AWS Secure Financial Architecture – Reflection Report

42035 Cloud Security Assessment
Task 3 - Design Implementation

Group 11 Team Members:

- Shruti Jambu - 25241050
- Rishikesh Vijay - 25021953
- Sakshi Chame - 25167065
- Vrinda Sati - 25202490

Student 1: Rishikesh Vijay

What I set out to do?

The project goal was to design and implement a secure and scalable Virtual Private Cloud (VPC) as part of a 3-tier application architecture in AWS. This involved creating a VPC with high availability across multiple Availability Zones (AZs), setting up internet access through gateways, and securing sensitive components like databases.

What I actually did?

I created the VPC with multiple subnets: public subnets for the presentation layer, and private subnets for application and database layers. I configured an Internet Gateway for public subnets and a NAT Gateway for private subnets, enabling secure internet access without exposing instances. Security groups controlled access at each tier, and NACLs managed inbound and outbound traffic.

What worked successfully?

Dividing the network into public/private subnets successfully separated accessible resources from internal services. Configuring the VPC across multiple AZs supported high availability, with load balancers enhancing reliability. Security groups and NACLs provided robust security, limiting unauthorized access while allowing required communication between tiers.

What modifications were made and why?

Originally, the NAT Gateway was in a single AZ, creating a failure point. To improve availability, I modified the setup to include multiple NAT Gateways across AZs. Managing NACL rules became complex, so I streamlined configurations by relying more on security groups, simplifying network management and reducing errors.

What were my key takeaways/insights?

Designing a secure VPC requires careful subnet separation to protect critical resources and ensure communication paths. High availability goes beyond resource addition; it requires strategic distribution across AZs and redundancy planning. Balancing cost and redundancy is essential, as features like multiple NAT Gateways increase reliability but also costs.

Key Challenges and Solution

Challenge 1: Misconfigured Routing Tables Leading to Network Isolation

- **Issue**: Private subnets couldn't access the internet due to incorrect route configurations.

  Solution: Adjusted the route table to direct private subnet traffic through the NAT Gateway**.**

Challenge 2: Difficulty in Managing Security Rules

- **Issue**: Balancing rules across security groups and NACLs was complex.
- **Solution**: Simplified rule management by focusing on security groups, reducing the risk of configuration errors.

Student 2: Shruti Jambu

What I set out to do?

The project aimed to leverage Amazon S3 for two essential functions within our 3-tier architecture: storing web and application server code and collecting VPC flow logs. The first S3 bucket was intended as a central repository for code access and deployment, facilitating seamless updates across environments. The second bucket was designed to gather VPC flow logs for comprehensive network traffic monitoring and troubleshooting.

What I actually did?

I created two distinct S3 buckets to fulfill these roles. The first bucket enabled secure, streamlined deployment of server code across multiple EC2 instances, ensuring that the latest application versions were readily accessible. The second bucket was established for the centralized storage of VPC flow logs, allowing easy access to critical network data. IAM roles were configured to provide read-only access to the code bucket for authorized personnel while restricting log bucket access to essential resources.

What worked successfully?

Using Amazon S3 for both code storage and log collection proved highly effective. S3's durability ensured reliable access to code, minimizing downtime. The flow log bucket facilitated real-time network monitoring, which was crucial for maintaining security. Additionally, S3's integration with IAM enabled straightforward, secure access control, ensuring that only authorized users could access sensitive resources.

What modifications were made and why?

Initially, I considered using a single bucket for both functions. However, I opted for two separate buckets to improve organization and access control. This separation allowed for more granular permissions, simplifying monitoring and maintenance without exposing sensitive data.

What were my key takeaways/insights?

This project highlighted S3's versatility in managing storage and security needs. I learned that well-defined permissions and robust access management via IAM can enhance both security and operational efficiency. The implementation of S3 buckets improved our architecture's scalability while reinforcing security, showcasing S3's adaptability for cloud-based applications.

Key Challenges and Solution

Balancing secure yet accessible code storage with restricted log data access was a key challenge. Custom IAM roles addressed this, granting precise permissions for each bucket, ensuring security, and supporting streamlined deployment. To address this challenge, I implemented custom IAM roles that granted precise permissions for each bucket. This approach ensured that security was maintained while also supporting a streamlined deployment process, allowing for efficient access to the necessary resources without compromising the overall integrity of the system.

Student 3: Vrinda Sati

What I set out to do?

My goal was to create a robust AWS environment for a financial application by configuring essential components like a **DB Subnet Group** and **RDS** for data storage and setting up a **Application Server** with autoscaling, internal load balancing, and secure access. I planned to establish a **multi-AZ RDS** for data resilience, create a **launch template** using a custom AMI to ensure server consistency and configure **target groups** and **internal load balancing** for managing traffic within the application layer.

What I actually did?

I created a DB Subnet Group to organize selected subnets and configured a Multi-AZ RDS for secure, redundant storage with automatic failover. I set up the Test Application Server on AWS EC2 for the application layer with necessary packages and an IAM role for secure access. I created an AMI of this server and used it for a launch template to streamline deployment. A target group and internal load balancer directed traffic within the application, while an autoscaling group across zones us-east-1a and us-east-1b ensured scalability and availability.

What worked successfully?

The multi-AZ RDS deployment provided automatic failover and high availability. The AMI and launch template ensured consistent instance replication while the internal load balancer managed traffic effectively. The autoscaling group maintained performance by scaling instances as needed, and the IAM role enabled secure resource access without storing credentials.

What modifications were made and why?

Initially, the Application Server was set up without an IAM role, which was later added for secure, credential-free access to AWS S3. The autoscaling group was also configured to deploy instances across us-east-1a and us-east-1b, enhancing redundancy and reliability.

What were my key takeaways/insights?

Configuring DB Subnet Groups and Multi-AZ RDS highlighted the importance of redundancy and availability for resilient data storage. Setting up an AMI and launch template emphasized the value of a standardized, secure environment. The internal load balancer and autoscaling group proved essential for traffic management and scaling flexibility, while IAM roles simplified permissions and enhanced security.

Key Challenges and Solution

Challenge 1: Ensuring secure, standardized configurations across all instances.

Solution: Using an **AMI** created from the Test Application Server and developing a **Launch Template** simplified the process of replicating secure, pre-configured instances.

Challenge 2: Balancing high availability with resource management.

Solution: Configuring the **autoscaling group** to span multiple availability zones allowed for on-demand scaling, improving availability without over-provisioning resources.

Student 4: Sakshi Chame

What I set out to do?

My team and I embarked on a project to design a 3-tier architecture for a cloud-based system. My specific role was to create the web tier and handle the security and monitoring components of our design.

What I actually did?

I researched Nginx and React JS for the web servers, setting up one in each availability zone. I created an AMI with launch templates for EC2 instances. For security, only the external load balancer had public access. I implemented AWS CloudWatch with CPU usage alarms linked to SNS for email alerts, set up Auto Scaling Groups to maintain instance numbers, and used CloudTrail for activity logging and troubleshooting.

What worked successfully?

Our team's collaboration was key. After submitting our initial design, we met to discuss implementation, identifying areas for improvement. We enhanced our solution by adding more AWS services, demonstrating our adaptability. Setting up web servers across multiple availability zones and limiting public access improved security. The monitoring and auto-scaling features we implemented effectively maintained system performance.

What modifications were made and why?

Following our team discussion, we enhanced our initial design by incorporating additional AWS services to improve robustness, scalability, and security. A key modification was restricting public access to only the external load balancer, keeping the app and database tiers private and secure.

What were my key takeaways/insights?

This project offered valuable lessons in cloud architecture. I learned the importance of flexibility in design, as our team's ability to adapt and improve our initial plan was crucial to our success. Working hands-on with AWS services deepened my appreciation for their power in creating scalable and secure solutions, significantly enhancing my understanding of cloud architecture. The experience of securing publicly accessible components emphasized the critical role of security in cloud design. Finally, implementing monitoring and auto-scaling features like CloudWatch, SNS, and Auto Scaling Groups demonstrated their importance in maintaining optimal system performance.

Key Challenges and Solution

The project presented several key challenges, including understanding complex web server technologies like Nginx and React JS, securing public IP addresses, finding correct installation commands, and balancing security with accessibility. To overcome these, I dedicated time to in-depth research, implemented a design with limited public access, carefully documented procedures, and utilized AWS services like CloudTrail for logging and monitoring. These challenges, though difficult, significantly enhanced my understanding of cloud architecture and AWS services, providing valuable skills for future projects.

Appendix

Main Presentation

The presentation video link: https://www.youtube.com/watch?v=K_WM0Fx9OTI

References

Amazon CloudWatch Features - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/cloudwatch/features/

Amazon EC2 security groups for your EC2 instances - Amazon Elastic Compute Cloud. (n.d.). https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-security-groups.html

An AWS Cloud architecture for web hosting - Web Application Hosting in the AWS Cloud. (n.d.). https://docs.aws.amazon.com/whitepapers/latest/web-application-hosting-best-practices/an-aws-cloud-architecture-for-web-hosting.html

Auto Scaling launch templates - Amazon EC2 Auto Scaling. (n.d.). https://docs.aws.amazon.com/autoscaling/ec2/userguide/launch-templates.html

Automatically scale your Amazon ECS service - Amazon Elastic Container Service. (n.d.). https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-auto-scaling.html

AWS Well-Architected Framework - AWS Well-Architected Framework. (n.d.). https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html

AWS::RDS::DBSubnetGroup - AWS CloudFormation. (n.d.). https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-rds-dbsubnetgroup.html

Create a monitor in Amazon CloudWatch Internet Monitor using the console - Amazon CloudWatch. (n.d.). https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch-IM-working-with.create.html

Creating an AMI from an Amazon EC2 Instance - AWS Toolkit with Amazon Q. (n.d.). https://docs.aws.amazon.com/toolkit-for-visual-studio/latest/user-guide/tkv-create-ami-from-instance.html

Hosted MySQL - Amazon RDS for MySQL - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/rds/mysql/

Multi-AZ DB instance deployments for Amazon RDS - Amazon Relational Database Service. (n.d.). https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.MultiAZSingleStandby.html

Organization design - AWS Cloud Adoption Framework: People Perspective. (n.d.). https://docs.aws.amazon.com/whitepapers/latest/aws-caf-people-perspective/organization-design.html

Route internet traffic with AWS Load Balancer Controller - Amazon EKS. (n.d.). https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html

Setting up AWS WAF and its components - AWS WAF, AWS Firewall Manager, and AWS Shield Advanced. (n.d.). https://docs.aws.amazon.com/waf/latest/developerguide/getting-started.html

What is Amazon S3? - Amazon Simple Storage Service. (n.d.). https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

What is an Application Load Balancer? - Elastic Load Balancing. (n.d.). https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html

What is IAM? - AWS Identity and Access Management. (n.d.-b). https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

Working with a DB instance in a VPC - Amazon Relational Database Service. (n.d.). https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_VPC.WorkingWithRDSInstanceinaVPC.html