Apache Kafka and AWS take Distributed Messaging to the next level

A Technical White Paper by CloudTern

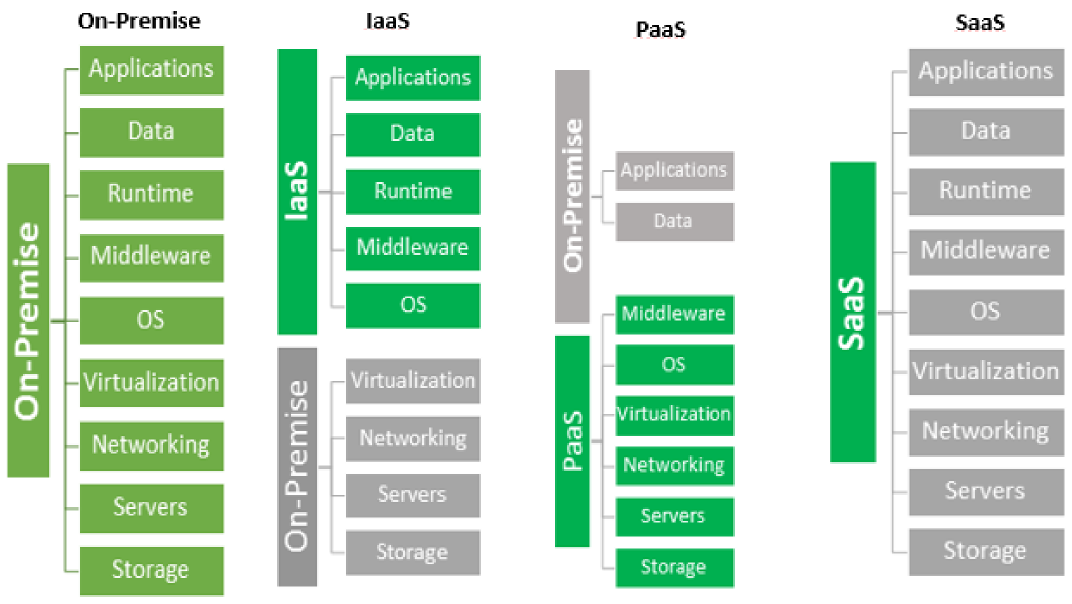
Abstract

With the cloud technology becoming an inevitable option, cloud providers are in great demand in recent times. Amazon Web Services (AWS) sits at the top of this revolution, enjoying 1/3rd or the public cloud market. Apache Kafka is a powerful distributed messaging system that has quickly evolved into a great option for AWS deployments. This white paper explores the functionality and performance of Apache Kafka in the AWS environment. It also compares Kafka with other messaging systems and concludes that AWS and Apache Kafka make a great combination.

Table of Contents

Introduction

From being an option, cloud computing has now become an inevitable technology in recent times. Right from the server and network to storage and applications, every resource is now moving to the cloud. The term 'cloud' is a generic term that is used as a synonym for the Internet. It actually represents the Internet, the communication system and the underlying infrastructure. From the time of its inception, cloud computing has been a disruptive technology for every vertical. The cloud centralizes resources, optimizes costs and improves business efficiencies. The pay per use model allows for a flexible financial management. Cloud computing is available in three major types; Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS).



The state of cloud computing

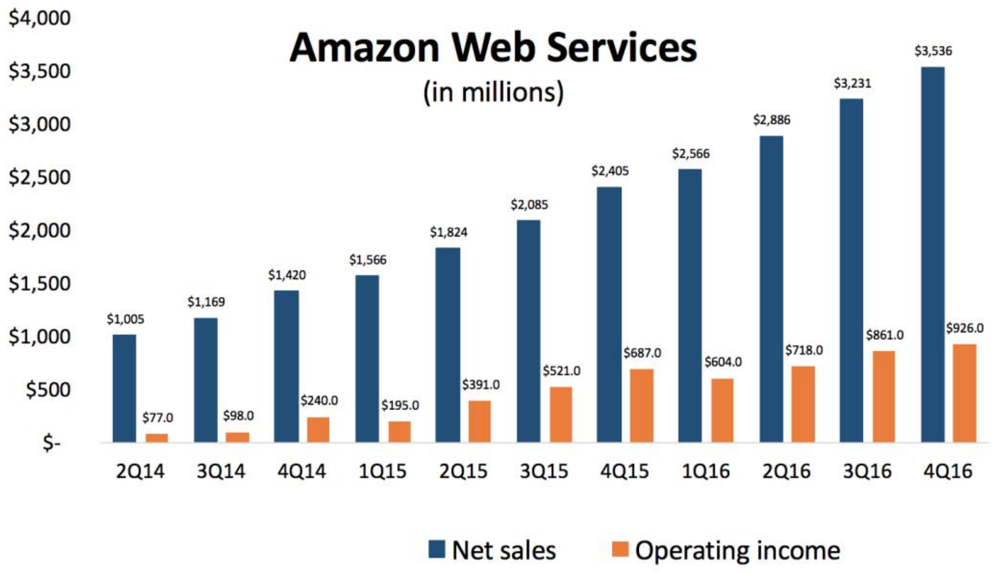
The Cloud computing is growing at a rapid pace. According to Gartner, the public cloud market has earned a revenue of $209.2 billion in 2016. This value is expected to touch $246.8 billion in 2017. Infrastructure-as-a-Service is expected to lead the cloud market with a growth of 36.8% in 2017 to reach $34.6 billion followed by Software-as-a-Service (SaaS) growing at a 20.1% to reach $46.3 billion. [Forbes](#) predicts that global cloud IT spent would increase to $390 billion by 2020 from $180 billion in 2015, growing at a CAGR of 17%.

Market leaders in the cloud

Amazon AWS, Google Cloud, Microsoft Azure and IBM Cloud are the leading providers of cloud solutions. Among these first tier companies, Amazon is the clear leader in the public cloud segment. According to Canalys report, Amazon Web Services, popularly known as AWS, holds 33.8% of the market share in the IaaS and PaaS sectoras of 2016. This value is more than the combined value of Google, IBM and Microsoft, 30.8%. [Canalys](#) predicts that this market value will touch $55.8 billion by the end of 2017.

AWS stands at the top

Amazon Web Services (AWS) is the cloud solution offered by Amazon.com. It is the most popular cloud computing solution in the world. According to a GeekWire report, AWS has earned a revenue of $3.53 billion in Q4 of 2016. This value is $300 million more than the revenues generated in Q3 of 2016. AWS revenues spiked up by 47% from the year-ago quarter. The company has now reached a $14 billion annual runrate.



Source: https://www.geekwire.com/2017/amazon-web-services-posts-3-5b-revenue-47-last-year/

Many organizations are completely dependent on AWS. For startups, AWS is a great option as you don't have to invest on the infrastructure. AWS is easy to use. You can quickly deploy your applications on AWS while monitoring the performance from a central location via the management console. The AWS solution is flexible which means you can choose your programming language, operating system, database, web application platform and other services. AWS solutions are highly scalable to suit your business needs on-demand. They are reliable and secure. As you only pay for the services used, AWS optimize costs while improving the performance of your business procedures.

Moving to AWS

Before moving the business infrastructure to the cloud, businesses need to consider certain aspects such as achieving resilience, providing consistent user experience, implementing network segmentation and centralizing monitoring of infrastructure. Most importantly, businesses have already made large investments for on-premise infrastructure and datacenters. Moving these variable workloads means you need powerful management tools that are augmented by expertise professionals. It is a major concern in terms of financial aspects as well as management aspects. A best practise is to move small batches and test environments to the cloud and proactively monitor the performance.

When it comes to Messaging system, AWS mainly supports two types of brokers.

a) Amazon Simple Queue Service (SQS)& Amazon Simple Notification Service (SNS) for simple and non-distributed networks.
b) Amazon Kinesis for distributed, highly scalable and fault tolerant networks.

While AWS offers powerful tools to manage your infrastructure, it is not a good idea to have over-reliance on AWS infrastructure. Firstly, it results in vendor lock-in. When you try to move the system to another infrastructure, it becomes a difficult task. Secondly, you are constrained by AWS restrictions such as number of messages per unit time, size and how long a message lives etc.

For a distributed and insanely scalable messaging system, Apache Kafka is the ultimate choice. Before looking at Kafka, check out the available messaging options.

Challenges with traditional messaging system

Businesses have multiple options when it comes to choosing a messaging system. However, traditional messaging systems come with several challenges. Here are the most common ones.

Single Point of Failure

The traditional messaging systems are designed for a Hub and Spoke topology. In this design, all the messages are stored on a central hub or a broker. So, each client application connects to one server or broker at any given point in time. As all topics are queued in the central hub, this design can turn out be a single point of failure. Even when you install a standby to the primary broker, the client application can still connect to a single server.

Difficulties in horizontal scaling

Though Hub and spoke designs evolved into multi-node networks, the architecture doesn’t allow for horizontal scaling. A single client application connects to a single node at a time. When you increase the number of nodes, the amount of inter-node traffic that are processing writes and reads increases as well.

Monolithic architecture

Traditional messaging systems are designed to address data challenges of a monolithic architecture. However, most organizations now run a distributed and clustered computing environment. In this design, large clusters of commodity hardware cannot be scaled horizontally. Moreover, messages have to wait in queues.

Apache Kafka is specially designed to address data management challenges in a distributed and clustered computing environment. Apache Kafka and AWS take the distributed messaging to the next level.

How Apache Kafka solves these challenges?

Apache Kafka is a fault-tolerant and highly scalable distributed messaging system designed by LinkedIn. It was the idea of a group of LinkedIn engineers Jay Kreps, Neha Narkhede and Jun Rao who were working on data streaming tasks. The event data from the LinkedIn websites and the entire infrastructure was ingested to Lambda architecture to be harnessed by Hadoop and other real-time event processing systems. During this process, the company experienced low-latency issues in processing real-time events. The result was Kafka. It was designed in 2010 and was made public in 2011. With Kafka, LinkedIn was able to deliver massive message streams to Hadoop clusters. It later became an Apache project.

The Kafka Architecture

The Kafka architecture consists of four basic components namely Brokers, Consumers, Producers and Zookeepers. The basic element is a Message which is actually a payload of bytes. Streams of messages belonging to a specific category is called a Topic. Each message is identified by its index called offset.

Topics

Kafka stores data in Topics. Each topic is partitioned as a set of segment files that are equal in size. Messages in the partition are organized into an immutable order sequence. Each topic has a minimum of one partition. The backup of a partition is called a Replica.

Brokers

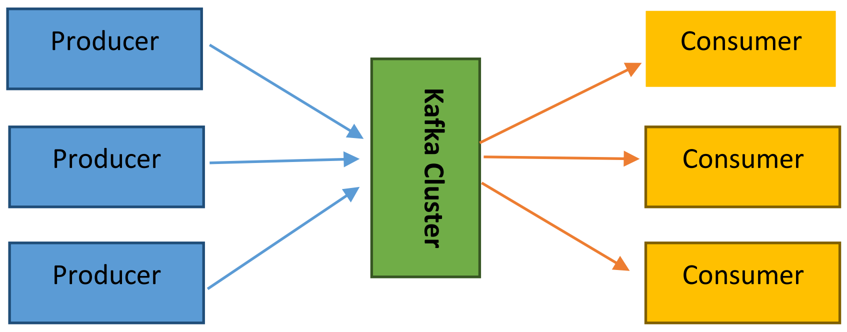
A broker is a stateless server that stores the published messages. A set of servers are also called clusters. One instance of Kafka broker can handle TB of messages and hundreds of thousands of reads and writes per second without impacting the performance. As brokers are stateless, a zookeeper is used to maintain the cluster state.

Zookeepers

The role of a zookeeper is to co-ordinate and manage Kafka brokers. When a new broker is started or an existing broker is stopped, the zookeeper sends a notification to producers and consumers. Based on these notifications, producers and consumers communicate with brokers accordingly. Many distributed systems such as Apache Hadoop, Neo4J and HBase are using Zookeeper to successfully run their projects.

Producers

A producer is the element that publishes messages to a topic. In technical terms, producers push data to brokers. Producers doesn't need an acknowledgement from a broker. They continuously push data to brokers as fast as the broker can handle. Every time a new broker is started, Kafka producers search for that new broker and send messages to that broker.

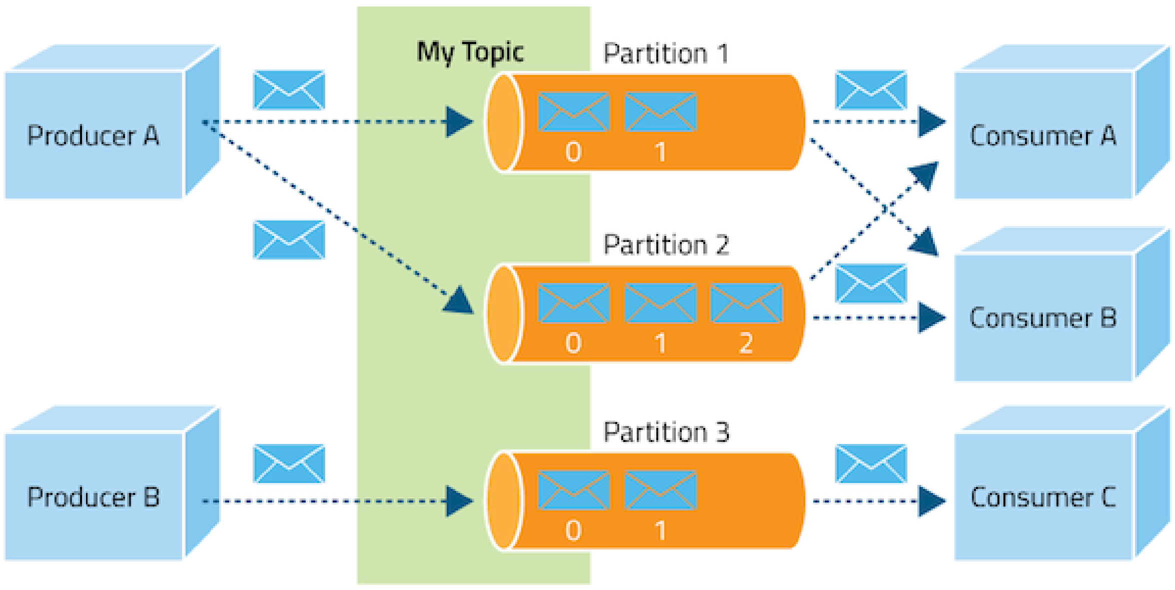


Consumers

A consumer is a Kafka element that subscribes to one or more topics to consume messages from brokers. Consumers use partition offset values to maintain how many messages have been consumed. This partition offset value is obtained from the zookeeper. When a particular message offset is acknowledged by a consumer, it means all the prior messages are consumed.

Kafka Storage Design

Kafka uses a simple storage layout in the form of logical logs. Topics are partitioned into logical logs. When a producer publishes a message to a partition, the message is appended to the last segment of the file. No explicit message ID is given. After a certain period of time is elapsed or when a configurable number of messages are published, the segment file is flushed to the disc and exposed to consumer. The logical offset in the log exposes the messages which means you don't need random-access index structure to map messages ID to the actual locations. Messages from a partition are sequentially consumed by the consumer. An asynchronous pull request is issued to the broker so that the broker prepares a buffer of bytes to be consumed. The sendfile API helps Kafka to efficiently log segment file bytes from brokers to consumers.



Source: https://thenewstack.io/apache-kafka-primer/

How does Apache Kafka messaging system work?

Kafka supports both publish-subscribe messaging system as well as queue based messaging system.

Here is an example of workflow in the publish-subscribe system

1. Producers are responsible for pushing data to brokers. Producers regularly send messages to topics.
2. Brokers categorize these messages into specific topics and stores them in corresponding partitions. All the partitions share equal number of messages. When there are two messages, each partition stores one message.

3. To pull data from these partitions, consumers first subscribe to a topic. Then the offset of the topic is provided to the consumer. This offset value is also saved in the zookeeper ensemble.
4. Consumers request for messages in regular intervals. When new messages are posted by producers, they are forwarded to consumers.
5. Consumers process these messages and send an acknowledgement to the broker after the processing of the messages is done.
6. After an acknowledgement is received, the broker updates the offset value with a new one.
7. This workflow is repeated till the consumer stops requesting for messages.
8. Consumers can rewind to any offset value to consume desired messages.

Here is an example of workflow in the Queue Messaging system

The queue messaging system is similar to publish-subscribe system. The only difference is that instead of one, consumers come in groups to pull data from partitions. They are segregated into groups and data is published.

1. Producers are responsible for pushing data to brokers. Producers regularly send messages to topics.
2. Brokers categorize these messages into specific topics and stores them in corresponding partitions. All the partitions share equal number of messages. When there are two messages, each partition stores one message.
3. To pull data from these partitions, consumers first subscribe to a topic. Then the offset of the topic is provided to the consumer. This offset value is also saved in the zookeeper ensemble.
4. Brokers supply messages to the consumer until another consumer subscribes to the same topic. With the arrival of a new consumer, the broker shares the messages between the consumers. This process will be carried out until the number of consumers doesn’t exceed the threshold for that partition. This group is called a consumer group.

5. After maximum number of consumers subscribe to a topic, the broker doesn't entertain new consumers till one of the consumer unsubscribes from that topic.
6. Consumers request for messages in regular intervals. When new messages are posted by producers, they are forwarded to consumers.

Apache Kafka is a powerful distributed messaging system that is fault-tolerant and highly scable. For both publishing and subscribing, Kafka offers a high throughput. It supports multiple subscribers. At the same time, it efficiently balances consumers without failures and downtime. As messages are persisted on the disk, Apache Kafka is best suited for real-time applications as well as ETL (Extract, Transform and Load) operations.

Managing Twitter feed - Apache Kafka Use Case

Here is an example of how Apache Kafka helps you to process streaming data in real time. Consider an instance wherein you have tomonitor trending twitter feeds and hashtags. You need a Kafka producer that collects data from Twitter, processes it, extracts hashtags and sends this data to the Kafka Ecosystem. You can install a data processing engine such as Apache Storm or Apache Spark.

Firstly, the Kafka producer collects data from the Twitter feed using the Twitter Streaming API. You can use any programming language to access the Twitter streaming API and get details of various subsets of public and protected Twitter data. You also need to sign up for a Twitter developer account. After signing up, you can get OAuth authentication details such as Customerkey, CustomerSecret, AccessToken and AccessTokenSecret. The Kafka producer can receive twitter feed and process it to extract hashtags. This information is sent to the Apache Storm/Spark ecosystem.

Apache Kafka is used by some of the popular applications such as Twitter, Netflix, Oracle, Mozilla and LinkedIn.

Why Kafka is better than other messaging systems?

There are several messaging systems that are alternate to Kafka. RabbitMQ, ActiveMQ and ZeroMQ are some of the popular ones.

RabbitMQ

RabbitMQ is a powerful messaging broker that was actually designed to implement Advanced Message Queuing Protocol (AMQP). This open-source tool is developed in Erlang and is easy to install and use. RabbitMQ supports both message persistence and replication. RabbitMQ comes with excellent routing capabilities based on rules. The performance is good. RabbitMQ is broker-centric which means it focuses on deliver guarantees between the consumer and the producers. Advanced capabilities such as routing, persistent messaging and load balancing can be performed with a few lines of code. However, RabbitMQ messaging system is not distributed. When you have an infrastructure that scales massively, RabbitMQ won't be able to match that capability.

ZeroMQ

ZeroMQ is a light weight messaging system that comes with strong documentation and active support. The tool is especially useful for instances wherein low latency and high throughout are required. It offers all advanced capabilities similar to RabbitMQ. However, the downside is that you have to combine various pieces of frameworks to create those solutions. While ZeroMQ offers strong documentation, there is a bit of learning curve.

ActiveMQ

ActiveMQ is another popular messaging system that is easy to implement and use. It enjoys largest number of installations. The deployment supports both P2P and broker topologies. With a few lines of code, you can implement advanced capabilities. It uses Java Message Service specification. ActiveMQ offers numerous options when it comes to clustering and distribution. ActiveMQ enjoys strong documentation and active support. It is highly scalable and handles tens of thousands of messages per second.

ActiveMQ is reliable and delivers high performance. While ActiveMQ offers more features, it is more suited for simple queue service (SQS). When it comes to distributed systems that massively scale up and down, ActiveMQfaces tough competition from newer technologies that deliver better performance and features. ActiveMQ writes messages to a journal before shipping them to consumers. It means the number of messages that it can store depends on the disk capacity. In case of high memory consumption, ActiveMQ pauses producers until the space is freed. In a distributed system wherein producers also act as consumers, the entire system can be locked up.

Kafka output is far better than its competitors. For instance, Kafka producers doesn't wait for acknowledgements from the broker. They send messages as fast as the broker can handle. Moreover, Kafka storage format is more efficient than ActiveMQ and other systems. On an average, Kafka message overhead is 9 bytes wherein ActiveMQ message overhead is 144 bytes. This is because ActiveMQhas to manage various indexing structures. Another advantage is that there are no disk write activities on Kafka broker. RabbitMQ and ActiveMQ containers have to maintain the state of each message. Kafka reduces transmission overhead with sendfile API.

AWS and Kafka bring the best of both worlds to your table.

Cloud (AWS) migration challenges

Optimized costs, increased efficiencies, centralized IT management and mobility solutions are the key driving factors to the cloud adoption. However, migrating the IT infrastructure from on-premise to the cloud is surely a daunting task. Here are some of the key challenges.

Lack of holistic cloud strategy

Many organizations are aggressively embracing the cloud without understanding how it works. The process has to be carefully planned. Firstly, you should identify apps or services that can be moved to the cloud in the initial stage. Most organizations migrate to the cloud in smaller increments. You should have modern monitoring and logging tools to proactively assess the changes. Secondly, network segmentation is a critical concern.

Extending network segmentation

Organizations that deal sensitive data or healthcare data should adhere to stringent government regulations. Financial data, patient data and confidential data has to be separated from regular traffic. To achieve this, they perform network segmentation in the datacenter using mechanisms such as VLANs and VRFs. Extending this segmentation to the cloud is a challenge.

Refactoring applications and services

One of the main challenges in cloud migration is to decide whether to rebuild an application or rehost it. When you do a rehosting, you simply take an application as it is and deploy it in the cloud. Rehosting an application gives you the advantage of quickly hosting the application in the cloud. However, horizontal scaling can be a concern as the application is designed for a vertically integrated hardware. When you rebuild the application, it gets all the power but takes demands time and efforts while being expensive too. Resource intense applications are better suited for rebuilding.

Consistent user experience

Another cloud migration challenge is providing a consistent user experience. Datacenters leverage the IPSec technology and connect to AWS via encrypted IP tunnels. While it offers greater security, it increases application latency and demands higher bandwidth which negatively affect the application performance.

Centralized monitoring

The main challenge in cloud migration is centralized monitoring of the infrastructure. To monitor and manage the performance of application, organizations use multiple monitoring tools. When you have datacenters in the cloud and on-premise, it results in fragmented monitoring between the cloud and on-premise datacenters. Owing to the disparities in operating procedures, monitoring the entire infrastructure from a single dashboard is a challenge.

CloudTern stands at the forefront of this transition

In today's highly competitive world, businesses are required to deliver great applications in quick time. To stay ahead of competition, businesses should choose the right tools for the right tasks at right times. AWS provides the infrastructure that helps businesses build sophisticated applications in quick time and at reduced costs. Apache Kafka provides a highly scalable and fault-tolerant distributed messaging system that processes streams of data in real time. Together, AWS and Kafka bring the best of both worlds.

While Apache Kafka is a great messaging tool, organizations are concerned about the manual configuration and the learning curve. Similarly, AWS offers standardized cloud solutions. It means businesses have to spend time in customizing AWS solutions for organization-specific requirements. This is where CloudTern comes to the rescue.

CloudTern is a leading provider of cloud-based IT solutions. The company offers customized AWS cloud solutions that are tailor-made for your business. Whether you want to move your infrastructure to the AWS cloud, build great applications or manage the infrastructure in the cloud, CloudTern provides customized solutions for the AWS cloud. While CloudTern handles the AWS infrastructure, you can concentrate on your core business processes. Most importantly, CloudTern solutions are significantly cost-effective.

CloudTern has helped several companies make a smooth transition to the AWS environment.
Contact us today to leverage the power of Apache Kafka on the AWS environment!
CloudTern Solutions
www.cloudtern.com
Phone: +1.408.512.1994