AWS Big Data Blog

Best Practices for Running Apache Kafka on AWS

by Prasad Alle | on 02 MAR 2018 | in Amazon Kinesis, Amazon Kinesis Video Streams, Amazon Simple Storage Services (S3), AWS Quest | Permalink | Comments | Share

This post was written in partnership with Intuit to share learnings, best practices, and recommendations for running an Apache Kafka cluster on AWS. Thanks to Vaishak Suresh and his colleagues at Intuit for their contribution and support.

Intuit, in their own words: *Intuit, a leading enterprise customer for AWS, is a creator of business and financial management solutions. For more information on how Intuit partners with AWS, see our previous blog post, Real-time Stream Processing Using Apache Spark Streaming and Apache Kafka on AWS. Apache Kafka is an open-source, distributed streaming platform that enables you to build real-time streaming applications.*

The best practices described in this post are based on our experience in running and operating large-scale Kafka clusters on AWS for more than two years. Our intent for this post is to help AWS customers who are currently running Kafka on AWS, and also customers who are considering migrating on-premises Kafka deployments to AWS.

AWS offers Amazon Kinesis Data Streams, a Kafka alternative that is fully managed.

Running your Kafka deployment on Amazon EC2 provides a high performance, scalable solution for ingesting streaming data. AWS offers many different instance types and storage option combinations for Kafka deployments. However, given the number of possible deployment topologies, it's not always trivial to select the most appropriate strategy suitable for your use case.

In this blog post, we cover the following aspects of running Kafka clusters on AWS:

- Deployment considerations and patterns
- Storage options
- Instance types
- Networking
- Upgrades
- Performance tuning
- Monitoring
- Security
- Backup and restore

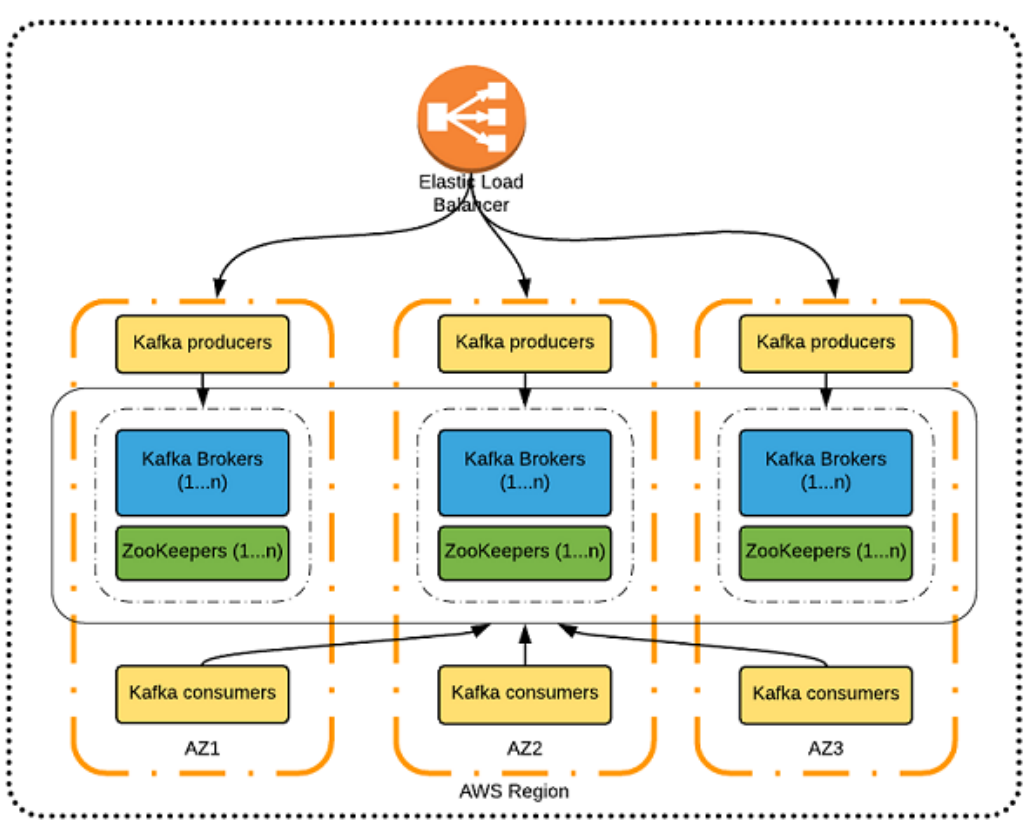
Note: While implementing Kafka clusters in a production environment, make sure also to consider factors like your number of messages, message size, monitoring, failure handling, and any operational issues.

Deployment considerations and patterns

In this section, we discuss various deployment options available for Kafka on AWS, along with pros and cons of each option. A successful deployment starts with thoughtful consideration of these options. Considering availability, consistency, and operational overhead of the deployment helps when choosing the right option.

Single AWS Region, Three Availability Zones, All Active

One typical deployment pattern (all active) is in a single AWS Region with three Availability Zones (AZs). One Kafka cluster is deployed in each AZ along with Apache ZooKeeper and Kafka producer and consumer instances as shown in the illustration following.



In this pattern, this is the Kafka cluster deployment:

- Kafka producers and Kafka cluster are deployed on each AZ.
- Data is distributed evenly across three Kafka clusters by using Elastic Load Balancer.
- Kafka consumers aggregate data from all three Kafka clusters.

Kafka cluster failover occurs this way:

- Mark down all Kafka producers
- Stop consumers
- Debug and restack Kafka
- Restart consumers
- Restart Kafka producers

Following are the pros and cons of this pattern.

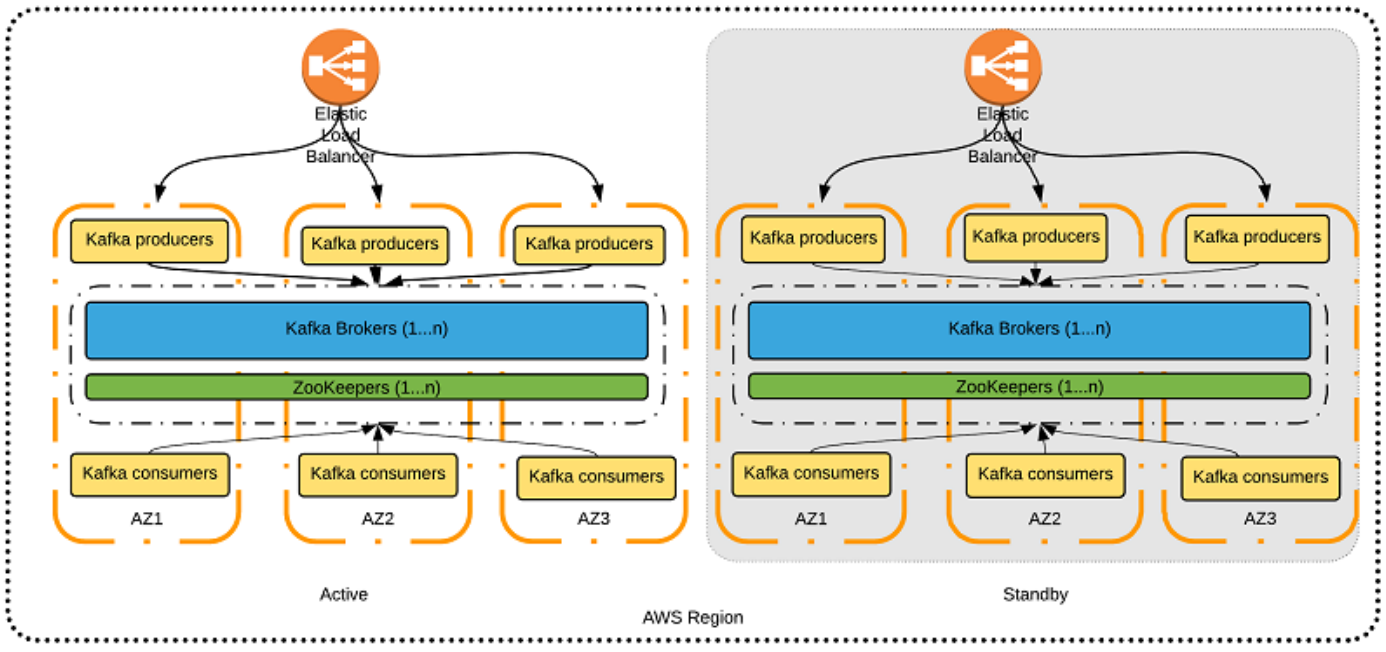
Pros	Cons

• Highly available
• Can sustain the failure of two AZs
• No message loss during failover
• Simple deployment
	• Very high operational overhead:
	  ◦ All changes need to be deployed three times, one for each Kafka cluster
	  ◦ Maintaining and monitoring three Kafka clusters
	  ◦ Maintaining and monitoring three consumer clusters

A restart is required for patching and upgrading brokers in a Kafka cluster. In this approach, a rolling upgrade is done separately for each cluster.

Single Region, Three Availability Zones, Active-Standby

Another typical deployment pattern (active-standby) is in a single AWS Region with a single Kafka cluster and Kafka brokers and Zookeepers distributed across three AZs. Another similar Kafka cluster acts as a standby as shown in the illustration following. You can use Kafka mirroring with MirrorMaker to replicate messages between any two clusters.



In this pattern, this is the Kafka cluster deployment:

- Kafka producers are deployed on all three AZs.
- Only one Kafka cluster is deployed across three AZs (active).
- ZooKeeper instances are deployed on each AZ.
- Brokers are spread evenly across all three AZs.
- Kafka consumers can be deployed across all three AZs.
- Standby Kafka producers and a Multi-AZ Kafka cluster are part of the deployment.

Kafka cluster failover occurs this way:

- Switch traffic to standby Kafka producers cluster and Kafka cluster.
- Restart consumers to consume from standby Kafka cluster.

Following are the pros and cons of this pattern.

Pros	Cons
• Less operational overhead when compared to the first option • Only one Kafka cluster to manage and consume data from • Can handle single AZ failures without activating a standby Kafka cluster	• Added latency due to cross-AZ data transfer among Kafka brokers • For Kafka versions before 0.10, replicas for topic partitions have to be assigned so they’re distributed to the brokers on different AZs (rack-awareness) • The cluster can become unavailable in case of a network glitch, where ZooKeeper does not see Kafka brokers • Possibility of in-transit message loss during failover

Intuit recommends using a single Kafka cluster in one AWS Region, with brokers distributing across three AZs (single region, three AZs). This approach offers stronger fault tolerance than otherwise, because a failed AZ won’t cause Kafka downtime.

Storage options

There are two storage options for file storage in Amazon EC2:

- Ephemeral storage (instance store)

- Amazon Elastic Block Store (Amazon EBS)

Ephemeral storage is local to the Amazon EC2 instance. It can provide high IOPS based on the instance type. On the other hand, Amazon EBS volumes offer higher resiliency and you can configure IOPS based on your storage needs. EBS volumes also offer some distinct advantages in terms of recovery time. Your choice of storage is closely related to the type of workload supported by your Kafka cluster.

Kafka provides built-in fault tolerance by replicating data partitions across a configurable number of instances. If a broker fails, you can recover it by fetching all the data from other brokers in the cluster that host the other replicas. Depending on the size of the data transfer, it can affect recovery process and network traffic. These in turn eventually affect the cluster's performance.

The following table contrasts the benefits of using an instance store versus using EBS for storage.

Instance store	EBS
• Instance storage is recommended for large- and medium-sized Kafka clusters. For a large cluster, read/write traffic is distributed across a high number of brokers, so the loss of a broker has less of an impact. However, for smaller clusters, a quick recovery for the failed node is important, but a failed broker takes longer and requires more network traffic for a smaller Kafka cluster. • Storage-optimized instances like h1, i3, and d2 are an ideal choice for distributed applications like Kafka.	• The primary advantage of using EBS in a Kafka deployment is that it significantly reduces data-transfer traffic when a broker fails or must be replaced. The replacement broker joins the cluster much faster. • Data stored on EBS is persisted in case of an instance failure or termination. The broker's data stored on an EBS volume remains intact, and you can mount the EBS volume to a new EC2 instance. Most of the replicated data for the replacement broker is already available in the EBS volume and need not be copied over the network from another broker. Only the changes made after the original broker failure need to be transferred across the network. That makes this process much faster.

Intuit chose EBS because of their frequent instance restacking requirements and also other benefits provided by EBS.

Generally, Kafka deployments use a replication factor of three. EBS offers replication within their service, so Intuit chose a replication factor of two instead of three.

Instance types

The choice of instance types is generally driven by the type of storage required for your streaming applications on a Kafka cluster. If your application requires ephemeral storage, h1**,** i3, and d2 instances are your best option.

Intuit used r3.xlarge instances for their brokers and r3.large for ZooKeeper, with ST1 (throughput optimized HDD) EBS for their Kafka cluster.

Here are sample benchmark numbers from Intuit tests.

Configuration	Broker bytes (MB/s)
• r3.xlarge • ST1 EBS • 12 brokers • 12 partitions	Aggregate 346.9

If you need EBS storage, then AWS has a newer-generation r4 instance. The r4 instance is superior to R3 in many ways:

- It has a faster processor (Broadwell).
- EBS is optimized by default.
- It features networking based on Elastic Network Adapter (ENA), with up to 10 Gbps on smaller sizes.
- It costs 20 percent less than R3.

Note: It's always best practice to check for the latest changes in instance types.

Networking

The network plays a very important role in a distributed system like Kafka. A fast and reliable network ensures that nodes can communicate with each other easily. The available network throughput controls the maximum amount of traffic that Kafka can handle. Network throughput, combined with disk storage, is often the governing factor for cluster sizing.

If you expect your cluster to receive high read/write traffic, select an instance type that offers 10-Gb/s performance.

In addition, choose an option that keeps interbroker network traffic on the private subnet, because this approach allows clients to connect to the brokers. Communication between brokers and clients uses the same network interface and port. For more details, see the documentation about IP addressing for EC2 instances.

If you are deploying in more than one AWS Region, you can connect the two VPCs in the two AWS Regions using cross-region VPC peering. However, be aware of the networking costs associated with cross-AZ deployments.

Upgrades

Kafka has a history of not being backward compatible, but its support of backward compatibility is getting better. During a Kafka upgrade, you should keep your producer and consumer clients on a version equal to or lower than the version you are upgrading from. After the upgrade is finished, you can start using a new protocol version and any new features it supports.

There are three upgrade approaches available, discussed following.

Rolling or in-place upgrade

In a rolling or in-place upgrade scenario, upgrade one Kafka broker at a time. Take into consideration the recommendations for doing rolling restarts to avoid downtime for end users.

Downtime upgrade

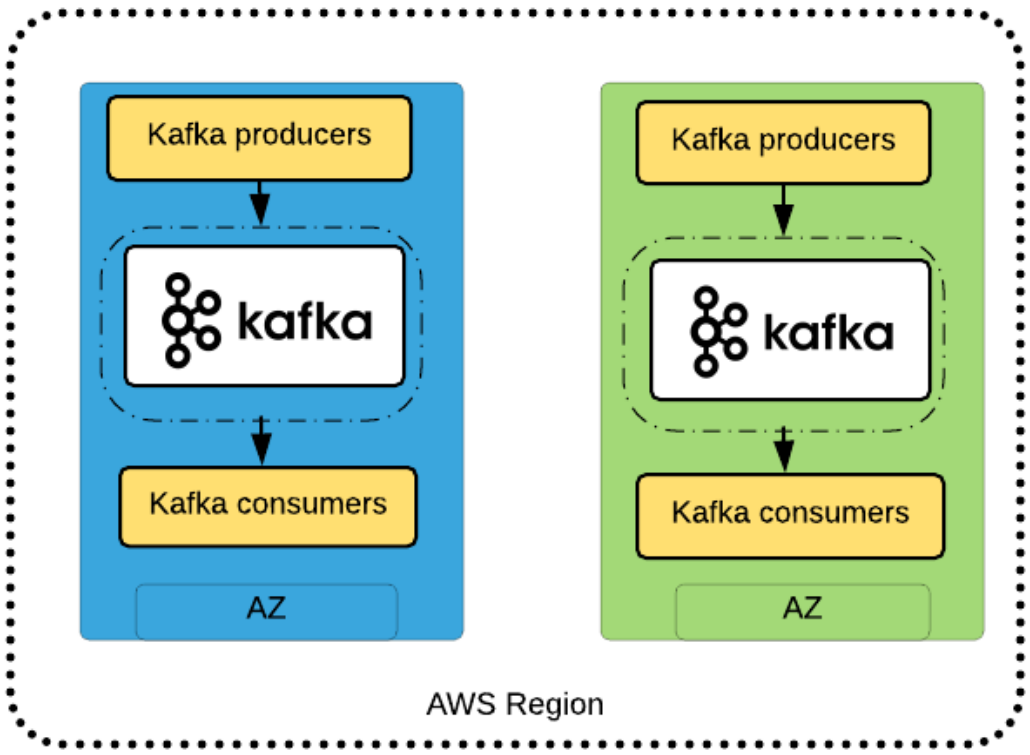
If you can afford the downtime, you can take your entire cluster down, upgrade each Kafka broker, and then restart the cluster.

Blue/green upgrade

Intuit followed the blue/green deployment model for their workloads, as described following.

If you can afford to create a separate Kafka cluster and upgrade it, we highly recommend the blue/green upgrade scenario. In this scenario, we recommend that you keep your clusters up-to-date with the latest Kafka version. For additional details on Kafka version upgrades or more details, see the Kafka upgrade documentation.

The following illustration shows a blue/green upgrade.



In this scenario, the upgrade plan works like this:

- Create a new Kafka cluster on AWS.
- Create a new Kafka producers stack to point to the new Kafka cluster.
- Create topics on the new Kafka cluster.
- Test the green deployment end to end (sanity check).
- Using Amazon Route 53, change the new Kafka producers stack on AWS to point to the new green Kafka environment that you have created.

The roll-back plan works like this:

- Switch Amazon Route 53 to the old Kafka producers stack on AWS to point to the old Kafka environment.

For additional details on blue/green deployment architecture using Kafka, see the re:Invent presentation Leveraging the Cloud with a Blue-Green Deployment Architecture.

Performance tuning

You can tune Kafka performance in multiple dimensions. Following are some best practices for performance tuning.

These are some general performance tuning techniques:

- If throughput is less than network capacity, try the following:
  - Add more threads
  - Increase batch size
  - Add more producer instances
  - Add more partitions
- To improve latency when `acks =-1`, increase your `num.replica.fetches` value.
- For cross-AZ data transfer, tune your buffer settings for sockets and for OS TCP.

- Make sure that `num.io.threads` is greater than the number of disks dedicated for Kafka.
- Adjust `num.network.threads` based on the number of producers plus the number of consumers plus the replication factor.
- Your message size affects your network bandwidth. To get higher performance from a Kafka cluster, select an instance type that offers 10 Gb/s performance.

For Java and JVM tuning, try the following:

- Minimize GC pauses by using the Oracle JDK, which uses the new G1 garbage-first collector.
- Try to keep the Kafka heap size below 4 GB.

Monitoring

Knowing whether a Kafka cluster is working correctly in a production environment is critical. Sometimes, just knowing that the cluster is up is enough, but Kafka applications have many moving parts to monitor. In fact, it can easily become confusing to understand what's important to watch and what you can set aside. Items to monitor range from simple metrics about the overall rate of traffic, to producers, consumers, brokers, controller, ZooKeeper, topics, partitions, messages, and so on.

For monitoring, Intuit used several tools, including Newrelec, Wavefront, Amazon CloudWatch, and AWS CloudTrail. Our recommended monitoring approach follows.

For system metrics, we recommend that you monitor:

- CPU load
- Network metrics
- File handle usage
- Disk space
- Disk I/O performance
- Garbage collection
- ZooKeeper

For producers, we recommend that you monitor:

- `Batch-size-avg`
- `Compression-rate-avg`
- `Waiting-threads`
- `Buffer-available-bytes`
- `Record-queue-time-max`
- `Record-send-rate`
- `Records-per-request-avg`

For consumers, we recommend that you monitor:

- `Batch-size-avg`
- `Compression-rate-avg`
- `Waiting-threads`
- `Buffer-available-bytes`
- `Record-queue-time-max`
- `Record-send-rate`
- `Records-per-request-avg`

Security

Like most distributed systems, Kafka provides the mechanisms to transfer data with relatively high security across the components involved. Depending on your setup, security might involve different services such as encryption, Kerberos, Transport Layer Security (TLS) certificates, and advanced access control list (ACL) setup in brokers and ZooKeeper. The following tells you more about the Intuit approach. For details on Kafka security not covered in this section, see the Kafka documentation.

Encryption at rest

For EBS-backed EC2 instances, you can enable encryption at rest by using Amazon EBS volumes with encryption enabled. Amazon EBS uses AWS Key Management Service (AWS KMS) for encryption. For more details, see Amazon EBS Encryption in the EBS documentation. For instance store–backed EC2 instances, you can enable encryption at rest by using Amazon EC2 instance store encryption.

Encryption in transit

Kafka uses TLS for client and internode communications.

Authentication

Authentication of connections to brokers from clients (producers and consumers) to other brokers and tools uses either Secure Sockets Layer (SSL) or Simple Authentication and Security Layer (SASL).

Kafka supports Kerberos authentication. If you already have a Kerberos server, you can add Kafka to your current configuration.

Authorization

In Kafka, authorization is pluggable and integration with external authorization services is supported.

Backup and restore

The type of storage used in your deployment dictates your backup and restore strategy.

The best way to back up a Kafka cluster based on instance storage is to set up a second cluster and replicate messages using MirrorMaker. Kafka's mirroring feature makes it possible to maintain a replica of an existing Kafka cluster. Depending on your setup and requirements, your backup cluster might be in the same AWS Region as your main cluster or in a different one.

For EBS-based deployments, you can enable automatic snapshots of EBS volumes to back up volumes. You can easily create new EBS volumes from these snapshots to restore. We recommend storing backup files in Amazon S3.

For more information on how to back up in Kafka, see the Kafka documentation.

Conclusion

In this post, we discussed several patterns for running Kafka in the AWS Cloud. AWS also provides an alternative managed solution with Amazon Kinesis Data Streams, there are no servers to manage or scaling cliffs to worry about, you can scale the size of your streaming pipeline in seconds without downtime, data replication across availability zones is automatic, you benefit from security out of the box, Kinesis Data Streams is tightly integrated with a wide variety of AWS services like Lambda, Redshift, Elasticsearch and it supports open source frameworks like Storm, Spark, Flink, and more. You may refer to kafka-kinesis connector.

If you have questions or suggestions, please comment below.

Additional Reading

If you found this post useful, be sure to check out Implement Serverless Log Analytics Using Amazon Kinesis Analytics and Real-time Clickstream Anomaly Detection with Amazon Kinesis Analytics.

About the Author

Prasad Alle is a Senior Big Data Consultant with AWS Professional Services. He spends his time leading and building scalable, reliable Big data, Machine learning, Artificial Intelligence and IoT solutions for AWS Enterprise and Strategic customers. His interests extend to various technologies such as Advanced Edge Computing, Machine learning at Edge. In his spare time, he enjoys spending time with his family.

TAGS: Amazon Kinesis Streams, Amazon S3

Resources

Amazon Athena
Amazon EMR
AWS Glue
Amazon DynamoDB
Amazon Kinesis
Amazon QuickSight
Amazon Redshift

Follow

Twitter
Facebook
LinkedIn
Twitch
Email Updates

Related Posts

Our data lake story: How Woot.com built a serverless data lake on AWS

How Disney built their streaming analytics pipeline on AWS

How Startups Can Grow Faster in 2019

ICYMI: Serverless Q4 2018

The most-viewed AWS Database Blog posts in 2018

Analyze user behavior using Amazon Elasticsearch Service, Amazon Kinesis Data Firehose and Kibana

Manage centralized Microsoft Exchange Server logs using Amazon Kinesis Agent for Windows

Stream Amazon CloudWatch Logs to a Centralized Account for Audit and Analysis