AWS Certified Solutions Architect Associate (SAA-C02)

The most comprehensive and up-to-date study guide for the AWS Solution Architect Associate certification available today

Written by
Neal Sanford,
Allison Cope, &
Michael Reeves.

ISBN: 9798744835972

TABLE OF CONTENTS

Introduction

AWS is the top leading cloud services provider, which has modified and revolutionized our IT approach. AWS is the hallmark of many top global organizations, such as Netflix, Ubisoft, Spotify, and Expedia. AWS offers all the advantages of high data availability and durability that an individual might expect from a cloud service. AWS is a much better choice than other cloud service providers because it helps their users focus more on their projects and customers without stressing about infrastructure problems. It does not just save the user's efforts, time, and money but also helps them focus on the important business aspects.

AWS rules over the cloud computing market, making AWS certification a required certification for IT experts working with the IT Giants. Presently, AWS certificates are considered the most famous and high-in-demand certifications in the IT sector. These certifications are suitable for individuals interested to become Cloud engineers or Cloud architects. These certifications show a certified individual's ability to develop and operate tech solutions on Amazon's cloud platform. They are consistently ranked as the world's highest-paid information technology certifications. The salary of an IT certified professional can be increased by 26% with the AWS Cloud certifications.

One of the most popular and most valuable IT Certifications is AWS Solution Architect Associate. AWS SAA is considered a common entry point for people joining cloud architecture. An AWS Solution Architect is a person with a profound knowledge of critical cloud engineering concepts to efficiently design and implement reliable and scalable cloud technology solutions. With AWS Solution Architect Certification, you can understand, learn, assess, and demonstrate your ability to design and install secure and stable AWS platform applications. The AWS-certified Solution Architect can earn an average annual salary of $130,000, which is 11.7% higher than the non-certified individuals. So, if you have this certification in your resume, you will be most likely get a higher salary, and you will be high in demand by many employers.

This textbook is one of the in-depth and exam-specific tools specially made to prepare individuals for the AWS Solutions Architect Associate (SAA-C02) certification exam. It covers all the important AWS services, both basic and newly added elements, needed across all the AWS domains. We use the process of incremental learning to build a strong base of theoretical knowledge for our learners. By the end of this course, you will surely be equipped with enough knowledge to pass the certification exam.

Details of Certification Exam

This *Certification Exam* is intended for the individuals devoted to the Solution Architect's role. Normally, it identifies the participant's ability to develop a solution based on architectural design concepts, which focuses on the clients' needs and helps them execute the proposed solution for their project.

You will need an AWS exam guide to getting all the details and the exam's objectives after you have made your mind to take the test. The official guide is available on the official website of AWS. This certification also includes two levels of a solutions architect, Associate level and Professional level.

Important exam details are listed below:

- The exam includes 65 questions.
- The passing percentage is 65% (720 out of 1000 points).
- The exam duration is 130 minutes.
- This test consists of two types of questions, including multiple answers and multiple-choice questions.
- It is offered in various languages, including Chinese, Korean, English, and Japanese.
- This certification is valid for two years.

- *AWS Solution Architect Associate-level* exam costs $150, *whereas the Professional-level* exam costs $300.

- The latest version of the exam is released in August 30, 2022, referred to as AWS SAA-C02.

To demonstrate your AWS knowledge in the exam, you will be tested across four different domains. You will be examined through four different domains and each with a total percentage of your overall performance. While proceeding along the course, you will be able to fully understand each domain and gain the requisite knowledge to meet the domain requirements. These domains are categorized as:

Domain 1: Design Resilient Architectures (30% of scored content)

Domain 2: Design High-Performing Architectures (28% of scored content)

Domain 3: Design Secure Applications and Architectures (24% of scored content)

Domain 4: Design Cost-Optimized Architectures (18% of scored content)

Intended Audience

This learning approach is ideal for individuals who want to pass the AWS SAA-C02 Certification exam on their first go by acquiring in-depth knowledge of all of the AWS services required to become an AWS Solutions Architect. As stated in the blueprint for the AWS SAA exam, it is particularly intended for those who can check, validate, and effectively demonstrate their skills for designing solutions safely and robustly.

The most suitable match for this textbook is the beginners who want to get familiar with the AWS infrastructure and want to enhance their skills. They do not need any previous AWS knowledge or programming expertise to start this course. With this textbook's help, one will be able to create the base for the exam preparation even though he is never signed on to the AWS platform before. The Network Administrators and existing Solution Architects can also use this edition to learn more about the AWS cloud platform's newly added components to add to their expertise. Furthermore, it can be used by the companies concerned to make their

solution architects certified for hosting their applications that are entirely scalable and fault-tolerant onto the AWS cloud. Therefore, this approach can surely help you in preparing for the role of AWS Solutions Architect.

Key Learning

This textbook aims to validate your AWS knowledge in many key areas. This learning process enables you to understand all the key components with the help of detailed graphical explanations and real-life use case scenarios. In this way, you can easily grasp the concepts and principles of all components. Through this approach, you can:

1. Determine the right Amazon web service based on the customer requirements for computing, databases, security, and cost control mechanisms.

2. Learn to develop, deploy and maintain secure and flexible applications on the cloud with Amazon Web Services.

3. Get a detailed understanding of fault-tolerant, highly scalable, and highly available systems on AWS.

4. Identify the correct use of Amazon's most basic and newest web services.

5. New users can easily get familiar with the role of each component within the VPC architectural design.

Now, let us get started with the new journey to become Solution Architect Associate!

Chapter 1: Cloud Computing with AWS

Overview

The very first chapter of the textbook introduces the very commonly used Amazon *Web Services.* This chapter gives you an insight into the overview of the AWS Cloud and the benefits of using AWS. Along with the basic services that make up the platform, it also covers the life cycle of the AWS Cloud computing platform. Let us get started with our first chapter.

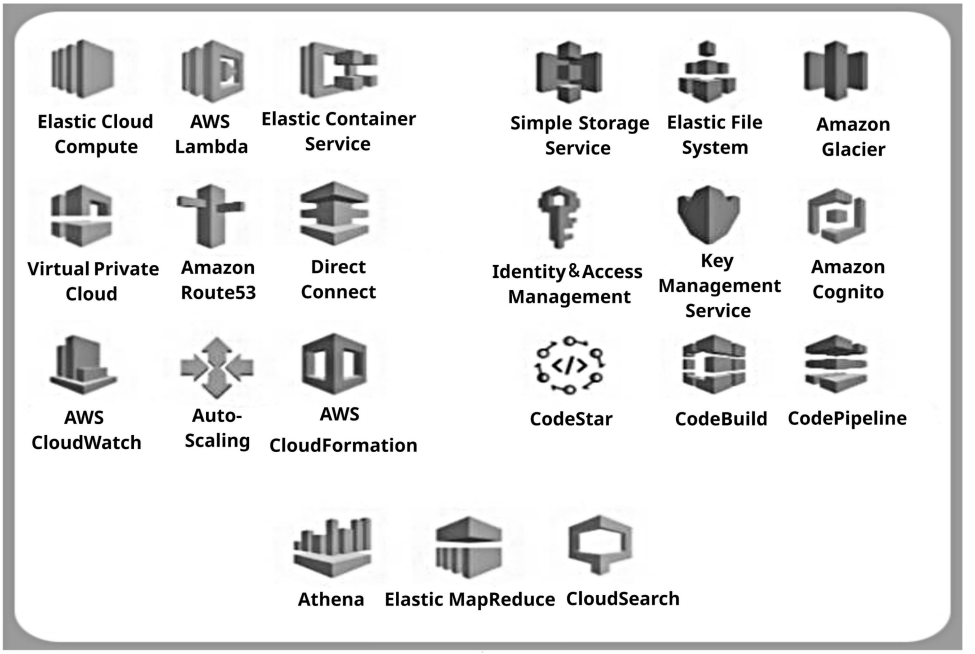
Lifecycle of the AWS Cloud Computing Solution

Before we learn more about Amazon Web Services, let us quickly look at the lifecycle of the AWS cloud solution as it will help you understand the AWS cloud infrastructure more clearly. Getting a *Correct Understanding of Requirements* is the first step in the lifecycle of Amazon's cloud computing solution. After that, the appropriate service offered by the provider will then be chosen. The second step would be to *Define the Hardware* after a fine understanding of your requirements. Defining hardware is selecting a compute service that provides the desired support so that the computing capacity can be resized to run applications. Knowing the exact requirement can also help you in choosing the right hardware.

As not all fit one size so, different services and hardware for different user requirements are available in the AWS marketplace. You can use *Elastic Cloud Compute* for IaaS, *Lambda* for serverless computing, and *ECS* for containerized service. You need to select the right hardware according to your requirements.

Now, the third step is *Defining storage*. Select the relevant service for storage to backup your data and a separate storage service archiving the data either locally within the cloud or on the internet. You have to pick up the right storage option, such as *Simple Storage Service (S3)*, especially for backups**.** There is separate storage to archive data, known as *Amazon Glacier*. Therefore, recognizing the difference between them help

you pick the best service for your desired requirement. You can see some of these services in the following diagram:



Next is to *Define the Network*, which delivers data, videos, and applications securely. You have to define the network services and identify them correctly. For instance, you use *Virtual Private Cloud* for Networking, *Direct Connect* for private connection of user's office to the AWS center, and *Route-53* for Domain Name System. The next step is to establish the *Right Security Services*; for example, *IAM* is used to authenticate and authorize the access while *KMS* is used for data encryption at rest. Therefore, several security services are accessible, so you must choose the correct one according to the requirements.

Then the next step is to *Define the Management Tools and Processes*. Here you can choose from various deployment, automation, and monitoring tools available in the AWS platform. CloudWatch is a monitoring service, while Auto-scaling is an automation service, and CloudFormation is a deployment service. If you identify the management tools properly to monitor the AWS resources and all customized applications on the AWS platform, you can efficiently control

your cloud environment. Understanding them would also help you to identify the cloud computing solution's life cycle properly.

Furthermore, many resources are available for *Testing a Process*, such as *CodeStar, CodeBuild,* and *CodePipeline*. These are tools that allow you to build, test, and deploy your code quickly. Finally, once everything has been set and done, select the *Analytics Services* to analyze and visualize the data. You can start to query the data immediately and get a response from the analytic services. The Amazon Athena can be used to view the events in your region visually. The *EMR-Elastic Map Reduce* and *CloudSearch are the* other analytical services. So, this is a brief introduction to the lifecycle of the AWS cloud platform. Let us move onto the overview of some basic Amazon web services.

Overview of AWS

AWS is the Cloud Computing technology operated by Amazon, which is available over the internet. Let us now discuss the various types of services provided by AWS. The services we describe here can be classified into categories, such as compute, storage, database, networking, security, management, monitoring services, and application integration services. If you are looking for a service that is not available, these services may be included in the certifications for Big Data or DevOps Engineer. Now, let us discuss a few AWS services that are frequently used for SAA Certification.

Within *Compute Services,* we have Amazon EC2, Amazon Lambda, and Elastic Beanstalk.

Amazon EC2: In cloud services, Amazon EC2 offers to compute capacity. It provides security and can be resized according to the end-users needs.

AWS Lambda: This refers to a compute service that enables users to execute the code without using and managing another service. AWS Lambda will only run the code when necessary and scales it from fewer requests a day to thousands a second. Moreover, the users will only be charged for the computing time that they consume.

Elastic Beanstalk: An Application Orchestration service that allows its users to scale and launch Web applications. Different programming languages are used for their development. A user needs to upload the code. It will then automatically monitor the whole deployment process and the application's performance as well.

Within *Storage Services,* Amazon provides S3, *Glacier, Amazon EBS, and EFS.*

Amazon S3: S3 is generally file storage and sharing service used for uploading and sharing files. It is the object storage that can store and recover data from any platform such as websites, mobile applications, IoT sensors, etc.

Amazon Glacier: This cloud storage service is safe, long-lasting, and exceptionally cost-effective that is used for data storage and permanent backups.

Amazon EBS: This service offers block store volumes for the EC2 instances. EBS service is highly accessible, and it provides secure storage volume, which can be connected to working EC2 instances in a similar AZ. The EBS volumes-attached instances are shown as Storage Volumes that remain independent of the instance's lifetime.

Amazon EFS: It can be used with the EC2 instance as a flexible cloud-based storage solution. It provides a simpler interface with the help of quickly generating and customizing file systems with ease. Amazon file system is designed to scale on users' demand elastically without affecting the application from expanding and shrinking.

Now let us talk about *Database Services in AWS.* We have two main flavors of databases in AWS: Amazon RDS and Amazon Redshift.

Amazon RDS*:* RDS lets you run and organize a database in the cloud. They have nearly every database from SQL Server to Oracle, MySQL PostgreSQL, and Aurora. RDS supports the process involved in the creation, operation, and scaling of a relational database. It offers a cost-effective and resizable capacity as it automates administrative activities

like providing hardware, Database setups, repairs, and data backup, which require a considerable amount of time.

Amazon Redshift: It provides a data storage service for analyzing data through SQL and other smart corporate tools. With the help of advanced query optimization against the petabytes of structured information, you can even run complex analytical queries. Moreover, most of the results are generated within seconds.

In the *Management and Monitoring Services,* we have CloudWatch, CloudTrail, and CloudFormation.

Cloud Watch: It is used for tracking applications and resources that run on the AWS platform. It is used Cloud Watch provides two types of monitoring that are Basic monitoring and Detailed monitoring.

Cloud Trail: CloudTrail is a web service in the AWS account that monitors your API operation. It is a very effective monitoring tool that can enable users to track logs, control, and maintain actions-related account activity within the AWS network.

Cloud Formation: It is a service that models and configures AWS resources, allowing you to concentrate more on your application and spend less time handling those AWS resources. In cloud formation, different templates are used to generate user applications instantly. Moreover, infrastructure is defined as code here.

AWS provides *Networking & Content Delivery* services such as Virtual Private Cloud, Route 53, Elastic load balancer, and Cloud Front.

VPC**:** With Virtual Private Cloud, you can set up virtual networks in the cloud for running your servers in those networks. You are not allowed to have full control over the cloud network; instead, you can have VPCs, which are the chunks of your cloud.

Route 53: This Networking and Content delivery service has high availability and is a fully managed DNS service. Being a scalable Domain

Name System, it is fully compatible with IPV6 and scales according to the requirement.

Elastic Load Balancer: It is used for network and content delivery, distributing incoming requests to multiple locations, including containers, EC2 instances, and IP addresses. By using ELB, the fluctuating load of your application traffic can be handled in both single and multiple Availability Zones.

Cloud Front: Cloud Front falls under the content delivery network domain that boosts the categorization process of Static and Dynamic Web Content for end-users.

Then in *AWS SECURITY*, we have IAM service and KMS.

AWS IAM: IAM is a security tool that allows its end-users to get and manage the Amazon web services and resources safely. With the help of IAM, creating and managing users, groups, and permissions to accept or reject the AWS resource access has become very easy.

KMS: This service creates and manages the user data encryption keys very easily. The HSMs are the Hardware Security Modules responsible for securing the customer master keys generated in KMS. AWS KMS uses a highly secure encryption method (AES 256-bit encryption) to access the user's data.

Finally, we have *Application Integration Services* which includes SQS and SNS.

***SQS*:** It is a completely managed queuing service that acts as a bridge of communication and connects isolated applications with the help of messaging and queues. It uses a queue, which is temporary storage for messages that need to be processed.

***SNS*:** The SNS is a fully managed and highly available pub/sub messaging service which allows you to decouple microservices-based architectures, serverless and distributed systems. When we say decoupling, we refer to

application integration which is like a family of AWS services connecting one service to another.

This was a quick overview of a few of the primary Amazon web services. We will be covering a lot more terminologies in the course ahead.

AWS provides 175 services that fall under various domains, but we only focus on the ones important for the Solutions Architect Associate exam preparation. These domains include Compute, Storage, Databases, Network & Content Delivery Services, Security, Management, & Monitoring Services, and Application Integration Services. Let us get started with the basic ones.

Chapter 2: Compute Fundamentals

Overview

AWS Compute Services are designed for capacity provisioning according to the user's demand. There is the problem of under-provision or over-provision the compute resources when using the cloud. AWS Compute Domain provides a solution by providing a set of computing services that are scalable, highly available, and customizable over the cloud to meet customer-specific needs. This section provides all the necessary elements and concepts of AWS Compute services, which can assist you in selecting the right service for your applications and projects.

Before we start, let us define the compute services. To support your business, you will need a compute capacity to run your applications. In a traditional environment, one needs to determine the required compute capacity first, then purchase the hardware to support that capacity, and finally, deploy all the servers to run the user applications. When you launch your application onto these servers, you need to maintain them from the software as well as physical perspective. On the other hand, if you build cloud-native applications, you can use compute as a service model, which allows you to provide and consume raw computing capacity over the internet with the pay-as-use model. It will eliminate the stress of installation and maintenance of these physical servers and allows you to use desired hardware and software to run over them. In addition to the maintenance and installation of computer infrastructure in a typical on-premises environment, when you miscalculate the initial capacity estimation, then there are two possibilities. In the case of limited resources, the user will face slow application performance, and the latency will result in user dissatisfaction that can affect your business. So, to overcome this issue, you will need to buy more servers and install, configure, and maintain those physical servers. While in the case of excessive resources, you will be paying for idle resources and bear the extra cost. Therefore, AWS enables you to provide the required capacity on user's demand and manage the entire process of installation, configuration along with the maintenance of virtual server for clients.

AWS Compute domain encompasses a variety of compute services. Amazon EC2 or Elastic Compute Cloud is the fundamental compute service of Amazon with customizable cloud computing capacity. It offers complete control over computing resources and enables deployment on the cloud environment of Amazon. AWS provides serverless solutions such as Lambda. The flexibility of AWS compute services allows you to execute all the applications in the cloud virtually. It provides container services as well, which enable us to use Docker via Elastic Container Service (ECS) or Kubernetes via EKS. Then there are managed compute options in AWS, such as Amazon Lightsail. You can use this service for the desired compute capacity without knowing the provision or management of the underlying hardware. Furthermore, there are a few options that extend beyond raw server capacity. In this section, you will see some commonly used fundamental compute services from an exam point of view. Regarding compute services, the Amazon EC2, Lambda, and Elastic Beanstalk will be covered in this chapter.

AWS EC2 (Elastic Cloud Compute)

Introduction

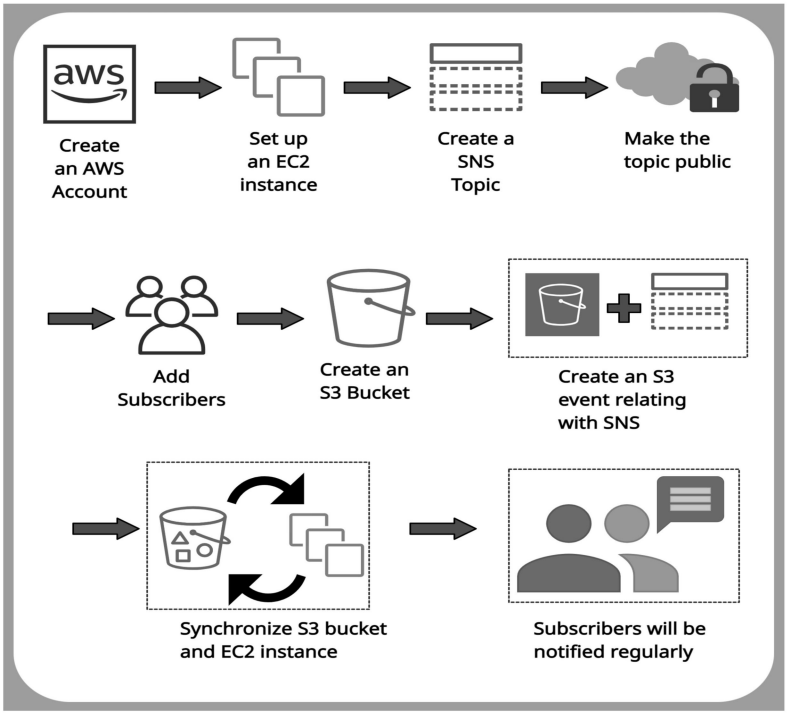
EC2, abbreviated as *Elastic Compute Cloud*, is the first and commonly used Amazon web service, which provides empty servers. It is used to launch a virtual machine and run the application on it. Based on the user's criteria, all types of virtual machines are available here. It is a web-based service that allows developers to deliver secure and resizable cloud computing capacity easily. It makes it easy to scale up or scale down the user’s infrastructure based on the requirement. For instance, let us say that the web traffic requirement that comes to your website keeps changing. Therefore, behind the scenes, EC2 can enlarge its workspace to several instances, and when there is no load, it can reduce the workspace to one resource. It follows a pay-as-you-go pricing model, and almost all Amazon web services get well integrated with this EC2 service. To configure new instances, EC2 uses Amazon Machine Images. AMI’s are the software and application packages that you need to run your

application. It provides the necessary information to configure an EC2 instance.

EC2-Use case

Let us discuss a use case in which a successful businessman has many users and a successful in-market product. Now, few more products are developed, which (he thinks) will be very successful in the future, and the customers will be satisfied with it. So, the businessman needs to figure out a practical marketing approach to get the customer's attention. This business use case can be accomplished using basic AWS services like EC2, SNS, S3 and then combining them all. Therefore, let us discuss how to get this environment started, essential measures to connect the environment, and make the applications start execution on it. In short, figure out the way to notify the users about a newsletter.

So, the team of a businessman can develop an architecture where whenever the company creates a newsletter, the group of users is informed about it. Here the components are AWS resources. Firstly, the architecture would need EC2 instance for computing, SNS for informing the users, and S3 for storage then, merge the EC2 and S3. That is all an infrastructure would require to inform a group of users about a new newsletter.



Let us discuss each step in detail now. So, first, his team needs to launch an EC2 instance. As discussed earlier, there are seven steps for launching an EC2 instance. By following these steps, they can successfully launch an EC2 instance. After that, they will move towards notifying the customers. *Simple Notification Service* is a service of Amazon which updates the customers through email. So, they need to search for SNS in the AWS account and create a *Topic* that will be used for public notifications. So, it will be made public, and the *Subscribe*r is also created. Now, these subscribers are the ones the team wants to send the newsletter information. Suppose that they already have an email database. So they can link this database with the subscriber list, and then new newsletters will be delivered to the subscribers when they publish the topic.

The next step is to create *a Bucket in S3*, where data will be stored. They will create an *Event* in that bucket, which will trigger and notify the *SNS*. In this way, the S3 bucket will be set up, and subscribers will get notifications if anything is added to the S3 bucket. S3 bucket will generate

a notification event that will ensure that the notification is delivered to the end-users because they already subscribed to the topic as subscribers. Finally, the S3 will be attached to EC2 so that the bucket and EC2 instance will be synchronized. Thus, when the user (businessman) puts some content in the S3 bucket, the email system will inform the customer, and the customer will be able to go to the website hosted in EC2. Since S3 and EC2 are synchronized, the items that a user puts in S3 will also appear in EC2. When everything is connected, the subscribers will be updated regularly whenever new content is placed in the S3 bucket. Moreover, the same content will be available in the EC2 instance over the website.

AWS Lambda

Introduction

AWS Lambda is one of the services offered by AWS that fall within the compute domain. This also belongs to the broad category of AWS compute services. With AWS Lambda, users can execute code for all application types and backend services virtually. They only need to provide the code in one of the AWS Lambda supporting languages.

Let us discuss Lambda in detail. It automatically executes your code without needing you to set up or manage service. You just need to write down the program and upload it, and then Lambda is responsible for executing that code. Thus, you neither have to provision and manage the services nor require any server to run them. By executing the code, Lambda is able to scale an application and answers every trigger. The program runs side by side, and each trigger is handled separately and scaled concisely to the size of the workload. Currently, this service supports *C#, Java, Node.js, Python,* and *Go*. It can execute, call, and answer to some events from other services, and it can execute the functions based on these events. These functions may belong to *Node.js, Java, C#,* etc.

In Lambda, billing is measured in seconds. You will only pay for the duration in which the code executes, which ensures that you do not receive any service charges. You only pay for the time the code is computed for every hundred milliseconds. You pay for the code when it runs, and a couple of times, the code is triggered, so you do not pay anything when it is not running.

How does it work?

Let us see the working of Lambda and how easily and smoothly the complex function works in the backend. Consider a use case in which the clients send data to Lambda. Anyone sending requests to AWS Lambda is a client. These clients may be an application or some other Amazon web service that sends data to Lambda, and Lambda receives the request. It runs on the specified number of containers determined by the size, amount, or volume of the data. If there are just one or a few requests, it will operate on a single container. To handle these requests, they are passed to the container, which executes the code provided by the user to satisfy those requests.

Containers are required to understand Lambda. So they are lightweight, self-executable packages in software that contain all necessary components, such as codes, run times, system libraries, and tools. They are currently running on both Linux and windows. The software will continue its execution without affected by the environment because the container separates the software from the surroundings. For instance, a developing and staging environment can differ, which is a kind of isolation that helps to reduce tension between the teams, which are running the various software on the same infrastructure.

Furthermore, if there are few requests, Lambda delivers them to a single container. However, when the number of requests grows, it will create several containers and share multiple requests to the different containers. Depending on the volume, size, and number of sessions, more containers are given to manage those requests. Whereas, when the number of requests reduces, the number of containers also reduces, which helps to save expenses. Whenever the users are not using any resources, they will

not pay for them. They do not pay for resources at all and are only charged when a function runs inside the container.

Cold Starts

Now one of the most important concepts of AWS Lambda is *Cold Starts***.** AWS has pre-configured servers which are lying in the switch-off mode for the user runtime environment. These servers need to be switched on when Lambda is invoked, and the code needs to be copied over. So, when the function initially runs, there will be a delay during this time, which refers to as Cold Start. Assume that you have a Lambda function that gets triggered, and to run this function, there is no server. So, the server will get started here, and you will have to copy the code due to which delay occurs. However, if you want to invoke that function again, the code is available, and the server is active so, you will not face the delay again. The cold start will not happen at this time, and now your servers are warm, which refers to as *warm server.*

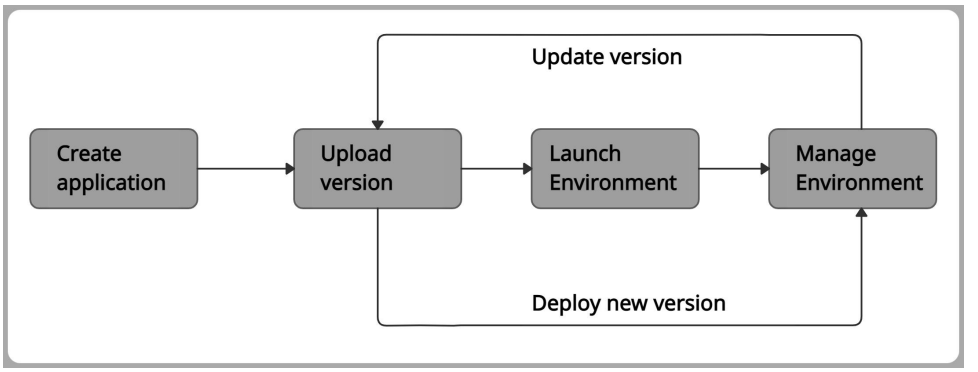
Cold Start is one of the drawbacks of using *serverless functions*. The serverless functions are inexpensive, but they also come with a tradeoff. A user may experience delays by using serverless functions cold starts. There are some strategies to overcome the cold start issue. One of them is called *Pre warming*, which allows the servers to run continuously. So you can invoke a function to start the servers prematurely, and as soon as someone starts using it, they will stay warm. The other approach for a cold start is that you can take a Lambda and assign more responsibilities to it so that more operations will process through it, making the *warm* more persistent. Thus, cloud providers are trying to find solutions to reduce these delays, making cold start less of an issue.

Elastic Beanstalk

Introduction

Let us now discuss *AWS Elastic Beanstalk*. It is used as a *Platform-As-A-Service* that allows coders to deploy and manage their applications easily without knowing the infrastructure that runs those applications. Elastic Beanstalk minimizes management complications without limiting the control or choice. One simply needs to upload the application, and it will manage the load balancing, health monitoring, capacity provisioning, and scaling of applications. Without Elastic Beanstalk, you may be able to run applications on AWS, but it does take time to pick and bundle services from a wide variety of AWS ecosystem options. Elastic Beanstalk allows you to extract the infrastructure layer to make it easy to install and manage your applications on AWS, but you must be aware of the underlying infrastructure.

Elastic Beanstalk workflow:



For using Elastic Beanstalk, you need to develop an application and then upload its version in application source bundle format (which includes Java .war file) to Elastic Beanstalk and describe a few details about the application. When you launch this application, its selected and supported platform version is deployed by Elastic Beanstalk. Then, the AWS resources that are necessary for your code execution will be created and configured. Once you have deployed the environment, you are able to handle the environment and launch new versions of that application. It

supports the applications built-in Java Node.js, Python, Go, .NET, Ruby, and PHP.

This service can set up the Auto-Scaling Group, database, EC2 instance, and Elastic Load Balancer pre-configured with your platform. You can also build a customized platform on Elastic Beanstalk and execute it. It can also run dockerized environments in the form of *Docker*s and *Multi-container Dockers*. The *Blue/Green* and *In-Place deployments* are a couple of deployment methods in Elastic Beanstalk. Now, it is in In-Place deployment by default, but blue/green deployment is also deployable. You can also get monitoring with the help of Elastic Beanstalk. It also contains interesting security features, such as it can rotate out passwords for you if an RDS is connected. It is not recommended by AWS for production applications. However, large enterprises and some startups are using Elastic Beanstalk. Amazon does not include any additional cost for using this service. You are only required to pay for those resources that are used to execute and store your applications.

Use Case Scenarios

We are going to discuss a couple of use cases for Elastic Beanstalk. Let us assume the first use case where the client demands a web application with no underlying infrastructure concerns. Thus, you can launch Elastic Beanstalk for such clients and offer multiple functionalities according to their needs that would normally require a lot of time for application configuration and maintenance. Your client will get an Elastic Beanstalk stack for a deployment process that has many AWS features in a single package.

- Log Rotation
- Load Balancer
- X-Ray
- Blue/ Green deployment
- Auto-scaling / auto-healing instances

- Security Groups
- Health checks/monitoring
- Networking / VPC Configuration
- Managed updates
- Integration with deployment tools such as Git and Visual Studio

Whenever a client demands a stable application deployed without any complication, then Elastic Beanstalk is the most suitable option in such cases.

For the second use case, let us assume that the client wants to manage a few infrastructure components, but he is not interested in learning the complete setup procedure of AWS. In such cases, Elastic Beanstalk allows the clients to manage some updates, such as managing deployment, the size of Auto Scaling Group, and switching over to the latest version without the need to access or train on all the other AWS systems. You can use this service to deploy and make changes from one place in multiple locations traditionally. For instance, a client can launch the latest version of the code, update the ASG min/max and instance size from a single screen.

Benefits

- It provides a complete stack of applications in few minutes.
- The developer may concentrate on the application with no infrastructure concerns.
- Dynamic and Automated scaling on the basis of usage/load.
- Automatic updates in the platform.
- You can integrate it with other AWS services.
- Accessibility control is role-based (IAM) across all AWS services.
- Audit logging and security of cross-service APIs via (CloudTrail),
- Maintain complete control of resources and frequent accessibility.

Chapter 3: Storage Fundamentals

Overview

Storage is one of the basic building blocks of *Infrastructure-as-a-service* (*IaaS*). AWS is undoubtedly leading the cloud computing market, which also includes the cloud storage market. In this module, the storage services offered by Amazon will be explained in-depth, including their key features and the reasons to use these services in your environment. This chapter presents an outline and introduction to the various AWS storage services and gives an insight into the process of data transfer from and to AWS along with an understanding of choosing the storage service that best serves your needs.

Amazon web services provide a variety of storage options, and five major storage options are available to the users. The first and most frequently used AWS cloud storage service is the *S3-Simple Storage Service*, which stores and collects data from the internet in any quantity. Then there is *Elastic Block Storage*, the Solid-State Drive (like C drive, D drive, or the E drive in your computer) connected to the EC2 instance. Then we have *Elastic File System* in the AWS storage options. The underlying technology of EFS and EBS is quite the same, but these services differ from each other in one way. EBS (D and E volumes) can only be accessed through the instances attached to it. On the other hand, the EFS is a shared file system, and multiple file systems can access it. EFS can be accessed from inside the Amazon environment as well as from on-premises.

We also have *AWS Storage Gateways*, which are used to shift the data from the user's local environment to the cloud environment securely and keep the local copy of data. So, the data can be accessed from on-premises as well as from the cloud. In the end, we have *Snowball* and *Snowmobile*. Snowball is a system for importing and exporting user's data. It is hardware sent to your on-premises where you can copy the data in it and then send it back to Amazon, and then Amazon will copy this data to the location which you have mentioned in your account.

Furthermore, if the size of the data is too big, then Snowmobile can be used, which is a data center on a Truck. Amazon would send a truck loaded with a data center in a container with high compute capacity, lots of storage space, and much more. So, when the truck arrives, it will be parked near the client's data center and is connected to it through cables. In this way, you can transfer the data into it and send it back to Amazon, where they would copy that data into your account and in any other storage that you notify them. So, if the data size is too big, you would call Snowmobile. However, it is not available in every region.

In the case of cloud storage, these popular services operate differently and deliver varying performance, availability, cost, and scalability levels. We will discuss these storage options with the help of use cases and compare their costs, accessibility to store data and performance. There is a common benefit of using these services that the users do not need to manage their servers independently, but they will be required to make a selection. This chapter explains these services in a simplified form, which will help select your desired storage service. Now let us move on and discuss the AWS storage in more detail.

AWS Simple Storage Service

Introduction

AWS S3 stands for *Simple Storage Service*, an *Object Storage Service* in which data is stored and recovered from the internet, but a user cannot install anything. There are two types of storage in AWS that are Object storage and block storage. In Object Storage, you can store and recover the data but cannot install anything. It can be accessed directly from the internet. On the other hand, the Block Storage needs to be connected to an EC2 instance as you cannot access it directly but can install the software in it.

AWS S3 is designed to store and collect data from the internet in any quantity. It is also accessible through the web interface. There are different

ways to access this storage service. One method is to drag and drop the content directly into S3, and the other way of recovering the data is to open a browser and click on the download button. In this way, you will be able to download any content with a size up to 5TB. You can download thousands of files like that, but a file can have a maximum of 5TBs. S3 is designed for developers where instead of storing the data locally in the server, they can push or retrieve the data from S3 anytime they want. You can also use S3 as a code repository to save your code so that the applications will read the code from that code repository. Furthermore, you can share the code with another user with full security and encryption using S3. Amazon also provides a service known as Import/Export to transfer huge data volumes that will allow you to upload and download the data in S3.

The Amazon S3 offers 99.999999999% *durability* and 99.99% *availability*. The term Durability means that the data gets lost if it is stored somewhere else, whereas it will not be lost with AWS due to its 99.999999999% durability. The other term, availability, means that data is available to users at any time they request it. In AWS, you can access your data with a 99.99% availability rate. So, whenever you request the data, it will be available to you without taking a lot of time. Amazon offers its S3 services in US East (N. Virginia), US West (Oregon), and Asia Pacific (Mumbai, Sydney & Tokyo).

Now let us move onto discussing the working and basic building blocks of AWS S3.

How does it work?

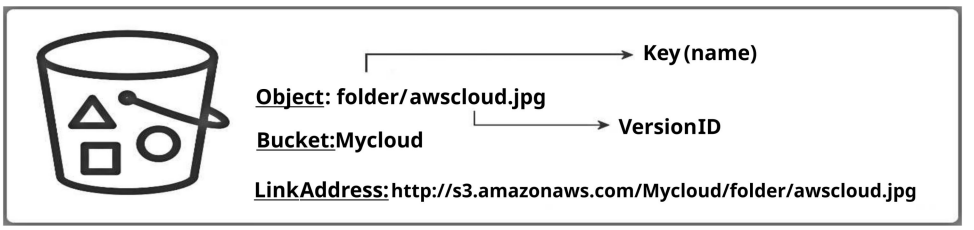
Let us take a look at the terms Buckets and objects in Amazon S3 before we learn about how it works. *Buckets and Objects* are known as the S3's basic building blocks. An Object is the original data with some additional information (a reference to the data) such as file type JPEG file or PNG file, name of the file, and the time it was added. This additional information is also called metadata. Therefore, we can say that an object is a combination of original data and metadata. A bucket is a container that receives the data and stores it securely. The name of the

bucket must be unique in all Amazon accounts because the name of the bucket is a piece of the URL. There is a concept of the folder in S3 with the management console, which refers to the grouping of several objects. You cannot create a bucket inside the other bucket, but it is possible to have a folder inside a bucket.

You can access the object via a public URL when it is uploaded. The Two URL types available in S3 to use are:

- bucketname.s3.amazonaws.com/objectname
- s3.amazon.aws.com/bucketname/objectname

The key refers to an object's distinctive identifier within the S3 bucket. You can have a separate key for every object in a bucket. These objects have an Access Control List with the help of which you can figure out that if you can share the object over the internet. A specific version ID is created and assigned to the object by S3 whenever you insert the data into a bucket to identify it in the future easily.



Let us discuss the working of the S3 bucket. Buckets are created in the S3 service to store files (based on objects), which act as folders. Whenever a bucket is created, the region must be specified to deploy that bucket. The expected latency, security policies, and service usage costs are generally kept in mind before selecting any region. You can store Objects in an S3 bucket. Amazon S3 objects are accessible and managed with the help of a web interface by default. All the objects that are uploaded to the S3 bucket are independent with respect to their characteristics and related permissions, meaning that who can and who cannot access the files. When

you upload a single file or set of files to the bucket, the S3 storage type must be specified for those specific objects.

Chapter 4: Database Fundamentals

Overview

AWS has a broad range of *Databases* that can satisfy the user data requirements. AWS databases will satisfy the demands, regardless of whether users need a relation, a time series, a graph, a document, in-memory, or a key-value database. All of these databases are able to handle applications that receive hundreds and thousands of requests per second. Amazon can automate every major database operation, including maintenance, upgrades, restoration, and backup. In this chapter, various types of database services are going to be discussed that are available in the AWS cloud platform. At the end of this chapter, you will be able to choose and identify the database that best suits a particular use case.

Before moving into the family of available AWS database services, the dissimilarities between a *relational* and a *non-relational database* will be addressed. After that, we will cover the broad topic of Amazon RDS in detail. It is a fully managed database service where users can choose among common database engines and allow AWS to handle their databases. Then we will discuss the two major Amazon SQL databases, such as Amazon Redshift and Amazon Aurora, along with their use cases in the context of a scenario. Then, we will look at the non-relational databases, which include DynamoDB, and Elasticache and the use case scenarios for using these types of database services. Let us start the chapter now.

Relational and non-relational databases (NoSQL) are the two groups of AWS database services. You will see an overview of these two groups of Amazon database services in this section.

AWS Relational Database Services

The data in *Relational Databases* is stored in tables in *rows* and *columns* where you can apply queries using *Structured Query Language*. Columns in such databases show the attributes, whereas the rows in the tables represent the records. Every field is a data value in the table. Amazon RDS, Redshift, and Amazon Aurora are the primary services that offer relational database services.

The use-cases for Amazon relational databases are:

- Transactions
- CRM applications
- Data warehousing
- Finance data
- Enterprise Resource Planning (ERP)apps

AWS NoSQL Database Services

The relational databases are not good for those use cases which require high speed or adaptive scalability in particular. *NoSQL databases* provide the non-tabular flexible database schemas to allow more efficient distribution and processing of data. These databases are generally required to deal with big data – a large volume of semi-structured and unstructured data. They are available in various forms on the basis of their data model. *Amazon Dynamo DB* and *ElastiCache* are the two types of managed NoSQL databases. Amazon Dynamo DB refers to a *Key-value Database* used to store the data as a series of key-value pairs with a key that serves as an ID. Different data types, such as plain and compound objects, can be stored in them. Customer preferences, e-commerce shopping carts, Real-time binding, and Product catalogs are the use cases of Dynamo DB. Based on your application, it is a managed non-relational database, which can be a more suitable solution than a traditional SQL-based database. Amazon ElastiCache, on the other side, is an *In-memory database type* in which the data is stored in the memory to ensure low-latency access.

These stores can be used as caches, message brokers, databases, or queues. The ElastiCache can be used for the following applications:

- Caching
- Session stores
- Real-time streaming
- Leaderboards
- Gaming
- Pub/sub messaging
- Geospatial services

The other services for AWS databases are *AWS DMS* for database transfers and migrations, *Amazon Neptune* for graph databases, and *MongoDB* for document databases. Let us proceed towards our first database now, which is known as Amazon RDS.

Amazon RDS

Introduction

RDS stands for Relational Database Service, a cloud computing solution in AWS to provide help in configuration, launch, and scaling of a relational database instance. RDS is the managed AWS solution for relational databases that supports several SQL engines**.** Amazon RDS manages many time-consuming database management tasks, including migration, patching and backup, and recovery, as a completely managed service. It also provides security and backup for the user databases. It can also backup RDS instances automatically, take a regular snapshot, and save the transaction logs for enabling recovery point-in-time. RDS also patches the software of the database engine automatically. It allows replication for high availability of production workloads and increased

reliability. The automatic failover in multiple AZ's can also be enabled with synchronous replication of data. You can control the Amazon RDS through AWS Management Console, AWS CLI, or its own APIs. So there are currently six relational database options for AWS. They are *MariaDB, Aurora, Oracle, PostgreSQL, Microsoft SQL, and MySQL.*

Being a database administrator, you can create, manage, delete, and launch an RDS instance. RDS instance is a cloud-based database environment that uses storage and compute resources. Based on the database engine, you can spin up several schemas or databases. Each customer can use a total of 40 Amazon RDS instances per account. In the case of SQL Server and Oracle instances, you can have only ten of each.

Features of Amazon RDS

Availability

You need to access the hosted data in the cloud at the time or the location of your choice. With the help of the Multi-AZ feature, RDS can provide high availability to maintain the redundant copy of your data in a different location. This Multi-AZ service level agreement will guarantee the database uptime of at least 99.95 percent a month.

Scalability

Scaling a self-hosted in-house database can become very challenging, but Amazon RDS has made it very simple. It provides two types of automated scaling: Horizontal scaling (introducing more machines into your infrastructure) and Vertical Scaling (introducing more power to your existing machines). In case the database gets overburdened with the requests, this service provides a load balancer that distributes the requests equally.

Performance

With the help of the Performance Insight dashboard in Amazon RDS, you can easily troubleshoot and analyze relational databases' efficiency.

Encryption

All RDS engines can be encrypted at rest. By turning on the encryption, automatic backups, snapshots, and read replicas associated with the database will also be encrypted. AWS KMS is used for encryption where the default key or KMS key is used to turn it on. However, enabling encryption in older versions of certain engines is not possible yet.

RDS Backups

Let us look at the *Backups* of *RDS*. There are two solutions to perform RDS backups in AWS. They are automated backups and manual snapshots. In *Automated Backups*, the backup value can be set to 7 to enable it and set it to 0 to disable it. The retention period of 1 to 35 days must be selected in automated backups. The transaction logs are stored here the whole day, and all the data will be stored in S3. No extra cost will be charged for these backups. You can define your required backup through a backup window. In a backup window, you have to specify the starting time, UTC, and then the storage. The time cannot exceed half an hour (0.5). During backups, the IO and the storage may be blocked so that you may face some problems during this time. So you might want to choose that time carefully.

On the other hand, the *Manual Snapshot* creation is a manual process in which you can drop down actions and take a snapshot. You can still have these snapshots even if the primary database or the RDS instance is deleted. So if you leave the RDS system, they do not disappear. It is also possible for you to restore the previous version of the snapshot.

Restoring Backups

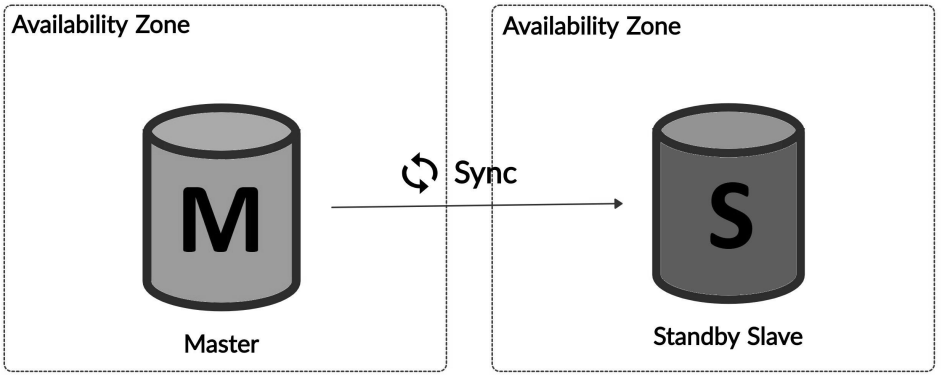
Let us now understand the method to *restore* a *backup*. It is the simple step of selecting restore to a single *point in time* from the drop-down actions menu. AWS then takes the recent daily backup while restoring it and applies the transaction log data related to that day. As a result, the point in time recovery will be reduced to a second during the retention period. Data backups are never restored over an existing EC2 resource. Once an automatic backup or manual snapshot is restored, a new instance must be created for the recovered database. So, a new DNS endpoint is

required when you create a new instance. You have to do it manually because you need to uninstall the previous instance and use this new endpoint for your applications.

Multi-AZ deployment

Let us now discuss the *Multi-AZ deployment*. It guarantees the availability of your database if another AZ is not available. This allows users to copy the database in another AZ, and AWS directly synchronizes all changes from the master DB to the standby DB automatically. The master DB is the primary database, and standby is a slave database that does not receive real-time traffic. In the case where AZ goes down, standby is simply a backup to replace the master database.

Assume that you have enabled automatic failover protection, and it will occur when our AZ goes down. Thus, it will point to the slave database, and the slave will be upgraded to the master database. Therefore, this database is now the master database.



Read Replicas

Read Replicas can be used to run several copies of your database. The write operations can not be performed to these copies, but you can read them. Read replicas are designed to reduce the primary database workload in order to maximize performance. You must enable automatic backups in order to use read replicas. So creating a read replica is very simple; you drop down actions and click "create read replica'. The process of

replication between read replicas and the master database is asynchronous. It allows you to have five replicas of your single database. Each read replica has a specific DNS endpoint. There is no automatic failover for read replicas, so if the primary replica fails, you need to change URLs manually to point at copy.

You can have the *Multi-AZ Replicas* and *Cross-Region Replicas.* You can even create replicas of replicas. You can promote read replicas to your master database, but this will break the replication. The read replica acts as a database instance, allowing only read links so that applications can interact with these replicas, just like any DB instance connection. Once the replica becomes its own master, it does not receive updates from its old master. Promoting one read replica does not affect the relationship between its old master and other read replicas.

Comparison between Multi-AZ & Read Replicas
Now let us differentiate the Amazon RDS Multi-AZ deployment and Read-replica.

- For replication, Multi-AZ has synchronous replication, and Read-replicas have asynchronous replication.

- The primary instance database is going to be active. The primary instance will be the instance currently active on Multi-AZ. Therefore the standby does nothing. It just becomes the primary instance in case the primary instance goes down, which uses all the replicas, read-replicas, and primary instances.

- You can have automatic backups taken from the standby database for multi-AZ, whereas no backups are enabled by default for read replicas.

- Multi-AZ is followed by two AZs within a single region. Read-replicas are available in one AZ but can be multi-AZ, Cross-AZ, or Cross-Region**.**

- When upgrades occur in multi-AZ deployment, they will be in a primary database. On the other hand, when upgrades occur in read replicas, they will be independent of the source instance.

- The automatic failover can occur on standby in multi-AZ deployment. DB instance Failover here is completely automatic and does not require administrative action. You do not have this automated failover for reading replicas. Each of these replicas must be manually promoted to become the standalone database instance.

Amazon Aurora

Introduction

Aurora is a completely maintained Postgres or MySQL compatible database service. It is designed to scale according to the workload and is very fast by default. High-end database availability and speed are combined with the open-source database's convenience and cost flexibility in Aurora. It can run on engines that are compatible with MySQL or Postgres. The advantage of using Aurora over a standard RDS Postgres or MySQL engine is that it is fine-tuned to deliver the best performance. If you use MySQL, Aurora is five times faster than the traditional MySQL, and its Postgres version is three times more efficient than the traditional Postgres. The cost is one of the biggest advantages, as it is a tenth of the price of other database solutions that provide similar performance and availability.

Aurora is a corporate-level database service that can scale up to 64 TBs in terms of performance and availability. It has a special feature called *Multi-AZ deployment* that allows it to replicate data across different availability zones. Depending on the use cases, you can select from various types of hardware specifications for your instances. *Serverless mode* is another feature of Aurora which provides a complete on-demand experience in which the database scales down automatically for the lower loads and

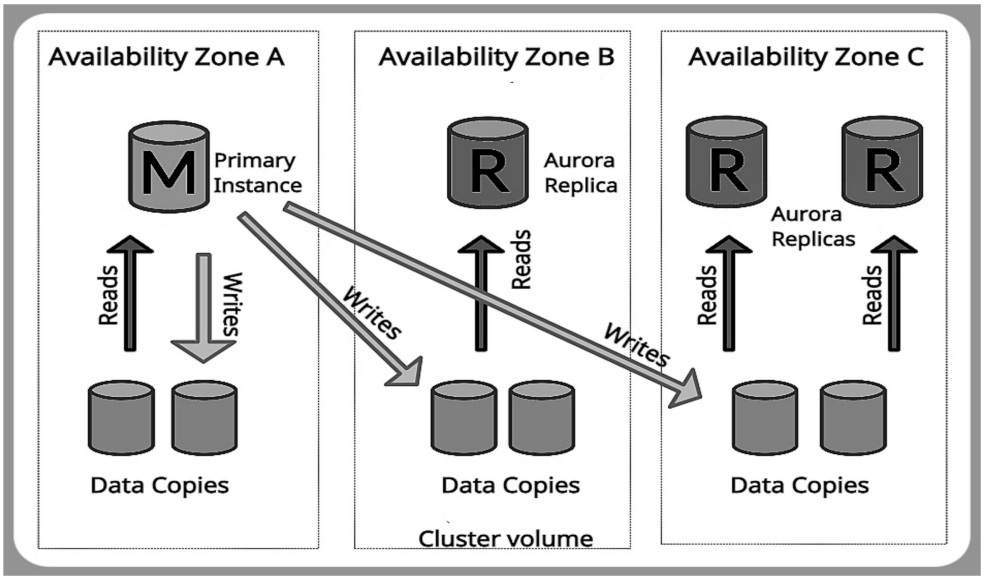
scale-up for the higher loads. In this mode, you are only charged for the time in which the database is operational. However, a small delay will be there in responding to requests when the database is fully scaled down.

Aurora architecture's working depends on a volume cluster that handles all DB instance data in that specific cluster. A volume cluster is essentially virtual DB storage that is shared across several AZ's. The underlying storage volume is located on top of several cluster nodes distributed around various AZ. Apart from this, the Aurora database may also provide Read Replicas. Generally, only a single instance works as a primary instance that supports read and write operations. In contrast, the remaining instances act as read-replicas, and the user is responsible for load balancing. It differs from the Multi-AZ deployment, in which the instances are located in the AZs and support automated failovers.

Features of Aurora

Availability

Aurora is a highly available database service because it replicates your data six times across three AZ’s, each with two copies. If you lose two copies of your data, it will not affect the write availability. If you lose three copies of your data, it will not affect read availability**.** To avoid storage failure and for safety purposes, AWS Aurora provides the data backup on a constant rhythm.



Higher Security

Aurora provides the database with several layers of security to make it better than the other services. Data in the underlying storage is encrypted on an encrypted Amazon Aurora instance. The management is done via the AWS KMS, and SSL is used to encrypt the data in transit. In addition, replicas, automatic backups, and snapshots are available in the same cluster.

High Scalability and Performance

Aurora Scaling is a managed feature that starts with 10GBs of storage initially and can scale up to 64TBs with increments of 10GBs. Auto-scaling is enabled as far as storage is concerned, which automatically scales up or scales down the storage. The computing resources can scale up to *32 vCPUs* with 244GBs of memory for its computing power.

Aurora allows both horizontal and vertical scaling. You can perform vertical scaling by upgrading the instance types, and there is a minimum downtime associated with it in the case of Multi-AZ deployment. If this is not possible, then you can schedule the scaling during the database maintenance time. On the other hand, horizontal scaling is performed through the read replicas where the Aurora database can have a maximum

of 15 read replicas simultaneously. There is a difference between storage scaling and Aurora compute scaling, and the one mentioned above is compute scaling. The storage scaling in Aurora is performed by modifying the maximum allotted storage space or hardware storage type such as HDD or SSD.

Aurora offers five times the throughput of the standard MySQL database. Such performance is good for enterprise databases at the rate of 1/10th price. As your needs change, you can simply scale your database preparation up to and down from smaller to larger instance options.

MySQL and PostgreSQL Compatibility

Aurora engine is completely compatible with the PostgreSQL and MySQL databases. Compatibility for the latest versions of the database is introduced on a regular basis, and you can also move the data from PostgreSQL/MySQL to Aurora using simple import/export tools. Your current applications that use PostgreSQL or MySQL can also be used with Aurora without requiring any code modifications.

Fault Tolerance & Durability

Aurora automatically manages backups and failovers when it comes to *Fault Tolerance* and *Durability*. Snapshots can be shared whenever data has to be exchanged with another AWS account. Aurora also comes with self-recovering storage, so data blocks and disks are regularly checked and repaired automatically.

Replicas

There are two types of replicas available in Aurora, including *Amazon Aurora replicas* and *MySQL Read Replicas.* You can have five read replicas for MySQL, and there is a high-performance impact on the primary database. It does not have automatic failover, yet it supports user-defined replication delay and various data/schema against the primary. On the other hand, you can have 15 *Aurora replicas,* and there is a low-performance impact on the primary database. It supports automatic

failover but does not support any user-defined replication delay or different data/schema.

Moreover, the Aurora database can span into multiple regions via *Aurora Global Database***.** Thus, you need to select the one that is more suitable for your enterprise. From an exam point of view, one must know the two different Aurora replicas types.

Aurora Serverless

Aurora is very expensive if it is not used for high production applications. However, if using Aurora is still a requirement, Amazon has a feature known as *Aurora Serverless.* It is just another Aurora mode that only runs when the user needs it to run and can scale up and down, depending on the application's requirement. The capacity settings are required when the serverless option is set in the database features, and the capacity for *Aurora Capacity Units* (*ACU*) can be set to a minimum and maximum. It lies between 2 and 384 ACUs, which only charge you when it is consumed. Aurora Serverless is used for low-volume blog sites such as for a chatbot or to build an MVP that is out to clients. It is not frequently used, but you can plan to use Aurora in upcoming times.

Redshift

Introduction

Amazon Redshift is a completely managed data warehouse of *Petabyte-size*, which is used to analyze the huge data size using complex SQL queries. It is necessary to understand the term data warehouse to comprehend the process of Redshift because Amazon Redshift is a columnar store database.

Let us compare the term data warehouse with the database to understand it better. Therefore, you need to have some basic knowledge and understanding of database transactions. It is defined as: 'A transaction represents a unit of work done within a database management system.**'** For

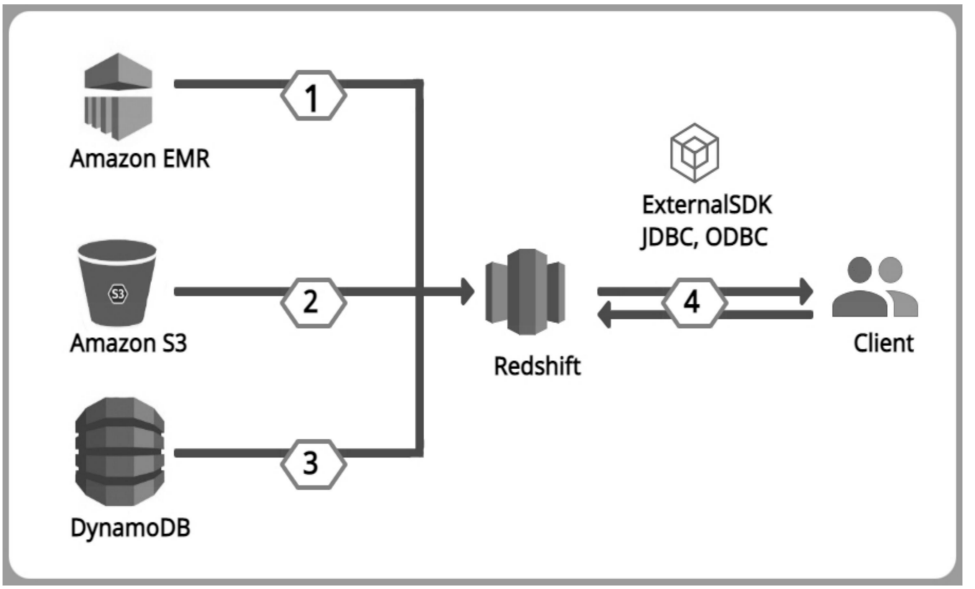
instance, reads and writes are an example of a transaction. Transactions are treated differently for databases and data warehouses. *OLTP* is an online transaction processing system for a database in which transaction is short, which means small and simple queries that focus on writes. In OLTP, you can create a database for storing existing transactions and enabling fast access to some transactions for ongoing businesses.

Consider a web application that needs to be very responsive regarding reads and writes for the current users. Read and write operations may include adding an item to a shopping list, signing up, or any operation in the web application. Generally, these operations are backed by a single database, Postgres, or might be running on RDS. The Data warehouse runs on an Online Analytical Processing System (*OLAP*). OLAP involves lengthy transactions, such as long and complex SQL queries that focus on read operations. Therefore, a data warehouse is built to store a huge amount of historical data that enables fast and complicated queries across all data. It is used for generating reports and Business intelligence tools. A data warehouse is not a single source because it takes data from multiple sources. The data is combined from DynamoDB, EMR, S3, and Postgres in one place to run complex queries.

You will prefer Redshift because the pricing starts at 25 cents per hour, with no capital costs or commitments. It scales up to petabytes of data for $1000 per terabyte a year. Redshift price is less than 1/10th the cost of most similar services. It is used for business intelligence, and it uses OLAP. It is a columnar storage database that's crucial for improving the performance of analytic queries. The size of the data to load from the disk and the overall disk IO needs instantly reduces. Redshift is fast due to columnar storage. Let us discuss it in more detail in the following use case.

Use Case

Let us discuss a use case to understand the process of Redshift. Consider that you want to build a business intelligence tool. You have many different sources, and the data is coming from EMR, S3, and DynamoDB, which you will copy into Redshift by using the copy command.



You can normally interact and access Redshift data using AWS SDK as most of the services do. However, in this case, you do not use any AWS SDK because you may need to make a generic SQL connection to Redshift. Moreover, if you are using Redshift, you must be using java. Therefore, you would use *JDBC* or *ODBC*, third-party libraries, to connect and query Redshift data. As mentioned earlier, columnar storage is very important to Redshift performance. So, let us understand its concept here. *Reading via the rows* is normally used with a database, whereas in an OLAP, it is *reading via the columns* because it is better to look at columns rather than looking at huge data and breaking it. OLAP applications look at multiple records at the same time. You can save memory because you only fetch the columns of data you need instead of whole rows. If you read columns, the data can be stored as the same data type, which allows easy compression. It means that you can load the data more quickly. Furthermore, you always look at large volumes of data simultaneously, so only columns needed in bulk can be retrieved. Thus, it gives much faster performance for a use case, such as the business intelligence tool.

DynamoDB

Introduction

Let us look at *DynamoDB*, which is a *key/value* in the document database. It is a *NoSQL database* service used by applications requiring stable latency (in milliseconds) at any scale. It can guarantee consistent reads and writes at any scale. The NoSQL database is a type of database that is neither relational nor uses SQL for querying the data. This database service uses key-value store, and document store.

A key/value store is based on the key-value model in which you have a key and a value, and they can only search by the key, whereas the document store is a document model where your data is more structured than KV models. The whole data is a single value in the database. Thus, the DynamoDB is a NoSQL key/value in the document database with many features for internet-scale applications. This database service is multi-feature, multi-region, durable, and multi-master with built-in protection, memory caching, recovery, and restoration.

DynamoDB provides developers with minimum configuration and management while offering high efficiency and scalability. It serves as a key-value-store database for NoSQL, which reflects that the data (JSON objects) is not stored in a relational or structured mapping, but it is stored in the format of simple key-value. In order to provide high data durability and availability, the data in DynamoDB is stored on SSD's and replicated synchronously in several AZ's in a region. Moreover, DynamoDB is a user's choice because it satisfies all the user requirements. For example, if you need a hundred reads per second or a hundred writes per second, you will surely get it using DynamoDB. It depends on the payment, so scaling this database is not an issue. Regarding durability, DynamoDB stores its data across three regions, and you can have fast reads and writes as it uses SSD drives.

How does it work?

DynamoDB stores the data in tabular form, just like other databases. There is a set of Items in each table, and there are many Attributes in each item. The DynamoDB table must contain a primary key in every table item, which is a single attribute or two of them, such as sort-key and partition-key. Using the primary key or making your indexes and using keys from those indexes, you can reference certain items in a table.

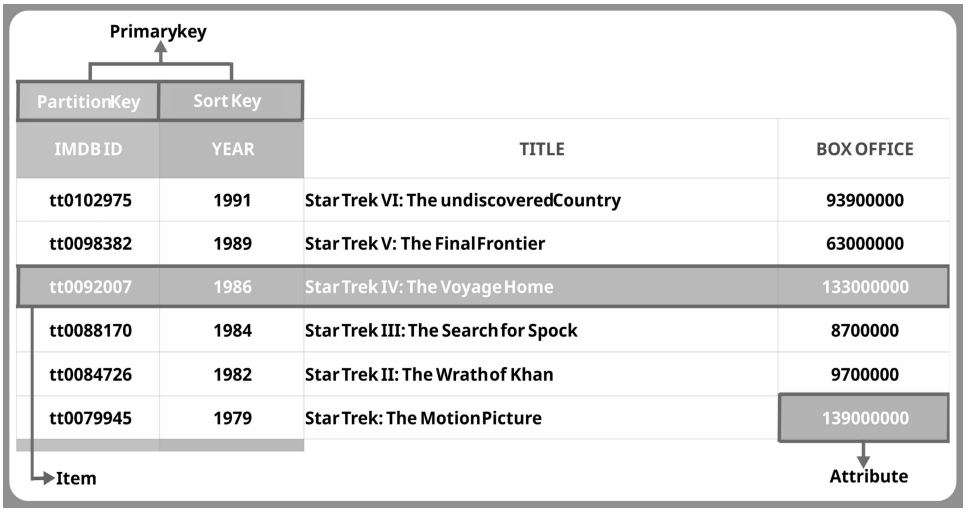
In the DynamoDB table, your data is shared over several devices instead of having a single hosting server, so you are guaranteed high performance and scalability. It also means that you cannot link directly to a database host and query the data. The DynamoDB HTTP API is used to write and read items from and to a DynamoDB table directly or via AWS SDK/CLI. You can also batch reads and writes to DynamoDB tables, even through various tables at once. It supports cross-region replication, transaction, and automatic backups.

Table Structure

It is important to understand the *Table Structure* of DynamoDB because it does not use the same terminologies as a relational database. In DynamoDB, there are three basic data model units, *Tables, Items,* and *Attributes*. The term *Item* is used instead of a *Role* and Attribute instead of a *Column/Cell*. Then we have the *primary key*, which is made up of two keys:

- Partition Key
- Sort Key

These table elements of DynamoDB can be shown in the following diagram.


Primarykey
PartitionKey
Sort Key
IMDB ID
YEAR
TITLE
BOX OFFICE
tt0102975
1991
Star Trek VI: The undiscovered Country
93900000
tt0098382
1989
Star Trek V: The Final Frontier
63000000
tt0092007
1986
Star Trek IV: The Voyage Home
133000000
tt0088170
1984
Star Trek III: The Search for Spock
8700000
tt0084726
1982
Star Trek II: The Wrath of Khan
9700000
tt0079945
1979
Star Trek: The Motion Picture
139000000
Item
Attribute

Chapter 5: Management & Monitoring services

Overview

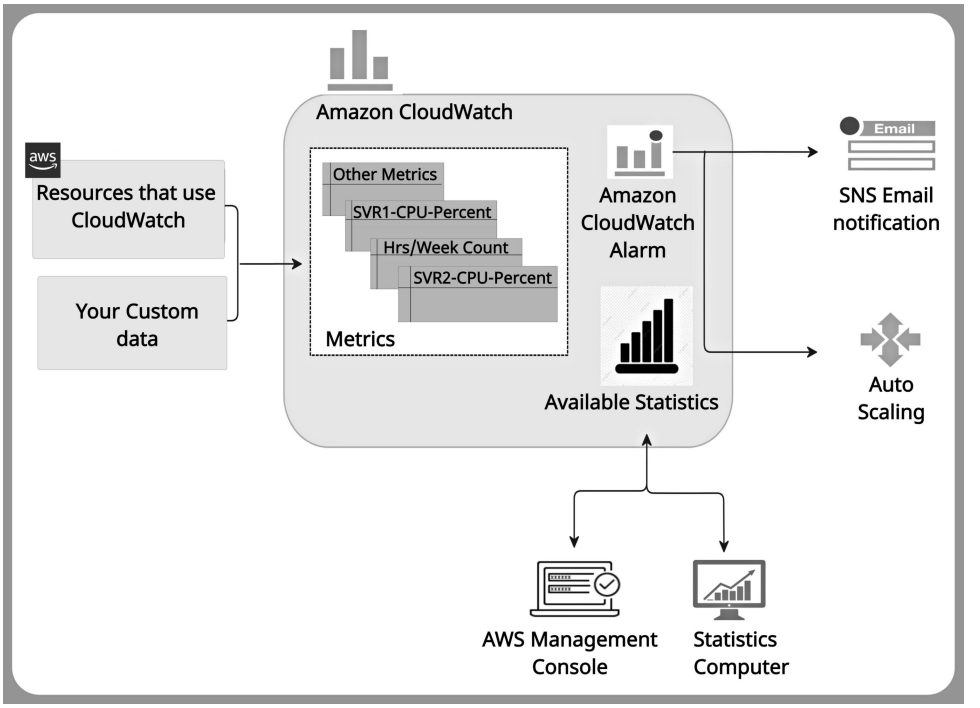
AWS Cloud Monitoring Services use various automated and manual tools. These tools are used to analyze, manage, and observe the cloud computing architecture, cloud framework, and Amazon web services. It integrates the complete cloud management strategy, which allows cloud administrators to monitor cloud resource status. The three basic Amazon Management and Monitoring services will be our focus for this chapter. These services include CloudWatch, Cloud Trail, and Cloud Formation. Let us discuss CloudWatch in detail first.

CloudWatch

Introduction

Amazon CloudWatch enables its users to monitor different applications and cloud resources running on Amazon cloud infrastructure. It enables you to observe EC2 instance metrics and track them, such as EBS volumes and RDS DB instances. Besides, it can perform different functions, such as setting up alarms, storing log files, viewing graphs and statistics. Moreover, it can also be used to monitor changes in AWS resources and respond to them. Amazon CloudWatch provides an overview of the system's health and performance, which is eventually used to enhance an application's functionality. An advantage of this monitoring solution is that you do not need to install any additional software. Let us observe the working of Amazon CloudWatch.

The following figure reflects the concept of monitoring in AWS CloudWatch.



For AWS applications and resources, CloudWatch provides full platform visibility. It monitors the files of resources and creates key metrics depending on the log files of the applications. Disk storage, processor latency, processor usage, and network traffic are all key metrics. CloudWatch displays the actual summary of every resource along with system activities, depending on the metrics. CloudWatch also offers insight into the AWS architecture, allowing it to monitor the performance of applications, notice trends, and fix operational issues. Let us suppose that the AWS environment undergoes unpredictable operational changes. So, in that case, it sets up alarms and sends proper notifications.

CloudWatch is a monitoring service that allows its users to respond, log, and observe the log data. Therefore, you have to understand that CloudWatch includes multiple services under one title. It is not a single service. These services include CloudWatch dashboards, CloudWatch alarms, CloudWatch metrics, CloudWatch Events, and CloudWatch logs.

As we have gone through Amazon CloudWatch concepts and its operations, let us discuss some of its services now.

CloudWatch Dashboards

CloudWatch Dashboards offer insight into the use of resources, operational health, and an application's performance. Therefore, customized dashboards are created to display multiple metrics and access images and texts. One can create multiple dashboards along with global dashboards that are used to fetch data from multiple regions. The AWS CLI or Management console is used to create them and then incorporate them into *Infrastructure as a Code* tool such as AWS CloudFormation.

Depending on the customer's preference, texts and graph widgets are introduced to the dashboard to display the metrics. For example, a graph widget is created to reveal CPU utilization for specific EC2 instances. Both new and existing customers can create three dashboards using up to fifty metrics without any extra cost. If it exceeds the limit, you need to pay for each additional dashboard monthly.

CloudWatch Logs

Using *CloudWatch Logs* help in accessing, monitoring, and storing AWS resources' log files such as Route 53, EC2 instances, Amazon CloudTrail, and few others. The three basic concepts of CloudWatch Logs are discussed below.

Log Events: It is an activity record registered by the applications or the monitored resource.

Log Streams: It shows the series of log events from the instance of an application. Thus, log streams are the series of log events with the same resource.

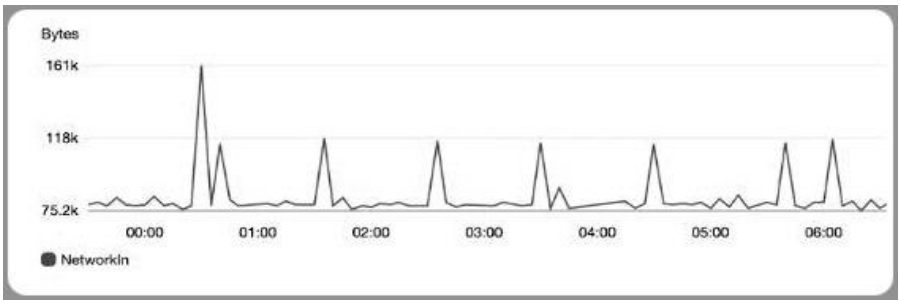
Log Groups: The log group comprises a set of log streams that share similar monitoring settings, access control, and retention.

Moreover, you can use CloudWatch logs to try and fix machine bugs. The log files are maintained and stored automatically. One can set the alarm so that it sends a notification if there is a bug in the system log. CloudWatch

log stores the original data. Therefore, by viewing the original files, you can remove the errors in minutes.

CloudWatch Metrics

Let us discuss *CloudWatch Metrics* that is built on top of CloudWatch logs. CloudWatch Metrics represents a sequenced set of data points. It is also considered a variable for monitoring. Therefore, you need to extract the data from logs as data points and graph it. Consider a use case in which network traffic '*NetworkIN'* reaches an EC2 instance. You select that specific metric and observe it by seeing it.



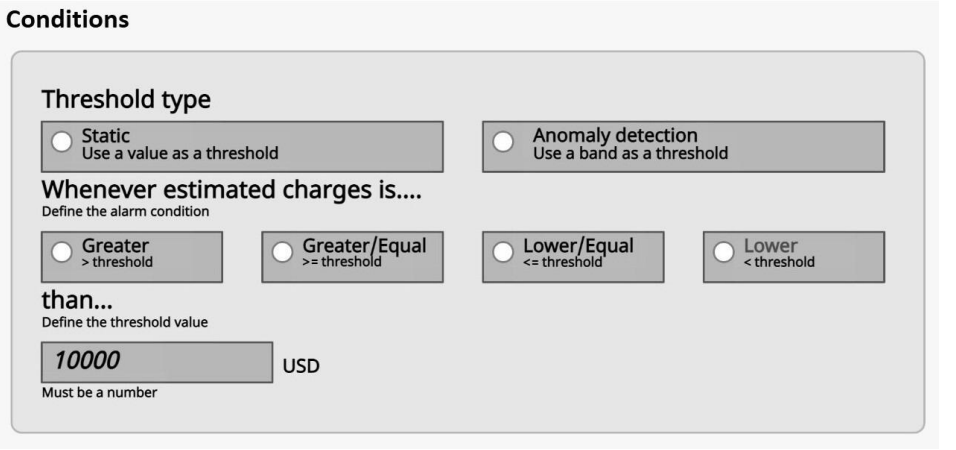
The above figure shows the CloudWatch metrics. These metrics are pre-defined, so you do not need to do anything. To support this, you must enable logs on some services so that when the data arrives, these metrics are available.

CloudWatch Alarms

CloudWatch Alarms send a notification to the users based on metrics when the defined threshold is exceeded. The most common use case is Billy alarms. It is one of the very first steps while setting up an AWS account. There are different options while setting the alarm. In the conditions tab, you need to mention if it is static or anomaly detection.

Furthermore, you need to mention the time to trigger the metrics, such as when it is greater than, equal to, or less than the threshold and the amount of limit. Let us say that you are watching for a thousand dollars, and you mentioned that it is fine while it is under a thousand dollars. However,

send me an email if it exceeds a thousand dollars. That is the utility of CloudWatch Alarms.



Availability

It is important to understand how often CloudWatch updates the available metrics to you as availability varies for different services. When you use an EC2 instance, CloudWatch monitors at an interval of five minutes by default. If you want to get down to one minute, you have to turn on detailed monitoring that costs money. It will be between a minute and five minutes for all other services. There may be few other services with detailed monitoring, but EC2 is not one of them by default. The majority of services are by default one minute, whereas the EC2 is five minutes. You can get one minute by turning on detailed monitoring for EC2.

CloudWatch Use Cases

Infrastructure Monitoring and Troubleshooting
CloudWatch monitors the key metrics and logs, generates alarms, checks applications, and compares the logs and metrics to sort out the operational challenges in AWS resources. It includes monitoring the user container environment through Kubernetes, AWS Fargate, Amazon EKS, and ECS.

Mean-Time-To-Resolution Development
The CloudWatch helps in comparing, evaluating, and visualizing logs and metrics, allowing you with instant response time to figure out the problems and incorporate them with the trace information from AWS X-Ray for end-to-end observation. The user's request can also be determined to accelerate the problem solving (debugging) and decrease the total mean-time-to-resolution or MTTR.

Proactive Resource Optimization
When you determine the metric values, CloudWatch alarms will evaluate those metric values against the threshold, and CloudWatch monitors the abnormal behavior of resources by using machine learning designs. When generated, an alarm can automatically select an action to perform EC2 Auto Scaling or stop an instance for presentation. It means that the resource planning and compute capacity are automatically available to you.

Application Monitoring
The CloudWatch will monitor the applications that work on-premises or on AWS architecture. The data is collected from all the layers of the performance stack, including metrics and logs on automated dashboards.

Log Analytics
The CloudWatch will analyze, visualize, and investigate the logs to improve the application performance by reporting operational issues. Inquiries can also be carried out to help you in an efficient and immediate response to the functional problems. Whenever you come across a problem, you can initiate querying by using the purpose-built query language for instant diagnosis of the root cause.

CloudTrail

Introduction

Cloud trail is used for monitoring API calls between AWS services and actions made on the AWS account. It helps in identifying the individual responsible for actions. AWS account holders can use CloudTrail to

record and write each API call made to AWS resources in their account. The API call is made when:

- You make a REST API call to an AWS resource.
- You run an AWS CLI command.
- You access a resource from the AWS console.

Such activities may come from another AWS service, an application, or human users such as:

- When the Lambda function writes to an S3 bucket (Another AWS service)
- When an AWS CLI command is called by a bash script (An application)
- When you spin an EC2 instance from the console (Human users)

CloudTrail preserves the API events in an immutable and secure manner which you can use to analyze later.

Let us consider a record to understand the working of CloudTrail in simple words. This record can identify the person responsible when something goes wrong by providing five W's, i.e., *what, where, when, why,* and *who.* The ‘where’ part provides the account ID's details, such as the account where it happened and the person's IP address who created the request. The ‘when’ part tells the time at which it happened. The ‘who’ part tells about the user agent, which is the operating system, the language, the process of making the API call, or the user itself. Finally, the ‘what’ part provides information about the service and the region from where the request is generated. So, CloudTrail works in this way. This level of visibility can be helpful for proactive account vulnerability management, post-breach investigations, and ensuring compliance standards are met.

From an examination point of view, it is important to remember that when keywords such as governance, compliance, operational audit, or risk audit are mentioned in the exam, it is probably referring to AWS CloudTrail.

Features of CloudTrail

This governance and monitoring tool has various features such as

- Multi-region configuration
- CloudTrail Insights, Data events, and management events.
- Validation and encryption of Logfile integrity
- Event History to allow you to observe changes.
- CloudTrail is 'Always On,' allowing the data visibility for the last 90 days.

Let us discuss some features in detail.

Event History

CloudTrail monitoring logs already exist in the AWS account, which collects records for the last 90 days under the *Event History*. There is an interface here from which the events can be filtered out. Now, if you need logging beyond 90 days, which is a very common use case, you have to create a custom trail. The only drawback is that it does not have a GUI like the event history, due to which some manual work is required to visualize the information. A very common method is to use Amazon Athena. Thus, CloudTrail enables you to search the CloudTrail logs with the help of Amazon Athena.

CloudTrail to CloudWatch

CloudTrail can also deliver its events to cloud watch. It helps in sending its insights of data, events, and data management to CloudWatch Logs. When a CloudTrail is being created, you can set up and forward events to CloudWatch logs, but only those events are sent that fit your trail setting.

Use Cases

Financial Issue Troubleshooting
Operational challenges can be solved by using the call history of AWS API developed by CloudTrail. You can observe the recent changes to the resources, such as adding, removing, or changing AWS resources in your environment.

Compliance Aid
In order to verify that your environment is compliant at any specified time, CloudTrail checks the history of every activity. Its event log files can be used by the IT auditors as a compliance aid. AWS resource modification log, log integrity verification and log analysis for unauthorized access are all part of a compliance audit activity's workflow.

Data Exfiltration
Data exfiltration can be detected using CloudTrail recorded object-level API events to collect activity data on S3 objects. When activity data has been collected, you can use additional resources (like Lambda or CloudWatch events) to provoke the response.

Abnormal Activity Detection
If you enable the CloudTrail insights, you can easily detect the abnormal activities in your AWS account. You immediately become alert and act upon the functional issues such as any service reaching its limit or faulty spikes in provisioning resources.

Analyzing Security
The integration of CloudTrail events with your log analytics and management solutions will identify client behavior patterns and run security analysis.

CloudFormation

Introduction

Let us discuss the concept of *CloudFormation* in detail. *CloudFormation* is a service that offers a convenient way for users to build and manage AWS resources in a very organized and predictable manner by creating and updating them. It is a service that lets you set up the AWS resources to concentrate more on the running application instead of wasting time on managing resources. You have to form a template for any EC2 or RDS instance and then upload it to the CloudFormation portal. Then, CloudFormation will build and manage the resources using a user-defined template. There is no need to worry about figuring out the dependencies and complexities between the applications. When a template is checked and verified, cloud formation manages all the applications' dependencies in an organized and consistent manner. When the template properly runs, it creates and makes those resources available for use. In short, it designs and creates the infrastructure and applications without performing manual operations. Take an example of a company, 'Expedia' that entirely manages and executes its front-end and back-end resources very easily running on AWS CloudFront. They spend less time on infrastructure management and more time on core business and applications.

How does it work?

With Cloud Formation, you can manage your complete infrastructure or AWS resources in a text file. This file is known as the *Template*, which is either in a JSON or a YAML format. The collection of resources that a template provides is known as a *Stack*. The stack will run the resources, which can be updated as well. Stacks are not limited, which means that the stack can be updated without creating another resource. Consider a use case in which you want to add two more servers to connect to other applications. As you are expanding the environment by introducing a new feature in your system, you can easily merge it into the stack and update it, clarifying that the stacks are updatable.

Now, let us discuss the functions and features of a template. With templates, you can add almost all applications and the resources that you might need. One can even make a template by provisioning EC2 instance and applications running on top of it. Moreover, these templates are portable, which means users can use them in multiple regions. If you use the same template in two regions, it can build a similar environment in both regions. It can also be shared with customers, and when the customer executes it, a portable template environment will be executed in his system from his AWS account with security guaranteed. You can build a resource, architecture, or an environment similar to other resources and the environment with the help of templates in CloudFormation. These templates can be reused by using some sections of the template. By reusing the parameter, mapping, and condition sections within the template, you can make a template reusable.

Now, let us understand how a template becomes infrastructure. Let us say you need a JSON format template that will be created based on various factors. These factors include user requirements, number of EC2 instances, time at which users want a load balancer, window server, a database server, and the applications running on top of it. All of this can be templatized into the code by you. Therefore, one creates a code and either put it in an S3 bucket or saves it locally from where the CloudFormation will call the template and provide the resources. Therefore, CloudFormation is used to create the stack or the resources defined in the template. The CloudFormation will analyze the templates first, validate them, and detect the dependencies between them. After that, it starts providing the resources gradually, one after the other. So, if you need five servers in a VPC, it will create the VPC first, then servers, and then the stack.

The infrastructure provisioning and updates are automated by CloudFormation in a managed and secure way. The *Rollback Triggers* can be used to configure CloudWatch alarms for CloudFormation to track while the stack creation and update are in process. The entire stack will be rolled back to the earlier state by the CloudFormation if any alarm is breached.

Chapter 6: Network & Content Delivery Services

Overview

Content distribution networks (CDNs) have started to emerge rapidly, utilizing cloud resources such as compute and storage. Cloud-based CDNs use geographical availability and the PAYG billing model, which is different from our conventional CDNs hosted in private data centers. The CCDNs support the AWS cloud model of content distribution as a service.

In this chapter, the AWS networking components will be discussed to design any environment using VPCs and content delivery solutions via AWS Global network, especially Edge Locations. Moreover, this chapter gives a general idea of the AWS VPC and the networking components. AWS networking services offer a vast spectrum of database and networking options that are scalable, on-demand, and available with a few mouse clicks. Let us look quickly at the *AWS Global infrastructure* before we get started with this segment.

AWS Global infrastructure

The cloud infrastructure of AWS is designed around the *Availability-Zones* and *AWS Regions*. In the *AWS Global infrastructure* environment, a physical location in the world in which there are several availability zones is known as Region. Each AWS Region comprises multiple AZ's that are physically separated. They ensure the high availability of an application. There are two or more AZ's available within a region. Normally, users keep their machines in separate AZ's while designing the AWS architecture. If one of the AZ goes down, their machine will be running in another AZ, and the application will have high availability.

On the other hand, an Availability Zone (AZ) has multiple distinct data centers kept in separate locations with redundant networking, connectivity, and power facilities. These Availability Zones allow you to run highly accessible, fault-tolerant, and scalable databases and production

applications from a single data center. The AWS Cloud works over 25 geographic regions and 80 Availability-Zones across the world, and it has declared plans for more Regions and Availability Zones.

Now, let us start with the first and most basic networking service known as Amazon VPC.

Amazon VPC

Introduction

Amazon VPC stands for *Virtual Private Cloud,* which enables its users to build a virtual networking system inside the cloud in their own logically isolated area. The EC2 instances can be launched into the VPC subnets. A VPC is like a standard network system that you can have in your data center. Every Single VPC is entirely customizable and logically separated from the rest of the cloud's virtual networks. You can select a number of IP addresses, set up network gateways, configure route tables, and create subnets. As a prerequisite, you must create an IPv4 CIDR Block. In addition, the VPC's IPv6 CIDR Block address is also available.

Moreover, you can configure security settings with the help of security groups and NACLs. VPCs are region-oriented, so they do not span regions. One can create five VPCs per region where each region has a default VPC. You can have 200 Subnets per VPC, which is a large number. DNS hostnames are not enabled by default while creating a VPC. When an EC2 instance runs with DNS hostnames enabled, it will give you a public IP. However, it will only get a public DNS that is a domain name, such as an address.

When you deploy an instance in the AWS account, it will have a private and public IP and will be deployed into the default VPC by default. The IP addresses that are not accessible over the internet are known as private IPs, which allow inter-instance communication in the same network. On the other hand, public IP addresses are used for intra-instance-communication

on the internet. You can use these IPs to communicate between your and others instances on the web. Then there are Elastic IP addresses which are persistent/static public IPs that come with your account. If an instance or software fails, you can immediately remap it to another instance by using an elastic IP. Furthermore, the default IP address contains all valid IP addresses and is used for giving access to user's public resources from the internet. It is represented as '0.0.0.0/0'. When you interact with AWS networking, you can use this address for sending traffic to the internet to get an internet gateway route. Therefore, you need to set this IP to enable any internet traffic to access your public resources using *SG inbound rules*.

When you create a VPC, there is no cost involved for using Route Tables, Internet Gateway, Security Groups, Subnets, and VPC Peering. However, there are resources in VPC that cost money. They are VPC Endpoints, NAT Gateway, Customer Gateways, and VPN Gateways. You mostly use those resources that do not cost anything. Therefore, you should not worry about getting over-billed.

Launch instance into a VPC

While launching an instance and specifying a VPC, the first thing is to define a subnet where you want to deploy the instance. If you do not mention a subnet, then AWS will choose the default subnet in the selected VPC. The next step after identifying the subnet is to identify an AZ, which can be selected by you, or AWS can pick one.

A primary private IP is allocated to the instance's default network interface from the IPv4 subnet range whenever an instance is created within a VPC. In order to resolve an instance's private IP address, the private DNS hostname must be provided to each instance. In case you do not specify a primary private IP, AWS selects it for you from the available subnet range.

Whenever you launch an instance in a subnet with an enabled public IP address attribute, the primary NIC generated for an instance will be assigned a public IP. Thus, the primary private IP is mapped to the primary public IP with the help of NAT.

Now let us look at the two types of VPCs.

Types of VPC

Default VPC and non-default (*Custom*) VPCs are the two types of VPCs. In Default VPC, Amazon creates it by default, and one can instantly launch an instance into it. The Default VPC is available for every region. You can instantly launch the EC2 instances without considering any network setup. A default VPC comes along with specific configurations such as:

- It creates a VPC of CIDR Block size /16 with attached default subnet (size /20) for each Availability Zone.
- The main Route Table is provided with a rule that transfers all IPv4 traffic to the IGW.
- An IGW is created and linked with the default VPC so that the EC2 instances have internet connectivity.
- Default SG is associated with the VPC. Running an instance will automatically enable the SG unless you override it.
- It also integrates with default NACL and DHCP options.

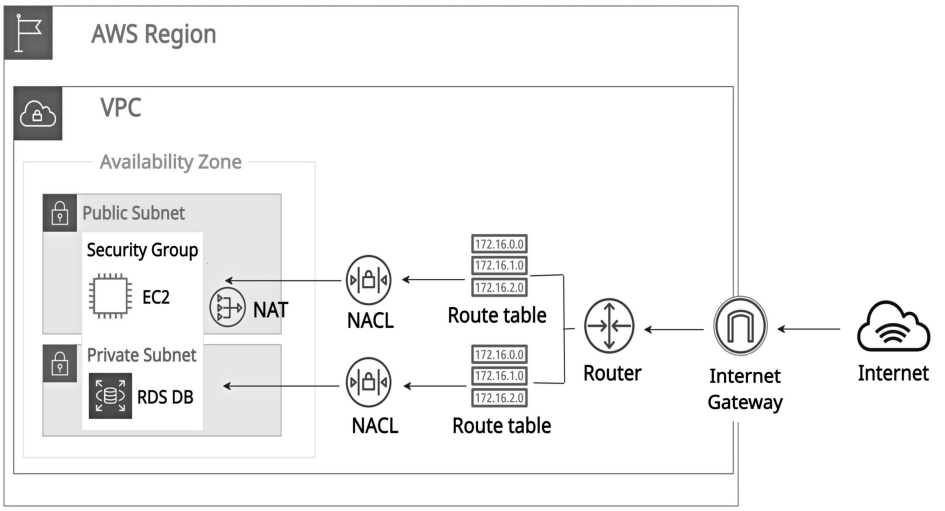
In contrast, custom VPCs can be created using out-of-the-box configuration for hosting the production workloads in default VPCs. AWS suggests that you need to customize your VPC when its environment becomes complicated. For instance, a default VPC implies that every EC2 instance will reside in internet accessible public subnet, which may not satisfy a company's security requirements. Thus, you need to generate a Custom VPC and configure it as you want. Creating a custom VPC enables its users to:

- Making resources secure
- Modify the virtual network to identify the user's IP address range.

- Create private and public subnets
- Reinforce security settings.

Core Components

VPC provides logically separated sections of the AWS cloud to its users, where the user initiates AWS resources in the virtual network. The easiest way to understand VPC is by considering it as a personal data center because it offers complete control over your virtual networking environment. Let us look at the architectural diagram of VPC with multiple networking components in it.



We have a Region, and it has VPC in it. Then VPC has multiple Availability Zones in it. So when the internet flows into the network, it enters from the *Internet Gateway* and passes through a *Router*. The Router sends the traffic to a *Route table*, which will pass it to the *NACL*. After that, NACL directs the traffic to the *Private* and *Public Subnets* in the user's VPC. Then there is a *Security Group* that contains all the resources within VPC. This is just an overview of how network traffic flows into the VPC from the internet. Now let us look at different networking components which are part of the Amazon VPC.

Subnet

It is basically a set of IP addresses in Virtual Private Cloud within the AWS environment. A VPC consists of multiple availability zones, but the *subnet* lies in each AZ. Therefore, an instance cannot be launched until the subnets are present in the VPC. However, you can launch AWS resources in a specific subnet. Furthermore, the resources connected to the internet use public subnets. In contrast, the resources not connected to the internet use private subnets. The *Netmask* for a default subnet is 20, with a maximum of 4,096 addresses for all subnets, of which AWS keeps only a few.

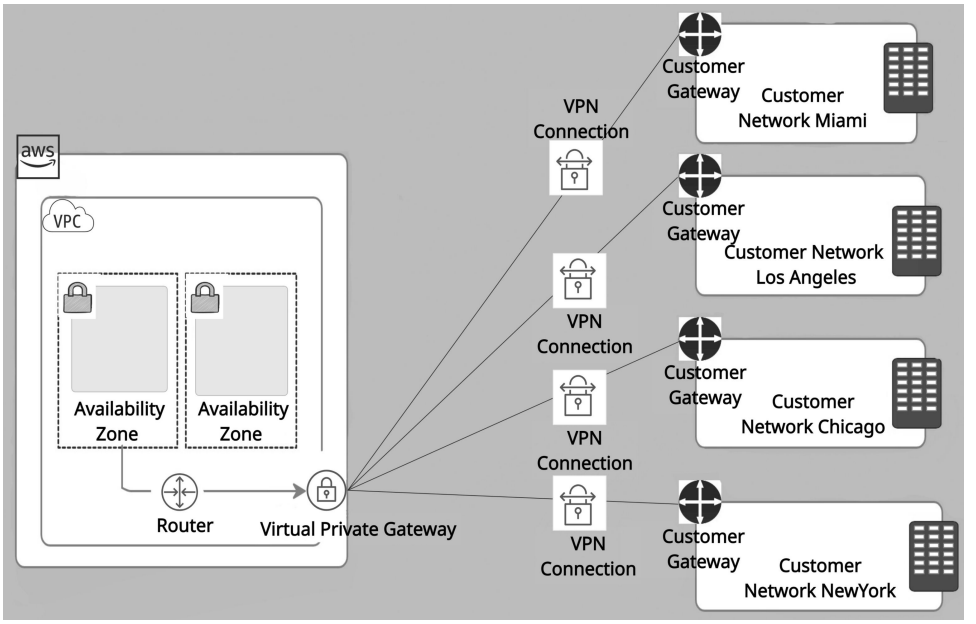
Private & Public Subnets

Let us look at the two types of VPC subnets. The resources that are linked with the internet use *Public Subnet*, such as web servers. The main Route Table delivers subnet traffic intended for the web to the internet gateway, so it is made public. On the other hand, the resources that do not need an internet connection use *Private Subnet*. One might use them to secure their resources from the internet, such as database instances.

Virtual Private Network

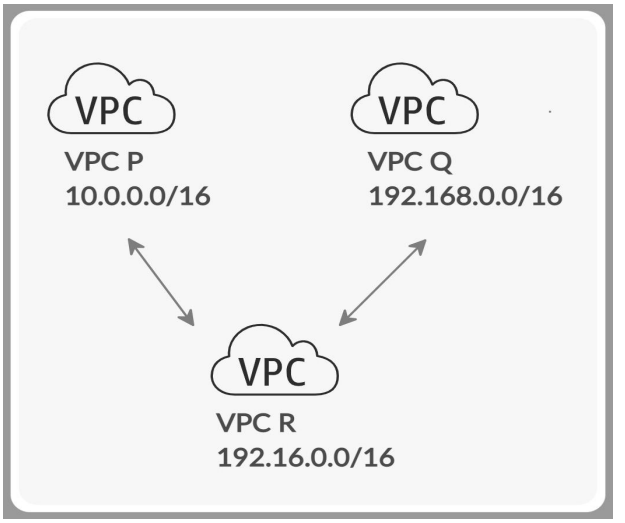
By default, instances launched in the Amazon VPC cannot communicate with the user's network. Users can link VPCs to an existing data center using *VPN* access. By doing this, they can create a Hybrid environment that will expand the data center into the cloud. They have to set up a virtual private gateway for this purpose.

In the following diagram, there is a VPN concentrator on the provider side of the VPN link. You will need the customer gateway for your data center. It is either a Software-application or a Physical-device that resides on the consumer side in the VPN link. When a VPN link is made, the traffic from the customer's side connection creates a VPN tunnel.



VPC Peering

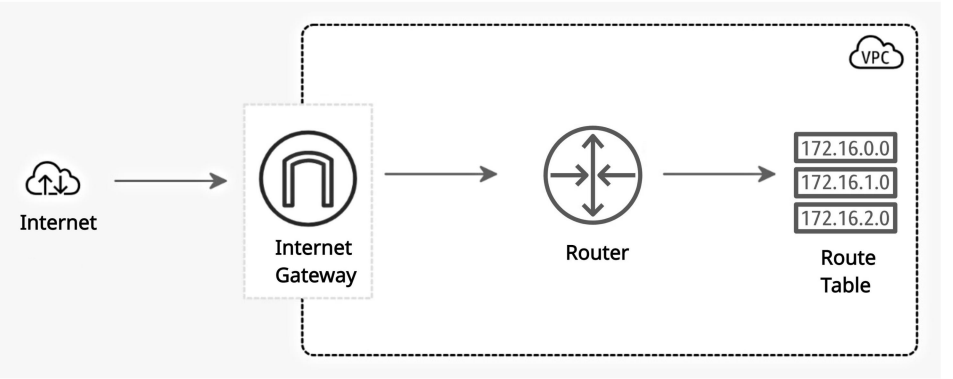
VPC Peering enables its users to link one VPC to another across a direct network route using private IP addresses. A peering link can be established between two VPCs in your AWS account if they are in the same region. Let us say you have instances in VPC A, and you want it to communicate with VPC B or C. In that case, you must create a peering link. Peering is a one-to-one relationship that means a VPC can be attached to other VPCs through multiple peering connections, but it does not allow transitive peering. In the following figure, VPC P can connect with VPC Q and R. In contrast, VPC Q cannot communicate with R until it gets paired directly. In addition, VPCs cannot pair if they have overlapping CIDRs. As seen in the figure, all VPCs have different IP ranges. They could not pair if they had the same IP ranges.



Now that we have learned how VPC operates and how instance launch works in it, let us expand this chapter by exploring further networking components.

Internet Gateway

This service allows internet access to the VPC and performs two activities. It creates a *Target* in the Route Table for internet traffic routing. It also performs Network Address Translation on instances with public IPv4 addresses. It allows instances in the VPCs of users to connect with the internet. This ensures that the network traffic does not face any bandwidth limitations or availability risks. To allow the VPC to connect to the internet, one must connect to an internet gateway. Moreover, only a single internet gateway is allowed to attach per VPC. So connecting an Internet gateway is the first step in allowing internet access to instances in your VPC.



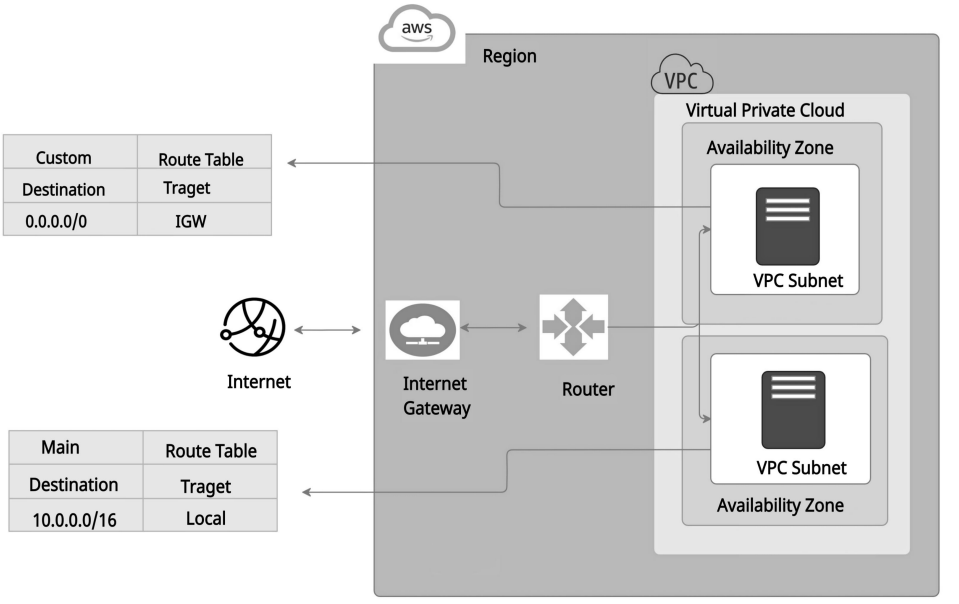
Let us look at the IGW working. When you want an IGW to access the web and route the traffic from EC2 instances, you have to pass through a Route table to get to a router. Thus, you will be generating a new Route in the Route-table for the Internet Gateway. For that, you have to provide the information for the target as ***igw-id*** and destination as ***0.0 0.0/0.***

Custom route table

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	*igw-id*

Route Tables

Route Tables are used to determine the path of network traffic. A route table should be connected to each subnet in your VPC. Only one route table can be used with a subnet at a time, but several subnets can be connected with the same route table. A common example of using the route table is that it permits EC2 instances to access the internet. In this way, you can have a public subnet where that EC2 instance belongs and links a route table to it. The route table will then provide the path that will give internet access to you through the Internet Gateway.



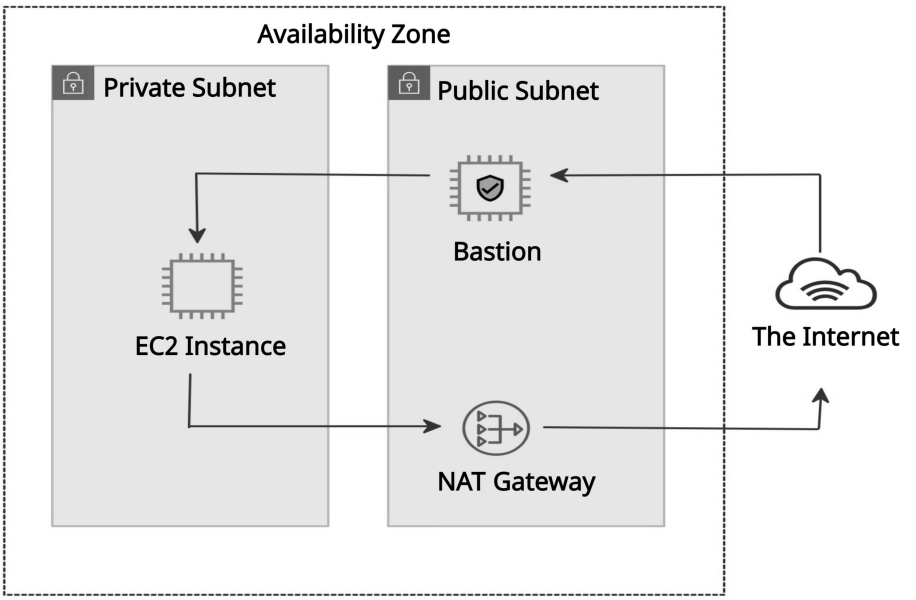
We added two route tables in this use case:

- Default route table
- Custom route table

A private subnet is attached to the default route table, which will not allow web traffic to come into the VPC. Therefore, all the traffic inside the private subnet remains local. On the other hand, Custom Route-table will notify the internet gateway to send the web traffic to the public subnet.

Bastion Host

It is a safe and robust instance that belongs to the public subnet of the VPC. It has classified security controls that allow limited inbound links from known and verified IP addresses. Generally, it uses RDP or SSH protocols for this purpose. These secure controls let the engineers establish a local and secure connection to the *Bastion Host*. Thus, the instance here serves as a '*jump server*,' which enables the engineers to RDP or SSH to the instances located in private subnets.



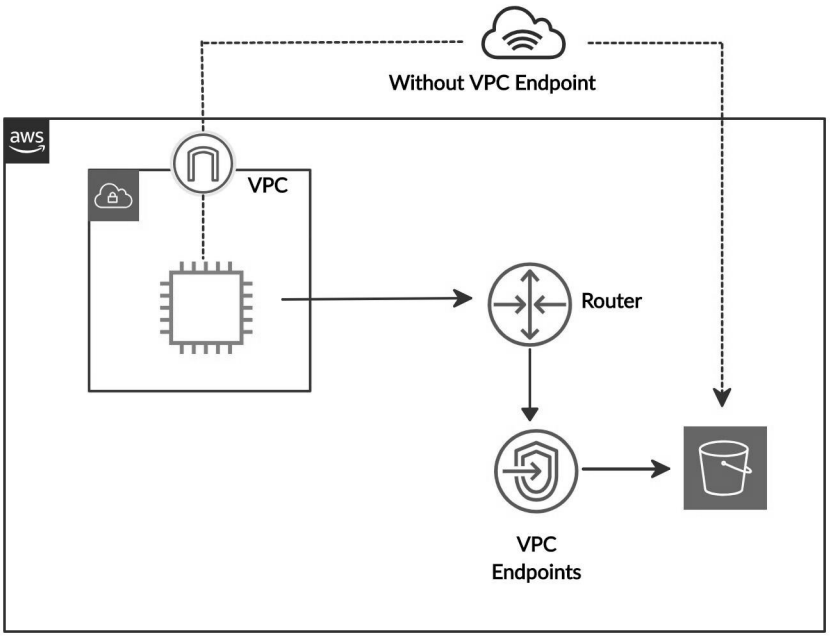
Although the NAT instance is an EC2 instance, one cannot turn it into a bastion like NAT Gateways. However, from a security point of view, you will not prefer to do so because of NATs' configuration. You would always prefer to have different EC2 instances as a bastion. Besides, a service called *Systems Manager's Sessions Manager* replaces the need for bastions where you need not launch your EC2 instance. Thus, AWS recommends it, but many companies are still using bastions because they meet their requirements.

Direct Connect

Direct Connect is an extremely fast AWS solution for establishing dedicated network connections from the user's on-premise locations to AWS. The configurations are based on the selection of the range of bandwidth. The lower bandwidth is between 50 MB to 500MB, whereas the higher bandwidth is 1GB to 10GB. Thus, the transfer rate of the network from the on-premises environment to AWS is considerably fast. Direct Connect helps to reduce network cost and increased bandwidth throughput. It offers a better network experience for individuals and enterprises as compared to typical Internet-based connections.

VPC Endpoints

Users can use *VPC Endpoints* to privately join their VPC to other VPC endpoints and AWS services by keeping the traffic within the AWS network. The public IP-Addresses are not required to communicate with the AWS services because it eliminates the requirement of an Internet Gateway. Let us consider a use case in which you need some data from the S3 bucket. So, you request it using SSDK. Furthermore, to get the data file from S3, the request would go out from the internet gateway and come back into the AWS network.



We have two types of VPC Endpoints in the AWS Network. They are *Interface Endpoints* and *Gateway Endpoints.* Interface Endpoint provides *Elastic Network Interface* (*ENI*), a NIC which connects multiple AWS services and has a private IP address from the VPC subnet IP address set. It acts as the entrance for traffic that is going to a supported service. Furthermore, the AWS Private Link supports the Interface-Endpoints. Therefore, one can have smooth and secure access to AWS's services by keeping the network traffic in the AWS network. It supports many AWS services such as *API Gateway, CloudFormation, CloudWatch,*

Kinesis, AWS KMS, Secrets Manager, SNS, SQS, Service Catalog, SageMaker, Codebuild, and *AWS Config.*

On the other hand, Gateway-Endpoint is the second VPC Endpoint type. In the Gateway Endpoints, the Route-table is used to access AWS services. The specific route in your route table used for the traffic flow of a supported AWS service is specifically targeted by the Gateway endpoint. In order to set up a Gateway Endpoint, you must identify a VPC, where you want to build an endpoint, and the service to which you want to build a link. Gateway endpoint is a 100% free service, and it is usually used for DynamoDB and Amazon S3. You will be charged for using the Interface-Endpoints. On the other hand, Gateway Endpoints are free.

Private Link

Introduction

Before discussing *AWS PrivateLink*, its functionality, and use cases, let us first understand the relation between VPC Endpoints and Private Link. With the help of *VPC Endpoints*, one can connect his VPC to VPC endpoint services and AWS services privately. It utilizes AWS Private Link to wipe out the necessity for AWS Direct Connect, a VPN connection, a NAT unit, or an Internet gateway. The Amazon network traffic between VPCs and other services remains secure by not leaving the network.

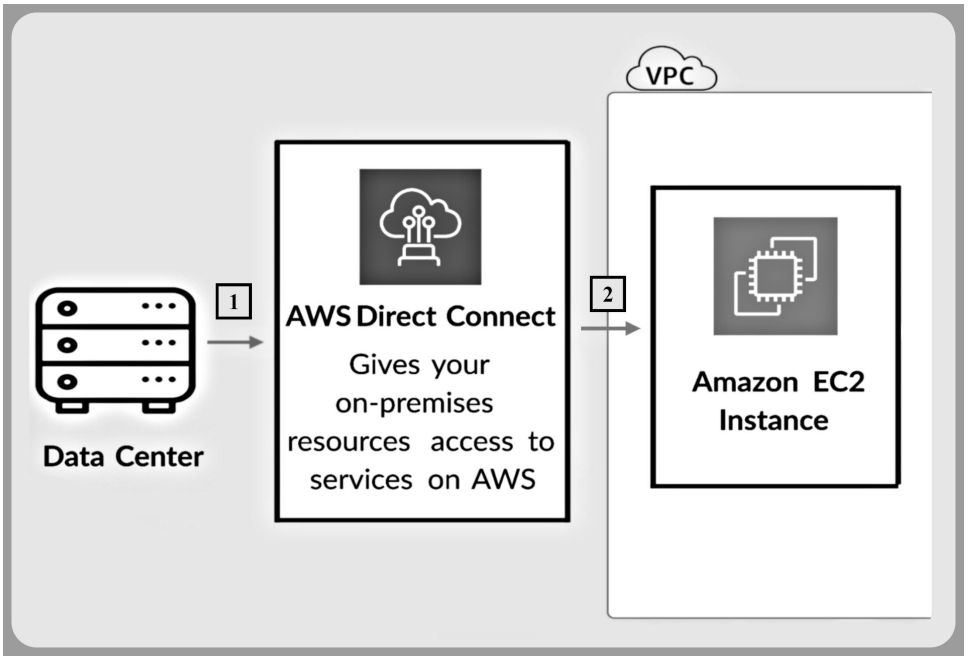
Now let us understand the concept of *AWS PrivateLink*. It offers a private connection among AWS VPCs, an on-premises application, and AWS services. It is highly scalable and available technology. It resembles Direct Connect, where it creates a private connection to the AWS cloud while Direct Connect connects the user's on-premises environments to AWS. However, Private Link secures the network traffic of the VPC environments of the users residing in AWS. Using the VPC endpoint services, it connects securely across the Amazon Network. The Interface Endpoints can, by default, allow 10 Gbps bandwidth in every AZ.

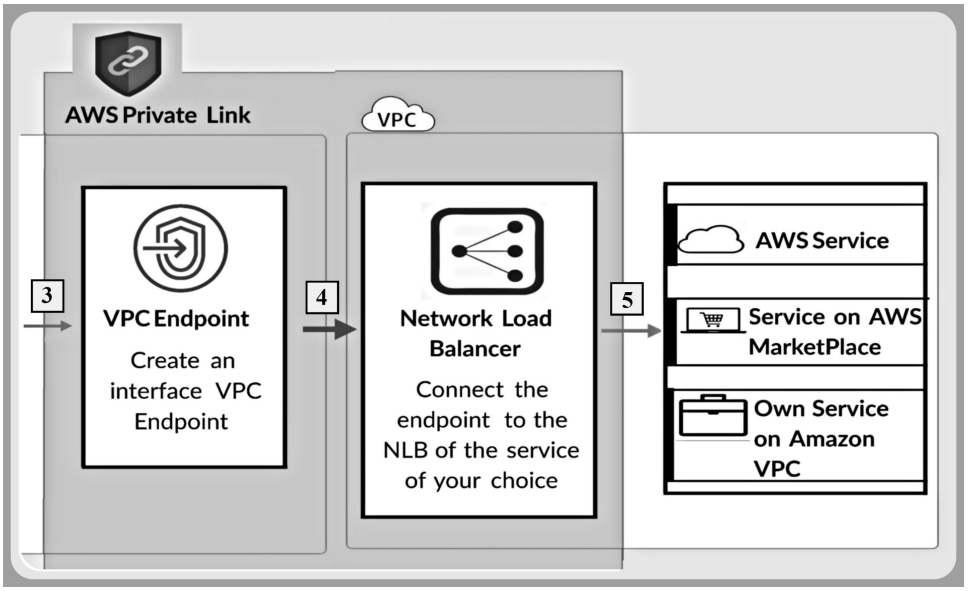
The traditional endpoints were like associating the virtual cables between the Virtual Private Cloud and Amazon Web Services. The connectivity for AWS service does not need any NAT gateway or an Internet; however, the endpoints stay outside of the VPC. While in the case of Private Link, the endpoints are made inside the user's virtual private cloud utilizing IP addresses and ENIs. The service is now available in the user's VPC and allows him to communicate to AWS services through private IP addresses. It implies that the AWS Direct Connect can be used for accessing Private Link endpoints from your in-house data center. Moreover, you can use VPC SG's to handle access to the Endpoints.

How does it work?

PrivateLink allows its users to connect VPCs to the AWS-supported services securely. It can link the VPCs to their own services on AWS, third-party services, and the services hosted by the other AWS accounts. There is a grid diagram below in which the in-house data center uses Direct Connect or a VPN to connect with AWS. In the checkbox below, AWS Private Link is created where a VPC generates an Interface endpoint which creates an ENI according to each AZ. The provider VPC is running a Network Load Balancer to keep the AWS services behind.

Usually, multiple VPCs can share the same services. They connect by using either a VPC public IP address across the internet or privately using the VPC peering or routing via on-premises location. AWS Private Link allows you to create service connectivity from service provider VPC to consumer VPC in a safe and scalable way. Besides, you can connect to the services that are available in a shared service environment or the AWS marketplace.



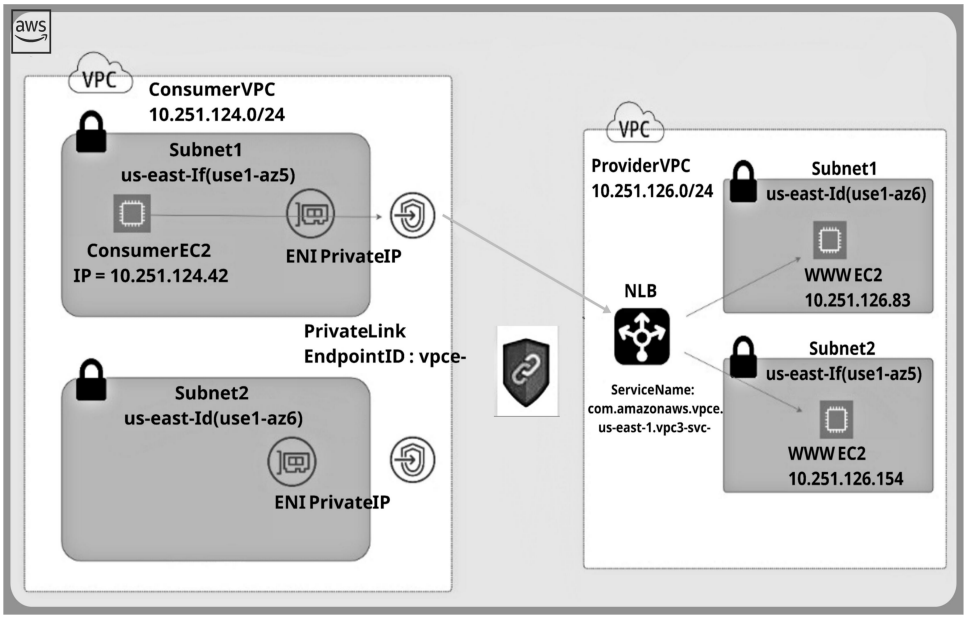


Use case

Let us have a look at the use case for AWS Private Link. In the same AWS region, there is a connection between a consumer VPC and the provider VPC. Thus, the provider VPC lies on the right side of the

architectural diagram, which consists of two subnets. It hosts web services for external clients and the customer's internal products. The VPC Endpoints and the host services lie under the NLB. The consumer VPC, on the other hand, lies on the left side that consumes the services via interface endpoint. It also consists of two subnets that are identical to the provider VPC. Also, the ENIs must be available in the same AZ and same region.

To utilize AWS Private Link, the consumer VPC establishes an interface endpoint for the service in its own VPC. It will create an ENI in a subnet containing a private IP, which acts as an entrance for the traffic expected for the services on the other end. Thus, the service endpoints available via AWS Private Link will appear as ENIs with private IPs in the VPC. Then, the consumer VPC will connect to its Endpoints, and they will connect to the NLB of the provider VPC on the back end. In this way, Private Link can be used for establishing secure connections between AWS services and VPCs.

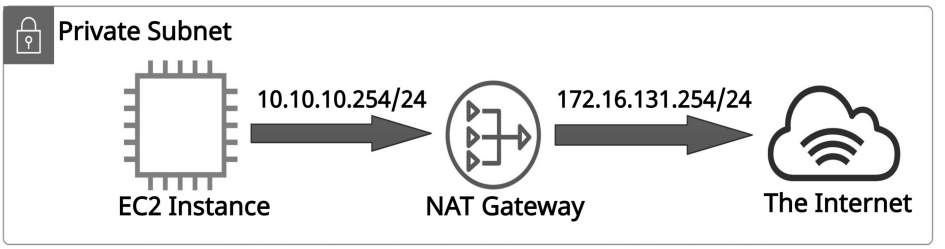


Network Address Translation

Introduction

Let us talk about the role of *NAT* in AWS network architecture. It is a method of remapping an IP address space into another. NAT devices are used to connect the private subnet instances with the AWS services or the internet. However, the internet is restricted from creating links with the private subnet's instances. With NAT devices, private subnet traffic is directed towards other AWS services or internet, and in return, the instance gets its response. As soon as the traffic is routed to the internet, the NAT device address takes the place of your instance's source IP address, and when the traffic returns, the NAT device converts it to your instance's private IP address.

Let us assume that there is a local network with its personal IP address space. The IP address changes as it passes through NAT. If you have private networking, you need help in getting outbound access to the internet. Therefore, you must use the NAT gateway to remap those private IPs. Secondly, when there are two networks with overlapping network addresses, you can utilize the NAT to make the addresses more appropriate for communication.



Route 53

Introduction

AWS Route 53 provides a cost-effective and secure Domain Name System for companies and developers to route users to their web applications. It is used to translate the domain names into IP addresses for routing traffic to a website. Users use AWS Route 53 for their domain management. It is a reliable and affordable DNS service for businesses and developers. Specifically, it is useful for managing AWS infrastructure. In simple words, there are three purposes of Route 53:

- Route Internet traffic,
- DNS Registration, and
- Health Check status

In Route 53, domain requests are resolved by a global network of DNS servers, which makes it fast and reliable. The Domain Name System is an internet routing layer. It maps domain names in human-readable format into machine addressable IP addresses such as *www.fulllhosting.com* into 23.21.47.33. It is used especially for registering and managing domains. Various record sets can also be created on a domain using Route 53 and implementing complex traffic flows like blue/green failovers. Besides, you can also resolve VPCs outside of AWS and continuously track the records via health checks. Using Route 53, an organization can easily track and route global data traffic. AWS provides a user-friendly and simple Restful API for its management in the Command Line interface. Thus, it serves as a programmable DNS for the infrastructure as well.

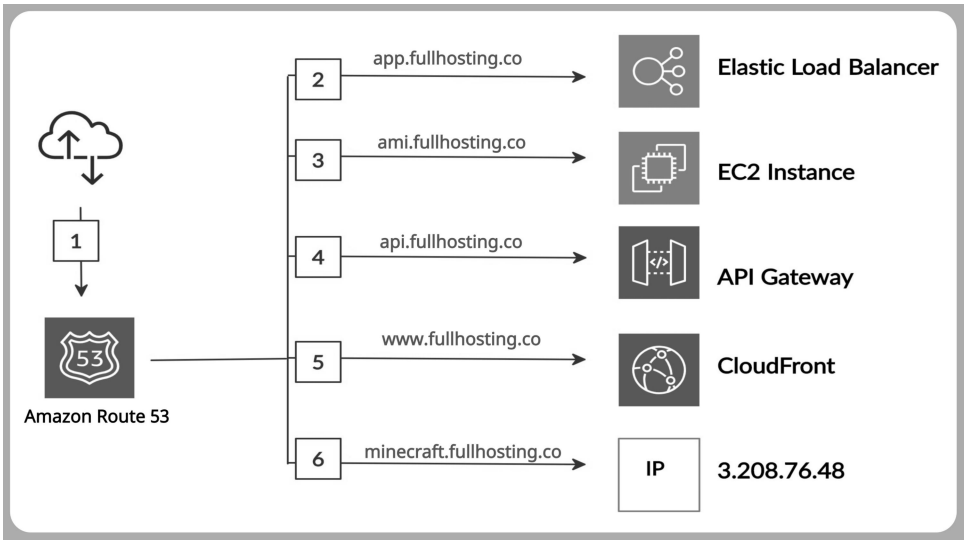
How does it work?

Lets us look at the overview for the working of Route 53 to route the traffic between the hosted web apps and end-users:

1. First, the domain name gets registered with Route 53 and then configured for routing the internet traffic to the servers. Both AWS public and private cloud infrastructures can act as the servers hosting the domain name.
2. A complete URL or the domain name is entered by the users in the search bar of any web browser.
3. The request is routed to the DNS resolver by the Internet Service Provider.
4. The DNS resolver forwards the user request to its root name server, and then it is routed towards the Top-Level Domain server. Eventually, it is directed to Route 53.
5. The domain name IP address is returned by the Route 53 name server to the DNS resolver.
6. When it receives the required IP address, it will direct the user request to the content hosted server according to the Route 53 service configuration.
7. The health of backend servers is also monitored by Rout 53. There is a feature known as *DNS Failover*. It is used to check the endpoints for availability. In case an endpoint turns out to be unhealthy, Route 53 will direct the traffic towards some other healthy endpoint. With the help of CloudWatch, an alarm can be triggered to notify the recipient to take necessary action.

Use Case

Consider a use case in which you have a domain name *'fullhosting.co'*. You have Route53 to manage the name servers allowing you to set the Record Sets within Route53. There is a set of various Record Sets for sub-domains, and you want these subdomains to point to different resources in AWS, as shown in the diagram below.



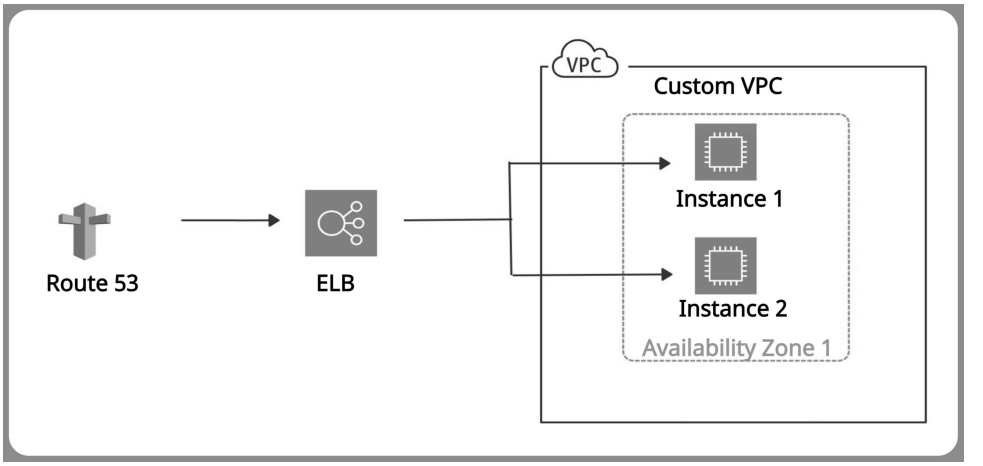
Thus, your app uses 'app.fullhosting.co' *to* point out to ELB because this app runs behind it. When you need to work on the AMI image, an EC2 instance can be launched and pointed to the subdomain 'ami.fullhosting.co' there**.** If the API gets powered by an API gateway, you can use 'api.fullhosting.co'. For static website hosting, you can point out to CloudFront and use **'***www.fullhosting.co'* for Cloud distribution. You might run a Minecraft server on a specific IP from a learning perspective, which is probably an elastic IP, as you do not want it to change. Therefore, you can use '*minecraft.fullhosting.co***'** for this purpose**.** This is a very simple example; however, now we will learn the features of Route53.

Features of Route53

Simple Routing Policy

Simple Routing Policy is the most fundamental routing policy. It always uses an *A record* to resolve a single resource without any specific rules. You can direct the traffic to the single resource with this policy, e.g., to the webserver of a service provider. In Route 53 console, multiple records cannot be created with the same name and type using a simple routing policy. However, multiple values can be specified in the same record, such as multiple IP addresses. Let us say that a DNS record gets created to settle

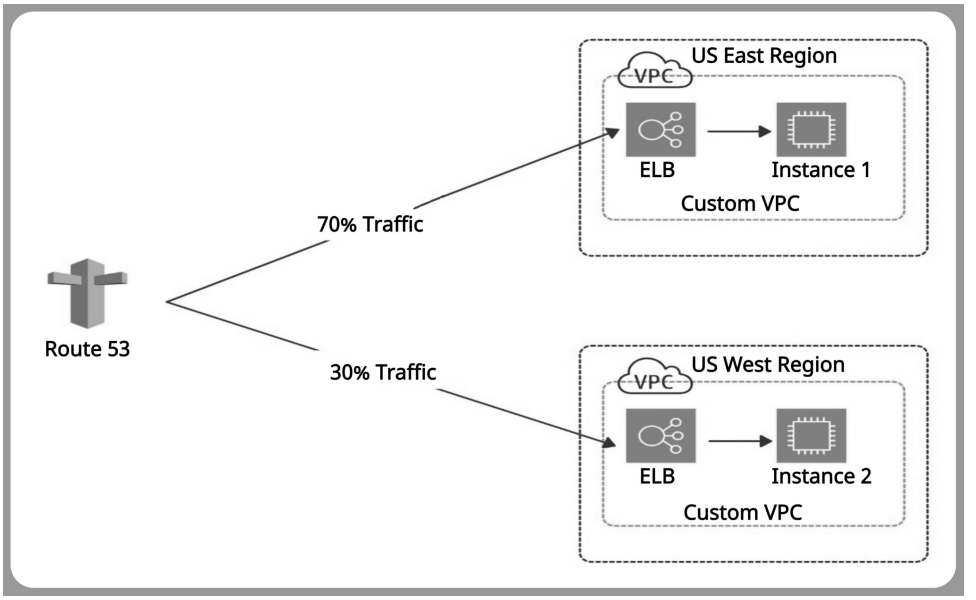
the domain to the ALIAS record. It will route the traffic to an ELB, which will load balance the set of instances.



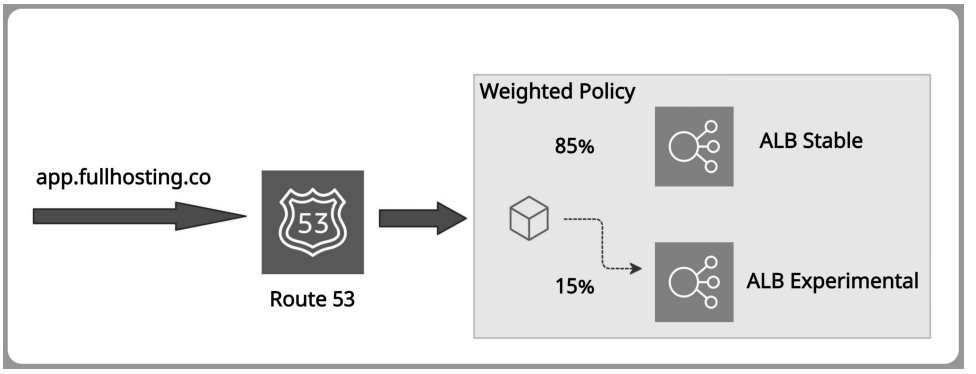
When creating a record, you need to set its name and type. In this case, let us assume that the name is Random, and the type is A-IPv4 address**.** The Routing Policy is always set to simple by default. It means that you can provide more than one IP address using a simple routing policy. A single IP means the Random will move towards the first IP address every time. On the other hand, if there are multiple IPs, it will randomly select one.

Weighted Routing Policy

When multiple resources are required for the same functions, users apply *Weighted Routing Policies* to split traffic between the resources relying on a predefined weight. It is useful for several reasons, such as testing new versions of software and load balancing. It allows you to split the traffic depending on various assigned weights.

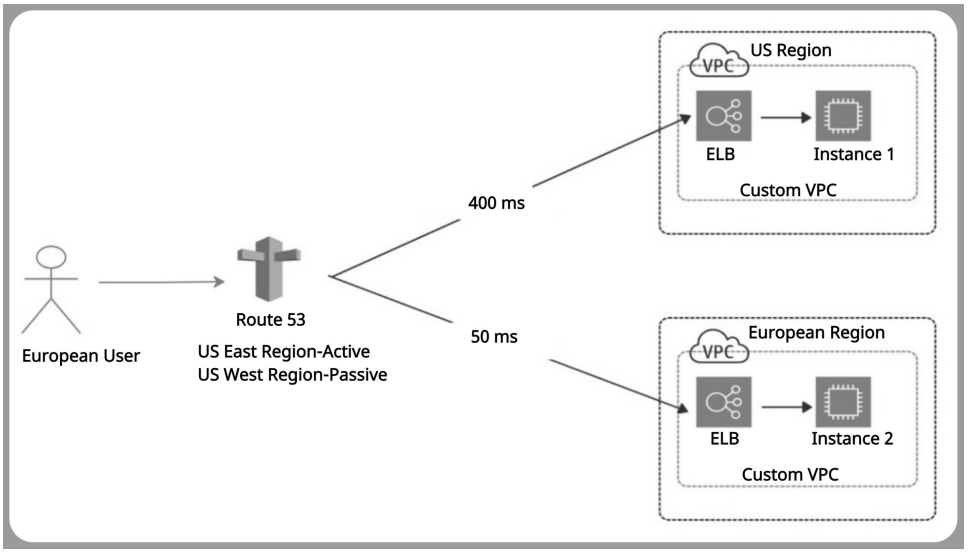


Let us discuss the use case for this policy. Let us say you have a domain ‘app.fullhosting.co,' and create two identical Record Sets in Route 53 using this domain. Therefore, for a weighted routing policy, you need to set two different weights for each of them and set them both to weighted policy. Let us suppose that you create the first recordset, '*Stable*,’ and it is set to *70*%. Then, the other record set, 'Experiment,' is created with the same sub-domain and is set to *30*%. Now, when the traffic hits ‘app.fullhosting.co,' it will observe the two weighted values. The *70*% network traffic will go to *Stable*, whereas *30*% of traffic will go to the *Experiment* recordset. It was a simple use case to help you grasp the concept. However, it is recommended to test a limited amount of traffic to mitigate the effect while testing new experimental features.

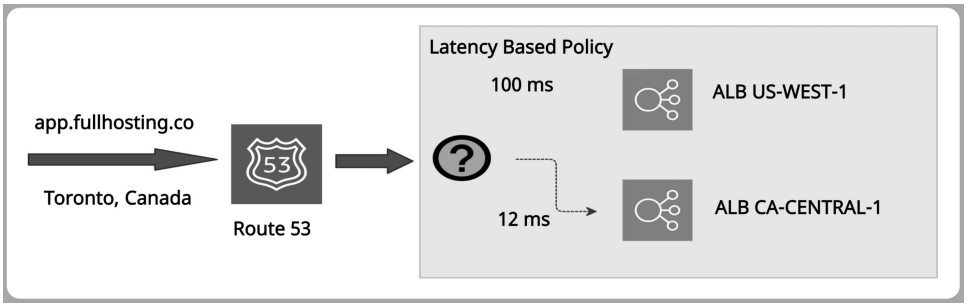


Latency Based Routing Policy

This policy enables you to direct traffic depending on the lowest possible network latency for the region-based end-users. Users apply this policy when there are multiple resources for the same functionality. Route 53 is intended to address DNS queries with the high latency possible, e.g., the region that offers the fastest response time.

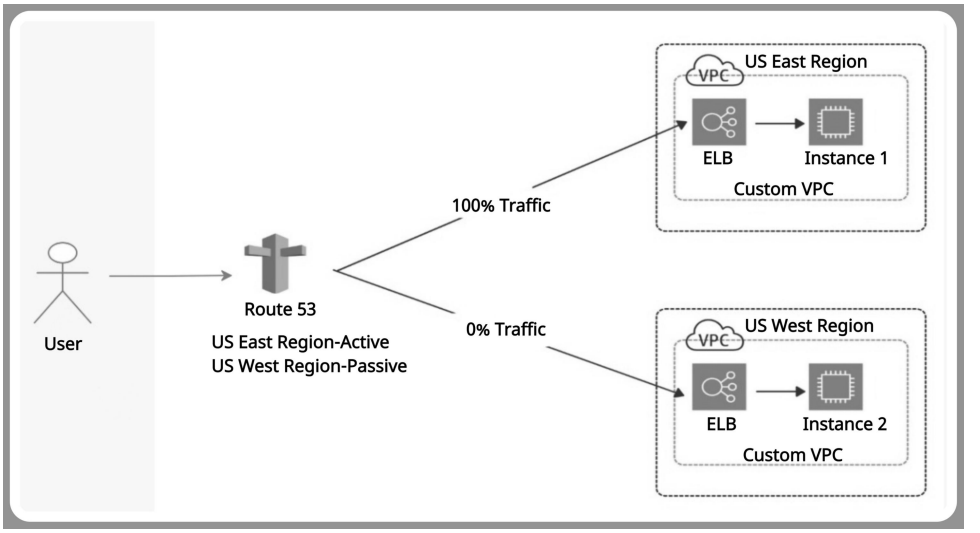


Consider a use case in which the traffic from Toronto hits your domain app.fullhosting.co**.** Therefore, you create two records having latency with the sub-domain. Let us say that one is set to *US-WEST*, which is located on the west coast, while the other to *CA-CENTRAL*, located in Montreal. Thus, you have set the policy for latency-based routing, which will observe the records with the least amount of latency. The one with the lowest return in milliseconds is the one to which the traffic will be redirected. Besides, it does not have to be the nearest one geographically. Logically, the closer ones should be faster. In this case, *CA-CENTRAL* has 12 milliseconds of latency. Thus, it will route the traffic to ALB *CA-CENTRAL*. That is how Latency Based Routing Policy works.



Failover Routing Policy

As the name reflects, this Routing policy is concerned with failovers. It allows you to create an active and passive setup in a situation where you want a primary site in one location and a secondary data recovery site in another location. Moreover, Route 53 automatically tracks the primary site through health checks to determine the endpoints' health. If the endpoint is not healthy, then it will automatically redirect the traffic towards the secondary location.

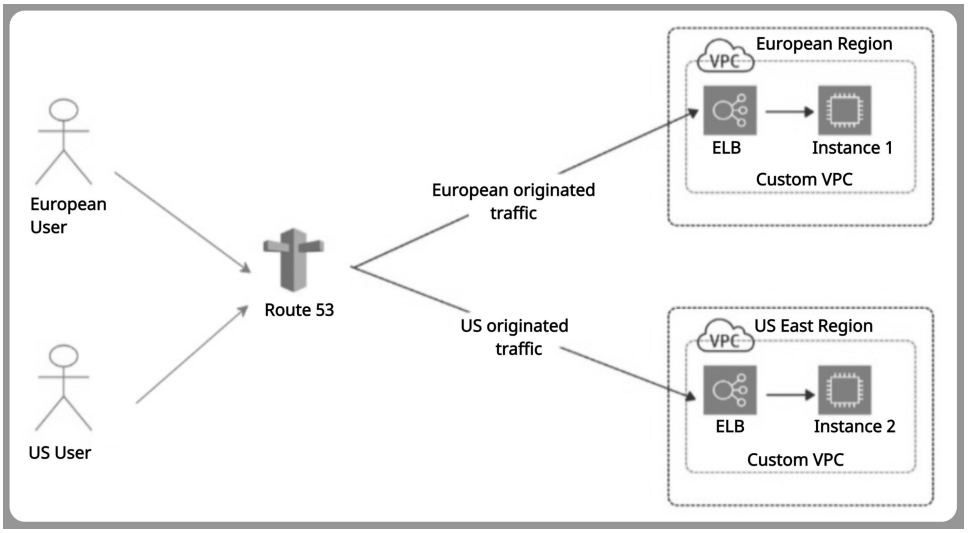


Let us assume the same domain ‘app.fullhosting.co’ with instances in primary and secondary locations. Thus, Route 53 will run the checks on these endpoints with the help of a health check, and if it finds the endpoint unhealthy, it will redirect the traffic to the secondary location. Therefore, you have to create two routing policies with the same domain and choose the primary and secondary locations.

Geolocation Routing Policy

Geolocation Routing Policy enables users to direct the traffic based on geographical location, which is the request's origin. This policy enables traffic to be routed to resources in the same location from which the request was sent, i.e., it allows user locations to have site affinity.

Let us assume that a request approaches 'app.fullhosting.co' from the US. The Record Set for Geolocation is set for *North America*. So as the US is in North America Region, it will go to this set of records.



Auto Scaling Groups

Introduction

Auto-Scaling Groups or *ASGs* allow their users to set scaling rules to automatically launch additional EC2 instances or shut down instances to meet current needs. ASGs consist of a set of EC2 instances treated as an automatic scaling and management group. The automatic scaling occurs via health check replacements, scaling policies, or capacity settings. Let us discuss autoscaling via capacity settings first.

Capacity Settings

Capacity Settings are the best way to use Auto-Scaling Group without any configurations. Thus, there are three options: '*Desired Capacity,' 'Min,'* and *'Max*.' The *'Min'* is the least number of EC2 instances allowed to run, and '*Max*' refers to the most instances that can be run. In the case of 'Desired capacity,' it is the number of EC2 instances a user wants to run.

Consider a situation in which you have a new auto-scaling group where *Min* is set to one. When you launch it, and nothing is running, then the ASG will spin up one instance. Furthermore, a server will always spin up at least one instance if it crashes for being unhealthy. Then there is an upper level where you cannot exceed two instances because the ASG could trigger more instances. It is like a safety net that ensures that you do not have multiple running servers. The 'Desired capacity' is the user's required capacity that he wants to execute. Therefore, AGS will try to obtain that value; however, there is no guarantee that it will always be the same value. The auto-scaling disables the EC2 installation and spins up a new one if Min functionality is one.

Health Check Replacements

Health Check is another way of auto-scaling to occur with Auto-Scaling Groups. ELB and EC2 health checks are the two forms of health checks. Let us discuss the EC2 type first. When you apply a health check, it will check whether the EC2 instance is healthy or not depending on two checks performed on the instance. EC2 is considered unhealthy if any of these two checks fail. If *Min* capacity is one, then the auto-scaling will spin up a new EC2 instance by disabling the current one.

In the case of ELB type, the health checks depend on the elastic load balancer. It performs health checks by pinning an endpoint on the server, which could be HTTP or HTTPS, and expects a response. Let us assume that you want 200 back at that specific endpoint. Therefore, if you have a web application, you need to create a small page called health check, which will return to 100. If it does so, it is considered healthy, or else, the

auto-scaling group will disable that instance and spin up a new one for ELB by setting *the Min* value to one.

Scaling Policies

Finally, the most important way of triggering auto-scaling with ASG is *Scaling Policies*. There are three types of scaling policies. Let us start with *Target Tracking Scaling Policy*. It maintains a specific metric at a target value, meaning that you can select the metric type with an average CPU utilization. Let us say that the target value is set to 75%. Thus, if it exceeds this value, then you can introduce another server. Besides, if you add instances, it means you are scaling out. While, if you remove instances, you are scaling in.

The *Simple Scaling Policy* is the second type of scaling policy. It scales when the alarm breaks out. You create the desired alarm and ask it to scale in or scale out. This scaling policy is not suggested for current times because it is old. There is another similar yet more robust policy known as *Scaling Policy with Steps.* This policy allows its users to scale, depending on when an alarm breaches, but based on the changing alarm value, it can also increase the number of instances. In the case of Simple Policy, there was a single value. However, in this case, the alarm value changes with time. Let us consider a use case where you set the alarm between one and two and then add one instance. When it moves between two and three, it will add another instance, and by exceeding three further, it adds another instance. Therefore, it helps you to grow depending on the alarms that change over time.

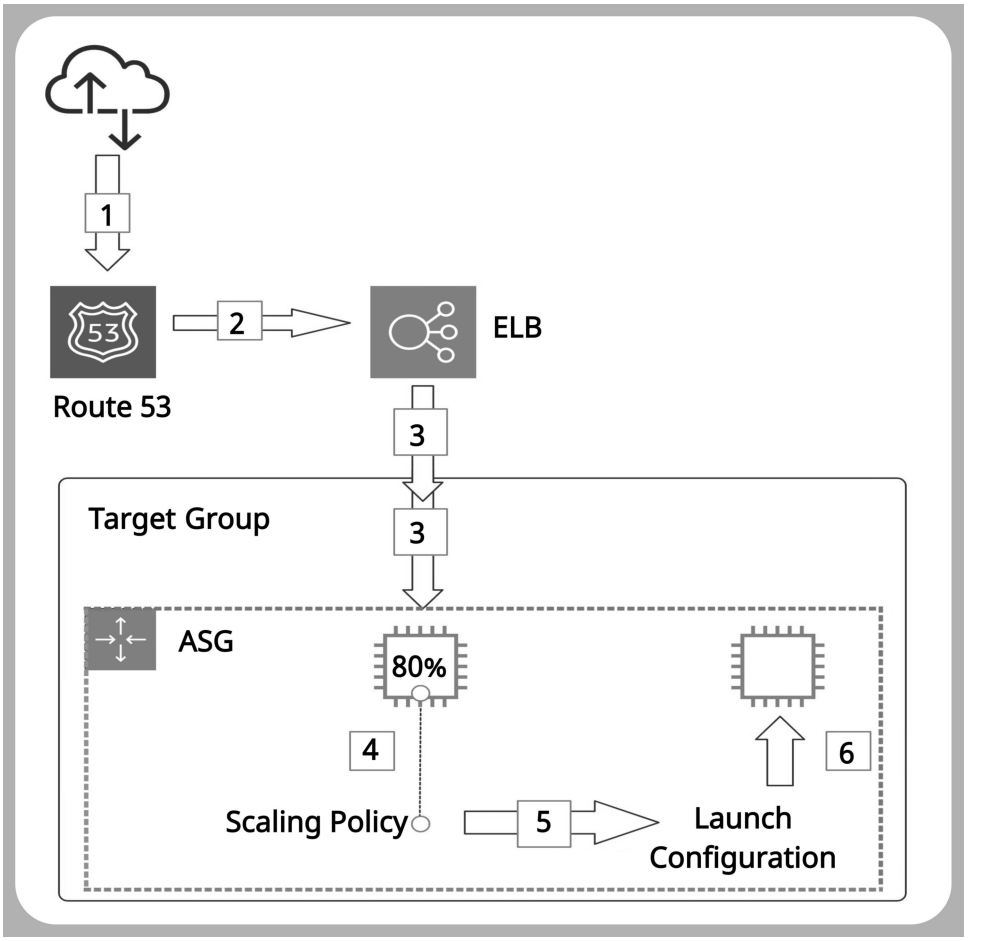
ELB Integration

We have learned that you can perform health checks based on ELBs. However, in *ELB Integration,* you can observe how ELBs are linked to an ASG. There is a bit of diversity in the ELB integration based on the kind of load balancers we will address in the next segment. The procedure to associate ELB with ASG is very simple. There are two fields in auto-scaling group settings, including *Classic Load Balancer* and *Target*

Groups. For a classic load balancer, all you need is to select the load balancer and associate it. On The other hand, you can associate the target group as it lies between the auto scaling group and the load balancers.

Use Case

Let us discuss the ASG use case in which you receive a burst of traffic, and auto-scaling occurs. The ASG uses a launch configuration attached to it to launch a new instance. Let us assume that a web server is running on an EC2 instance. A sudden wave of Internet traffic hits your domain and columns into Route 53, which in turn directs the traffic to the ALB that has a listener. This listener directs the traffic to the target group. The EC2 instance is associated with the target group here. In the target scaling policy, you have already mentioned that if CPU utilization goes over 75%, it will spin up a new EC2 instance. So, the excessive traffic flow causes the CPU utilization to exceed 75%. Thus, ASG uses a launch configuration and spins up a new instance. This is the functionality of the auto-scaling group. The architectural diagram reflects the overall visibility of the working of the entire pipeline.



Launch Configuration

For *Launch Configuration*, when an ASG launches an EC2 instance, it contains the information about the configurations required for an instance launch. Launch configurations can be set with the help of an ASG, which is like launching an EC2 instance. Thus, you follow the same procedure and set the options. The difference is that in the end, you save the configurations instead of launching the instance. It is known as Launch Configuration. You cannot edit these configurations after creation. Therefore, if you want to update or replace the existing launch configurations, you need to make a new one or clone the current configuration. AWS offers a useful service to clone an existing configuration and make some changes using a button. Also, the Launch

Configurations with versioning are known as *Launch Templates*. They are a new version of launch Configuration a user can use.

Elastic Load Balancer

Introduction

With the help of *Elastic Load Balancer* (*ELB*), you can distribute incoming application traffic to different targets. *Targets* may include IP addresses, containers, Lambda functions, or EC2 instances. So ELB includes physical hardware or virtual software, accepting incoming traffic and distributing it across multiple targets. It can manage the load using specific rules, and these rules vary depending on the types of load balancers. Thus, we have three load balancers to choose from for the elastic load balancer: ALB, NLB, and CLB.

Before discussing types of ELB, you first need to understand three components for ELB traffic flow. These components include *Listeners, Rules,* and *Target Groups*. Depending on the load balancers, these components can differ. The listeners listen for incoming traffic and assess it against a particular port (port 80 or 443). Then rules decide the actions against the traffic. Lastly, there are target groups which help you gather EC2 instance (to which you want to direct the traffic to) in logical groups. So, starting from ALB, let us discuss these load balancers in detail.

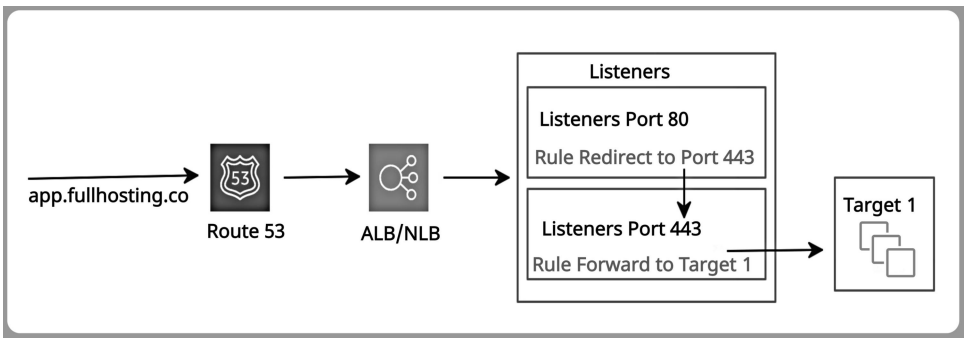
Types of ELB

ALB

ALB refers to the *Application Load Balancer* used to balance *HTTP* and *HTTPS* traffic. It works at the OSI model's seventh layer, which is the application layer. You can add routing rules to the listeners by using a feature of ALB called *Request Routing*. It depends on the HTTP protocol. Therefore, Request-Routing rules are only used for ALB. You can add a *Web application firewall* to ALB as they both are application-specific. If

we consider a use case for an application load balancer, then it is suitable for web applications.

Let us assume that the incoming traffic from Route 53 points to the user application load balancer. As soon as it goes there, the ALB goes to the listener that checks the port it runs on. If it is on port 80, the traffic will be redirected to port *443*. After that, it will move towards the listener that has a rule attached to it. This rule will forward the traffic to *Target* 1, which contains all the EC2 instances. The target group will distribute the traffic equally to the instances registered with it. You can also locate the position of the listeners. In this case, you have ports 80 and *443*, and this is the position where the listeners and ALB's reside. SSL certificate is also attached here because these rules appear for ALB, not for the NLB. Thus, there are a few more complex rules here. Thus, the ALB only forwards the traffic to the target.



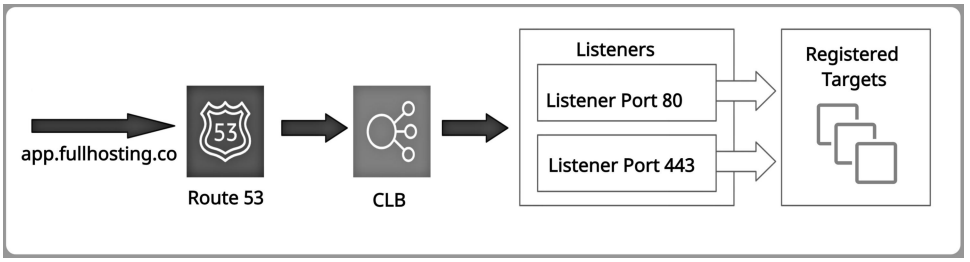
NLB

NLB is a *Network Load Balancer* used to balance *TCP/UDP* traffic. It works at the OSI model's 4th layer, which is the transport layer. It can process tens of millions of requests every second while retaining a low latency at the same time. It also performs *cross-zone load balancing*. It is suitable for the primary network performance of an application and supports multiplayer video games as well.

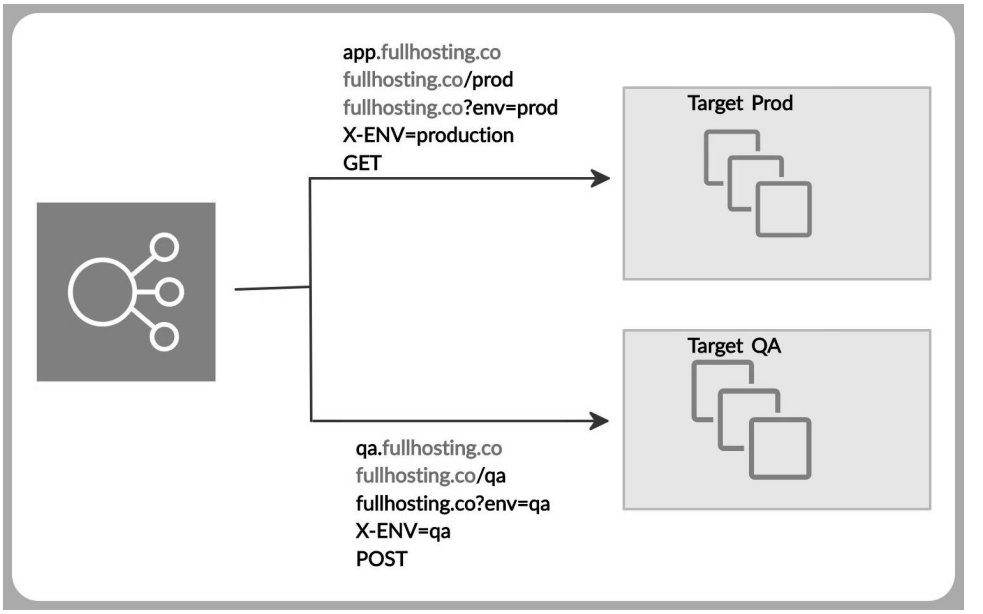
CLB

CLB is the *Classic Load Balancer* which is the AWS first load balancer. It can balance HTTP or TCP, but not simultaneously. It can use layer 4 to

balance TCP applications and specific features of layer seven, like sticky sessions. So, when the internet traffic flows to CLB, there are listeners who listen to ports, and instead of target groups, you have registered targets. There are EC2 instances only linked with the classic load balancer. CLB can also perform cross-zone load balancing. If an application is not responding, it responds with a 504 error in the case of a timeout. Since CLB is the earliest load balancer so, it is not preferred anymore. However, network load balancers and application load balancers are frequently used.



Let us assume a use case in which five different examples use Request Routing. The first approach to use this feature is to direct traffic depending on the subdomain. A subdomain app can go to a *Target Prod*, and *QA* can go to *Target QA*. The other way to use it is on the path where you can have *fullhosting.co/prod* and *fullhosting.co/QA* that will route it towards respected target groups. You can do it using a query string like *fullhosting.co?env=prod* and *fullhosting.co?env=qa*. You can also do it with the HTTP header. Lastly, you can use *GET* and *POST* methods here so each GET method can go to Prod, and each POST method can go to QA. Thus, the above five use cases explain Request Routing.



CloudFront

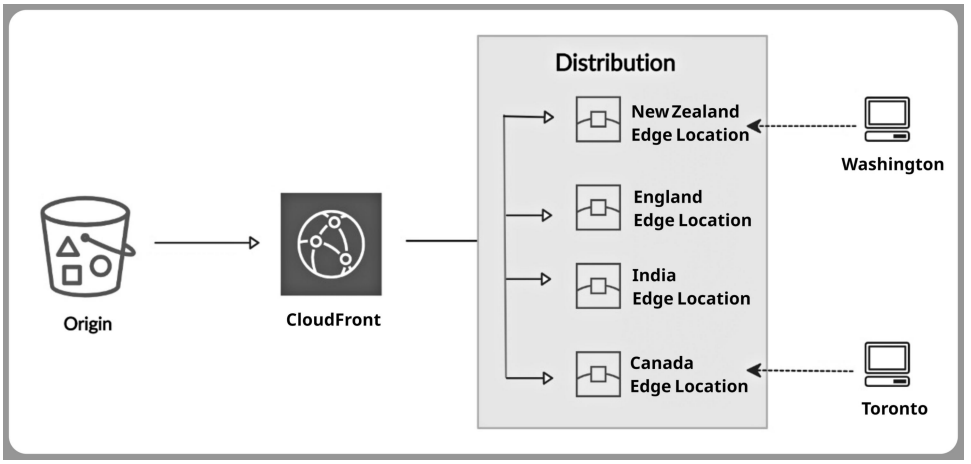
Introduction

CloudFront produces cached copies of a user website at different edge locations around the world. It is *a content distribution network (CDN)*. One must be aware of a content delivery network to understand the concept of CloudFront. CDN is a massive distribution of caching servers around the world. It contains the content stored in the origin servers and directs the viewers to the most suitable location to see the content stored in the cache. The nature of the content can be *Static* (does not change) or *Dynamic* (does change). CDN improves the scalability and performance of applications.

CloudFront is Amazon's worldwide content delivery network with massive capacity and scale. You can also use built-in security features for the best services. A user is in control of the service and can make changes during the work. It includes real-time reporting to monitor the performance and modify the application or the way CDN interfaces with

the application. Moreover, it has been designed for static and dynamic objects and video delivery.

Let us explain the concept of CloudFront by using a graphical representation of a CDN. You host your content on the internet, with S3 being the origin. The CloudFront server distributes a copy of the website on several edge locations. These edge locations are the servers that are closer to users worldwide. When the user accesses content from Toronto, it will not go to the S3 bucket; rather, it will send the traffic to the CloudFront. Then, the Cloud front will send it to the nearby edge locations to ensure the lowest latency for the user.



Chapter 7: AWS Security

Overview

We will discuss *AWS security* in this chapter as Cloud Security is the highest priority in Amazon Web Services. The AWS data is as secure as in traditional data centers. Security is the key concern for most cloud technology adoptions. However, many people who implemented cloud technology in the early days claim that the cloud solution has better security than on-premises security. AWS security works on *a Shared Security Responsibility model,* meaning that Amazon will secure its infrastructure. At the same time, you can have your security controls for the data and the applications you launch and store in the cloud. Not only the security level of AWS is higher than on-premises, but the cost of using this type of security for your applications is also very low as compared to on-premises.

Amazon has different security tools to help in implementing the best AWS security practices. AWS commonly uses some of these services, such as IAM, KMS, and Amazon Cognito. Let us start this chapter with IAM as it is the most important security feature in AWS.

AWS IAM

Introduction

IAM stands for *Identity and Access Management.* It is a web server that manages access to the AWS resources in a safe and secure way. With IAM, you can safely manage contact to AWS resources. You can also create different groups and let them access specific servers or limit them from using certain services. All of this is possible with identity and access management, and it comes without any extra charges. It allows you to authenticate or limit a certain set of users to access the AWS account or specific resources in the AWS account. Let us assume a use case in which there are three groups, including group1, group2, and group3, and the

IAM administrator wants to allow permissions to a different group of users. Let us assume the administrator wants to restrict group2 to access AWS resources. The IAM authorizes the administrator to allow access for a certain group to certain resources. In this case, the IAM admin can grant access to group1 and group 3 and restrict access to group2. So, that is the function of IAM.

How does it work?

Let us discuss a few elements that complete *IAM workflow*. There are six categories: Actions, Principle, Authentication, Resources, Request, and Authorization. Let us explain these core elements of IAM workflow.

Principle

An application or a person that can request operations (actions) on AWS resources is known as a *Principle*. An action on an AWS resource can be performed only by a principle as it is the main element of IAM. It is an entity that takes actions on AWS resources, which may be a role, a user, or an application that tries to access the AWS environment.

Authentication

It is the process of identity verification of the principle that wants to enter into the AWS network. It is the second element of IAM. You will be considered authentic if you enter the credentials or if you provide the required keys to verify that you are the same person as in records. To get Root User authorization, you have to use your email and password to log in from the console. As an IAM user, your account ID must b entered first, then your username and password. You must provide the two secret keys (access and secret access keys) for AWS CLI/API authentication.

Request

The *request* is another element that makes the IAM. If a principal wants to use the AWS console, a request is sent from the user (principal) to the AWS. The request carries information about the actions of the principal. A request can be of three types such as *Put*, *Get*, or *Delete* request. This data

includes resource and request origins on which the action needs to be done, such as EC2 or S3 bucket. The origin may be an AWS environment or any other Cloud Service provider. So, this data will be available in the request. AWS collects the request data into a request context, which evaluates and authorizes the request.

Authorization

In *Authorization*, IAM uses data from the Request Context to find similar policies and determines whether to accept a request or not. By default, all resources are denied according to the evaluation logic, so a request is only authorized by IAM if the matching policy allows every part of the request. In any permission policy, an Explicit allow overrides this default behavior. A permission policy may be identity-based or resource-based. If it is not the explicit allow, then it overrides the default denied**.**

On the other hand, if you deny service, it will overwrite all the allow statements and other statements. As a result, explicit deny will take place. By default, only the root users in the AWS account can access all the resources. You must be authorized by the policy if you do not log in as a root user.

Actions

Let us briefly discuss Actions now. After authentication and authorization of the requests, AWS approves the actions that you are going to take. With Actions, you can view, create, edit the content, and delete a resource. It is all based on the action that you are allowed to do. For instance, IAM supports about forty actions for a user resource, including:

- UpdateUser
- GetUser
- DeleteUser
- CreateUser

Resource

An object that exists in service is called a *Resource*. So, you can perform a set of actions on related resources. Anything is considered incomplete if it does not get mentioned in the action or resource. For instance, requesting access to the instance for an IAM role after removing it will result in a denied request.

IAM Components

Let us discuss a few more IAM components, which include Roles, End Users, Groups, and Policies. The End Users are the basic building blocks. When different end-users combine, they form a Group. Policies are the engines that allow or deny a connection. It means that access to a resource depends on the policy. Lastly, the Roles are temporary credentials that you can expect for an instance when they are needed. Let us discuss them one by one.

End-User

In an AWS environment, an *IAM End-User* refers to an entity that a user can create. It indicates a person or a service that communicates with the AWS environment. IAM User has a few credentials with it. By default, it does not have any secret key or a password. The root user creates a new user in the AWS account and assigns the desired credentials. These credentials get linked with the end-user. By default, a user cannot perform any operation in AWS. The benefit of having a one-to-one user is that you can assign permissions to the users individually. For instance, you might want to give administrative permission to some users to manage your AWS environment.

A user is an entity in the AWS account, so it is not bound to pay separately. It can start using the servers, and that will build under the account it is associated with. A separate user does not indicate separate payments for each user. They are just users of that account, and the entire account pays for the services it provides. The IAM user does not always represent a person. It is an entity that has credentials and related

permissions. Therefore, access to AWS resources can be securely managed by IAM.

Group

The collection of IAM users is known as the *IAM Group*. It specifies permissions for several users. It ensures that any permission granted to a group is also applicable to the user associated with that group. A group may have multiple users. However, a group cannot be nested. It means that a group can not belong to another group. Instead, only users can belong to a group or several groups. A group needs to be created because there is no such thing as a default group that automatically includes all users in a group.

Features of IAM

The main features of IAM are:

- *AWS account shared access:* The key feature of IAM is that it allows you to create unique usernames and passwords for individual users or resources and representative access.

- *Free of cost:* If you want to create additional users, groups, and policies, they are available for free. There are no extra charges for IAM security.

- *Multifactor authentication (MFA):* IAM supports MFA, where users can enter their username and password. Then they enter a one-time password from their devices - an additional authentication factor that uses randomly generated numbers.

- *Identity Federation:* You can make IAM trust Facebook or Google Account authentication method and allow access on its basis. It also enables its user to keep the same password for the cloud environment and on-premises.

- *Granular Allowance*: You can apply restrictions on requests. For instance, you can allow users to download data but prevent them from updating data through policies.

- *Password Policy*: You can reset or rotate a password locally with the IAM password policy. You can set rules for passwords as well. For instance, you can set a rule such as picking a password or the number of attempts to enter a password before denied access.

- *PCI DSS Compliance: IAM complies with the PCI DSS,* which is the data safety standard for the companies to manage branded credit cards from major card schemes.

Chapter 8: Application Integration Services

Overview

With *message queuing service* and *automating tasks* together in the AWS application integration, you can create reliable, asynchronous, and scalable applications. You may need some additional features while building an IoT, web, or mobile application on the cloud, as they play an important role in upgrading your application's functionality and efficiency. For instance, you can use SNS to create push notifications for your web or mobile users. You can add such types of features with AWS App integration. This chapter focuses on learning Amazon's Message Queuing Services, which are considered important from an exam perspective. These services include SQS and SNS. Let us discuss them in detail.

SQS

Introduction

SQS is an abbreviation of *Simple Queue Service,* which is a completely managed queuing service for scaling and setting apart the architectures of microservices, serverless and distributed systems. Let us have a comparison between a queuing system and a streaming system, which will help you understand the SQS. So, the *Queuing System* is a messaging system that offers asynchronous communication and sets apart processes via messages. These messages are known as events from the sender and receiver point of view.

On the other hand, the sender and receiver in the streaming system are known as producers and consumers. In queuing systems, the incoming messages are generally deleted when they get consumed, which is a simple communication process. It is not good for real-time scenarios because it is reactive. It means that the sender and receiver have to pull to decide the

action so that they can communicate with the queue and the messages. There are a few examples of queuing systems such as *Sidekiq, SQS,* and *RabbitMQ*, which is debatable because one may consider it as a streaming service.

In contrast, a *Streaming System* can respond to multiple consumers' events. If multiple users want to perform any event, they can do it since it will not be deleted instantly. The messages stay in the event stream for a longer time, enabling you to apply complicated operations. A big difference between both queuing systems is that one is reactive while the other is non-reactive. One allows you to perform multiple actions on messages and keeps them in the queue, whereas the other remove the messages and does not care about its operations. So, it was a comparison between streaming and queuing systems. Let us continue with SQS, which is a queuing system.

SQS is an *application integration service* that acts as a bridge of communication and connects isolated applications with the help of messaging and queues. It uses a queue, which is temporary storage for messages that need to be processed. You can understand it as going to a bank, and everyone is waiting in line, that is the queue. SQS is a pull-based service instead of push-based. You need the AWS SDK to interact with the queue and write the code that will publish messages to the queue. Then you will need AWS SDK to pull the messages to read them. Let us discuss the uses of SQS.

What is SQS used for?

In cloud applications, the most common ways of using SQS and other messaging systems are as follows:

Decouple Microservices: Messages are one of the best ways to establish contact between various components of the system in the microservice architecture. If your microservices (particularly serverless services) run in AWS, then SQS is a great option for that kind of communication.

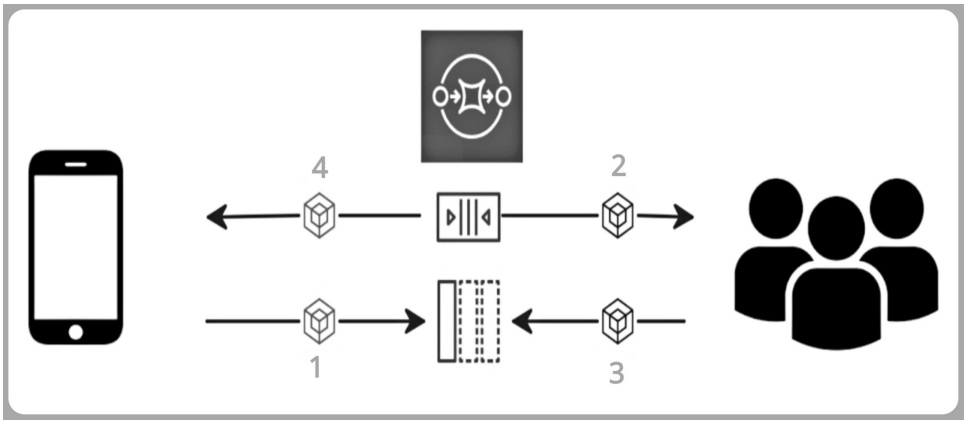
Send tasks between different parts of your system: To take advantage of SQS, you do not have to run a microservices-based application. You can also use it to communicate the tasks of one application to another system/application.

Distribute workloads from a central node to worker nodes: In wide distributed workflows like map-reduction operations, you will also find messaging systems. It is important for such operations that all tasks which need to be processed remain in the queue, distribute them quickly between the machines or functions which perform the work, and make sure that each part of the work is done only once.

Schedule Batch Jobs: SQS is an excellent choice to schedule batch jobs due to two reasons. First, it keeps a long-lasting queue for all the scheduled tasks, which means you do not have to monitor the job status. You can entirely depend on SQS to get the jobs done and to manage any retrieval even if the operation fails and the batch system return messages to the queue. Secondly, it merges with AWS Lambda to perform batch operations; if you use AWS Lambda with SQS, it automatically runs the Lambda functions for data processing once the data is available to them.

Use Case

Let us assume a use case for SQS. Consider a mobile app and a web app, and you want them to communicate with each other. The mobile app will send a message to the queue by using the AWS SDK. Now the web app will use AWS SDK and pull the queue whenever it wants. It is, therefore, up to this web application to code in how often it will check for a specific message in the queue. If a message appears, it will pull the message down, use it, and report the queue that it has consumed the message. It will inform the queue to proceed and remove this message from the queue.



Now, the mobile app wants to know if the web app has consumed the message. It will check from time to time that if the message is still in the queue or not. If it is not in the queue, the mobile app will know that the other app has consumed the message. Therefore, it is the process of using SQS between two applications.

Limits & Retention

Let us discuss some SQS limitations. Message size in SQS ranges from 1byte to 256KB. If you need to go farther than the limit, you can use *Amazon SQS extended client library for Java* which allows you to extend the message size up to 2GB. It works in a way that the message will be stored in S3, and the library will refer to that S3 object.

Message Retention is the duration in which SQS keeps a message in the queue before dropping it. It is set to four days by default. This message retention can be set from 60 seconds to 14 days maximum.

Queue Types

The Standard Queue and *FIFO* are the two different queue types. *Standard Queue* provides a nearly unlimited number of transactions per second while transacting. It is similar to messages and guarantees to deliver the message at least once. You can potentially deliver multiple copies of a message, but it can cause things to go wrong. Standard queue tries best to keep the messages in the order they are sent, but there is no

surety of it. On the other hand, you need to use *FIFO* (First In, First Out) if you want the guarantee that the messages stay in their order. The message is sent exactly once in the FIFO Queue and remains visible until the consumer deletes it without introducing duplicates into the queue. It provides one-time processing where the order of sending and receiving a message is well-preserved. However, it has a limit of 300 transactions per second for a single API action.

Visibility Timeout

Visibility Timeout is the duration defined for a queue object that is hidden from the queue and other users once it is fetched and processed by a consumer. The main goal is to avoid multiple consumers (or the same consumer) who repeatedly consume the same object. With this feature's help, you can prevent an application from reading a message when some other application is already reading it, or you can avoid doing the same amount of work that someone else has already performed.

The Visibility Timeout is the duration in which the messages are not visible in the SQS queue. When a reader selects a message, you can set a visibility timeout between 0 to 12 hours for that message so that no one else can access or see it. Visibility timeout is set to 30 seconds by default. When the reader completes the processing successfully, it can delete the message from the queue; otherwise, the message would appear again in the queue for another customer to select it again. If the consumer fails to complete it in the given visibility timeout frame, he must kill the job; otherwise, it might end up with the same messages delivered twice.

Polling Types

Polling is the way of retrieving messages from the queue. *Short Polling* and *Long Polling* are the two ways of polling in SQS. The short polling is used by SQS by default. It returns the messages immediately, even if the message queue is empty. When there is no message to pull, the short polling will not come in handy because you are simply making calls without any reason. However, there are some scenarios where you need to

retrieve the message immediately, and short polling is useful in such situations. In most use cases, the long polling should be used because it waits for the message to appear in the queue or till the timeout expires. As soon as the messages are available, you can retrieve them from the queue with the help of long polling. It is not set by default, so you have to set it manually. It also reduces the price because you can limit the number of empty receives. You can enable long polling within SDK and schedule the received message requests with a waiting time.

SNS

Introduction

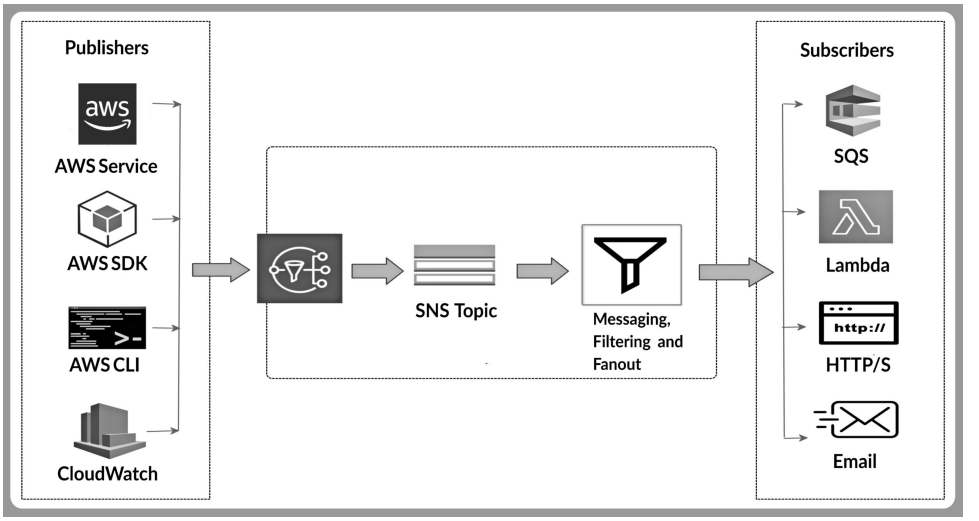
SNS stands for *Simple Notification Service*. Using SNS, you can send notifications and subscribe via *email, text messages, Lambdas, web books, mobile notifications,* and *SQS*. Let us learn the concept of the *pub/subsystem* for a better understanding of SNS. A Pub/Sub is a *publish-subscribe* pattern widely used in messaging systems to convey messages between various components of the system without revealing their identities to each other.

In this system, *the publisher* is the sender of messages, and *the subscriber* is the one who receives them. The publisher sends a message to an *Event bus* instead of sending it to the receiver directly. Then, the Event bus categorizes the messages into different groups/classes, and the receiver of the message subscribes to these groups. So if a new message arrives within their subscription, it is immediately sent to them.

The *message buses* are the servers on which messages are sent, stored, and collected. So, a variety of topics can be included in each message bus. *Topics* are used to differentiate between various types of communications since not all subscribers need to view each message. Publishers do not know about their subscribers and vice versa. Subscribers need not pull the messages from the event bus as these messages get pushed towards

subscribers automatically. Furthermore, Events and messages are interchangeable terms in the pub/sub environment.

Now, let us discuss the SNS. The SNS is a fully managed and highly available pub/sub messaging service which allows you to decouple (set apart) microservices-based architectures, serverless and distributed systems. When we say decoupling, we refer to application integration which is like a family of AWS services connecting one service to another, and SNS is an application integration service. A typical example of an AWS pub/sub system is shown in the following diagram. The publishers are on the left side, subscribers on the right side, and the event bus in the center is SNS. The publisher refers to any service which programmatically uses the AWS API. In this case, there are AWS SDK and AWS CLI as publishers that use AWS API underneath to publish their messages onto an SNS Topic. The CloudWatch can also publish to SNS because you would use them for Billy alarms. There can be another service on AWS that can publish to SNS topics.



On the other hand, the subscribers subscribe to a specific topic in the event bus. We have subscribers with several outputs, such as *SQS, Email, HTTP/S protocol,* and *Lambda*. The benefit of the pub/sub system is that you can have as many subscribers as you need and scale them all separately. It depends on the amount of time a single subscriber takes

for every message to process and the traffic on the given topic they subscribe to.

Therefore, the publishers will push the messages to the SNS topic to get into the event bus. After that, the subscribers will subscribe to the SNS topic so that the events get pushed towards them. The event bus 'SNS' takes published messages as an SNS topic, filters them, and then forwards them to all the registered subscribers.

SNS Topic

A *Topic* is an access point or a communication channel where you can subscribe using a phone number, email address, or a URL to receive its relevant notifications. A topic combines multiple subscriptions together. You can deliver a topic to multiple protocols at once. When a new notification is posted, it is filtered to comply with the policies of subscribers and, if successful, sent to the specified endpoint. The publisher is not concerned about the subscribers' protocol. You can encrypt your topic with the help of KMS. Note that SNS does not guarantee the delivery of messages as the SQS does. Moreover, if the sending is unsuccessful for some reason, it will not retry. If it happens, it will delete the message.

Use cases

SNS can be used frequently for sending system notifications to the subscribers of a given topic. They include warnings when the server is down, a new user registers, or when it performs a regular check. Since different users take an interest in different topics, some of them are created to avoid unnecessary emails or texts to the entire company. There is another popular use case, which includes a ‘fan-out’ scenario that uses Amazon SQS. In this scenario, a new message is sent to the topic at every new purchase from the store. After that, SNS sends the same message among its subscribers. These messages are directed to an email address (to send a notification of a bought product for a manager), an HTTP/HTTPS endpoint to update a client record in *CRM*, and to several SQS queues as well. These queues are as follows:

- Queue 1 – send to a warehouse to allow the workers to select a product and forward it to the next team.
- Queue 2 – deliver to a purchasing department to keep them updated about remaining stock.
- Queue 3 – send to the shipping team to create a shipping tag and prepare the packaging.

Testing the internal workflows is another use case of the SNS-SQS collaboration. You can add a fourth SQS queue in the above example. It will deliver all order confirmations to the development environment of your application. After that, you can use that production data to get rid of the errors and introduce changes to the production environments. However, if you want to test the email workflows, an external tool will be good enough to test them.

Subscriptions

Subscriptions are created on a topic. So, if there is an email subscription, then the endpoint will be an email as well. For getting an email subscription, first, you enter an email address, click on '*Create Subscription*,' and fill in the options. Then you need to select your protocol from a detailed list. We have different protocols such as *platform application endpoints, HTTP/s, Email, Email-JSON, SQS, SES, SMS,* and *Lambda*. A *Platform Application Endpoint* is used for the mobile push, which enables you to get a notification system on various devices such as cell phones and laptops. The protocols *HTTPS* and *HTTP* are used for web books. HTTP/S is actually an API endpoint to the users' web applications that listens to incoming SNS messages for sending emails.

The *Email-JSON protocol* will send you JSON objects via email, where you can get email notifications that are text-based messages. After Email-JSON, there is *SQS* where you can send an SNS message to SQS. Then we have another service for sending out emails, which is the *Simple Email Service*. It is very useful for internal email alerts to the customers as you do not necessarily need your customized domain name. It can be used for

billing alarms or checking any sign-up activity on your platform. You can also use *SMS protocol* to send text messages. Lastly, one can also use *SNS* to trigger the *Lambda* function for messages alert, which is a very effective feature.

Application as Subscriber

We are going to discuss Platform-Application-Endpoint in a bit more detail as it is used for mobile push. The mobile endpoints available are *Amazon Device Messaging* (*ADM*), *Google's Firebase Cloud Messaging, Apple's APNs*, and *Baidu*. For *Microsoft*, we have got *Window Push* (*WNS*) and *Microsoft Push* (*MPNS*). Using the *Application as a Subscriber protocol*, you can send alert messages to the mobile endpoints. The advantage is that notification messages will appear as a message alert, sound alerts, or even badge updates in the mobile application when you push messages to these mobile endpoints.

To be continued…