Amazon OpenSearch Service Upgrade Guide

A Guide To Safe and Easy Migration

and How To Avoid Extended Support Fees

BIGDATA BOUTIQUE

Produced and maintained by BigData Boutique,
your Amazon OpenSearch Service partners,
and the creators of Pulse – the proactive AI SRE for OpenSearch

Contents

About This Guide **3**
The Safe Upgrade Procedure **5**
Pre-Upgrade Checklist 5
Select An Appropriate Upgrade Method 6
During the Upgrade 9
Post-Upgrade Tasks 9
Rollback Considerations 10
Upgrade Paths **11**
Elasticsearch 6.x → 6.8 11
Elasticsearch 7.x → 7.10 11
OpenSearch 2.x → 2.11+ 12
OpenSearch 1.x → 1.3 12
OpenSearch 1.x → 2.11+ 12
Elasticsearch 7.x → OpenSearch 2.11+ 14
Elasticsearch 6.8 (AWS) → OpenSearch 2.11+ 14
Upgrading to 3.x 14
Elasticsearch < 6.8 → OpenSearch 3.x 17
Upgrade Methods **18**
Quick Decision Matrix 18
In-Place Upgrade (Blue/Green) 19
Important Notes 19
From AWS Console 20
CloudFormation 20
Terraform 21
Snapshot and Restore 21
Reindexing to New Cluster 22
Using OpenSearch Migration Assistant 25

About This Guide

Starting November 7th, 2025 an Extended Support Fee is going to be imposed on all Amazon OpenSearch Service clusters (aka Domains) running the following versions:

- Elasticsearch versions 1.5 and 2.3
- Elasticsearch versions 5.1 to 5.6, 6.0 to 6.7
- Elasticsearch versions 7.1 to 7.8
- OpenSearch versions 1.0 through 1.2
- OpenSearch versions 2.3 to 2.9

As an example, for a medium-sized cluster with 10 r7g.2xlarge nodes and 3 m7g.large master nodes running a version under extended support, the surcharge would be $26.83 per day ($806.96/month or $9,793.68/year) in US East, calculated as: data nodes at $0.0065 × 24 hours × 16 (normalization factor) × 10 nodes = $24.96/day, plus master nodes at $0.0065 × 24 hours × 4 (normalization factor) × 3 nodes = $1.87/day.

While being on recent versions of OpenSearch is also a good idea in terms of price performance, starting November it's also going to be cheaper.

This guide was created to provide a detailed, step by step process on planning and executing your upgrade to make sure you don't pay unnecessary fees.

This guide is based on the official guidance from AWS as well as our 15+ years of experience supporting thousands of customers over the years.

In this guide:

- We explain how to execute a Safe Upgrade Procedure and how to plan for it.
- We help you Select the right Upgrade Method for you and provide a Quick Decision Matrix for reference.

- We provide all possible Upgrade Methods and guide you through each.
- A good starting point is the list of Upgrade Paths, given your current cluster version we'll guide you on what to look for and what to do next to reach the latest versions.

If you are still confused or unsure, we are happy to provide support and consultation. Reach out to us here: https://bigdataboutique.com/services/opensearch-consulting.

The Safe Upgrade Procedure

Pre-Upgrade Checklist

1. Check cluster health - Ensure your cluster status is green and all indices are healthy. Your domain must have:
   - Green or yellow cluster status (not red)
   - Low shard count per node
   - Low error rate
   - Disk usage < 90%
   - CPU average < 85% on all nodes
   - JVM memory pressure < 75%
2. Consider connecting your cluster to Pulse (Free tier is sufficient) to automate those checks, before and during the migration; alternatively set up proper Grafana or CloudWatch dashboards and alerts
3. Take a manual snapshot - Create a backup of your cluster data before proceeding to a dedicated, separate S3 bucket. Do not rely on the backup (snapshot) provided by the service itself in case you need to rollback or recover to a new cluster.
4. Review breaking changes - Check the OpenSearch release notes for any breaking changes between versions. We have highlighted any specifics in the Upgrade Paths section.
5. Test in non-production first - Perform the upgrade on a staging/development cluster first, including any necessary client upgrades (producers and consumers)
6. The most important element to check is the various applications connecting to the cluster, and their functionality. Each upgrade step needs to verify the applications are using a client library that is supporting the new version before starting the upgrade, for example by deploying a branch to test on the two cluster versions (previous and future versions).

7. Plan for downtime - You can never be too careful. Understand whether your chosen upgrade method requires downtime and plan for it
8. Check domain configuration - Review instance types, storage, and ensure compatibility with the target version
9. Monitor resource utilization - Ensure adequate storage and memory headroom (recommended 25% free)

Select An Appropriate Upgrade Method

In-Place Upgrade (Blue/Green)

Best for:

- Single major version upgrades (e.g., Elasticsearch 7.1 to 7.10, OpenSearch 2.1 → 2.11, or 2.x → 3.x if one version jump)
- Straightforward upgrades without significant mapping or configuration changes
- When cluster architecture doesn't need significant changes
- Upgrades with zero ingestion and operations downtime (in some cases, your OpenSearch Dashboards might experience some downtime)

Limitations:

- Cannot skip multiple major versions
- Very old Elasticsearch versions (1.x, 2.x, 5.x, 6.x) cannot directly upgrade to OpenSearch 2.11+ or 3.x
- Cannot change fundamental cluster architecture during upgrade
- No cutoff decision point and cannot allow for validations in canary-like environment
- No rollback capability once completed

Use when: Your cluster is in a major version that has a latest version that is still under standard support (e.g. 6.0 to 6.8 , 7.1 to 7.10), or it is relatively recent (OpenSearch 1.x or 2.x, or Elasticsearch 7.10) and you need a quick, simple, zero-downtime upgrade to a more modern version which is still under standard support, with minimal operational overhead.

Using OpenSearch Migration Assistant

Best for:

- Complex upgrades - from very old versions, where validations are required safety is a priority and zero downtime is required
- Multi-version jumps - Upgrading from very old versions directly to latest (e.g., Elasticsearch 2 → OpenSearch 2.11+), especially legacy Elasticsearch clusters on Amazon OpenSearch Service
- Zero-downtime requirements with validation - This is the only option that offers side-by-side validation with zero downtime, when you deed to test the target cluster with live traffic before cutover
- Large, mission critical, active clusters - High-volume production workloads that cannot tolerate downtime
- Complex migrations needing metadata transformation - Handling type mapping deprecation, field type changes, flattened fields
- Rollback capability required - Want to validate new cluster and revert if issues arise

Limitations:

- Requires additional infrastructure deployment (ECS/Fargate, capture proxy)
- Live capture recommended only for < 4 TB/day workloads
- Does not migrate ISM policies and security constructs
- Automatically generated document IDs not preserved during replay
- More operationally complex than in-place upgrade
- Significant set up time and non-negligible cost
- Requires specific networking configurations

Use when: You have very old Elasticsearch versions (5.x, 6.x, 7.x, even 1.x) that cannot use in-place upgrade, need zero-downtime for large production workloads, or require traffic validation before final cutover.

This is the recommended approach for legacy Elasticsearch clusters on Amazon OpenSearch Service migrating to OpenSearch 2.11+ or 3.x.

Snapshot and Restore

Best for:

- When you need to change cluster configuration significantly (VPC settings, deployment location)
- Moving between AWS accounts or regions
- Smaller clusters (< 1 TB) where downtime is acceptable
- Disaster recovery scenarios
- Quick migrations where operational simplicity is preferred over zero-downtime

Limitations:

- Requires downtime
- Version compatibility constraints (cannot restore snapshot from very old versions to newest versions directly)
- Manual process requiring coordination
- Slower for very large datasets

Use when: You have a maintenance window available, need to change cluster infrastructure significantly, and you are dealing with a manageable data size where downtime is acceptable.

Reindexing to New Cluster

Best for:

- Restructuring data or changing index mappings during migration
- When you need full control over the migration process
- Breaking changes in mappings or field types between versions
- Opportunity to clean up or optimize data during migration
- Selective data migration (only migrate specific indices)

Limitations:

- Most operationally complex method
- Requires managing two clusters simultaneously

- Application changes needed to handle dual-write or cutover
- Higher cost during migration period (running two clusters)
- Time-consuming for large datasets

Use when: You need to transform data during migration, have breaking mapping changes, or want the cleanest possible upgrade path with data optimization.

During the Upgrade

1. Monitor the upgrade progress - Track status in the AWS Console
2. Recommended: Connect your cluster to Pulse (Free tier) to monitor and get proactive health assessment in real-time during the upgrade.
3. Avoid cluster changes - Don't modify configuration, indices, or mappings during upgrade
4. If possible halt unnecessary operations (read, writes) to ease the load on the cluster. The upgrade will finish faster this way.
5. Watch for failures - The upgrade may roll back automatically if issues occur
6. Check cluster metrics on CloudWatch or Pulse - Monitor cluster performance and resource utilization to ensure no service disruptions
7. Be patient - Upgrades can take 30 minutes to several hours depending on cluster size

Post-Upgrade Tasks

1. Verify cluster health - Confirm cluster status is green, all nodes are active and that node performance is within norm
2. Test application connectivity - Ensure applications can connect and query successfully, as well as all data producers and data updates from applications and utilities
3. Validate data integrity - Spot-check critical indices and document counts
4. Review deprecation warnings - Check logs for deprecated features you're using

5. Update client libraries - Upgrade application SDKs to versions compatible with the new OpenSearch version
6. Monitor performance - Watch metrics for several days to identify any degradation
7. Update cluster settings - Take advantage of new features or recommended settings
8. Remove old snapshots - Clean up pre-upgrade snapshots after confirming stability

Rollback Considerations

In-place upgrades cannot be rolled back automatically. If issues occur, you must restore from a snapshot to a new cluster, which is why pre-upgrade snapshots are critical.

Upgrade Paths

To avoid the Extended Support fees, it is enough to get to the latest 2.x or to the latest versions at the recent majors (6.8, 7.10, 1.3).

We generally recommend upgrading to the latest 2.x version (currently 2.19). Upgrading to 3.0 might require additional steps and therefore might pose additional risk. Nevertheless, the process is detailed below.

If you are on older versions of Elasticsearch, it is most likely going to be faster and easier for you to upgrade to 3.x immediately using the Migration Assistant or using a side-by-side approach (Reindexing, for example).

We have listed all common scenarios below.

Elasticsearch 6.x → 6.8

While we recommend upgrading to 7.10, this is a seamless upgrade that'd still keep you at standard support and away from Extended Support Fees. An upgrade within the same major version would guarantee there are no breaking changes. Follow The Safe Upgrade Procedure and execute using In-Place Upgrade (Blue/Green).

Elasticsearch 7.x → 7.10

This is a seamless upgrade that'd still keep you at standard support and away from Extended Support Fees. An upgrade within the same major version would guarantee there are no breaking changes. Follow The Safe Upgrade Procedure and execute using In-Place Upgrade (Blue/Green).

However it's worth noting that 7.10 is still an outdated version, and OpenSearch at recent versions provides significant benefits, from performance to important features.

OpenSearch 2.x → 2.11+

This is a seamless upgrade: All are within the same 2.x major version family and there are no breaking changes. Follow The Safe Upgrade Procedure and execute using In-Place Upgrade (Blue/Green).

OpenSearch 1.x → 1.3

While we recommend going to 2.11 or above, this is a seamless upgrade that'd still keep you at standard support and away from Extended Support Fees. An upgrade within the same major version would guarantee there are no breaking changes. Follow The Safe Upgrade Procedure and execute using In-Place Upgrade (Blue/Green).

OpenSearch 1.x → 2.11+

Amazon OpenSearch Service supports in-place upgrades from OpenSearch 1.0+ directly to 2.x versions.

However, this crosses from 1.x to 2.x major versions, requiring careful attention to some breaking changes, specifically with client libraries being used.

Critical Breaking Changes (1.x → 2.x):

1. Mapping Types Removal (Most Critical)
   - What changed: The type parameter removed from all OpenSearch API endpoints
   - Impact: All indexes must use _doc as the mapping type starting in 2.x
   - Action required:
     - Review all application code for mapping type references
     - Update API calls to use _doc instead of custom type names
     - Modify bulk operations, GET/DELETE/POST requests

- Update filter queries to remove type references

2. Index Compatibility
- Elasticsearch 6.8 indexes cannot be used directly in OpenSearch 2.x
- Required action: Must reindex incompatible indexes before upgrading to 2.x, if any exist
- Process: Upgrade to OpenSearch 1.x first, reindex using Reindex API, then upgrade to 2.x

3. Alerting & Notifications
- Change: Notification channels replaced alerting destinations (2.0+)
- In 2.3+, existing destinations are automatically migrated to notification channels
- Requirements:
  - Install notifications-core and notifications backend plugins
  - Use notificationsDashboards plugin for dashboard management
  - Most Destination APIs removed (except Get Destinations)

4. Client Libraries
- Client libraries in the 2.x version, for all platforms, are officially supporting 1.x clusters, but we have seen gaps in the support before
- There is a known issue with the Java client library version 2.x performing bulk writes to 1.x and having issues parsing the response.
- There may be other issues as well, so treat with caution.

5. Data Considerations
- Documents with >10,000 nested JSON objects may cause migration failures from 1.x

Elasticsearch 7.x → OpenSearch 2.11+

This requires a two-step upgrade (ES 7.x → OpenSearch 1.x → OpenSearch 2.11+), but it is fully supported via In-Place Upgrade (Blue/Green).

OpenSearch 1.x maintains high compatibility with ES 7.x. Migrate ES 7.x to OpenSearch 1.x following the in-place upgrade instructions and our The Safe Upgrade Procedure, and then repeat a similar process following "OpenSearch 1.x → 2.11+" instructions above.

Elasticsearch 6.8 (AWS) → OpenSearch 2.11+

A multi-step upgrade is required, following this path: ES 6.8 → ES 7.10 → OpenSearch 1.x → OpenSearch 2.11+. Essentially, following this process with the In-Place Upgrade method:

1. ES 6.8 → ES 7.10:
   - In-place upgrade
   - Address mapping types deprecation warnings
2. ES 7.10 → OpenSearch 1.x:
   - In-place upgrade
   - Update clients and plugins
3. Reindex all data in OpenSearch 1.x
4. OpenSearch 1.x → OpenSearch 2.11+:
   - Follow "OpenSearch 1.x → 2.11+" instructions

Alternatively, using the OpenSearch Migration Assistant (no downtime) or Reindexing (with potential downtime) methods to do this in one go.

Upgrading to 3.x

OpenSearch 3.1 is now available on Amazon OpenSearch Service (released September 2025) in all AWS regions. This is a major version upgrade from 2.x with significant

breaking changes, which is why we recommend reaching 2.19 first following the paths above, and then performing the upgrade to 3.x; unless if you are coming from a significantly older version and leveraging the Migration Assistant or Reindex methods, in which case you could upgrade to latest 3.x directly.

Critical Breaking Changes in OpenSearch 3.x:

1. Mandatory Upgrade to 2.19 First before upgrading to 3.x
 - OpenSearch 3.0 is only wire-compatible with the last minor version of 2.x (2.19)
 - Rolling upgrades support only adjacent major versions
 - Cannot skip from 2.11 (or any other 2.x version) directly to 3.0/3.1
 - Must reindex all OpenSearch 1.x/Elasticsearch 7.x indexes before upgrading to 3.x

2. Document ID Length Enforcement
 - Version 3.x introduced a breaking change: Document ID length limit of 512 bytes now consistently enforced
 - Action Required: audit existing document IDs for length. If long document IDs exist, update document generation logic if creating long IDs and may need to hash or truncate long IDs.

3. New Limits Introduced to JSON Document Processing
 - Maximum nesting depth is now 1,000 levels
 - Deeply nested JSON objects/arrays limited
 - Maximum property name length: 50,000 units
 - Action Required: Review documents with complex nested structures. May need to flatten deeply nested documents
 - Test with representative production data

4. Query Nesting Depth Limit
 - New Index Setting: index.query.max_nested_depth limits complexity of nested queries to 20

- Action Required: Review complex nested queries and adjust setting if needed for specific indexes

5. k-NN Engine Changes

- NMSLIB engine was deprecated in OpenSearch 3.0. If in use, migrate to faiss or Lucene.
- Reindex k-NN indexes with new engine configuration
- k-NN Setting Changes (must remove deprecated settings before upgrading to 3.1):
    - index.knn.algo_param.ef_construction
    - index.knn.algo_param.m
    - index.knn.space_type
    - index.store.hybrid.mmap.extensions

```JSON
// Deprecated (set in index settings):
{
  "settings": {
    "index.knn.algo_param.ef_construction": 512,
    "index.knn.algo_param.m": 16,
    "index.knn.space_type": "l2"
  }
}

// Correct (set in field mapping):
{
  "mappings": {
    "properties": {
      "my_vector": {
        "type": "knn_vector",
        "dimension": 128,
        "method": {
          "name": "hnsw",
```

```
        "space_type": "l2",
        "engine": "faiss",
        "parameters": {
         "ef_construction": 512,
         "m": 16
        }
       }
      }
     }
    }
   }
```

6. JavaScript Client Breaking Changes
 - Strict parameter naming: No more camelCase flexibility
 - Typing system changes: May cause compatibility issues
 - Parameter names must match API exactly
 - Action Required: Update JavaScript/TypeScript client code and test all client interactions

Elasticsearch < 6.8 → OpenSearch 3.x

Follow paths above to reach 2.11+, then upgrade to 3.x. This is going to be a long multi-step process if going with the In-Place Upgrade (Blue/Green) , and so we recommend going with the OpenSearch Migration Assistant approach (for large clusters, and when no downtime is desired) or deploying a new cluster side-by-side and migrating through Reindexing or full applicative data rebuild if possible.

Upgrade Methods

There are several possible upgrade methods you could use. The most common, and recommended, ones are In-Place Upgrade (Blue/Green) (for most cases) and Using OpenSearch Migration Assistant for more complex or high stakes situations.

However, we provide additional methods which we sometimes see used. Those could be useful and even recommended in some scenarios, so they are brought forward here as well.

The pros and cons are described above under Select An Appropriate Upgrade Method.

For your convenience, here is a mapping of the various paths and the recommended method for each:

Quick Decision Matrix

Scenario	Recommended Method
Same major (e.g. Elasticsearch 7.x → 7.10, OpenSearch 2.x → latest 2.19)	In-Place Upgrade (Blue/Green)
Single major jump (eg OpenSearch 1.x → 2.x, Elasticsearch 6.x → Elasticsearch 7.10)	In-Place Upgrade (Blue/Green) noting any breaking changes
Elasticsearch 7.10 → OpenSearch 2.x	In-Place Upgrade (Blue/Green) (multiple steps)
Elasticsearch 6.x/7.x → OpenSearch 2.11+ or 3.x	In-Place Upgrade (Blue/Green) (many steps) or Migration Assistant
Very old Elasticsearch (1.x through 6.x) → latest OpenSearch	Migration Assistant or Reindexing to New Cluster if you can afford partial or full downtime

Need to change cluster architecture	Snapshot and Restore or Reindexing to New Cluster if downtime is acceptable, or Use Migration Assistant if not
Need traffic validation before cutover	Migration Assistant
Zero downtime + multiple version jumps	We recommend using the OpenSearch Migration Assistant

If you are still confused or unsure, we are happy to provide support and consultation. Reach out to us here: https://bigdataboutique.com/services/opensearch-consulting.

In-Place Upgrade (Blue/Green)

This is Amazon OpenSearch Service's default and most recommended upgrade method. The service creates a new environment with the upgraded version, migrates data, and switches traffic automatically. Provides zero downtime for most configurations. Best for production clusters.

Important Notes

- Some versions require incremental upgrades (e.g., 1.x → 2.x → 3.x)
- An upgrade is only recommended once the cluster is under control, and CPU is consistently below 85%. Do not start an upgrade process before this is achieved.
- Duration: Varies based on cluster size and data volume (typically 15 minutes to several hours)
- It is possible for the upgrade to "get stuck". It sometimes happens. Make sure you monitor the upgrade closely, and we recommend notifying AWS Support *prior* to executing the upgrade as they can help monitor and unstick when needed
- Service software updates: May be required separately from version upgrades

From AWS Console

1. Navigate to Amazon OpenSearch Service → Select your domain
2. Click Actions → Upgrade domain
3. Select target version (e.g. 2.11)
4. Enable Dry Run (recommended) to validate the upgrade without applying changes
   - Reviews compatibility issues and generates a report
   - No downtime or configuration changes occur
5. Review the dry run report for any issues
6. If successful, perform the actual upgrade by repeating steps 2-3 without dry run
7. Monitor the upgrade progress in the domain dashboard

CloudFormation

```
None
 Resources:
  OpenSearchDomain:
   Type: AWS::OpenSearchService::Domain
   Properties:
    DomainName: my-domain
    EngineVersion: OpenSearch_Y  # Update version here
```

Dry run using AWS CLI:

```
Shell
 aws opensearch upgrade-domain \
  --domain-name my-domain \
  --target-version OpenSearch_Y \
  --perform-check-only
```

Update stack with aws cloudformation update-stack or through the console.

Terraform

Update the aws_opensearch_domain resource to use the desired target version:

```
None
 resource "aws_opensearch_domain" "example" {
  domain_name    = "my-domain"
  engine_version = "2.19"  # Update version here
  # ... other configuration
 }
```

Dry run using AWS CLI:

```
Shell
 aws opensearch upgrade-domain \
  --domain-name my-domain \
  --target-version 2.19 \
  --perform-check-only
```

Apply with terraform apply after validating dry run results.

Follow similar steps to other IaaC tooling such as Pulumi, Ansible, etc.

Snapshot and Restore

Take a manual snapshot from the old cluster, create a new cluster with the desired version, and restore the snapshot. Requires manual coordination and involves downtime. Useful for major version jumps or when changing cluster architecture significantly (for example moving to a different AWS account or VPC configurations).

You can follow this guide for creating and restoring from manual snapshots. Official documentation: https://docs.aws.amazon.com/opensearch-service/latest/developerguide/managedomains-snapshots.html .

Reindexing to New Cluster

Create a new cluster with the target version and reindex data from the old cluster using the remote reindex API.

This approach provides the most control and cleanest upgrade path. Suitable for complex migrations or when restructuring data, but is definitely not the easiest.

1. Create Target Cluster
 - Provision new Amazon OpenSearch Service domain with target version
 - Configure networking to allow connectivity from source cluster

2. Set Up Cross-Cluster Connectivity
 Add source cluster as remote:

```
JSON
 PUT /_cluster/settings
 {
  "persistent": {
   "cluster": {
    "remote": {
     "source_cluster": {
      "seeds": ["source-cluster-endpoint:9300"]
     }
    }
   }
  }
```

```
}
```

3. Migrate Index Templates and Settings

```JSON
# Export from source cluster
GET /_template/my-template

# Create on target cluster with any necessary modifications
PUT /_template/my-template
{
 "index_patterns": ["logs-*"],
 "settings": {
  "number_of_shards": 3,
  "number_of_replicas": 2
 },
 "mappings": { ... }
}
```

4. Reindex Data

Reindex from remote can be done with the _reindex API. You can also leverage parallel reindexing using the ?slices=5 argument.

```JSON
POST /_reindex
{
 "source": {
  "remote": {
   "host": "https://source-cluster:443",
   "username": "admin",
```

```
    "password": "password"
   },
   "index": "source-index",
   "query": {
    "match_all": {}
   }
  },
  "dest": {
   "index": "target-index"
  }
}
```

6. Handle Active Writes

For active clusters, implement dual-write or use aliases:

JSON

```
# Create alias pointing to both old and new
POST /_aliases
{
 "actions": [
  { "add": { "index": "old-index", "alias": "my-alias" } },
  { "add": { "index": "new-index", "alias": "my-alias" } }
 ]
}

# After migration complete, remove old index from alias
POST /_aliases
{
 "actions": [
  { "remove": { "index": "old-index", "alias": "my-alias" } }
```

```
  ]
}
```

7. Verification and Cutover
- Compare document counts between clusters
- Validate sample data and queries
- Update application to point to new cluster
- Monitor for several days before decommissioning old cluster

Using OpenSearch Migration Assistant

The Migration Assistant is a comprehensive migration toolkit for complex, multi-version upgrades with zero downtime and traffic validation capabilities.

The Migration Assistant deploys on your AWS account several components:
- Capture Proxy - Intercepts and duplicates traffic to both clusters
- Traffic Replayer - Replays captured traffic to target cluster
- Migration Console - Manages and monitors migration
- Metadata Migration - Handles transformations for breaking changes

Those are required to facilitate live traffic capture and replay for validation, and metadata migration with automatic transformations.

With the Assistant, there are three migration patterns:
- Backfill only - For lower-traffic scenarios or when downtime is acceptable
- Live capture only - For continuous data streams without historical data
- Live capture + backfill - Complete migration with zero data loss

- Live capture recommended for < 4 TB/day workloads

Limitations:
- Auto-generated document IDs not preserved during replay
- ISM policies require manual migration
- Security configurations (roles, users) need separate migration
- Some plugins may not be supported

Network Requirements:
- VPC peering or PrivateLink between source and target
- Security groups allowing traffic between components
- Load balancer for Capture Proxy

Follow the official documentation for the actual process, or contact us at BigData Boutique to consult further.