seconds to run those six interactions instead of 30 from the source cluster.

However, if a source cluster had the same interactions and latencies but sent them via different connections, the total time to send the requests to the target service could be 35 seconds, since requests would overlap by 5 seconds. The schedules are isolated to a given connection. Since a connection is defined by a sequence of requests, we must wait for the previous requests to finish before proceeding. Without the connection isolation though, requests will be sent without those constraints.

> **Note**
>
> If the target cluster cannot keep up with the pacing of the source cluster, it won't be able to match the timing patterns that the source cluster experienced.

Throttling

The Traffic Replayer has a `TrafficCaptureSource` that it uses as an abstraction over a Kafka topic. If the Kafka topic has days of data and the replayer needs to replay that traffic over many hours, it's critical that the Traffic Replayer consume only what is necessary to keep its memory footprint manageable. However, it also must consume enough so that it can reconstruct the `TrafficStreams` into requests. To accommodate the guarantee that we can reconstruct `TrafficStreams`, the code takes advantage of expiration of streams. Within a user-specified number of seconds, we're guaranteed to either observe progress on a `TrafficStream` or expire it and give up. Setting the backpressure limit to a multiple of the expiration window provides backpressure to limit memory. However, it doesn't put a total bound on the peak amount of memory required. Such a bound would be problematic as there could be an unlimited number of simultaneous connections with ongoing traffic. Right-sizing in these cases will be an exercise in

understanding peak load to the source cluster and/or trial and error for provisioning the Traffic Replayer.

Likewise, the Traffic Replayer may be asked to send too many requests simultaneously. This can happen if one replayer cannot handle the load, such as if the time speedup factor was extremely high on a modestly sized cluster. In these cases, an additional throttling mechanism is required to restrict how many simultaneous connections can be made. A command line option is available to limit how many requests can be in-progress at any point in time. When a request has been reconstructed, if the Replayer has already saturated the total number of requests that it can handle, the new request will be blocked from sending until other requests have finished. Notice that this is currently bound to the number of requests, not connections.

You can add additional throttling measures in the future to account for message sizes or the number of connections.

Outputting results

The Traffic Replayer uses Log4j2 for its application logs. It also uses Log4J2 to output some of its other output, including logs destined for metrics and the results of the source and target traffic interactions. Be careful when adjusting the `log4j2.properties` files.

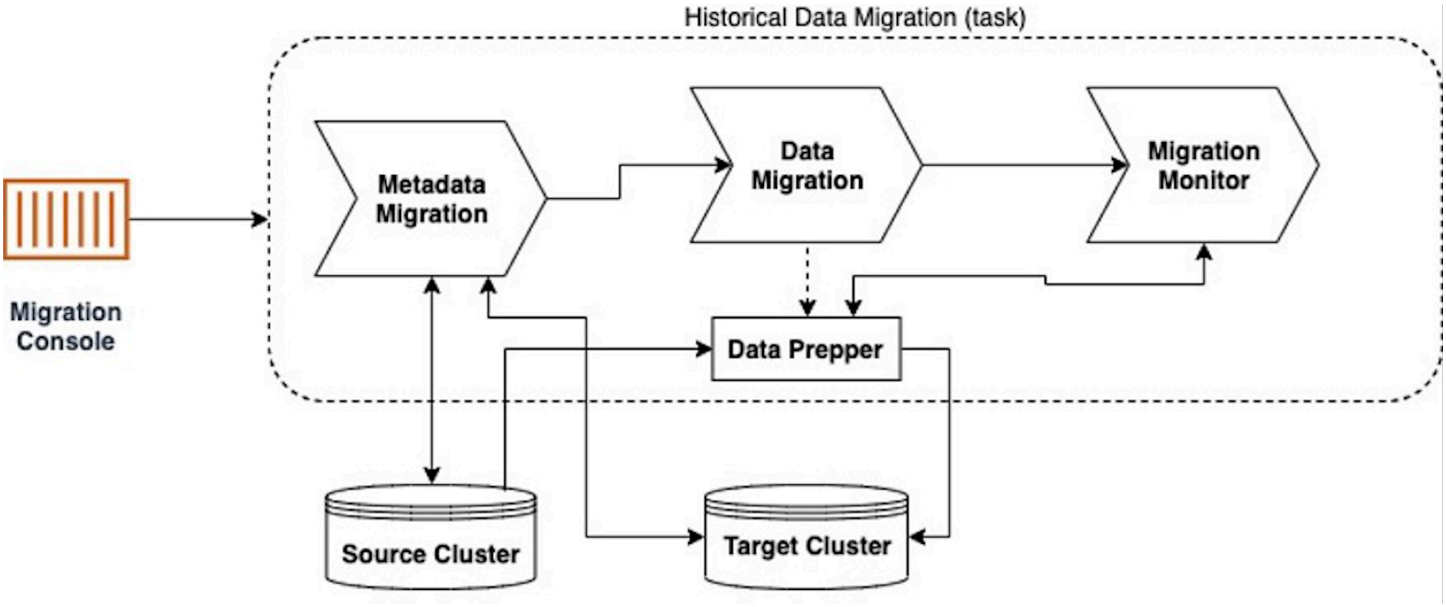
The results, which are logged not just through Log4j2 but also to a file, which is provided by a command-line parameter. This result output will be a stream of JSON-formatted objects with the source/target requests/responses. Those will include headers, timestamps, and the full bodies base64 encoded.

Historical Data Migration

Historical Data Migration

If the Historical Data Migration option was enabled at deployment time, users can initiate historical data migration from the Migration Console by using a pre-populated environment variable (`$FETCH_MIGRATION_COMMAND`) whose value is specified in an AWS CLI command.

This command spins up a new ECS task in the same cluster that houses the services described previously. The Historical Data Migration task is modeled as a workflow, with each step run in turn to replicate indices and data already present in the source cluster to the target cluster.



Historical Data Migration task workflow

Metadata migration

The first step in the workflow compares the indices present on the source and target clusters to identify which indices must be migrated. The indices are grouped into three buckets:

1. New indices – Indices that are present on the source cluster but not on the target cluster.
2. Identical indices – Indices with the same name that are present in both clusters and whose metadata (index settings and index mappings) are identical.
3. Conflicting indices – Indices with the same name that are present in both clusters but whose index metadata does not match.

All new indices are selected for Historical Data Migration, and API calls are made to the target cluster to create these indices with metadata (index settings and index mappings) matching that of the source cluster.

Additionally, identical indices that do not have any documents in the target cluster (that is, document count is zero) are also considered eligible for data migration. Conflicting indices and identical indices with existing documents in the target cluster are not migrated. Indices that are present on the target cluster but not on the source cluster are left as is.

The list of indices selected for migration are merged with the pipeline configuration file uploaded at the time of deployment to form the final Data Prepper pipeline definition.

Data migration using Data Prepper

The next step of the workflow initiates a Data Prepper process within the task to perform data migration. The Data Prepper process creates a pipeline based on the configuration file produced by the previous step and begins the data transfer process.

Internally, Data Prepper reads index documents using scroll queries, or point-in-time APIs if supported. These documents are buffered in-memory and then written to the target cluster by worker threads using the bulk API. You can fine tune these parameters by changing the uploaded Data Prepper pipeline file. Refer to Configuring Data Prepper for more details.

> **Important**
>
> Data Prepper (and therefore the Historical Data migration process) relies on the presence of the `_source` field to transfer documents from indices on the source cluster to the target cluster. Further, the migration process involves continuous API queries to both the source and target clusters, which will increase traffic and resource usage. This may impact cluster clients or operations running on each cluster (such as merges and snapshots).

Once the Data Prepper process has been kicked off, the workflow immediately moves to the next step.

Migration Monitor

The Migration Monitor polls the Data Prepper metrics endpoint to determine progress metrics for the migration. This is computed by comparing the number of documents processed by the pipeline to the total target document count set by the Metadata migration step. Once the target document count is achieved, the migration is considered complete. The Migration Monitor emits progress information (as a percentage) to the ECS task logs for user visibility.

Additionally, the Migration Monitor also tracks activity metrics for the Data Prepper pipeline to stop it. Once the target document count has been reached, the Migration Monitor validates that the pipeline is idle and no other data is flowing through it. Once this is verified, the Migration Monitor shuts down the Data Prepper process via its /shutdown API and concludes the Historical Data Migration workflow. This finally shuts down the ECS task.

Limitations

The Historical Data Migration container is only designed to scale vertically – this can be achieved by increasing the CPU and memory values in the ECS task definition and by updating the worker threads in the pipeline configuration file. Running multiple instances of the container is not supported.

Historical Data Migration also does not implement a rollback mechanism for changes made to the target cluster. In the event of an incomplete migration *(due to cluster connectivity loss, throttling, or termination of the ECS migration task)*, the user must delete partially migrated indices on the target cluster before re-launching the container to achieve accurate Historical Data Migration.

Plan your deployment

This section describes the cost, security, Region, and quota considerations for planning your deployment.

Cost

You are responsible for the cost of the AWS services used while running this solution. As of this revision, the cost for running this solution with the default settings in the US East (N. Virginia) Region is approximately **$2754.00 for a 1 TB migration**. These costs are for the resources shown in the sample cost table.

We recommend creating a budget through AWS Cost Explorer to help manage costs. Prices are subject to change. For full details, refer to the pricing webpage for each AWS service used in this solution.

Sample cost table

The following table provides a sample cost breakdown for deploying this solution with the default parameters in the US East (N. Virginia) Region and assumes a 1 TB migration with Migration Assistant running for one week.

AWS service	Dimensions	Cost [USD] / month
Amazon Managed Streaming for Apache Kafka (MSK)	Amazon Managed Streaming for Apache Kafka RunBroker - $0.21 per kafka.m5.large broker hour in the US East Region Amazon Managed Streaming for Apache Kafka RunVolume - $0.1 per GB per month for storage in the US East Region 2 instance(s) x 0.21 USD hourly x 730 hours in a month	$367.56

AWS service	Dimensions	Cost [USD] / month
	= $306.60 (MSK cost) MSK broker instance charges (monthly): $306.60 2 broker nodes x 100 GB x 0.10 USD = $20.00 USD (storage cost) Storage pricing (monthly): $20.00 All other Regions: 2 GB x 0 USD per GB = 0.00 Intra Region: (2048 GB x 0.01 USD per GB outbound) + (2048 GB x 0.01 USD per GB inbound) = $40.96 Data transfer cost (monthly): $40.96	

AWS service	Dimensions	Cost [USD] / month
Amazon OpenSearch Service (AOS)	Amazon OpenSearch Service ESDomain 2 instance(s) x 0.743 USD hourly x (100 / 100 utilized/ month) x 730 hours in a month = $1084.7800 (Amazon OpenSearch Service data instance cost) Amazon OpenSearch Service data instance cost (monthly): $1,084.78 1 instance(s) x 0.743 USD hourly x (100 / 100 utilized/ month) x 730 hours in a month = $542.3900 USD (Amazon OpenSearch Service dedicated master instance cost) Amazon OpenSearch Service dedicated master instance cost (monthly): $542.39 Amazon OpenSearch Service dedicated master instance cost (upfront): $0.00 Amazon OpenSearch Service data instance cost (upfront): $0.00 1,000 GB x 0.122 USD x 1 instances = $122.00 EBS Storage Cost (gp3)	$1749.17

AWS service	Dimensions	Cost [USD] / month
	General Purpose SSD (gp3) - Storage monthly cost: $122.00 1,000 GB x 3 IOPS = 3,000.00 calculated gp3 IOPs Max (3000 IOPS, 3000.00 IOPS) = 3,000.00 minimum gp3 IOPS Min (16000 IOPS, 3000.00 IOPS) = 3,000.00 minimum IOPS for free 1,000 GB / 3000 GB = 0.333333 Throughput incremental factor RoundUp (0.333333) = 1 1 throughput incremental factor x 250 MB/s = 250.00 MB/s Min (1000 MB/s, 250.00 MB/s) = 250.00 MB/s minimum throughput for free Amazon OpenSearch Service EBS storage cost (monthly): $122.00	

AWS service	Dimensions	Cost [USD] / month
Amazon Elastic Container Service (ECS)	**Migration Console:** Number of tasks or pods: 1 per day * (730 hours in a month / 24 hours in a day) = 30.42 per month Average duration: 7 days = 168 hours **Pricing calculations** 30.42 tasks x 0.50 vCPU x 168 hours x 0.04048 USD per hour = $103.44 for vCPU hours 30.42 tasks x 1.00 GB x 168 hours x 0.004445 USD per GB per hour = $22.72 for GB hours 20 GB - 20 GB (no additiona l charge) = 0.00 GB billable ephemeral storage per task 103.44 USD for vCPU hours + 22.72 USD for GB hours = $126.16 total Fargate cost (monthly): $126.16 **Replayer:** Number of tasks or pods: 1 per day * (730 hours in a month / 24 hours in a day) = $30.42 per month	$532.04

AWS service	Dimensions	Cost [USD] / month
	Average duration: 7 days = 168 hours **Pricing calculations** 30.42 tasks x 1 vCPU x 168 hours x 0.04048 USD per hour = $206.88 for vCPU hours 30.42 tasks x 4.00 GB x 168 hours x 0.004445 USD per GB per hour = $90.87 for GB hours 20 GB - 20 GB (no additiona l charge) = 0.00 GB billable ephemeral storage per task 206.88 USD for vCPU hours + 90.87 USD for GB hours = $297.75 total Fargate cost (monthly): $297.75 Historical Backfill: Number of tasks or pods: 1 per day * (730 hours in a month / 24 hours in a day) = 30.42 per month Average duration: 3 days = 72 hours **Pricing calculations**	

AWS service	Dimensions	Cost [USD] / month
	30.42 tasks x 1 vCPU x 72 hours x 0.04048 USD per hour = 88.66 USD for vCPU hours 30.42 tasks x 2.00 GB x 72 hours x 0.004445 USD per GB per hour = $19.47 for GB hours 20 GB - 20 GB (no additiona l charge) = 0.00 GB billable ephemeral storage per task 88.66 USD for vCPU hours + 19.47 USD for GB hours = $108.13 total Fargate cost (monthly): $108.13	

AWS service	Dimensions	Cost [USD] / month
Amazon Elastic File System (EFS)	Percentage of data that is frequently accessed: 10 / 100 = 0.1 **Pricing calculations** **GB (Storage capacity): 2,000** 2,000 GB per month x 0.10 = 200.00 Data stored in Standard storage 2,000 GB per month - 200.00 GB per month (Standard storage) = 1,800.00 Data stored in Standard-Infrequent Access storage 200.00 GB per month (Standard storage) x 0.30 USD = 60.00 USD (Standard Storage monthly cost) 1,800.00 GB per month (Infrequent Access) x 0.025 USD = 45.00 USD (Standard -Infrequent Access Storage monthly cost) 60.00 USD + 45.00 USD = $105.00 (Total data storage monthly cost) 105.00 USD / 2,000 GB per month = $0.0525 (Effective rate per GB for data storage)	$105.00

AWS service	Dimensions	Cost [USD] / month
	Effective rate per GB for data storage (monthly): $0.0525 60.00 USD + 45.00 USD = $105.00 **Storage (monthly): $105.00** 200.00 GB per month (Standard) x 730 hours in a month = 146,000.00 GB-Hours (storage monthly) 146,000.00 GB-Hours / 20 GB-Hours = 7,300.00 GB-Hours 7,300.00 GB-Hours x 1 MB/s-Hour = 7,300.00 MB/s-Hours (default throughput) 10 MB/s per month x 730 hours in a month = 7,300.00 MB/s-Hours (total provisioned throughput) 7,300.00 MB/s-Hours - 7,300.00 MB/s-Hours = 0.00 MB/s-Hours (total billable provisioned throughput) Max (0 USD, 0 USD) = $0.00 **Throughput cost (monthly): $0.00**	

AWS service	Dimensions	Cost [USD] / month
Amazon EC2	Amazon Elastic Compute Cloud NAT Gateway $0.045 per GB Data Processed by NAT Gateways $0.045 per NAT Gateway Hour	$0.00
Amazon S3	Amazon Simple Storage Service Requests-Tier1 $0.00 per request - PUT, COPY, POST, or LIST requests under the monthly global free tier $0.005 per 1,000 PUT, COPY, POST, or LIST requests	$0.00
AWS Lambda	AWS Lambda USW2-Lambda-GB-Second AWS Lambda - Compute Free Tier - 400,000 GB-Seconds - US West (Oregon) AWS Lambda - Requests Free Tier - 1,000,000 Requests - US West (Oregon)	$0.00
AWS Secrets Manager	AWS Secrets Manager USW2-AWS Secrets Manager-Secrets $0 per Secret $0 per 10000 API Requests	$0.00

AWS service	Dimensions	Cost [USD] / month
Amazon Route 53	Amazon Route 53 HostedZone $0.50 per Hosted Zone for the first 25 Hosted Zones	$0.50
Amazon EC2 Container Registry (ECR)	Amazon EC2 Container Registry (ECR) USW2-TimedStorage-ByteHrs 500MB-month Free Tier	$0.00
Virtual Private Cloud	$0.005 per Idle public IPv4 address per hour $0.005 per In-use public IPv4 address per hour	$0.00
	Total:	**$2754.27 [USD] / month**

Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared responsibility model reduces your operational burden because AWS operates, manages, and controls the components including the host operating system, the virtualization layer, and the physical security of the facilities in which the services operate. For more information about AWS security, visit AWS Cloud Security.

IAM roles

AWS Identity and Access Management (IAM) roles allow customers to assign granular access policies and permissions to services and users on the AWS Cloud. This solution aims to create IAM roles with least privilege where resource access is required. This includes allowing some required Migration ECS services to produce/consume from MSK, make requests to the target cluster, and access provided secrets stored within AWS Secrets Manager needed for target cluster authentication and authorization.

Security groups

The solution creates security groups designed to control and isolate network traffic between Migration ECS containers, as well as between certain Migration ECS containers and associated services such as Amazon MSK, Amazon OpenSearch Service, and Amazon EFS. We recommend that you review the security groups and further restrict access as needed once the deployment is up and running.

AWS Secrets Manager

Migration Assistant for Amazon OpenSearch Service allows accessing stored secrets from AWS Secrets Manager in the Replayer Migration container to construct a proper auth header when replaying captured traffic to the target cluster, if it is required.

Supported AWS Regions

This solution uses Amazon OpenSearch Service, which is not currently available in all AWS Regions. For the most current availability of AWS services by Region, see the AWS Regional Services List.

Migration Assistant for Amazon OpenSearch Service is available in the following AWS Regions:

Region name	
US East (Ohio)	Asia Pacific (Sydney)
US East (N. Virginia)	Asia Pacific (Tokyo)
US West (Oregon)	Europe (Frankfurt)
US West (N. California)	Europe (Ireland)
Asia Pacific (Singapore)	Europe (London)

Quotas

Service quotas, also referred to as limits, are the maximum number of service resources or operations for your AWS account.

Quotas for AWS services in this solution

Make sure you have sufficient quota for each of the services implemented in this solution. For more information, refer to AWS service quotas.

Use the following links to go to the page for that service. To view the service quotas for all AWS services in the documentation without switching pages, view the information in the Service endpoints and quotas page in the PDF instead.

AWS CloudFormation quotas

Your AWS account has CloudFormation quotas that you should be aware of when launching the stack for this solution. By understanding these quotas, you can avoid limitation errors that would prevent you from deploying this solution successfully. For more information, refer to AWS CloudFormation quotas in the *AWS CloudFormation Users Guide*.

Deploy the solution

This solution uses AWS CloudFormation templates and stacks to automate its deployment. The CloudFormation template specifies the AWS resources included in this solution and their properties. The CloudFormation stack provisions the resources that are described in the template.

Deployment process overview

Follow the step-by-step instructions in this section to configure and deploy the solution into your account.

Before you launch the solution, review the cost, architecture, network security, and other considerations discussed earlier in this guide.

Time to deploy: Approximately 45-60 minutes (with MSK requiring the bulk of time)

Step 1: Launch the bootstrap stack

Step 2: Setup the bootstrap instance

Step 3: Customize the migration options

Step 4: Deploy the migration stacks

Important

This solution includes an option to send anonymized operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. AWS owns the data gathered though this survey. Data collection is subject to the AWS Privacy Notice.

To opt out of this feature, download the template, modify the AWS CloudFormation mapping section, and then use the AWS CloudFormation console to upload your updated template and deploy the solution. For more information, see the Anonymized data collection section of this guide.

AWS CloudFormation template

You can download the CloudFormation template for this solution before deploying it.

View template

migration-assistant-for-amazon-opensearch-service.template - Use this template to launch the solution and all associated components. The default configuration deploys the minimum resources needed for a bootstrap ECS container, which will be accessed for customizing and deploying the needed migration resources for the solution.

> **Note**
>
> AWS CloudFormation resources are created from AWS Cloud Development Kit (AWS CDK) constructs.

Step 1: Launch the bootstrap stack

Follow the step-by-step instructions in this section to configure and deploy the bootstrap stack into your account.

Time to deploy: Approximately 5 minutes

1. Launch solution Sign in to the AWS Management Console and select the button to launch the `migration-assistant-for-amazon-opensearch-service.template` CloudFormation template.
2. The template launches in the US East (N. Virginia) Region by default. To launch the solution in a different AWS Region, use the Region selector in the console navigation bar.
3. On the **Create stack** page, verify that the correct template URL is in the **Amazon S3 URL** text box and choose **Next**.
4. On the **Specify stack details** page, assign a name to your solution stack. For information about naming character limitations, see IAM and AWS STS quotas, name requirements, and character limits in the *AWS Identity and Access Management User Guide*.
5. Under **Parameters**, review the parameters for this solution template and modify them as necessary. This solution uses the following default value.

Parameter	Default	Description
Stage	`dev`	Specify the stage identifier which will be used in naming resources, for example, `dev`, `gamma`, `wave1`.

6. Select **Next**.
7. On the **Configure stack options** page, choose **Next**.
8. On the **Review** page, review and confirm the settings. Select the box acknowledging that the template will create IAM resources.
9. Choose **Create stack** to deploy the stack.

   You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive a `CREATE_COMPLETE` status in approximately 5 minutes.

Step 2: Setup the bootstrap instance

1. From the local environment where you will access the bootstrap instance, configure the required AWS credentials to allow access to the bootstrap instance. The identity used must have permissions that allow `ssm:StartSession` on the deployed bootstrap instance and SSM document resource.

   > **Note**
   >
   > We recommend being restrictive as to who has access to this bootstrap instance. Ideally, a deployment or admin role needs to have access to the bootstrap instance, as the bootstrap instance deploys resources into the given account.

   Example policy:

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

```
            "Effect": "Allow",
            "Action": "ssm:StartSession",
            "Resource": [
                "arn:aws:ec2:us-west-2:12345678912:instance/<instance-id>",
                "arn:aws:ssm:us-west-2:12345678912:document/SSM-<stage>-
BootstrapShell"
            ]
        }
    ]
}
```

2. To output the instance id of the bootstrap instance that was deployed, run the following command:

> **Note**
>
> If a **Stage** other than dev was used, the name value in this command must be updated accordingly. Alternatively, the instance id could also be retrieved from the EC2 dashboard in the AWS Management Console.

```
aws ec2 describe-instances --filters "Name=instance-state-name,Values=running" --
query 'Reservations[].Instances[].[InstanceId]' --output text
```

3. Using the instance id obtained from the previous step, run the following command to access the bootstrap instance:

> **Note**
>
> Update the **Stage** if it isn't dev.

```
aws ssm start-session --document-name SSM-dev-BootstrapShell --target <instance-id>
 --region (AWS Region where target is deployed, that is, us-east-1)
```

4. To prepare the bootstrap instance for deploying the migration pieces, run:

> **Note**
>
> The initial setup can take approximately 10-15 minutes.

```
./initBootstrap.sh && cd deployment/cdk/opensearch-service-migration
```

Step 3: Customize the migration options

1. From the same shell on the bootstrap instance, find the `cdk.context.json` file in the current, `/opensearch-migrations/deployment/cdk/opensearch-service-migration`, directory. The `cdk.context.json` file contains a context block with the label demo-deploy, which includes options to setup an end-to-end sample for testing. Refer to Configuration options for more details.
2. For a demo setup, these options can be deployed as is. Alternatively, define a custom context block in the `cdk.context.json` file with a required label, such as cit-deploy, and specify its own context options.
3. For Historical Data Migration, find a template Data Prepper pipeline configuration file (`dp_pipeline_template.yaml`) in the same `/opensearch-migrations/deployment/cdk/opensearch-service-migration` directory. You can modify the configuration file as needed before deployment to fine-tune the migration process.

Step 4: Deploy the migration stacks

Follow the step-by-step instructions in this section to deploy the migration stacks into your account.

Time to deploy: Approximately 45-60 minutes

1. Specify the Region in which you want to deploy the migration stacks by setting the **AWS_DEFAULT_REGION** environment variable, for example:

```
export AWS_DEFAULT_REGION=us-east-1
```

2. If this is the first time you're deploying CDK in this Region of your account, you must bootstrap the account to work with CDK. In the `cdk.context.json` file, run the following command with any existing context block label:

```
cdk bootstrap --c contextId=demo-deploy
```

3. If this is the first time you're deploying Amazon OpenSearch Service or an ECS cluster with CDK in this account, create the service linked role initially by running:

```
aws iam create-service-linked-role --aws-service-name opensearchservice.amazonaws.com
```

```
aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

4. To deploy the demo setup, run the CDK command:

```
cdk deploy "*" --c contextId=demo-deploy --require-approval never --concurrency 3
```

   Alternatively, you can deploy any context block you've setup in the `cdk.context.json` file using this command:

```
cdk deploy "*" --c contextId=<context_block_label> --require-approval never --
concurrency 3
```

> **Note**
>
> If providing your own context file, values may be dependent on your currently configured infrastructure. For example, if you have a source cluster that is provisioned behind a VPC and you want to deploy Migration Assistant within this VPC, the number of MSK brokers (`mskBrokerNodeCount`), and OpenSearch data nodes (`dataNodeCount`) must be modified so that there is at least one of each for each availability zone.

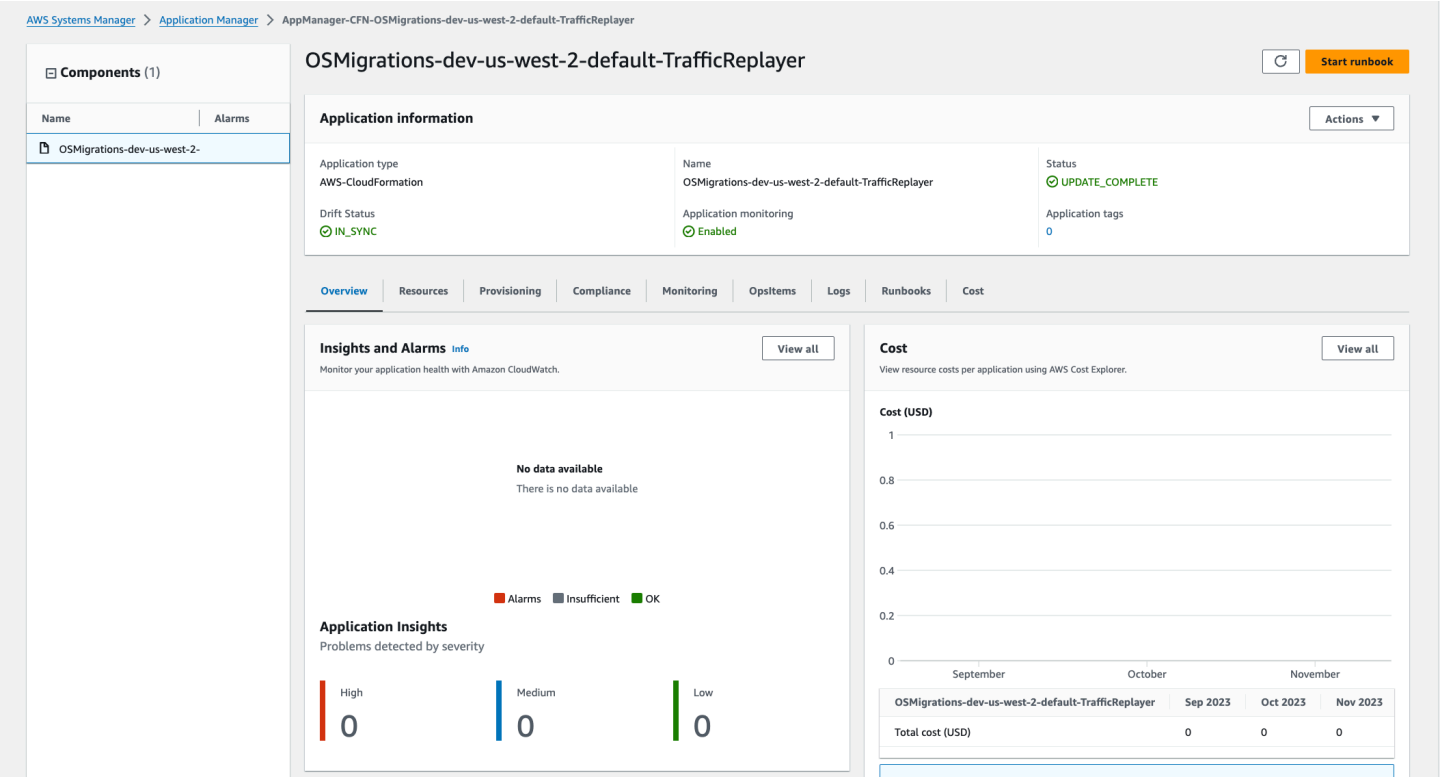
For additional help and tips, see the README.md file in the GitHub repository.

Monitoring the solution with Service Catalog AppRegistry

The solution includes a Service Catalog AppRegistry resource to register the CloudFormation template and underlying resources as an application in both Service Catalog AppRegistry and AWS Systems Manager Application Manager.

AWS Systems Manager Application Manager gives you an application-level view into this solution and its resources so that you can:

- Monitor its resources, costs for the deployed resources across stacks and AWS accounts, and logs associated with this solution from a central location.
- View operations data for the solution's AWS resources (such as deployment status, Amazon CloudWatch alarms, resource configurations, and operational issues) in the context of an application.



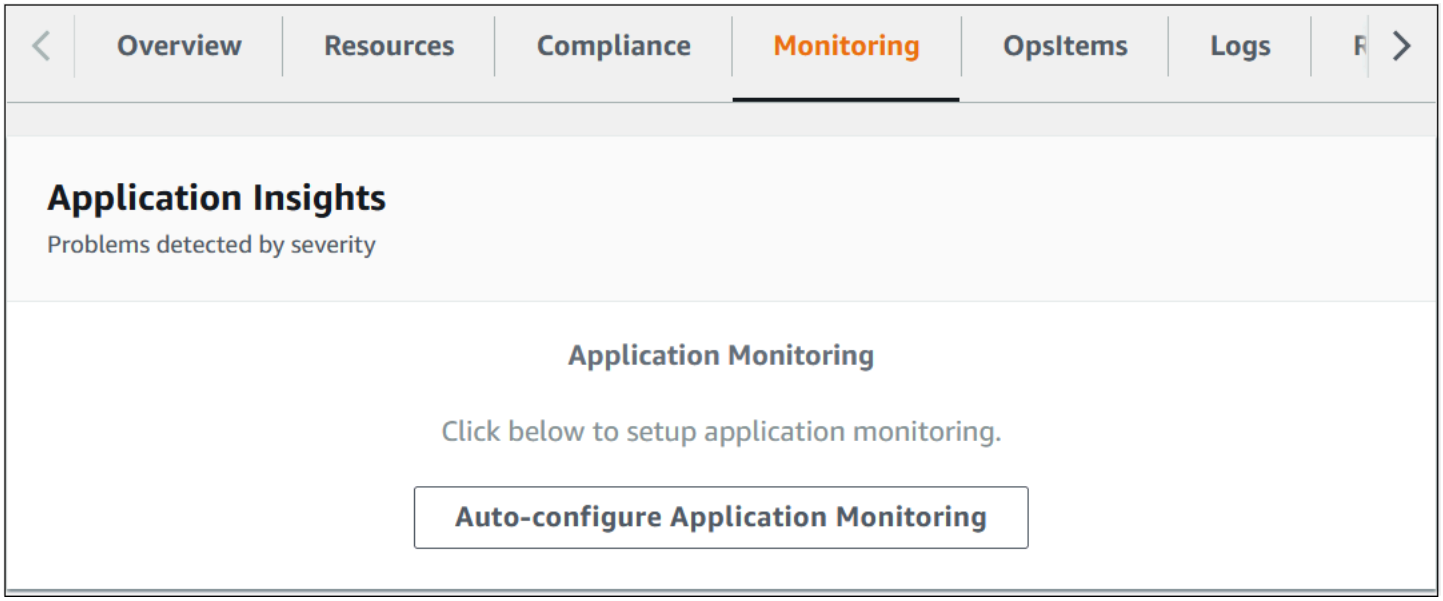
Migration Assistant Replayer stack in Application Manager

> **Note**
>
> Activate CloudWatch Application Insights, AWS Cost Explorer, and cost allocation tags associated with this solution. They are not activated by default.

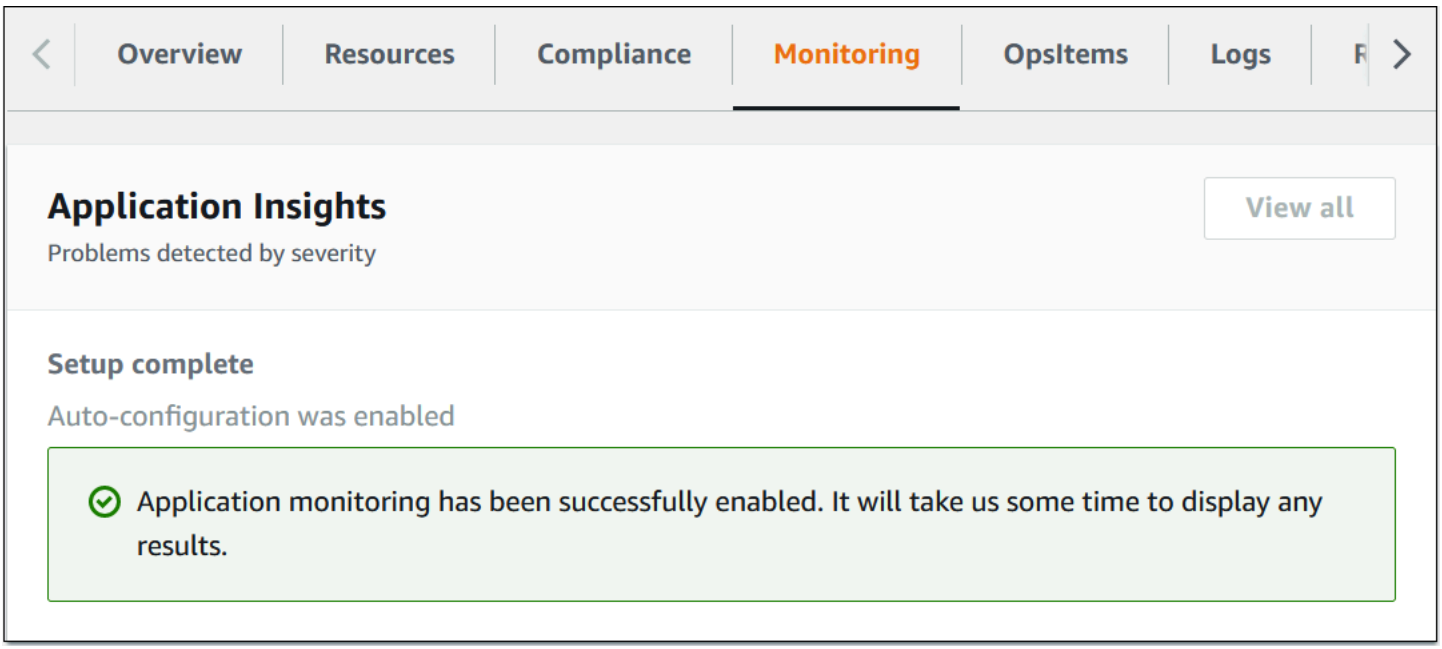
Activate CloudWatch Application Insights

1. Sign in to the Systems Manager console.
2. In the navigation pane, choose **Application Manager**.
3. In **Applications**, choose **AppRegistry applications**.
4. In **AppRegistry applications**, search for the application name for this solution and select it.

   The next time you open Application Manager, you can find the new application for your solution in the **AppRegistry application** category.
5. In the **Components** tree, choose the application stack you want to activate.
6. In the **Monitoring** tab, in **Application Insights**, select **Auto-configure Application Monitoring**.



Monitoring for your applications is now activated and the following status box appears:



Activate AWS Cost Explorer

You can see the overview of the costs associated with the application and application components within the Application Manager console through integration with AWS Cost Explorer which must be first activated. Cost Explorer helps you manage costs by providing a view of your AWS resource costs and usage over time. To activate Cost Explorer for the solution:

1. Sign in to the AWS Cost Management console.
2. In the navigation pane, select **Cost Explorer**.
3. On the **Welcome to Cost Explorer** page, choose **Launch Cost Explorer**.

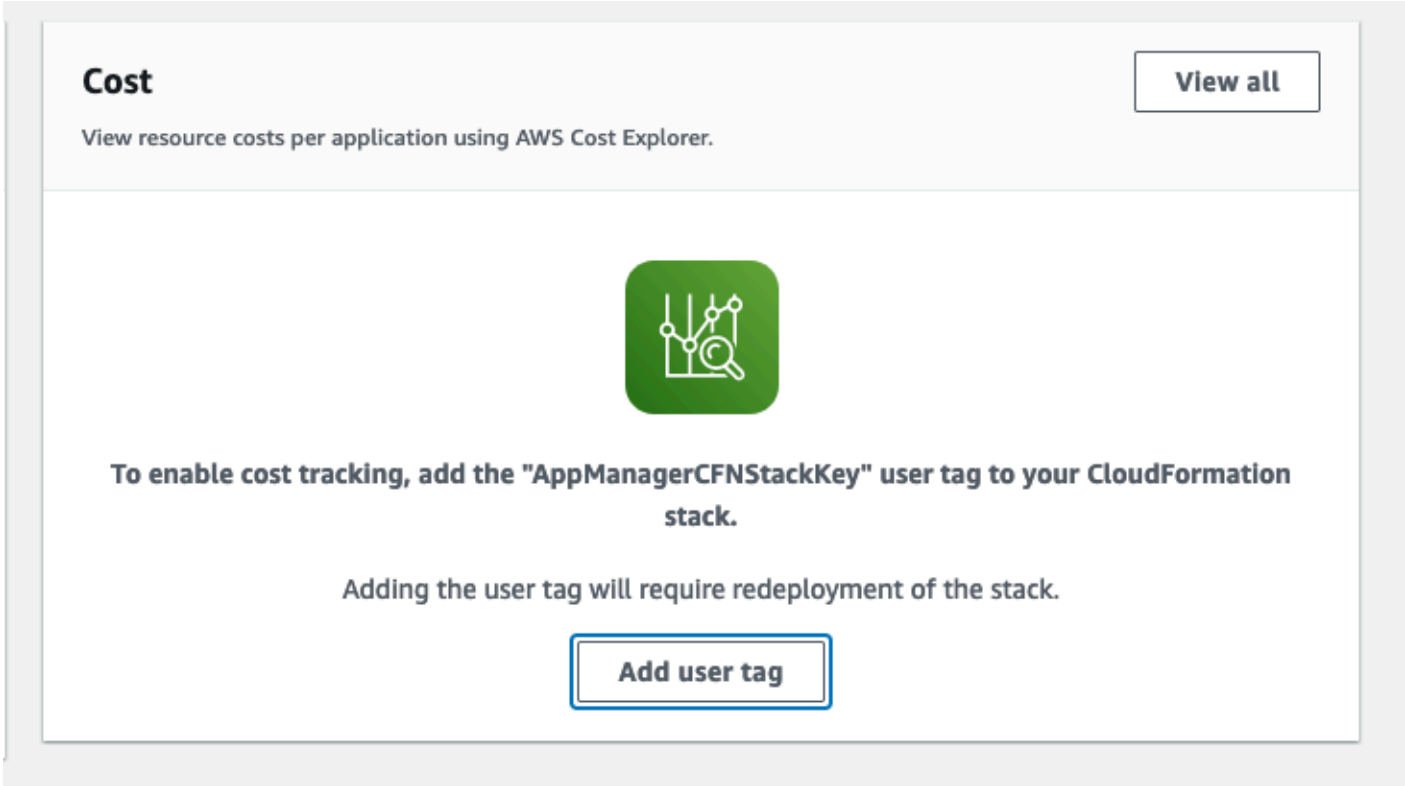
The activation process can take up to 24 hours to complete. Once activated, you can open the Cost Explorer user interface to further analyze cost data for the solution.

Confirm cost tags associated with the solution

After you activate cost allocation tags associated with the solution, you must confirm the cost allocation tags to see the costs for this solution. To confirm cost allocation tags:

1. Sign in to the Systems Manager console.
2. In the navigation pane, choose **Application Manager**.
3. In **Applications**, choose the application name for this solution and select it.

4. In the **Overview** tab, in **Cost**, select **Add user tag**.



5. On the **Add user tag** page, enter `confirm`, then select **Add user tag**.

The activation process can take up to 24 hours to complete and the tag data to appear.

Activate cost allocation tags associated with the solution

After you activate Cost Explorer, you must activate the cost allocation tags associated with this solution to see the costs for this solution. The cost allocation tags can only be activated from the management account for the organization. To activate cost allocation tags:

1. Sign in to the AWS Billing and Cost Management and Cost Management console.
2. In the navigation pane, select **Cost Allocation Tags**.
3. On the **Cost allocation tags** page, filter for the `AppManagerCFNStackKey` tag, then select the tag from the results shown.
4. Choose **Activate**.

The activation process can take up to 24 hours to complete and the tag data to appear.

Uninstall the solution

You can uninstall the Migration Assistant for Amazon OpenSearch Service solution from the AWS Management Console or by using the AWS Command Line Interface. Manually remove the contents of the bucket that matches cdk-*<unique id>*-assets-*<account id>*-*<region>* created by this solution. Migration Assistant for Amazon OpenSearch Service does not automatically delete S3 buckets and the target AWS OpenSearch cluster in case you have stored data to retain.

Using the AWS Management Console

1. Sign in to the CloudFormation console.
2. On the **Stacks** page, select this solution's installation stack. The stacks must be deleted in the following order:
   a. `migration-console`
   b. `traffic-replayer-default`
   c. `capture-proxy-es` (only available for demo install)
   d. `fetchMigrationStack`
   e. `mskUtilityStack`
   f. `migrationInfraStack`
   g. `openSearchDomainStack-default`
   h. `networkStack-default`
3. On the **Stacks** page, delete the bootstrap stack `Migration-Assistant-Bootstrap`.
4. Choose **Delete** for each of the above.

Using AWS Command Line Interface

Log in to the `migration-<stage>-deployment-box` container and run the following command:

```
$ cdk destroy "*" --c contextId=<context id>
```

> **Note**
>
> The default retention policy for the OpenSearch domain is to `RETAIN` the resource when the stack is deleted. To delete the domain on stack deletion, the **domainRemovalPolicy** must be set to `DESTROY`.

Alternatively, the domain can be manually deleted through the AWS Management console or AWS CLI.

After the Migration Assistant for Amazon OpenSearch Service solution stacks *(see step 2. in the Using the AWS Management Console section)* have been removed, remove the bootstrap CloudFormation template.

```
$ cdk destroy "Migration-Assistant-Bootstrap"
```

Determine whether the AWS Command Line Interface (AWS CLI) is available in your environment. For installation instructions, see What Is the AWS Command Line Interface in the *AWS CLI User Guide*. After confirming that the AWS CLI is available, run the following command.

```
$ aws cloudformation delete-stack --stack-name <stack
names as listed in step 2 (a-h) above>
```

Use the solution

For information on how to use this solution, refer to the User Guide on the OpenSearch Migrations GitHub page.

Developer guide

Source code

Visit our GitHub repository to download the source files for this solution and to share your customizations with others.

> **Note**
>
> Report technical issues with the solution on the Issues page of the GitHub repository.

AWS CDK generates the solution's bootstrap template. See the README.md file for additional information.

After the Migration Assistant for Amazon OpenSearch Service bootstrap is installed, install additional resources using CDK templates. For more information, refer to the opensearch-migrations GitHub repository.

Reference

This section includes information about an optional feature for collecting unique metrics for this solution, pointers to related resources, and a list of builders who contributed to this solution.

Anonymized data collection

This solution includes an option to send anonymized operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. When invoked, the following information is collected and sent to AWS:

- **Solution ID** - The AWS solution identifier
- **Unique ID (UUID)** - Randomly generated, unique identifier for each Migration Assistant for Amazon OpenSearch Service deployment
- **Timestamp** - Data-collection timestamp

AWS owns the data gathered though this survey. Data collection is subject to the Privacy Notice. To opt out of this feature, complete the following steps before launching the AWS CloudFormation template.

1. Download the AWS CloudFormation template to your local hard drive.
2. Open the AWS CloudFormation template with a text editor.
3. Modify the AWS CloudFormation template mapping section from:

```
AnonymizedData:
    SendAnonymizedData:
      Data: Yes
```

   to:

```
AnonymizedData:
    SendAnonymizedData:
      Data: No
```

4. Sign in to the AWS CloudFormation console.
5. Select **Create stack**.

6. On the **Create stack** page, specify template section, then select **Upload a template file**.
7. Under **Upload a template file**, choose **Choose file** and select the edited template from your local drive.
8. Choose **Next** and follow the steps in Launch the stack in the Deploy the solution section of this guide.

Contributors

- Kartik Ganesh
- Chris Helma
- Omar Khasawneh
- Tanner Lewis
- Brian Presley
- Greg Schohn
- Himanshu Setia
- Mikayla Thompson

Revisions

Date	Change
November 2023	Initial release

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents AWS current product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided "as is" without warranties, representations, or conditions of any kind, whether express or implied. AWS responsibilities and liabilities to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Migration Assistant for Amazon OpenSearch Service is licensed under the terms of the Apache License Version 2.0 available at The Apache Software Foundation.