AWS Whitepaper

Amazon MSK Migration Guide

Amazon MSK Migration Guide: AWS Whitepaper

Table of Contents

Amazon MSK Migration Guide

Publication date: **September 30, 2021** (*Document history*)

This guide covers options for your Apache Kafka migration to Amazon Managed Streaming for Apache Kafka (Amazon MSK). It provides guidance on the AWS Cloud infrastructure and migration fundamentals. This whitepaper details the Amazon MSK architecture and discusses architectural best practices around the AWS Well-Architected pillars of operational excellence, security, reliability, performance efficiency, and cost optimization.

Introduction

To be competitive and able to scale, businesses are reinventing their analytics applications around data streams, to reduce time-to-insight and to improve agility. Apache Kafka is one of the most widely adopted open source streaming platforms for ingesting and processing real-time data streams, enabling customers to decouple and independently scale data producing and data consuming applications. Apache Kafka is a distributed data store optimized for ingesting and processing streaming data in real time. Streaming data is data that is generated continuously by thousands of data sources, which typically send in the data records simultaneously, and in small sizes (order of Kilobytes). Streaming data includes a wide variety of data such as log files generated by customers using your mobile or web applications, ecommerce purchases, in-game player activity, information from social networks, financial trading floors, geospatial services, and telemetry from connected devices or instrumentation in data centers.

Businesses may encounter challenges operating Apache Kafka on their own, or when trying to migrate open source Apache Kafka clusters to AWS. These challenges include a lack of agility deploying clusters, engineering obstacles when setting up self-managed Apache Kafka, and administrative operational overhead.

Operating a self-managed Apache Kafka environment requires customers to spend time on tasks including:

- Managing capacity to meet demand fluctuations
- Provisioning, configuring, and replacing servers on failure
- Orchestrating patch management and software upgrades
- Architecting for high availability, data durability, and security
- Monitoring and alerting infrastructure

- Enabling governance and security at scale

Amazon Managed Streaming for Apache Kafka (Amazon MSK) makes it easy for customers to build and run production applications on Apache Kafka without needing any Apache Kafka infrastructure management expertise. This means customers can spend less time managing infrastructure and more time building applications.

Customers choose to migrate to Amazon MSK for the following benefits:

Eliminate operational overhead -- Amazon MSK takes care of all operational overhead for your Apache Kafka environment, including the provisioning, configuration, and maintenance of highly available Apache Kafka clusters. Amazon MSK continuously monitors Apache Kafka and cluster health, automates patching and version upgrades, and shares key performance metrics in-console.

Migrate without changes to application code -- Amazon MSK deploys the latest versions of Apache Kafka so applications and tools built for Apache Kafka work with Amazon MSK out of the box, with no application code changes required.

Reduce time to production with native AWS integrations -- No other provider offers the breadth and depth of AWS integrations than Amazon MSK. These native integrations, such as private connectivity to an Amazon VPC or AWS IAM for authentication and authorization, allow you to quickly and easily deploy secure, production-ready applications.

Keep costs low with the most cost-effective provider -- Amazon MSK is the lowest-cost option for running managed Apache Kafka. The typical price, per gigabyte ingested, is as low as 1/13th the cost of the next best provider. Get started with Amazon MSK for less than $2.50 per day.

Amazon MSK provides the environment to integrate managed Amazon Managed Service for Apache Flink for Apache Flink applications natively. Apache Flink is a powerful open source streaming framework that can elastically scale to process data streams within Amazon MSK. Additionally, AWS Lambda supports Amazon MSK and Apache Kafka as events, which provides customers with more choices to build serverless streaming applications.

Amazon MSK also offers multiple levels of security for your Apache Kafka cluster including virtual private cloud (VPC) network isolation, AWS Identity & Access Management (IAM) for control-plane and data-plane API authorization, encryption of data at rest, Transport Layer Security (TLS) encryption in-transit, TLS-based client certificate authentication, SASL/SCRAM authentication secured by AWS Secrets Manager, and support for Apache Kafka Access Control Lists (ACLs) for data plane authorization.

This guide covers a range of scenarios to assist with your Apache Kafka migration to Amazon MSK. It provides guidance on:

- AWS Cloud infrastructure and migration fundamentals
- Migrating self-managed Apache Kafka to Amazon MSK
- Security, cost, performance, reliability (disaster recovery and high availability), and performance of Amazon MSK clusters
- Operating, monitoring, and maintaining an Amazon MSK cluster

Establishing reasons for migrating

Before you begin migration, you'll want to take some time to narrow down your reasons for migrating to Amazon MSK. This will help you identify or define key performance indicators (KPIs) that are reflective of your goals and methods.

AWS Cloud infrastructure basics

The AWS Global Cloud Infrastructure is the most secure, extensive, and reliable cloud platform, offering over 175 fully featured services from data centers globally. Whether you need to deploy your application workloads across the globe in a single click, or you want to build and deploy specific applications closer to your end-users with single-digit millisecond latency, AWS provides you the cloud infrastructure where and when you need it.

With millions of active customers and tens of thousands of partners globally, AWS has the largest and most dynamic ecosystem. Customers across virtually every industry and of every size, including startups, enterprises, and public sector organizations, are running every imaginable use case on AWS.

AWS uses the concept of a Region, which is a physical location around the world where AWS clusters data centers. Each group of logical data centers is called an Availability Zone. Each AWS Region consists of multiple, isolated, and physically separate AZs within a geographic area. Unlike other cloud providers, who often define a region as a single data center, the multi-AZ design of every AWS Region offers advantages for customers. Each Availability Zone has independent power, cooling, and physical security and is connected via redundant, ultra-low-latency networks. AWS customers focused on high availability can design their applications to run in multiple Availability Zones to achieve even greater fault-tolerance. Infrastructures in AWS Regions meet the highest levels of security, compliance, and data protection.

AWS offers a broad set of global cloud-based products including compute, storage, databases, analytics, networking, mobile, developer tools, management tools, IoT, security and enterprise applications. These services help organizations move faster, lower IT costs, and scale. AWS is trusted by the largest enterprises and the hottest startups to power a wide variety of workloads including: web and mobile applications, streaming applications, game development, data processing and warehousing, storage, archival, and many more.

Migrating to AWS

AWS customers cite several reasons for moving their workloads to the cloud, including the following reasons:

- **Increased agility.** Customers, whether they are born-in-the-cloud newcomers or well-established, long-lived enterprises, choose to leverage the cloud to allow their teams to experiment more quickly, learn from successes and mistakes expediently, and iterate more tightly.
- **Improved capacity delivery with reduced effort.** On-premises enterprises report repeatedly spending multiple months of effort estimating their data center capacity needs only to discover that their estimation error margins are far too wide and their delivery timelines far too long, resulting in overspending precious capital resources to mitigate capacity risks.
- **Shift from capital, fixed expenses to variable, operating expenses.** Fixed depreciation schedules that can't be changed after big, stair-step capital purchases lock companies in to commitments for longer than their ever-evolving business plans demand. Moving to the cloud allows quicker reallocation of resources based on learnings from experimentation and allows expenses to reflect actual demand for technical capacity over time.
- **A broad and deep library of pre-built technical services and capabilities**. The cloud providers bring with them a library of services and capabilities that can be assembled in an infinite number of ways to meet business requirements. This broad and deep library of capability frees technical staff to spend their time building capabilities that deliver real business value to their firm's customers, instead of the undifferentiated heavy lifting that is repeated across virtually every firm and that fails to differentiate them from their competition.
- **Global delivery capability in minutes.** Because cloud resources can be deployed using declarative templates or imperative programming languages, resources can be deployed repeatedly around the globe in minutes. Rather than repeatedly justifying, planning, and deploying data centers globally, cloud customers can deploy to the regions closest to their customers or in other preferred geographies in only minutes.

Defining Migration KPIs

Once you have established your reasons for migrating from a self-managed Apache Kafka cluster to Amazon MSK, you should be able to identify or define key performance indicators (KPIs) that are reflective of those goals and methods.

For example, if you are seeking to improve your agility in using new Apache Kafka features, you might measure the number and size of new features utilized by the team. If you are seeking to reduce your total cost of ownership (TCO), you might measure the number of staff-hours used to maintain the cluster. If you are trying to improve your ability to go global, you might track the effort-time or calendar-time it takes to stand up your Apache Kafka clusters in a new region to serve local customers. Additional KPIs can also include time-sensitive factors such as uptime, total time to migrate and performance measured by throughout, consumer/producer latency, consumer lag, etc.

Regardless of the goals, it is important to plan ahead to define and measure KPIs that reflect those goals before, during, and after the migration. Now is the time to put thought into how you will monitor the success of your migration. By planning for and implementing a mechanism to measure KPIs ahead of time, you can validate the business case for your migration and monitor KPIs at various phases of the migration to take corrective action if your work is not delivering as you had planned.

Prototyping

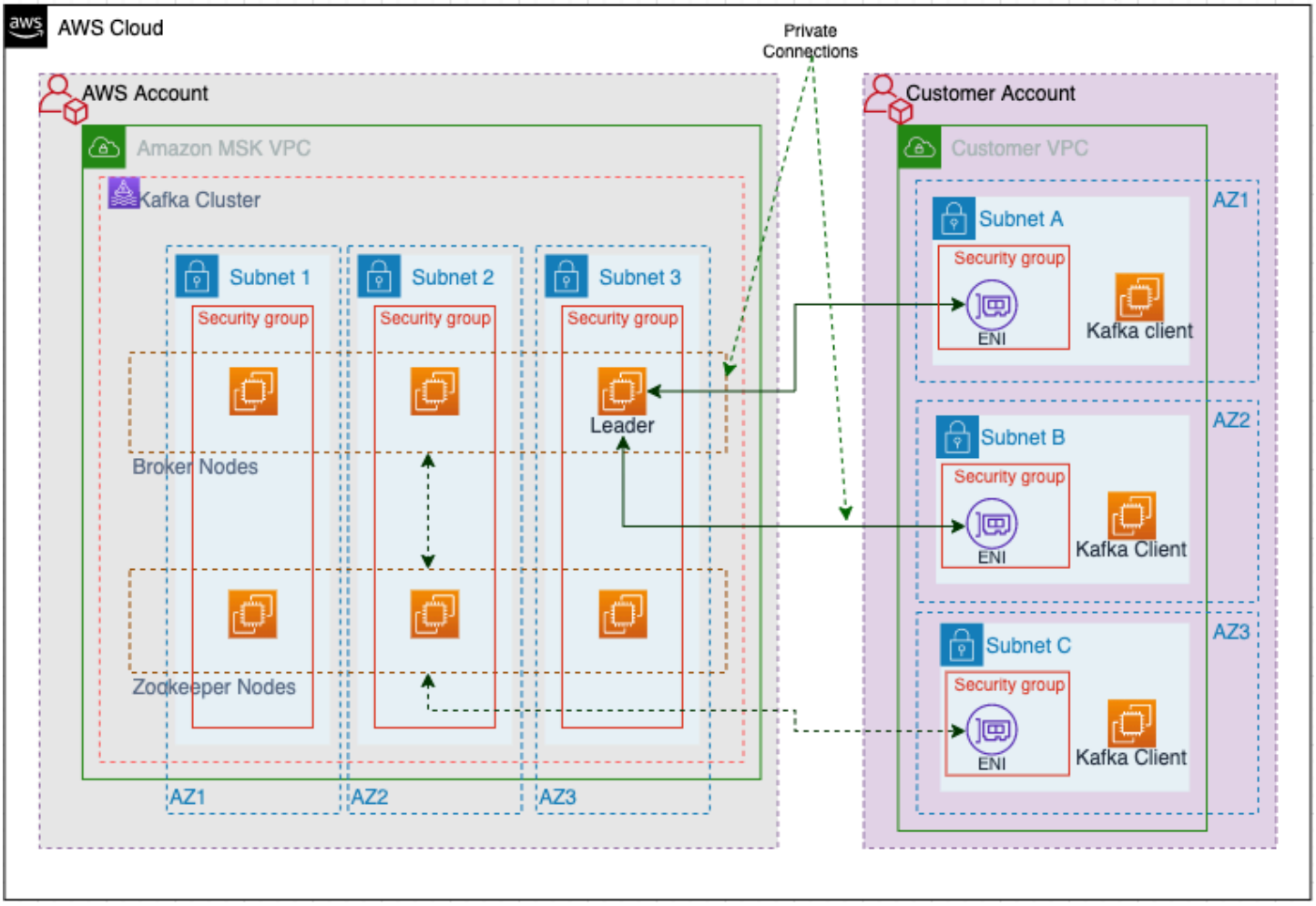
When migrating a workload to the cloud, you may run into many assumptions that were valid when the workload was originally built, but now need to be re-evaluated. These differences can occur as a result of changes between component versions, iterating business goals, or the evolution of technology. It is important to identify and revalidate these assumptions during the migration. One of the most successful methods for the validation of assumptions is prototyping.

Prototyping enables active learning on the part of builders prior to building the actual deliverable. This learning can result in a more effective final delivery. A common assumption to validate for streaming application migrations is that an architecture should function identically before and after migration. Oftentimes, the deep catalog of cloud services available during a migration can fundamentally change the architecture in beneficial ways. The question you need to ask is, “Can this function be achieved more efficiently or effectively by a separate service post-migration?”

Amazon Managed Streaming for Apache Kafka (Amazon MSK)

Infrastructure

Before you can create an Amazon MSK cluster, you need to have an Amazon Virtual Private Cloud (Amazon VPC) and subnets set up within that VPC. In most AWS Regions where Amazon MSK is available, you can specify either two or three subnets. The subnets must all be located in different Availability Zones. When you create a cluster, Amazon MSK distributes the broker nodes evenly across the subnets you specify. The following reference architecture shows an example of an Amazon MSK cluster infrastructure.



Amazon MSK cluster infrastructure

In this reference architecture, the brokers and Apache ZooKeeper nodes are deployed into an AWS service-managed VPC, dedicated to each cluster. You can limit access to the Apache ZooKeeper nodes by assigning them a separate security group. The brokers in the cluster are made accessible to clients in the VPC through elastic network interfaces that appear in the customer account. Traffic between clients and brokers is private by default, meaning it doesn't transfer over the public internet. The Security Groups on the network interfaces activate protocol and port-based traffic filtering on the brokers.

The IP addresses from the customer VPC are attached to the network interfaces, and by default all network traffic stays within the AWS network and is not accessible to the internet. Connections between clients and an Amazon MSK cluster stay private.

Amazon MSK provides control-plane operations for creating, updating, and deleting clusters. Customers use Apache Kafka data-plane operations for producing and consuming data. For a complete list of operations, see the Amazon MSK API Reference Guide. For more information on cluster provisioning, see Getting Started Using Amazon MSK.

At the time of this publication, Amazon MSK supports instance types from the M and T instance families. For the most up-to-date list of supported instance types, see Broker types.

Reliability

The following reliability best practices are recommended when running Amazon MSK:

- Deploy Amazon MSK clusters across three Availability Zones to ensure high availability. This is the default cluster provisioning configuration.
- Ensure a replication factor (RF) of 3 or at least 2 for each topic. A RF of 1 can lead to offline partitions during a rolling update or broker failure scenarios.
- Set minimum in-sync replicas (minISR) to RF -1. A minISR that is equal to the RF can prevent producing messages to the cluster during a rolling update. With a minISR of 2, three-way replicated topics are available when one replica is offline.
- Upgrade to the recommended version of Apache Kafka on Amazon MSK. Upgrading your cluster software ensures that you are using the latest stable features from open source applications, including:
  - Performance enhancements that help applications run faster.
  - Apache Kafka bug, and security fixes, and feature enhancements.

- Depending on the scale of your workload, evaluate the use of multiple clusters. Amazon MSK uses a rolling broker update process for many of its operations, including version updates and updating broker types. Operations are carried out one broker at a time and can vary in cycle time. By using multiple clusters, you can parallelize these operations, reducing overall time to completion per cluster. In addition, you can reduce the impact of a cluster-level issue such as an improper configuration by segmenting the workload into multiple clusters.

Security

Amazon MSK has a range of tools and methods to help secure your data processing in the AWS Cloud.

Network Security

By default, clients can privately connect to an Amazon MSK cluster within their Amazon VPC, with traffic traversing only within the AWS network.

- Enable Transport Layer Security (TLS) encryption between brokers and clients.
- Use Security Group rules to control the protocols and port numbers allowed on your brokers.
- Use Apache Kafka APIs to configure your cluster properties and restrict Apache ZooKeeper access with dedicated security groups.

Authentication and Authorization

Amazon MSK provides various methods for access management. AWS recommends using AWS Identity and Access Management (IAM) for managing client authentication and authorization, based on security, ease of use, and low cost. For more information see, IAM Access Control in the *Amazon MSK Developer Guide*.

Alternatively, you can use mutual Transport Layer Security (mTLS) with ACM Private CA or SASL/SCRAM authentication with Apache Kafka Access Control List (ACL) authorization.

Logging and Auditing

Amazon MSK makes it easy for you to troubleshoot clients and analyze communication by streaming Apache Kafka broker logs to Amazon CloudWatch, Amazon Simple Storage Service (Amazon S3), or Amazon Data Firehose. AWS recommends enabling logging. Amazon MSK service API actions are logged in AWS CloudTrail where you can view and search for recent events.

Encryption

- Encrypt data-at-rest using Customer-managed CMK.
- Amazon MSK recommends enabling encryption-in-transit.

Cost

With Amazon MSK, there are no minimum fees or upfront commitments. You do not pay for Apache ZooKeeper nodes that Amazon MSK provisions for you, or for data transfer that occurs between brokers and nodes within clusters.

You pay for the time your broker instances run and the storage you use monthly. Additionally, outside of Amazon MSK costs, there could be data transfer charges applied.

> **Note**
>
> Amazon MSK now supports Apache Kafka version 2.4.1 for new clusters so that consumers can fetch from the closest replica, which reduces latency and potential cross-AZ data transfer.

- Use the Amazon MSK sizing and pricing spreadsheet to correctly size the cluster.
- Configure the message retention period or log size to minimize storage costs.
- Track Amazon MSK cluster costs with custom tagging.
- Use T-family broker instance types for low-cost development, for testing small-to-medium streaming workloads, or low-throughput streaming workloads that experience temporary spikes in throughput. M5 brokers have higher baseline throughput performance than T3 brokers and are recommended for production workloads.

Performance

Amazon MSK is designed to provide performance that is identical to an Apache Kafka cluster running on Amazon Elastic Compute Cloud (Amazon EC2).

- Use Amazon MSK sizing and pricing spreadsheet to right-size your cluster for performance.

- Test your workloads after provisioning the cluster.
- Use M5 broker instance types for production workloads.
- Follow the Number of partitions per broker guide to get the expected performance. For guidance on partition counts, see the Kafka documentation. A high number of partitions will lead to higher CPU usage and can lead to a delay in metrics availability or cluster downtime. Decrease the number of partitions by deleting unused topics or migrating to a larger instance type for your brokers.
- Set minimum in-sync replicas (minISR) to at most RF - 1. A minISR that is equal to the RF can prevent producing to the cluster during a rolling update. With a minISR of 2, three-way replicated topics are available when one replica is offline.

Capacity

When building out capacity for an Amazon MSK cluster, consider the following two key factors.

Instance type for the broker

When creating an Amazon MSK cluster, you can specify the instance type of the brokers at cluster launch. You can start with a few brokers within an Amazon MSK cluster. Then, using the AWS Management Console or the AWS Command Line Interface (AWS CLI), you can scale up to 90 brokers per account and 30 brokers per cluster. These are soft limits, and can be adjusted by requesting a quota increase. Alternatively, you can scale your clusters by changing the size or family of your Apache Kafka brokers, which is a best practice because you don't need to reassign Apache Kafka partitions in order to scale up or down.

The M5 brokers have higher baseline throughput performance and support more partitions per broker than the T3 brokers, and are therefore recommended for production workloads. For more information about M5 instance types, see Amazon EC2 M5 Instances.

T3 brokers have the ability to use CPU credits to temporarily burst performance. Use T3 brokers for low-cost development if you are testing small to medium streaming workloads, or if you have low-throughput streaming workloads that experience temporary spikes in throughput. For more information about T3 instance types, see Amazon EC2 T3 Instances.

The broker instance type will affect the maximum number of partitions that each broker can support. This recommendation provides the recommended maximum number of partitions (including replica partitions) per broker. Perform testing to determine the right instance type for your brokers.

Number of brokers per cluster

Use the Amazon MSK Sizing and Pricing spreadsheet to determine the correct number of brokers for your Amazon MSK cluster. This spreadsheet provides an estimate for sizing an Amazon MSK cluster and the associated costs of Amazon MSK compared to a similar, self-managed, EC2-based Apache Kafka cluster.

Input parameters in the spreadsheet

Basic parameters:

- Ingestion rate (MBs): Number of messages/second divided by average message size going into the cluster.
- Replication factor: replication factor setting for Apache Kafka.
- Total data out (MBs): similar to ingestion rate, but for messages leaving the cluster.
- Retention: how long is data retained in cluster in hours. It is recommended to choose an average retention period across topics for the sizing exercise.
- Encryption in-transit: TLS encryption in-transit activated. TLS encryption consumes additional CPU.
- EBS disk utilization: Desired percentage usage of EBS volumes. It is good to have some headroom to scale. You can also leverage the auto expansion feature to set target disk utilization and maximum scaling capacity.

Advanced parameters:

- Number of AZs: AWS recommends three AZs.
- Nearest replica fetching (Apache Kafka 2.4.x): reduce latency and cross-AZ data transfer.
- Lagging consumer percentage.
- Broker Safety Factor: Number of brokers that can go offline without affecting performance.
- RAM data cached, in seconds.

Based on these inputs, the following outputs are returned:

- Storage load (required, provisioned storage and Total IOPS)
- Networking (Read/Write throughput, MB/s)

- Data Transfer (Cross-AZ and inter-broker replication traffic)
- Memory (RAM, GB)

Based on these outputs and the instance type selection in the summary section of the Sizing and Pricing spreadsheet, depending on production/test instances and number of partitions, the spreadsheet will provide the recommended number of brokers. The Sizing and Pricing spreadsheet is the result of running a test workload with three consumers, and ensuring that P99 write latencies are below 100 ms. This may not necessarily reflect your workload or performance expectations. Use this spreadsheet as a starting point and run performance tests for your workloads after provisioning the cluster.

It is possible to change the number of brokers in the Amazon MSK cluster after creation. After resizing, the cluster will need to be rebalanced by reassigning partitions to the new brokers.

Monitoring

There are two monitoring options: Amazon CloudWatch and Open Monitoring for Prometheus. If you already ingest metrics from Apache Kafka into a metrics solution such as DataDog or New Relic, it is recommended to use Open Monitoring with Prometheus to maintain compatibility throughout your migration, and supplementing these metrics by those offered in Amazon CloudWatch. The Open Monitoring with Prometheus option also provides an integration with Cruise Control to manage Apache Kafka partitions.

With Amazon CloudWatch, you can set the monitoring level for an Amazon MSK cluster to one of the following:

- `DEFAULT`
- `PER_BROKER`
- `PER_TOPIC_PER_BROKER`
- `PER_TOPIC_PER_PARTITION`

> **Note**
>
> The `DEFAULT` level is included with the Amazon MSK cluster whereas there is a cost associated with the other monitoring levels. The `PER_TOPIC_PER_PARTITION` level is only relevant for consumer lag.

Consumer-lag metrics quantify the difference between the latest data written to your topics and the data read by your applications. Amazon MSK provides the following consumer-lag metrics, which you can get through Amazon CloudWatch or through Open Monitoring with Prometheus:

- `OffsetLag`: Partition-level consumer lag in number of offsets.
- `MaxOffsetLag`: The maximum offset lag across all partitions in a topic.
- `EstimatedMaxTimeLag`: Time estimate (in seconds) to drain `MaxOffsetLag`.
- `EstimatedTimeLag`: Time estimate (in seconds) to drain the partition offset lag.
- `SumOffsetLag`: The aggregated offset lag for all the partitions in a topic.

Recommended metrics to monitor with Amazon CloudWatch

- **`KafkaDataLogsDiskUsed`:** Cluster should not reach or exceed 85% of disk space usage. Higher disk space usage can impact receiving and transmitting. Configure an Amazon CloudWatch alarm that monitors `KafkaDataLogsDiskUsed` metrics and alerts you if disk space used is too high.

  To mitigate storage issues, you can do one of the following:

  - Seamlessly scale up the amount of storage provisioned per broker
  - Create an auto scaling policy to automatically expand your storage
  - Change the retention period of messages and logs.
  - Delete unused topics.
- **`UnderMinIsrPartitionCount`:** The In-Sync Replication (ISR) count indicates the set of replicas up-to-date with the leader. The expected value for `UnderMinIsrPartitionCount` is zero.
- **`ActiveControllerCount`:** Monitor ActiveControllerCount and set an alarm to alert you in the event `ActiveControllerCount` (Maximum) statistics is greater than 1.
- **`OfflinePartitionsCount`:** This metric reports the number of partitions that don't have an active leader and are hence not writable or readable. You should configure an alert on a value greater than 0.
- **`UnderReplicatedPartitions`:** The number of in-sync replicas (ISRs) should be equal to the total number of replicas. As a general best practice, you should set an alert using a value greater than 0.
- **`UnderMinIsrPartitionCount`:** This metric should not be greater than zero and you should set an alert on a non-zero value.

- **CPU:** Amazon MSK strongly recommends that you maintain the total CPU utilization for your brokers under 60%.
- **`RequestTime`:** The average time in milliseconds spent in broker network and I/O threads to process requests. Indicates the percentage of time spent in broker network and I/O threads to process requests from client group.
- **`BytesInPerSec/BytesOutPerSec`:** The number of bytes per second received from clients and sent to clients.

For more information, see Amazon MSK Metrics for Monitoring with CloudWatch.

Testing

To minimize risks when migrating from a self-managed Apache Kafka cluster to Amazon MSK, AWS recommends migrating between identical Apache Kafka versions and then updating the version on Amazon MSK. It is also recommended to measure sustained performance, not peak performance, by running tests for an extended period of time. Smaller brokers in the M5 family can achieve higher throughput and lower latencies by leveraging Amazon Elastic Block Store (Amazon EBS) optimized networks and Amazon EC2 network burst credits.

When running performance testing on Apache Kafka, consider the following:

- Producer performance (throughput, latency)
- Consumer performance (throughput, latency)

Monitor the following metrics:

- Throughput (messages/sec) for size of data.
- Throughput (messages/sec) for number of messages.
- Total data.
- Total messages.

To test performance, download Apache Kafka into a test environment. The package contains shell scripts including `kafka-producer-perf-test.sh` and `kafka-consumer-perf-test.sh`. These scripts can be used to test the performance of your producers and consumers.

Migrating to Amazon MSK

Apache Kafka includes command-line tools and scripts to manage and administer an Apache Kafka cluster. The same Apache Kafka tools can be used to manage Amazon MSK, interact with brokers, and perform administrative tasks.

Included in the toolset are mirroring options that make it possible to maintain an image of an existing Apache Kafka cluster. Prior to the 2.4.0 release of Apache Kafka, there was a version of MirrorMaker (MM) tool that was bundled and released as part of Apache Kafka command-line tools. In the 2.4.0 release of Apache Kafka, a new version of MirrorMaker (MM2) was released.

The following sections discuss the migration path for each of the following options:

- MirrorMaker 2.0 (MM2)
- Apache Flink
- AWS Lambda

MirrorMaker 2.0 (MM2)

> **Note**
>
> This whitepaper is only meant to be used as a reference for deploying MirrorMaker 2.0 (MM2) on Kafka Connect to migrate data between multiple Amazon MSK clusters. AWS does not offer any support for it.

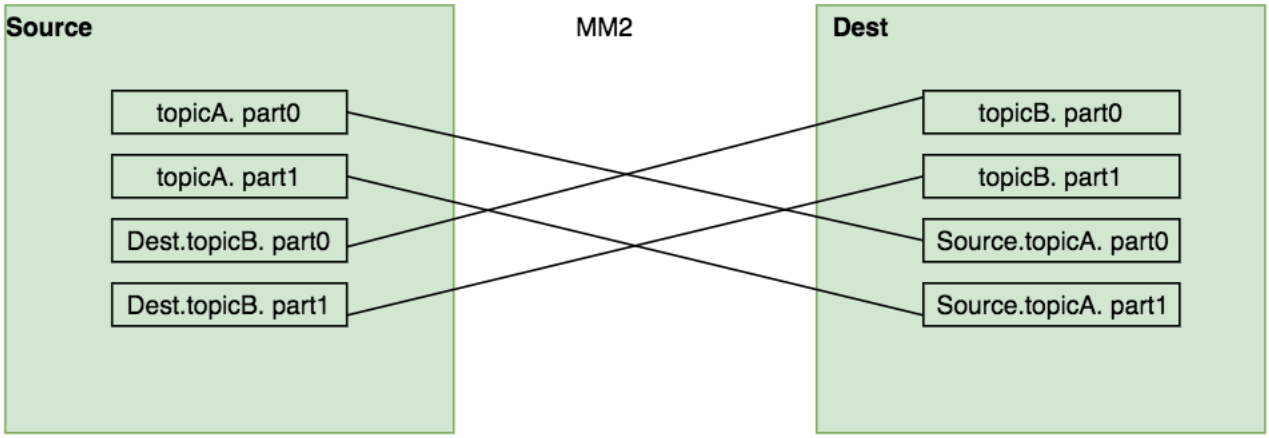
MirrorMaker 2.0 (MM2) is a multi-cluster data replication engine based on the Kafka Connect framework. MM2 is a combination of an Apache Kafka source connector and a sink connector. You can use a single MM2 cluster to migrate data between multiple clusters. MM2 automatically detects new topics and partitions, while also ensuring the topic configurations are synced between clusters.

MM2 is available as part of the Apache Kafka 2.4.0 release and supersedes all capabilities of MM1. AWS recommends using MM2.

MM2 remote topics

Replication policies help consumers perform migrations between self-managed Apache Kafka clusters and Amazon MSK clusters with no downtime. MM2 comes with a default replication policy that can be customized by providing a custom replication policy class.

Remote topics in MM2 are replicated topics on the target cluster. These topics reference the source cluster using a naming convention as shown in the following figure.



Remote topics

For the default replication policy, *topicA* is the source topic. *Source* is a source cluster alias used in the configuration properties file and the remote topic name is *source.topicA*. This distinguishes between a replicated remote topic and a topic created on the destination or target cluster.

MM2 ensures ordering in partitions when it replicates the topic from source cluster to the remote topic in the destination cluster. The following diagram shows an example of remote topics created on source and destination cluster when bi-directional replication is configured in MM2.

MM2 internal topics

MM2 uses the following internal topics for replication purposes:

Heartbeat topic: Emitted from the source cluster and replicated to demonstrate connectivity through connectors. This can be used by downstream consumers to verify that the connector is running and the corresponding source cluster is available. Messages in this topic contain information on the source cluster, target cluster, and timestamp when the heartbeat was created.

Checkpoint topic: Emitted in the target cluster by the connector and contains consumer offsets for each consumer group in the source cluster. The connector will periodically query the source cluster for all committed offsets of consumer groups (except for replicated topics) and emit a message to this topic. Information in this message includes the consumer group id, topic, partition, upstream offset, downstream offset, metadata, and timestamp. Consumers use the checkpoint topic via *MirrorClient* or *RemoteClusterUtils* class to get the replicated offsets in the target Apache Kafka cluster.

Offset sync topics: Encodes cluster-to-cluster offset mapping for each replicated topic-partition. Messages in this topic contain topic, partition, upstream offset, and downstream offset.

MM2 Connectors

MM2 has the following connectors for enabling complex flows between multiple Apache Kafka clusters and across data centers via existing Kafka Connect clusters.

- **`MirrorSourceConnector`** replicates a set of topics from a single source cluster into the primary cluster.
- **`MirrorCheckpointConnector`** emits consumer offset checkpoints and syncs the offset with `__consumer_offsets` (as of Apache Kafka 2.7.0).
- **`MirrorHeartbeatConnector`** emits heartbeats.

MM2 Deployment Methods

MirrorMaker 2.0 can be deployed in several modes:

- As a dedicated MirrorMaker 2.0 cluster
- As connectors in a distributed Kafka Connect cluster
- In legacy mode

Deploying a dedicated MM2 cluster

The *connect-mirror-maker.sh* script sets up the `MirrorSourceConnector`, `MirrorCheckpointConnector`, and `MirrorHeartbeatConnector` connectors based on the provided MM2 properties file. See the following example file.

```
#Apache Kafka clusters
```

```
clusters = #comma separated list of Apache Kafka cluster aliases
source.bootstrap.servers = apache-kafka-source:9092
target.bootstrap.servers = apache-kafka-target:9092

source -> target.enabled = true

#Source and target cluster configuration
source.config.storage.replication.factor = 1
target.config.storage.replication.factor = 1
source.offset.storage.replication.factor = 1
target.offset.storage.replication.factor = 1

#Mirror Maker configuration
offset-sync.topic.replication.factor = 1
heartbeat.topic.replication.factor = 1
checkpoint.topic.replication.factor = 1

topics = .*
groups = .*

tasks.max = 1
replication.factor = 1
refresh.topics.enabled = true
sync.topic.configs.enabled = true

#Enable heartbeats and checkpoints
source->target.emit.heartbeats.enabled = true
source->target.emit.checkpoints.enabled = true
```

Ensure MM2 has successfully connected to the source and target clusters using the `kafka-topics.sh` script to list and compare the topics created on both clusters. See the following example.

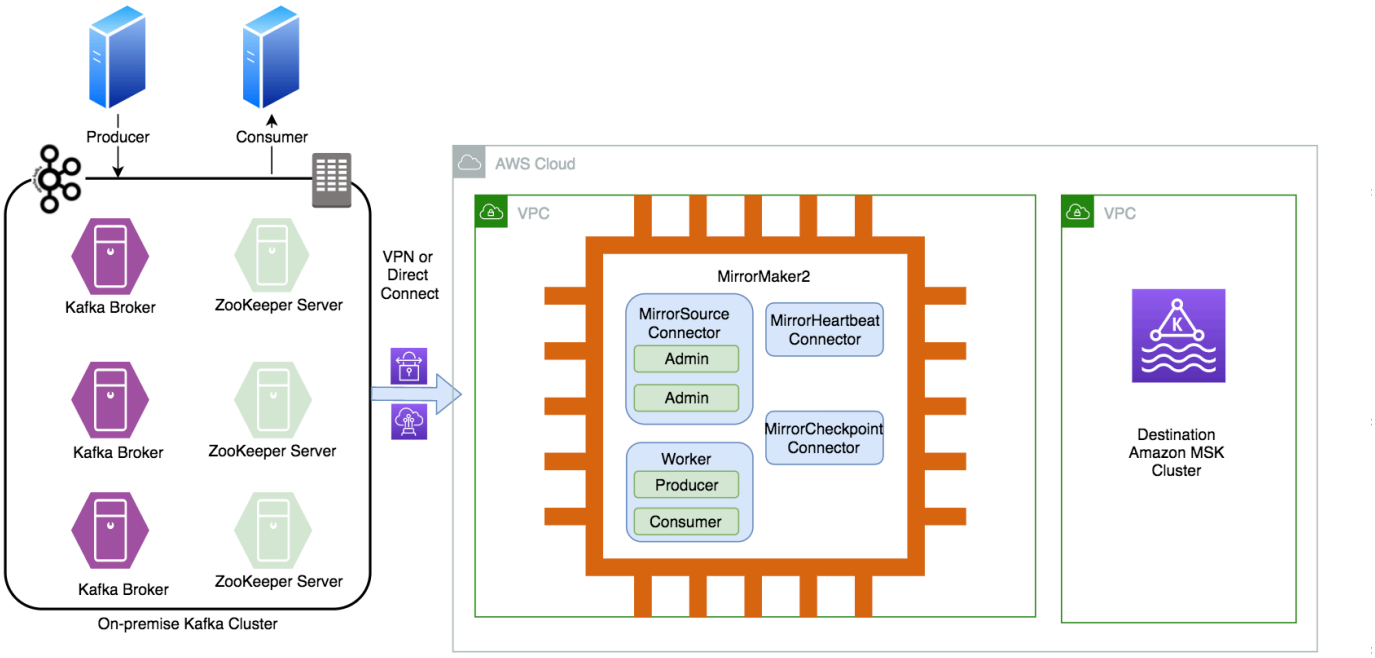
```
#Source Topics
./kafka-topics.sh --bootstrap-server <source bootstrap string> --list
__consumer_offsets
heartbeats
mm2-configs.target.internal
mm2-offset-syncs.target.internal
mm2-offsets.target.internal
mm2-status.target.internal

#Target Topics
```

```
./kafka-topics.sh --bootstrap-server <target bootstrap string> --list
__consumer_offsets
heartbeats
mm2-configs.source.internal
mm2-offsets.source.internal
mm2-status.source.internal
source.checkpoints.internal
source.heartbeats
```

Deploying MM2 on a Kafka Connect cluster

Customers that already have an existing Kafka Connect cluster running on EC2 instances or containers can run MM2 on the same cluster by starting the `MirrorSourceConnector`, `MirrorCheckpointConnector`, and `MirrorHeartbeatConnector` connectors.

The AWS provided self-service Amazon MSK workshop details all of the necessary steps required to set up MM2 on a Kafka Connect cluster. See the following figure for an overview.



MM2 on Kafka Connect

In legacy mode

After legacy MirrorMaker is deprecated, the existing `./bin/kafka-mirror-maker.sh` scripts will be updated to run MM2 in legacy mode:

```
$ ./bin/kafka-mirror-maker.sh --consumer consumer.properties
--producer producer.properties
```

Replication policy and filters for Apache Kafka client migration

The consumer offset data is migrated from source to target using MM2. With this approach, consumers can migrate the Amazon MSK cluster from source to target with little to no downtime. There are five possible migration approaches:

- Automated consumer offsets sync
- Custom replication policy with background process to sync offsets
- Default replication policy with background process to sync offsets
- Default replication policy with consumer using MM2 checkpointed offsets
- Custom replication policy with consumer using MM2 checkpointed offsets

Automated consumer offsets sync

AWS highly recommends this configuration. Starting with Apache Kafka 2.7, the automated consumer offsets sync can be enabled using a new configuration property in MM2, called `sync.group.offsets.enabled`, along with the `emit.checkpoints.enabled` property. Setting both properties to true will ensure the existing `MirrorCheckpointTask` will also sync the selected and translated consumer group offsets to the target cluster. The frequency of offset sync is equal to that of emitting checkpoints. This can be used for both custom and default policies.

With the automated consumer offsets sync configuration, consumers can use the translated offsets to resume the consumption from where they left off at the source cluster, without losing messages or consuming many duplicate messages. When enabled, MM2 translates the consumer offsets recognizable to a target cluster. This can be combined by periodically synchronizing the translated offsets to the `__consumer_offsets` topic. This provides a seamless one-time migration of consumers from one to another cluster, and also facilitates the failover of consumers from a source

to a target cluster to pick up from the last known offsets. A difference between the automated consumer offsets sync and checkpointed offsets configuration is that in the latter, the consumer is modified to fetch offsets based on the checkpoint topic in the target cluster and uses the `RemoteClusterUtils` class to translate the offset.

Automated consumer offsets sync is not enabled by default. Add the `source->target.sync.group.offsets.enabled = true` property to the MM2 config to enable a one-way sync from a *source* to a *target* cluster.

Custom replication policy with background process to sync offsets

By default, topics created in the target cluster have a prefix provided by the configuration file. This is to avoid cyclical topic references. Since the topics are different in the target cluster, the consumers require changes to start fetching from the new topics.

An alternative to this method is to create a custom replication policy that does not add a prefix to the topic name. This will avoid any changes to the consumer as topic names are the same across source and target. For more information, see the mirrormaker2-msk-migration sample replication policy in GitHub.

MM2 uses an internal checkpoints topic to store offsets for each consumer group in the target cluster. It uses an offset-sync topic to store cluster-to-cluster offset mappings for each topic-partition being replicated. Although offsets may not be the same between clusters, MM2 tracks the offset for consumer groups for all consumer groups in the source cluster as specified in the configuration. The checkpoints are done periodically (the interval is configurable); however, the checkpointed source offsets may lag behind the last consumed offsets by the consumer at the source.

From Apache Kafka 2.4.0 onwards, Apache Kafka offers the `RemoteClusterUtils` class that translates source offsets to offsets on the target. In this migration approach, the topic is created when MM2 starts replicating and syncs the topic. During migration, the consumer pointing to the target can start and consume data from where it left off without any changes.

For Apache Kafka Connect versions 2.7.0 and later, the automated consumer offsets sync feature is available in MM2. For earlier versions, see the mirrormaker2-msk-migration migration sample in GitHub to asynchronously perform the same function. This background sync process periodically syncs the MM2 checkpointed offsets in the checkpoint topic on the target cluster to the `__consumer_offsets` internal topic. With this feature consumers can pick up from where they left off after failover.

Default replication policy with background process to sync offsets

The MM2 default replication policy creates target topics with a prefix. This requires a change in the consumer to consume from new topics in the target Amazon MSK cluster as they are migrated.

This method uses an asynchronous background sync process to sync the MM2 checkpointed offsets in the checkpoint topic on the target cluster to the `__consumer_offsets` internal topic. With this feature consumers can pick up from where they left off after failover.

The procedure to start Apache Kafka connectors using a default replication policy, with a background process to sync offsets, is available in the AWS self-service Amazon MSK workshop.

Default replication policy with consumer using MM2 checkpointed offsets

In this approach, the consumer is modified to fetch offsets based on the checkpoint topic in the target cluster and uses the `RemoteClusterUtils` class to translate the offset in the target cluster. The procedure to start Apache Kafka connectors to use a default replication policy is available in the AWS self-service Amazon MSK workshop.

Custom replication policy with consumer using MM2 checkpointed offsets

In this approach, MM2 uses a custom replication policy class that can be provided to modify the replication behavior. Just like the preceding approach, the consumer is modified to fetch offsets based on the checkpoint topic in the target cluster and uses the `RemoteClusterUtils` class to translate the offset in the target or target cluster.

Apache Flink

MM2 supports at-least-once semantics, where multiple attempts are made at delivering a message so that at least one succeeds. Records can be duplicated to the target and the consumers are expected to be idempotent in handling duplicate records. In scenarios where exactly-once semantics are required, customers can use Apache Flink. However, they will lose the ability to sync offset translation and metadata migration. Apache Flink can also be used for scenarios where data requires mapping or transformation actions before submission to the target cluster.

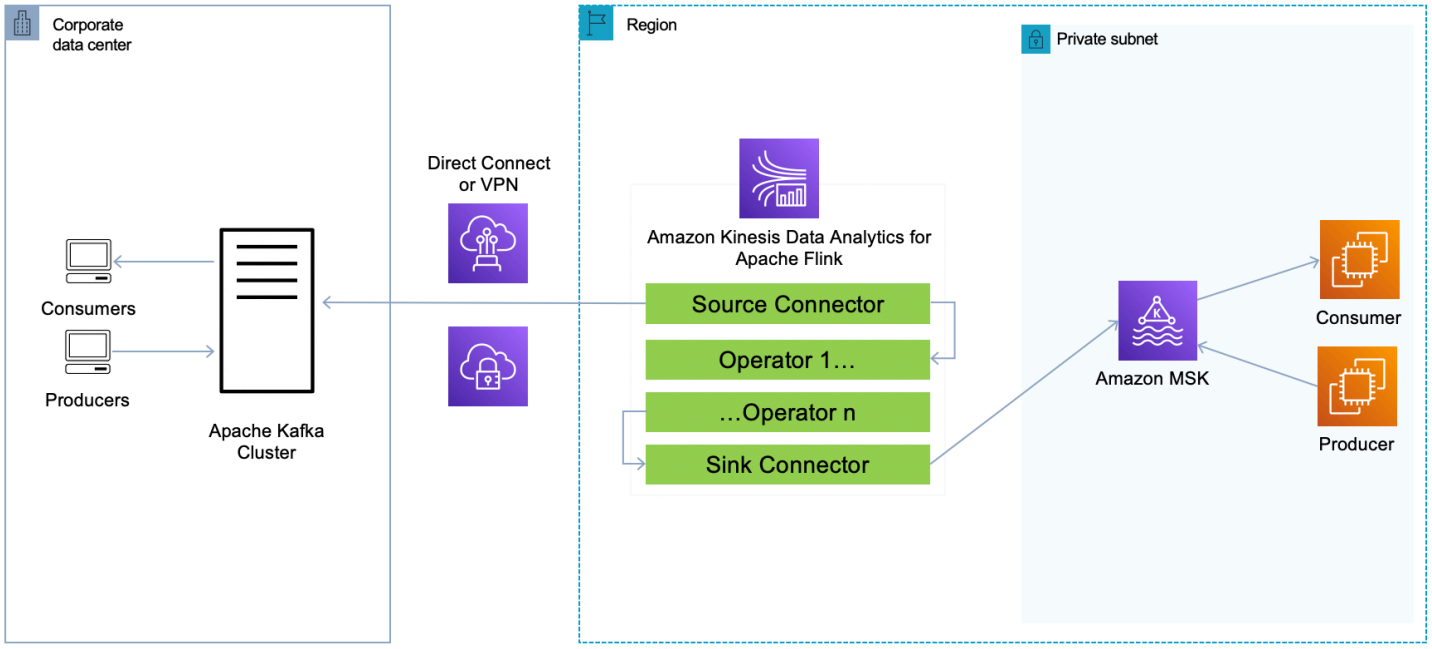
Apache Flink provides connectors for Apache Kafka with sources and sinks that can read data from one Apache Kafka cluster and write to another. Apache Flink can run on AWS by launching an

Amazon EMR cluster or by running Apache Flink as an application using Amazon Managed Service for Apache Flink.

Amazon Managed Service for Apache Flink is a fully managed, serverless service that provides the underlying infrastructure for your Apache Flink applications. It handles core capabilities like provisioning compute resources, parallel computation, automatic scaling, and application backups (implemented as checkpoints and snapshots). You can use the high-level Apache Flink programming features (such as operators, sources, and sinks) the same way you would on a self-hosted Apache Flink deployment or with Apache Flink running on an Amazon EMR cluster.

By using Amazon Managed Service for Apache Flink to replicate, you can focus on the data and business implications of the migration, instead of focusing on managing the infrastructure and operations required to do the migration. You can also schedule parallel execution of tasks in Apache Flink to implement scaling. Amazon Managed Service for Apache Flink has the ability to scale automatically to accommodate the data throughput of your source Apache Kafka cluster, enabling you to achieve the throughput requirements of your migration.

The following figure shows the high-level architecture of using Amazon Managed Service for Apache Flink to replicate an on-premises Apache Kafka cluster to Amazon MSK, using source and sink connectors.



Migrating to Amazon MSK with Amazon Managed Service for Apache Flink

AWS Lambda

With support for Apache Kafka as an event source for AWS Lambda, you can now consume messages from a topic via an AWS Lambda function. AWS Lambda internally polls for new records or messages from the event source, and then synchronously invokes the target AWS Lambda function to consume these messages. AWS Lambda reads the messages in batches and provides the message batches to your function in the event payload for processing. Consumed messages can then be transformed and/or written directly to your target Amazon MSK cluster.

AWS Lambda is a consumer application for your Apache Kafka topic. It processes records from one or more partitions and sends the payload to the target function. You can use a self-hosted Apache Kafka cluster as an event source for AWS Lambda or an Amazon MSK cluster as an event source for AWS Lambda to consume data from and send to your target cluster, all within the same function. AWS Lambda offers a serverless option for the migration if your scenario does not factor migrating topic state and requires you to move data from one cluster to another, although it can cause duplicates.

Migrating Apache Kafka State with Apache Flink and AWS Lambda

Using Amazon Managed Service for Apache Flink for serverless Apache Flink or AWS Lambda as an Apache Kafka migration solution provides benefits such as:

- Reduced operational overheads with managed and serverless infrastructures
- Ability to transform data between source and target

These methods however, do not support the migration of topic state (for example consumer offsets or topic configuration), but you can work around this using the following steps:

1. Stop the Apache Kafka producer in the source cluster.
2. Stop accepting new messages on the source cluster.
3. Ensure the consumers pointing to the source cluster have fully consumed the data available in the source Apache Kafka cluster topics. Use the Apache Kafka Consumer Group offset tool to check and ensure the consumers are caught up and there is no lag.
4. Terminate the consumers connected to the source cluster.

5. Start the consumers on the target cluster. Ensure the consumers are configured to retrieve messages from the latest point.
6. Start the producers on the target cluster.
7. Confirm the consumers are reading new messages pushed using the Apache Kafka Consumer Group offset tool.

Amazon MSK Migration Program

The Amazon MSK Migration Program packages best practices, expertise, hands-on technical assets, and migration incentives to make it easy for businesses to migrate Apache Kafka workloads currently running on-premises, on Amazon EC2, or on another managed service provider to Amazon MSK.

The program provides the following:

- **A two-day deep dive workshop**: A no-cost engagement, where customers can get familiar with Amazon MSK, learn how it aligns with their architecture, and receive hands-on migration guidance from AWS Specialist Solution Architects.
- **Ready to use technical assets:** Hands-on access to migration labs and best practices documentation.
- **Migration incentives:** Qualified use cases receive Proof-of-Concept credits to start their migration.
- **ProServe engagement:** Jump start migrations with experts from AWS Professional Services (ProServe).

For additional information on this program, reach out to your account team or email <msk-bd@amazon.com>.

Conclusion

Similar to many distributed systems, migrating to Amazon MSK involves planning, assessing various components and evaluating architectural choices. This guide includes the benefits of migrating to Amazon MSK, details the available migration strategies and provides best practices that AWS has compiled to help customers with their migrations.

Contributors

Contributors to this document include:

- Rajeev Chakrabarti, Sr Manager, Solutions Architecture
- Imtiaz (Taz) Sayed, Analytics Tech Leader
- Francis McGregor-Macdonald, Principal Solutions Architect
- Saurabh Shrivastava, Manager, Partner SA, GSI
- Kalyan Janaki, Senior Technical Account Manager
- Jared Warren, Senior Solutions Architect
- Jeremiah O'Connor, Senior Solutions Architect
- Masudur Rahaman Sayem, Analytics Specialist Solutions Architect
- Priyanka Chaudhary, Data Architect
- Gautam Srinivasan, Senior Solutions Architect
- Ajit Singh, Data Streaming Specialist

Further reading

For additional information, see:

- AWS Architecture Center
- Getting started Using Amazon MSK
- Amazon MSK FAQs
- Learn Amazon MSK
- Amazon MSK Migration Lab
- Best practices from Delivery on migrating from Apache Kafka to Amazon MSK

Document history

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Minor update	Added note that the MirrorMaker 2.0 deployment information is for reference only.	February 10, 2023
Minor update	Fix non-inclusive language.	April 6, 2022
Initial publication	Whitepaper first published.	September 30, 2021

Note

To subscribe to RSS updates, you must have an RSS plug-in enabled for the browser that you are using.

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 current AWS 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. The responsibilities and liabilities of AWS 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.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.