



AWS MSK

Verified 2026-10-24

Managed Kafka — cluster types, CLI, ports, IAM and SCRAM auth, client setup, storage, configuration, monitoring, and scaling.

Contents

Mental model	2
Cluster types	2
Create and inspect with the CLI	3
Connectivity and ports	6
Authentication and authorization	7
Client setup and Kafka CLI	9
Storage	11
Configuration	12
Monitoring	13
Scaling and operations	15
Further reading	16

Mental model

MSK runs real Apache Kafka brokers for you, so every Kafka client, tool, and config keeps working. AWS manages the broker hosts, patching, replacement, and (with ZooKeeper or KRaft) the controllers; you still own topics, partitions, replication factors, and consumer groups. The brokers live inside your VPC, so networking and authentication decide whether a client can connect at all.

Cluster types

Type	You choose	Best for
Provisioned, Standard	Broker size, count, storage	Full control of config and storage
Provisioned, Express	Broker size and count only	Faster scaling, no storage sizing

Type	You choose	Best for
Serverless	Nothing; pay per use	Spiky or unknown load

Standard and Express brokers run in 2 or 3 Availability Zones, with brokers spread evenly. Express brokers manage storage for you and expose fewer configs. Serverless accepts IAM authentication only and sets partition and throughput limits for you.

Gotcha: The broker count must be a multiple of the AZ count. A 3-AZ cluster takes 3, 6, 9 brokers, never 4.

Create and inspect with the CLI

Command	Does
<code>aws kafka create-cluster-v2</code>	Create any cluster type
<code>aws kafka list-clusters-v2</code>	List clusters and ARNs

Command	Does
<code>aws kafka describe-cluster-v2</code>	State, version, current version token
<code>aws kafka get-bootstrap-brokers</code>	Broker endpoints per auth type
<code>aws kafka list-nodes</code>	Per-broker details
<code>aws kafka list-kafka-versions</code>	Versions you may create

Most commands take `--cluster-arn`. Creation takes tens of minutes; poll `describe-cluster-v2` until State is `ACTIVE`. Every update call needs the cluster's `CurrentVersion` token, so read it first.

```
aws kafka create-cluster-v2 \
  --cluster-name orders \
  --provisioned file://cluster.json
aws kafka get-bootstrap-brokers \
  --cluster-arn "$ARN" \
  --query BootstrapBrokerStringSaslIam
```

```
{
  "BrokerNodeGroupInfo": {
    "InstanceType": "kafka.m7g.large",
    "ClientSubnets": ["subnet-a", "subnet-b",
      "subnet-c"],
    "SecurityGroups": ["sg-0123"],
    "StorageInfo": {"EbsStorageInfo":
      {"VolumeSize": 1000}}
  },
  "KafkaVersion": "3.9.x",
  "NumberOfBrokerNodes": 3
}
```

Note: Run `list-kafka-versions` and pick a supported version. Old versions are retired on a published schedule.

Connectivity and ports

Port	Protocol	Auth
9092	PLAINTEXT	None
9094	TLS	None or mTLS
9096	SASL_SSL	SASL/SCRAM
9098	SASL_SSL	IAM
9198	SASL_SSL	IAM, public access

Clients must reach the brokers' private IPs: run them in the same VPC, a peered VPC, or over Transit Gateway, VPN, or Direct Connect. The security group must allow the port from the client. Public ports add 100 to the private ones (9194 TLS, 9196 SCRAM, 9198 IAM). For another VPC or account without peering, enable multi-VPC private connectivity.

```
aws kafka update-connectivity \  
  --cluster-arn "$ARN" \  
  --current-version "$VER" \  
  --connectivity-info \  

```

```
'{"PublicAccess":  
  {"Type": "SERVICE_PROVIDED_EIPS"}}'
```

Gotcha: Public access needs public subnets, an authenticated cluster (unauthenticated access off), TLS in transit, and `allow.everyone.if.no.acl.found=false`. Without them the update is rejected.

Authentication and authorization

Method	Credentials	Authorization
IAM access control	IAM role or user	IAM policy
SASL/SCRAM	Secrets Manager secret	Kafka ACLs
mTLS	ACM Private CA cert	Kafka ACLs
Unauthenticated	None	None

IAM is the simplest choice for AWS-hosted clients: no secrets to rotate, and policies cover Kafka actions on cluster, topic, and group ARNs. SCRAM secrets must be named `AmazonMSK_*` and encrypted with a customer-managed KMS key, then associated with `batch-associate-scam-secret`.

```
{
  "Effect": "Allow",
  "Action": [
    "kafka-cluster:Connect",
    "kafka-cluster:DescribeTopic",
    "kafka-cluster:ReadData",
    "kafka-cluster:WriteData"
  ],
  "Resource": [
    "arn:aws:kafka:REGION:ACCT:cluster/orders/*",
    "arn:aws:kafka:REGION:ACCT:topic/orders/*"
  ]
}
```

Gotcha: A consumer also needs

`kafka-cluster:AlterGroup` and `DescribeGroup` on the group ARN (`.../group/orders/*`). Without them the consumer fails with an authorization error on join.

Client setup and Kafka CLI

Setting	Value
<code>security.protocol</code>	<code>SASL_SSL</code>
<code>ssl.mechanism</code>	<code>AWS_MSK_IAM</code>
<code>ssl.jaas.config</code>	<code>IAMLoginModule required;</code>
Callback handler	<code>IAMClientCallbackHandler</code>

Put the `aws-msk-iam-auth` JAR on the classpath of Java clients and Kafka CLI tools. Pass the same properties file to every tool with `--command-config`. Use two or three brokers from the bootstrap string; the client discovers the rest.

```
security.protocol=SASL_SSL
sasl.mechanism=AWS_MSK_IAM
sasl.jaas.config=\
    software.amazon.msk.auth.iam.IAMLoginModule \
    required;
sasl.client.callback.handler.class=\
    software.amazon.msk.auth.iam.\
    IAMClientCallbackHandler
```

```
kafka-topics.sh --bootstrap-server "$BS" \
    --command-config client.properties \
    --create --topic orders \
    --partitions 6 --replication-factor 3
```

Tip: Java properties files accept a trailing backslash to continue a value on the next line, as above.

Storage

Feature	Detail
EBS per broker	1 GiB to 16 TiB
Scaling	Up only, never down
Provisioned throughput	On m5.4xlarge and larger
Tiered storage	Offloads old data to cheaper storage

Storage is per broker and holds all replicas on it. You can raise it with `update-broker-storage`, or let Application Auto Scaling add it when disk use crosses a target. Tiered storage keeps recent data local and older data remote, so retention can grow without adding brokers.

```
aws kafka update-broker-storage \
  --cluster-arn "$ARN" \
  --current-version "$VER" \
  --target-broker-ebs-volume-info \
  '[{"KafkaBrokerNodeId":"ALL",
    "VolumeSizeGB":2000}]'
```

Gotcha: A full disk takes the broker down. Alarm on `KafkaDataLogsDiskUsed` at 85%, and remember a storage increase can only be repeated after a cooldown of about six hours.

Configuration

Command	Does
<code>create-configuration</code>	Upload a <code>server.properties</code> file
<code>update-cluster-configuration</code>	Apply a revision to a cluster
<code>describe-configuration</code>	Show revisions
<code>kafka-configs.sh</code>	Change topic-level configs live

Cluster configs are versioned revisions. Applying one triggers a rolling restart of the brokers. Topic-level settings such as `retention.ms` are changed with Kafka tools and need no restart.

```
auto.create.topics.enable=false
default.replication.factor=3
min.insync.replicas=2
log.retention.hours=168
```

```
aws kafka create-configuration \
  --name orders-cfg \
  --server-properties file:///server.properties
```

Gotcha: Topics with replication factor 1 are unavailable while a broker restarts for patching or a config change. Use RF 3 with `min.insync.replicas=2`, and produce with `acks=all`.

Monitoring

Metric (AWS/Kafka)	Watch for
KafkaDataLogsDiskUsed	Above 85%
CpuUser + CpuSystem	Sum above 60%
OfflinePartitionsCount	Anything above 0

Metric (AWS/Kafka)	Watch for
UnderReplicatedPartitions	Above 0 for long
MaxOffsetLag	Consumer falling behind

CloudWatch metrics come at monitoring level `DEFAULT` for free; `PER_BROKER`, `PER_TOPIC_PER_BROKER`, and `PER_TOPIC_PER_PARTITION` add detail at a cost. Consumer lag metrics only appear for groups that are active. Broker logs ship to CloudWatch Logs, S3, or Firehose. Open monitoring exposes Prometheus endpoints on ports 11001 (JMX) and 11002 (node).

```
aws kafka update-monitoring \
  --cluster-arn "$ARN" \
  --current-version "$VER" \
  --enhanced-monitoring PER_TOPIC_PER_BROKER
```

Note: Keep CPU under 60%. If a broker is down for patching, the survivors take its load, and a hotter cluster goes unstable.

Scaling and operations

Task	How
More brokers	<code>update-broker-count</code>
Bigger brokers	<code>update-broker-type</code>
Spread load	Reassign partitions
Upgrade Kafka	<code>update-cluster-kafka-version</code>

Scale out when CPU or network is the limit, and scale up when per-broker memory or throughput is. Rolling operations keep the cluster online if topics are replicated. Back up or mirror data across Regions with MSK Replicator, and move data to and from other systems with MSK Connect.

```
aws kafka update-broker-count \  
  --cluster-arn "$ARN" \  
  --current-version "$VER" \  
  --target-number-of-broker-nodes 6
```

Gotcha: New brokers start empty. Existing partitions do not move, so add brokers and then run `kafka-reassign-partitions.sh` (or Cruise Control) or the old brokers stay hot.

Further reading

- [Amazon MSK Developer Guide](#)
- [MSK best practices](#)
- [IAM access control for MSK](#)
- [AWS CLI kafka commands](#)
- [Apache Kafka documentation](#)