Ecole Supérieure
d'Informatique et du Numérique
COLLEGE OF ENGINEERING & ARCHITECTURE

Building a Highly Available, Scalable AWS Web Application

ING 4 CLOUD COMPUTING & VIRTUALIZATION

Developed by:

ALI EL BELLAJ

AHMED REDA SIKILLI

OUIAM NORI

REDA STITOUANE

College year :2023/2004

1- INTRODUCTION:

The AWS project focuses on the deployment of a cloud-based web application utilizing Amazon Web Services (AWS) infrastructure and services. The project aims to demonstrate the capabilities of AWS in building scalable, reliable web applications for various purposes. the project also involved estimating the associated costs and implementing cost optimization measures to ensure efficient resource utilization and budget management while staying within the boundaries of reliability and AWS best practice.

During the deployment phase, careful consideration was given to the selection and configuration of AWS services to minimize costs while meeting the requirements of the application. AWS offers a range of pricing options, including pay-as-you-go, reserved instances, and free tiering (under certain condition), allowing for flexibility in resource provisioning based on budget constraints and usage patterns.

To optimize costs, the project employed various strategies such as rightsizing, which involves selecting the appropriate instance types and sizes based on workload requirements to avoid over-provisioning and unnecessary expenses. Additionally, resource monitoring and insights tracking were implemented to track resource usage and identify opportunities for optimization.

2- ARCHITECTURE OVERVIEW:

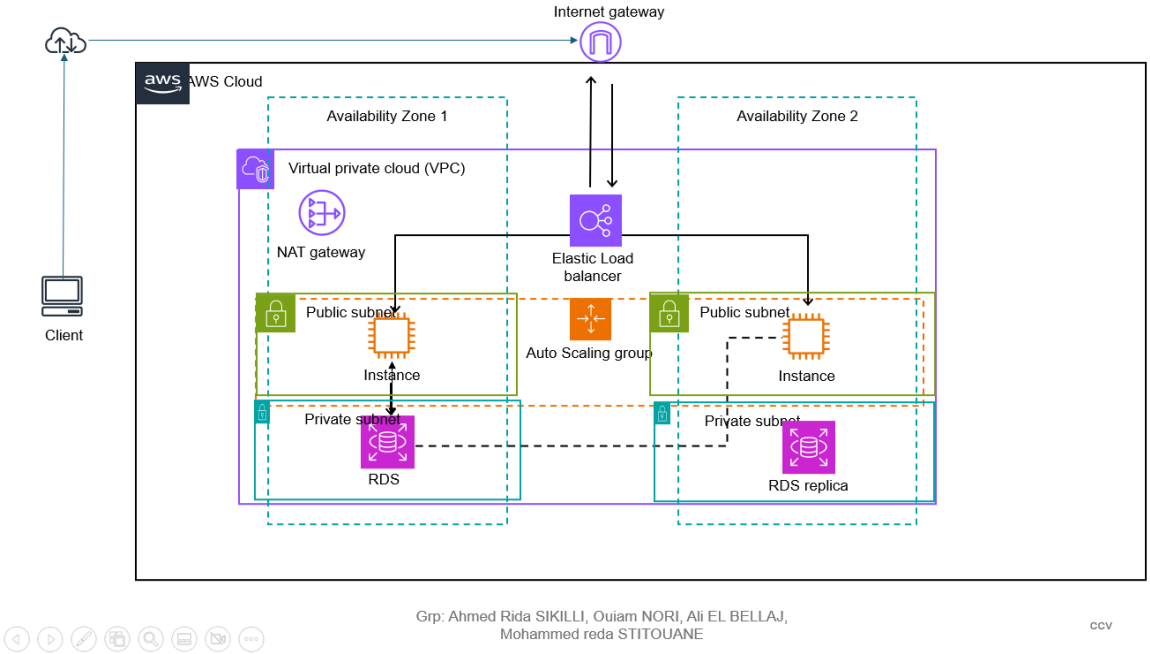

Figure1: main architecture

The architecture diagram provides a comprehensive view of the AWS infrastructure setup for hosting the web application. It encompasses networking, compute, and storage components to ensure a scalable, resilient, and secure environment.

Networking:

The networking architecture of the web application was designed to provide a secure and scalable environment for hosting the application components on the AWS cloud infrastructure. The architecture utilized Amazon Virtual Private Cloud (VPC) aggregating two availability zones, along with multiple subnets, both public and private (two for each availability zone), to control and manage traffic flow effectively.

Virtual Private Cloud (VPC): A dedicated VPC was provisioned to provide a private network environment containing an average of 65000 IP addresses. The VPC acted as a isolated networking environment in the AWS cloud, allowing full control over network settings, including IP addressing, routing, firewall setting and access control.

Subnet configuration: Two public subnets were created on two different availability zone, to host resources that needed to be accessible from the internet, such as load balancers and EC2 instances. These subnets were associated with a route table that had an internet gateway attached, allowing outbound and inbound internet traffic. The same internet gateway was used for both public subnets to reduce the overall cost. Two private subnets were created to host resources that did not require direct internet access, such as databases, and spare EC2 instance. These subnets were not directly accessible from the internet and were associated with a route table that routed traffic through a NAT (Network Address Translation) gateway for outbound internet access, as demonstrated in the diagram below.

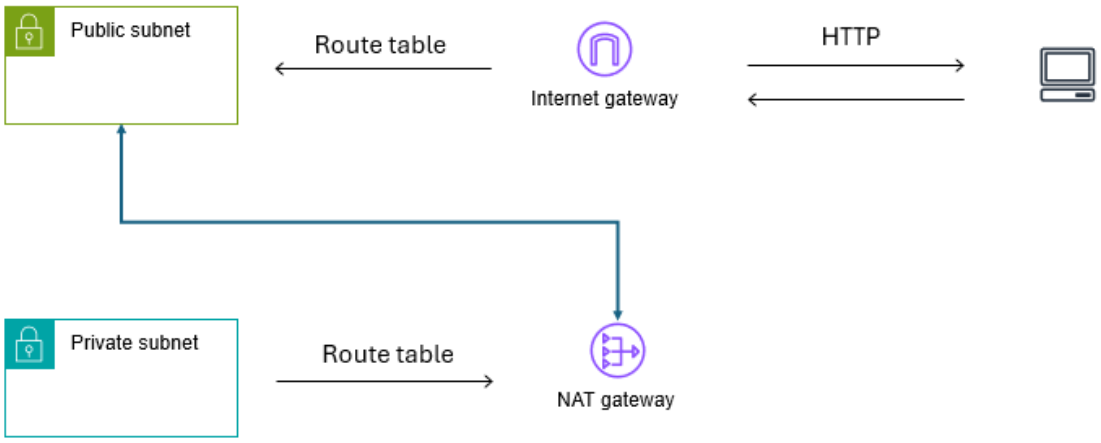

Figure 2: subnet configuration

Security groups: many Security groups were configured to control inbound and outbound traffic. Acting as virtual firewalls for EC2 instances and managed databases, security groups defined the rules that allowed or denied traffic based on protocols, ports, type of request and IP addresses. By applying security groups, access to resources was restricted, enhancing the overall security posture of the infrastructure.

Internet Gateway (IGW): An internet gateway was attached to the VPC to allow communication between the VPC and the internet. This facilitated inbound and outbound internet traffic for resources hosted in the public subnets.

NAT Gateway: A NAT gateway was deployed in each of the public subnets to provide outbound internet access for resources hosted in the private subnets. The NAT gateway allowed private subnet resources to initiate outbound connections to the internet while preventing inbound traffic from reaching those resources directly.

Optionally, both NAT and INTERNET gateways can be replicated for enhanced scalability, redundancy, and workload distribution, mitigating the risk of single point of failure (SPOF).

Compute:

The compute architecture of the web application was designed to provide scalable, efficient, and high-performance resources to handle the processing needs of the application. The architecture utilized a combination of Amazon EC2 instances, load balancing, and auto-scaling to ensure high availability and reliability.

Amazon EC2 Instances: Amazon Elastic Compute Cloud (EC2) instances were deployed to host the application to ensure high availability.

Note: This report focuses on the deployment of the web application on AWS, rather than its development.

The compute configuration included:

Main EC2 Instance: An EC2 instance of type t3.micro was deployed in the public subnet to handle direct traffic from the internet through the internet gateway. This instance served as the primary application server, processing incoming requests and data.

Spare EC2 Instance: A secondary EC2 instance of the same type was deployed in the private subnet to serve as a backup. This instance provided redundancy and could take over in case the main instance failed or needed maintenance.

Load Balancer: An Elastic Load Balancer (ELB) was configured to distribute incoming traffic across multiple EC2 instances. The ELB distributed incoming application traffic across the EC2 instances in the public subnet of both the availability zone within the main vpc, ensuring even load distribution and preventing any single instance from becoming a bottleneck.

Auto Scaling Group: An Auto Scaling group was set up to automatically adjust the number of EC2 instances based on the application's demand, ensuring optimal performance. The Auto Scaling group was configured with an Amazon Machine Image (AMI) of the main EC2 instance in the public subnet containing the application. This allowed new instances to be launched with the same configuration and software as the main instance, it was deployed across two Availability Zones (AZs) to ensure high availability and fault tolerance. This setup is an AWS best practice and ensures that the application remained available even if one AZ experienced an outage. Scaling Policies: Scaling policies were defined to automatically add or remove instances based on CPU utilization or other performance metrics. This ensured that the application could handle varying levels of traffic efficiently, scaling out during peak times and scaling in during low-traffic periods.

Additionally, security groups were configured for the instance to control the inbound traffic on necessary port (HTTP, HTTPS), and for allowing the ec2 instance to access the RDS database. A private EC2 Instances Security Group was configured to allow inbound traffic only from within the VPC with the notation (0.0.0.0/0), ensuring that the private spare instance is not directly accessible from the internet.

Database:

The database architecture was designed to provide a robust, scalable data storage solution for the web application. Leveraging Amazon Relational Database Service (RDS), which is a fully infrastructure managed AWS Platform as a service solution. the setup ensures efficient data management, high availability, and fault tolerance. The architecture also incorporates a read replica deployed on a different availability zone than the main database to optimize read-heavy operations and enhance overall database performance.

The primary database instance was set up using Amazon RDS, offering a managed relational database service that handles routine database tasks such as provisioning, patching, backup, recovery, and scaling.

The RDS instances were a type t3.micro and a allocated storage of 20 GB, the setup is a balance between cost and performance, making it suitable for low to moderate database workloads.

To improve read scalability and offload read-heavy traffic from the primary database, a read replica was deployed. It asynchronously replicates data from the primary database instance, ensuring that read operations are handled efficiently without impacting the performance of the primary database.

When creating the RDS instance, it automatically creates a default private subnet within the project vpc. It can be changed to public by adding a route to the internet gateway or the NAT gateway.

Linking the database to the ec2 instances:

Security groups play a crucial role in controlling access between the EC2 instances and the RDS database. Both the EC2 instances and the RDS database are configured with security groups that allow necessary traffic while ensuring security. AWS provide an easy way to link the instance, by doing so, security groups are automatically created enabling inbound and outbound traffic as shown is figure down below.

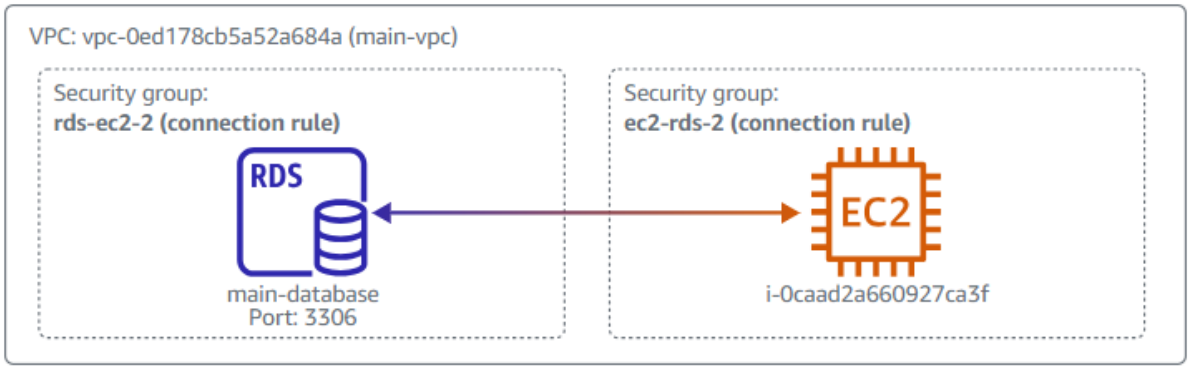

Figure3: database connection

Security groups associated with the EC2 instances allow outbound traffic on the database port to the RDS instance by specifying the type MYSQL/aurora traffic.

The security group associated with the RDS instance allows inbound traffic from the security groups of both EC2 instances. This ensures that only these instances can access the database, enhancing security.

3- TESTING AND DEPLOYMENT:

The final phase of the project involved testing and deployment of the web application to ensure it met performance, reliability, and scalability requirements.

To validate the performance and scalability of the application, extensive load testing was conducted. The goal was to ensure the application could handle high traffic volumes and maintain low latency. This was done by sending an average of a thousand request per second to ensure if the auto scaling is properly working. The load test results are summarized as follows:

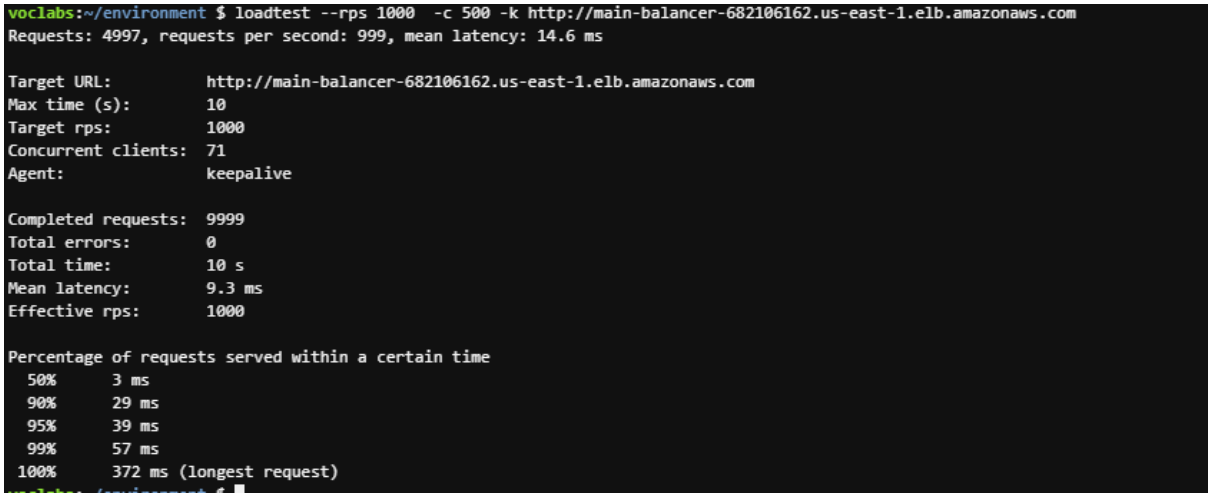

Figure 4: Load test results showing high performance with an average latency of 9.3 ms and 99.9% of requests served within 57 ms.

Test Tool: loadtest command-line tool was used for generating high volumes of traffic.

Target URL: The load test targeted the Elastic Load Balancer (ELB) URL.

4- COST ESTIMATION:

As part of the AWS deployment project, a detailed cost estimation was conducted to forecast the expenses associated with running the infrastructure and services. The estimation covered various AWS services used in the deployment, considering both monthly and annual costs.

The tool used for the estimation is the AWS Pricing Calculator which is a free web-based planning solution to create cost estimates provided by AWS.

It is important to know that the AWS PC provide only estimates, the final cost may be higher or lower than stated.

The overall cost estimation for the project is as follows:

Upfront Cost: $0.00 USD

Monthly Cost: $971.75 USD

Total 12 Months Cost: $11,661.00 USD (includes any upfront costs)

The figures down below provides a detailed breakdown of the monthly cost:

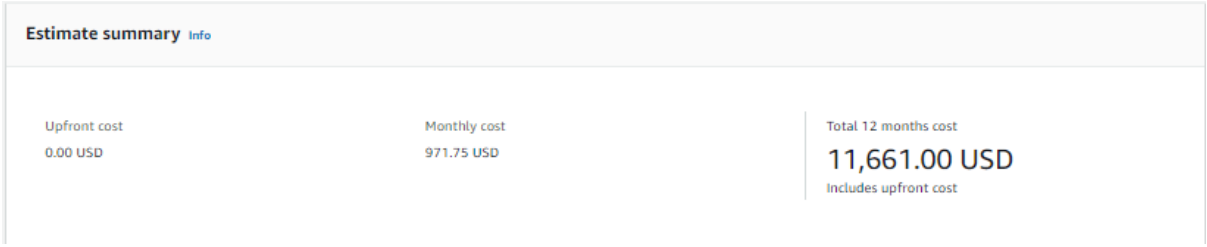

Figure 5: Monthly cost and total twelve months cost.

Service Name	Status	Upfront cost	Monthly cost	Description	Region	Config Summary
Amazon Virtual Priv...	-	0.00 USD	720.40 USD	-	US East (N. Virginia)	Working days per month (22), Number of Site-to-Site VPN Connections (0), Number of subnet associ...
Amazon EC2	-	0.00 USD	16.94 USD	-	US East (N. Virginia)	Tenancy (Shared Instances), Operating system (Linux), Workload (Consistent, Number of instances: 2)...
Elastic Load Balancing	-	0.00 USD	52.20 USD	-	US East (N. Virginia)	Number of Application Load Balancers (1) Number of Classic Load Balancers (1), Processed bytes per ...
Amazon RDS for My...	-	0.00 USD	53.72 USD	-	US East (N. Virginia)	Storage amount (30 GB), Storage for each RDS instance (General Purpose SSD (gp2)), Nodes (1), Inst...
Amazon RDS for My...	-	0.00 USD	128.08 USD	-	US East (N. Virginia)	Storage amount (30 GB), Storage for each RDS instance (General Purpose SSD (gp2)), Nodes (1), Inst...
AWS Secrets Manager	-	0.00 USD	0.40 USD	-	US East (N. Virginia)	Number of secrets (1), Average duration of each secret (30 days), Number of API calls (1 per month)

Figure 6: The cost estimation summary and detailed breakdown for the AWS deployment project.

The cost estimation provides a comprehensive view of the financial requirements for running the web application on AWS. By leveraging cost-effective instance types and optimizing resource usage, the deployment ensures a balance between performance and cost. Additionally, using services within the AWS Free Tier where applicable helps minimize costs, making the deployment economical while staying within the boundaries of reliability and best practices.

5- CONCLUSION:

The AWS Cloud Web Application Deployment project successfully achieved its objectives of deploying a scalable, reliable, and cost-effective web application infrastructure. By leveraging a comprehensive suite of AWS services, including EC2, RDS, ELB, VPC, and AWS Secrets Manager, the project demonstrated the ability to build a robust cloud environment capable of handling high traffic volumes while maintaining performance and security.

Regular monitoring and performance tuning should be maintained to adapt to changing workload patterns and optimize resource utilization. Exploring additional AWS services like AWS Lambda for serverless architecture to further enhance scalability and reduce operational overhead.

Additionally, Periodic security audits and compliance checks should be conducted to ensure ongoing adherence to industry standards and best practices.