AWS: AWS Fundamentals Project

Goals

This AWS Fundamentals project aimed to understand AWS services through hands-on experience by using AWS Management Console and CloudFormation templates with VS Code. After acquiring AWS certified Cloud Practitioner, I aspire to deepen my understanding of AWS through hands-on experience as I am not familiar with using cloud services while working in a food manufacturing industry and cloud services are not tangible compared to food products which makes me hard to understand the back-end processes occurred in AWS environments. This project helped me understand fundamental AWS services by manually deploying and automating resources. Particularly, I created a VPC manually using the AWS Console and configured public subnets and private subnets across different availability zones to understand how AWS resources connect to the Internet and control both inbound and outbound traffic while enhancing security. Furthermore, I automated this process through Infrastructure as Code by implementing CloudFormation templates and creating YAML files in VS Code. I built stacks including different AWS resources such as VPCs, subnets, security groups, route tables, auto scaling groups, an Amason S3 bucket hosting a simple static website and a relational database. All these hands-on activities were proceeded with AWS fundamental concepts which enhances my deep understanding of AWS global infrastructure and its applications.

AWS Global Infrastructure Review

AWS operate distinct AWS regions such as us-west-1 where independently isolated availability zones are designed to provide stable and fast services for users. Each availability zone has its own data centers which have redundant power, network and connectivity for fault tolerance to provide reliable, scalable, low latency services for users. CloudFront is AWS content delivery network service providing fast and reliable connectivity for users using AWS services around the globe by strategically locating Edge Locations near them.

I learned the functions of public subnets and private subnets located within VPC. Public subnets connect directly to the Internet using a public route table, a router and an Internet Gateway (IGW). In contrast, private subnets cannot directly connect to the Internet for enhanced security. However, they can indirectly communicate with the Internet through a NAT gateway located in a public subnet. Within a VPC, public subnets and private subnets can communicate with each other using private route tables. Typically, outbound traffic responding to users' requests is allowed but inbound traffic from users to access AWS resources within VPC is not allowed. Web servers handling users' requests are located in public subnets while back-end databases and other business logics are located in private subnets to enhance security.

When launching a VPC, CIDR Blocks are used to assign an IP address range to the VPC. CIDR notation specifies the IP address range and determines the number of distinct AWS resources which can be assigned within the VPC. There can be distinct VPCs within AWS whereas they are independent of each other. CIDR blocks enabled effective management of grouped AWS resources and subnets within a VPC Security groups are attached to instances and other AWS resources while Network Access Control Lists (NACLs) are attached to subnets to control both inbound and outbound traffic. Security groups are stateful, effectively remembering allowed traffic and applying the same rule to the same traffic whereas NACLs are stateless, checking every inbound and outbound traffic individually whenever traffic accesses subnets.

Additionally, I learned about Identity and Access Management (IAM) which provides credentials to users, groups and roles, enabling secure access to AWS resources while enhancing security management. IAM users allow individual users to access AWS resources while IAM groups allow groups including several users with similar roles to access AWS resources rather than controlling individual users. IAM roles are temporary credentials provided for applications or services to allow them to interact with AWS resources rather than providing permanent credentials which can pose security issues. Policies define permissions

specifying whether access to AWS resources is allowed or denied. The principle of least privilege means that users, group or roles must be provided the least permissions necessary to perform their specific tasks to enhance security management within AWS.

Explaining Results for configuring a VPC with public and private subnets

(1) Creating VPC

I created a VPC and assigned an IP address range using a CIDR block. I configured 2 distinct availability zones within VPC, each containing 1 public subnet and 2 private subnets. Public subnets have a common route table connecting to the Internet Gateway (IGW) while private subnets have their own private route tables communicating with public subnets and other AWS resources locally. When private subnets route outbound traffic to the Internet, this must be routed through a NAT gateway in a public subnet indirectly and securely connected to the Internet while enhancing security. VPC endpoints can be Amazon S3 bucket and Dynamo DB, enabling direct connections to AWS resources within VPC without using the public connectivity. However, installing a NAT gateway in each public subnet costs a lot of money. Instead, I installed a Bastion Host (EC2 instance) in a public subnet, connected to an EC2 instance in a private subnet within the same Availability Zone (AZ) and then to another EC2 instance in a different Availability Zone. I also generated a key pair as bastion to securely connect to the Bastion server from my local computer and a security group for this bastion server only allowing inbound traffic from my own local computer. I can send inbound traffic to the bastion instance by using its public IPv4 address.

(2) SSH access to a private subnet

SSH is an encrypted protocol using a key pair to securely control remote computers and servers effectively. I used the key pair when connecting to an EC2 instance in the public subnet and then SSH connection to another E2 instance in the private subnet within the same availability zone. Rather than creating another bastion host in a public subnet in a different availability zone, I sent a request to another EC2 instance in the private subnet in the different availability zone. The public IPv4 address of my local computer must be allowed by the security group of the instance installed in the public subnet while using the bastion key pair for the public connection. In addition, the security group for the instance in the public subnet must be allowed by the security group for the private subnet within the same availability zone. In contrast, when connecting to another instance in the private subnet in the different availability zone, SSH connection using the key pair was not necessary. Instead, sending the request from the instance in the private subnet must be allowed by the security group for the counterpart private instance.

(3) Automate the same process by using CloudFormation

I learned about Infrastructure as Code and CloudFormation automating the entire processes of establishing AWS resources in AWS environments consistently using VS Code and creating YAML. Infrastructure as Code means installing AWS resources within AWS environments using code not relying on manual processes to configure resources fast and efficiently. CloudFormation is the most famous tool for Infrastructure as Code, which can reduce manual errors and generate several versions to help developers track the changes made. These CloudFormation templates can be reused, which automates the configuration processes and improves the efficiency. CloudFormation templates are created by JSON or YAML.

First, I create a ‘MyVPC’ and DNS server which support AWS resources within VPC to communicate with other resources effectively using IP address. And then, I created one public subnet and 2 private subnets in each availability zone. When configuring the public subnet named as ‘PublicSubnet1A’, I mapped public IP address to ensure the public access to the Internet. In contrast, when configuring private subnets, I did not map the public IP address as they are not directly connected to the Internet. Finally, I deployed ‘PublicSubnet1A’, ‘AppPrivateSubnet1A’ and ‘DataPrivateSubnet1A’ in us-west-1a and then ‘PublicSubnet2B’, ‘ppPrivateSubnet2B’ and ‘DataPrivateSubnet2B’ in the other availability zone in us-west-1b.

Second, I defined an Internet Gateway and then attached it to the 'MyVPC' to ensure public access to the public subnet in us-west-1a. I generated a public route table attached to the public subnet and public route allowing all outbound IPv4 traffic to the Internet Gateway (IGW).

Third, before creating an EC2 instance called as Bastion Host in the public subnet in us-west-1a, I learned about AWS computing services such as Amazon EC2, AWS Lambda, AWS Elastic Beanstalk and Amazon ECS. Amazon EC2 has diverse instance types such as general purpose, compute-optimized, memory optimized and storage optimized and different pricing models such as on-demand instances, reserved instances and spot instances. On-demand instances are ideal for short-term and unpredictable usage as consumers are allowed to pay for what they used. In contrast, reserved instances offer significant discounts on long-term usage such as 1 to 3 commitments and are ideal for steady workloads. Spot instances provide cost savings by bidding on unused EC2 instances but they can be interrupted when prices exceed bids. AWS Lambda is a serverless computing, completely managed by AWS by uploading code. It does not need any manual management process of underlying infrastructure. AWS Elastic Beanstalk is a Platform as a Service for managing and deploying applications, managed by AWS but provides more flexibility for managing underlying EC2 instances. Amazon ECS manages docker containers and runs applications on cluster of EC2 instances.

I created an EC2 instance in the PublicSubnet1A named as a BastionHost and used the same key pair I have created through AWS Management Console. I created a security group of the instance in the public subnet and put the IPv4 address of my local computer to SSH to the Bastion Host, which allows inbound traffic rule in the security group. Additionally, I created a private instance named as 'App1' in AppPrivateSubnet1A with the security group allowed SSH access from the Bastion Host in the public subnet within the same availability zone. I configured another private instance named as 'App2' in AppPrivateSubnet2B with the security group. I did not SSH to the App2 instance. Instead, I made the security group to allow the ping request from App1 by allowing ICMP echo requests originating from App1.

Explaining results for IAM, Load Balancing, S3 bucket, RDS and Auto Scaling Group

(1) IAM users, IAM groups, IAM roles, and IAM policies

I created an IAM user and granted it 'AdministratorAccess' permission and IAM group named 'Developergroup' to attach this user to the group. IAM users are individuals which are granted permissions through IAM policies. IAM groups involve several IAM users who have similar roles and allow us to manage users effectively by providing permissions collectively to the users in the same group. IAM roles can be attached to AWS resources, such as EC2 instances, to grant permissions for specific actions on other resources in AWS environments. IAM policies define permissions to IAM users, groups and roles by specifying actions and effects (allow or deny) applied to specific resources. In this project, I created a S3 bucket custom policy which allows to get all objects stored in any configured S3 bucket.

(2) AWS Elastic Load Balancing

Before configuring an Application Load Balancer through a CloudFormation template, I learned about what is Elastic Load balancing. ELB distributed incoming traffic across back-end target servers to enhance the fault tolerance of applications. If demands peaked posing a huge number of requests, ELB is necessary to distribute those incoming traffic across multiple EC2 instances to retain low-latency and high performance of applications. There are 2 different Elastic Load Balancers, such as Application Load Balancers and Network Load Balancers. Application Load Balancers (ALB) distribute HTTP/HTTPS requests across multiple EC2 instances based on host names and URL paths. It allows more advanced incoming traffic distribution based on their requests ideal for moder applications, such as containers, microservices and Kubernetes. Network Load Balancers (NLB) distribute TCP, UDP and TLS traffic to enables ultra-low latency and high performance necessary for real-time and high throughput applications.

I created 2 public subnets with the public route table to connect instanced to the public Internet. I configured an application load balancer distributing incoming HTTP request from the public Internet to the target configured WebServerInstance1A and WebServerInstance2B in distinct public subnets. If incoming traffic reached out one of the web servers, a web page would be displayed hosted on the EC2 instance.

(3) S3 bucket

An Amazon S3 bucket is a storage managed by AWS, storing files, documents and images ensuring scalability, availability and durability. It also hosts a static web page and bucket policies are attached to a S3 bucket to manage access to the bucket. I created a S3 bucket hosting a static website by creating a index.html file and allowed the public access to the bucket. In addition, I created a bucket policy attached to the bucket to allow public access to the bucket to get all objects stored.

(4) RDS

There are 2 main categories of database such as relational database and non-relational database. Relational database stores data in an organized and structural way such as tables. Tables are associated with each other to provide more detailed information. In contrast, non-relational databases typically have more flexible schemas and may handle relationships between entities differently compared to relational database. Amazon RDS is relational database and their provisioning, configuration, patching and backups are automatically managed by AWS. RDS is built on the top of EC2 instances and RDS supports various database engines such as MySQL, PostgreSQL, MariaDB and ORACLE Database. RDS can be replicated in different availability zones to enhance fault tolerance and high availability. If primary database fails, standby database is automatically operated for failover. Read replicas prevent master database from being overloaded by separating read and write operations and assuming read operations instead of master database. RDS automatically backs up to Amazon S3.

(5) Auto Scaling Group

An Auto Scaling Group (ASG) consists of a collection of EC2 instances automatically increasing or decreasing the number of instances based on the amount of incoming traffic and pre-defined parameters. These parameters defines both minimum and maximum number of instances allowed in the scaling group. ASG can be combined with AWS Elastic Load balancers to distribute incoming traffic across multiple instances by adjusting the number of instances to improve performance and fault tolerance of applications.

I configured an Auto Scaling group in distinct public subnets with a minimum of 1 instance, a maximum of 3 instances and a desired capacity of 2 instances. I set up a CloudWatch alarm to monitor CPU utilization. When CPU usage exceeds 70%, the alarm triggers a scale-out policy to increase 1 instance in the auto scaling group. This scaling process occurs every 300 seconds which ensures high performance applications.

Conclusion

Through these practical activities through AWS Management Console and CloudFormation templates, I could imagine the back-end processes interacted with a front-end web page by sending incoming traffic or requests. One thing that I struggled with when learning about AWS services for certification was imagining the entire behind-the-scenes process for deep understanding. This project enables not only reviewing fundamental concepts of AWS services but also applying the concepts to real scenarios through hands-on experience. Infrastructure as Code is the most effective way to configure AWS services effectively and establish consistent environments by adjusting templates while reducing manual errors posed by human interventions. It was a meaningful experience for me to configure essential AWS services by utilizing CloudFormation templates in person.