CSCI 5409 Term Assignment – A Simple Tic Tac Toe game with WebSockets.

Description:

This is a simple Tic-Tac-Toe game running as a web application hosted on AWS, the game has the following features.

1. Multiplayer: The game can be played by multiple players across the internet once the game ID is known.
2. Real-Time: The player input and board state are synchronized in real-time through WebSockets [1].
3. Private Lobbies for hosting the game: The use of DynamoDB as a data store can allow for the use of individual lobbies to play games.
4. Discord Bot integration: The winner can send a brag message to the loser in Discord through a bot.
5. Data security through encryption at rest and encryption in transit [2].

Services and functionality used:

1. EC2: The web application is written in spring boot and runs on an EC2 instance in the private subnet.

   Alternative services explored:

   - Lambda: A serverless solution cannot be used to synchronize player data across clients using WebSockets.
   - ECS/EKS: Containerization can be appealing for web applications in organizational settings for standardization, but I believe it was not necessary in this case, as I'm the only developer and would rather prefer the application to run natively without any virtualization for the best possible performance.

2. DynamoDB: The Database tier uses DynamoDB to store data, A NoSQL [3] option is preferred as the application does not involve complex joins and the low latency provided by DynamoDB is vital for games.

   Alternative services explored:

   - RDS: Relational DBs are suited for strongly structured data with a lot of joins, the data for the game uses a loose JSON like format with the primary key as the Game ID. Using RDS wouldn't make too much sense here.
   - AWS DocumentDB: Document DB is for semi-structured data segregated as documents. The application can do with a few handful of tables and using DocumentDB seems too excessive.

3. AWS API Gateway: The discord bot integration is done by creating a REST API through the API gateway to provision a rest API to interact with a lambda function and send the discord message [4].

   Alternative services explored:

- EC2: EC2 instances maybe useful in creating REST APIs for the web application, but since the discord bot is integrated with a third-party services and uses player Discord ID information, it is vital that this mechanism be decoupled from the web application to add security. Besides, this minimizes running the python backend at the instance level.

4. VPC: A Virtual Private cloud with a private subnet where the EC2 instance resides and a public subnet with an Elastic Load Balancer [5].

   Alternatives Explored:

   - S3 static website hosting: Server-side scripts synchronize data between players using WebSockets and Server Sent Events, this is not possible in an S3 bucket configured for static site hosting.
   - A completely Serverless solution: Wouldn't work because the game renders a board and synchronizes events in real-time, while Step functions seem like a good fit here, they are relevant only for analysis.

5. Elastic Load Balancers (ALBs): Application load balancers in a public subnet with EC2 instances behind, in the private subnet. ALBs are perfect to host web applications and distribute traffic to multiple instances. In this case, having an ALB in the public subnet also prevents direct access to EC2 instances and servers to implement TLS through an SSL certificate provisioned in ACM (Amazon Certificate Manager).

   Alternatives Explored:

   - Other Load Balancers (CLBs and NLBs): Classic Load Balancers are not suited for high-performance Web Applications, Network Load Balancers may offer the necessary performance, but they are better suited for TCP applications, not HTTP/HTTPS.
   - CloudFront: Cloudfront is good for catching content and not as a secure, load balancing application. The game involves real-time interaction between clients and caching is of little use here.

6. AWS Secrets Manager: Secrets manager stores the secure API token for the discord bot. Secrets manager provides a secure, encrypted store for secrets. Secrets manager seamlessly integrates with CloudFormation and KMS for encrypting data[6].

   Alternatives Explored:

   - SSM Parameter Store: The SSM parameter store can store encrypted secure strings, but integration with CloudFormation requires more time to setup than using Secrets Manager.

7. Lambda: Lambda function uses the secure API token in AWS Secrets manager and the discord.py library to send the winner message to the losing player's discord as a Direct message.
   Alternatives Explored:
   - AWS Step Functions: Step functions can do the same thing, but requires more administrative overhead and inevitable lambda integration, so it's better to invoke a lambda function directly.

Common Services used to ensure security and encryption.

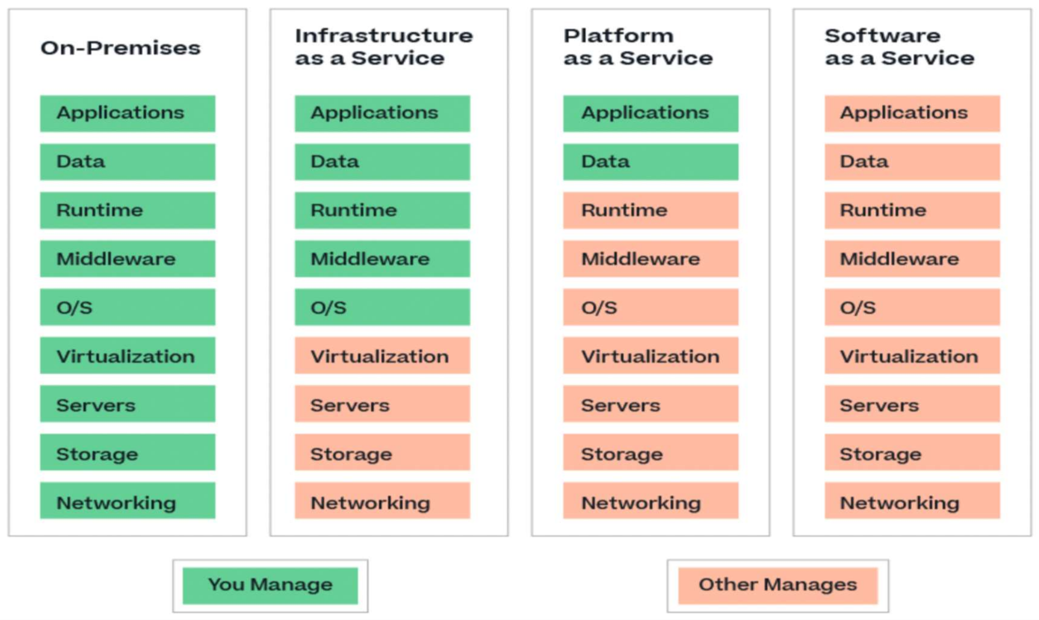
The project aims to maintain security throughout communication channels, both encryption at rest and encryption in transit is enforced through a variety of services like AWS KMS to store and manage AES 256 Keys, ACM to manage TLS certificates and Route 53 to setup a hosted zone to access the applications through the Load Balancer Alias [7]. Data at rest is encrypted through KMS keys and in-transit encryption occurs using SSL certificates at both the Load Balancer and the Instance level.

Describe your deployment model. Why did you choose this deployment model?

A Public Cloud deployment model is a good fit for this scenario. This application leverages the power and flexibility offered by AWS to provide a highly available and fault-tolerant experience. Since I don't have the hardware resources to run the application on premises, a public cloud provider would make sense. Also, the cost of hosting, ensuring high availability and networking infrastructure can be offloaded to the cloud provider.

A case for increased security can also be made here. Services like KMS allow for encryption of data at rest and in transit by SSL certificates. A private cloud solution would leave security in my hands, and I lack the expertise to setup data encryption at rest.

A community cloud would also not be feasible here as I am the only developer. Community clouds are suitable in cases where multiple organizations share common infrastructure because of how similar their workloads are.



Visualization of various deployment models [8].

Describe your delivery model. Why did you choose this delivery model?

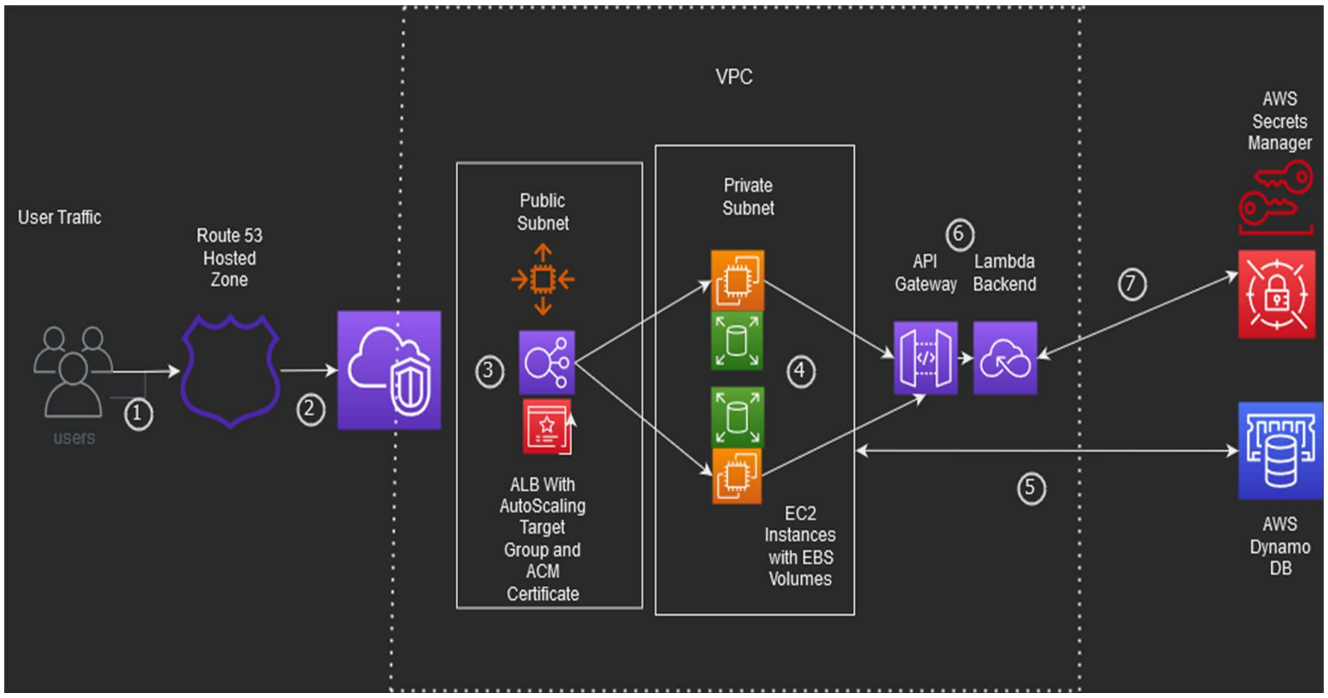
I chose the IaaS delivery model for this project. The IaaS delivery model allows the cloud provider to manage low-level IT infrastructure such as networking, compute and storage hardware while allowing access to the infrastructure to developers. The Iaas model provides fine grained access to services and allows for greater power in provisioning and scaling applications.

For this project, the IaaS model endows me with more control to provision where my infrastructure resides, my web application running on an EC2 instance resides in a private subnet while the public subnet hosts the application load balancer with an SSL certificate configured, such fine grained control increases security standards and enables the use of scaling services such as AutoScaling groups. I can also define custom parameters and criteria for scaling my game to a large player base.

Furthermore, my application is real-time and uses Websockets to sync data between clients. I would need to ensure the lowest possible latency; this involves complex provisioning like using cluster placement groups for my instances and using an appropriately provisioned DynamoDB tables. A PaaS or an FaaS solution would not allow for such flexibility.

The use of IaC (Infrastructure as Code) through CloudFormation allows me to provision resources through code, this is possible if I choose the IaaS delivery model.

Architectural Description:



Architectural depiction of application infrastructure in the cloud.

How do all the cloud mechanisms fit together to deliver your application?

Any application residing in the cloud can be broadly classified into the following layers:

- Storage Layer: Where the application data, database and Code resides. This includes EBS volumes and DynamoDB tables.
- Network Layer: Networking components such as VPCs, subnets, Route 53 Hosted Zone, and Load Balancers.
- Compute Layer: Compute components such as EC2 instances and Lambda functions for specific operations.
- Security Layer: This comprises secure key stores, SSL certificates and the Secrets manager for storing secure secrets like the API keys for the Discord bot.

Where is Data Stored?

Game Data is stored in DynamoDB tables, while lambda function data is stored in a public S3 bucket. Application code and scripts reside locally, on Instance EBS volumes.

Programming Languages used – Java and Python

I used Java to write the part of the application that resides on the EC2 instance, Java with Spring Boot is an excellent, lightweight stack to build web applications. Additionally, The AWS Enhanced SDK for Java makes interacting with DynamoDB so much easier.

Python is ideal for lambda functions as AWS provides native boto3 integration.

The Spring Backend consisting of Game Logic, Handlers for the WebSocket messages and the lambda function required code.

Traffic Flow and Services at each level (How is your system deployed in the cloud?)

Resources are created through CloudFormation templates, enabling rapid deployment and teardown.

The numbers in the diagram depict the order of traffic flow, from 1-7, with each stage consisting of various AWS services. They're explained below:

1. The public traffic coming from the users over the internet, the users access the application through a Route 53 Hosted zone with the domain name - ad368540-ta5409.com , the Name Servers are managed by AWS, an Alias A-record is setup to route traffic to the Application Load Balancer.
2. A VPC (Virtual Private Cloud) is setup in the us-east-1 region, the VPC is divided into multiple subnets, with CIDRS 10.0.1.0/24 and 10.0.2.0 for the public and private subnets respectively.
3. The Public Subnet comprising of the Application Load Balancer with an HTTPS listener configured to listen to public internet traffic. The Load Balancer uses an SSL certificate provisioned by ACM (AWS Certificate Manager) for Transport Layer Security. The target group for the Load Balancer consists of EC2 instances in two Availability zones to maintain High Availability.
4. The EC2 instances reside in the private subnet, The instances are of the t3 family as they excel for general purpose computing, the application code resides on general purpose (gp2) EBS volumes. The instances don't have any SSH key configured, they use the Systems Manager Sessions manager for establishing secure SSH sessions.
5. DynamoDB tables to store game state information. The DynamoDB tables are accessed via regional https endpoints as they reside outside the VPC. They run in an on-demand configuration to minimize read/write capacity management. A single table architecture is used for the game.
6. API Gateway with a Lambda backend to run discrete python functions that uses the discord.py framework with a secret key to send brag messages to players via a bot.
7. Secrets Manager stores secret keys used by the database and the discord bot. The seamless integration with CloudFormation allows for easier access and development.

How does your application architecture keep data secure at all layers? If it does not, if there are vulnerabilities, please explain where your data is vulnerable and how you could address these vulnerabilities with further work.

Encryption at rest and encryption in transit ensures that the data is secure at all layers, encryption in transit is handled using ACM to provision SSL certificates and encryption at rest is handled through the KMS service that uses an AWS Managed AES-256 key to encrypt data. DNS queries are secured using Route 53 which employs DNSSec to establish secure DNS connections and prevent man in the middle attacks.

Additionally, VPC level access control lists and the instance and load balancer security groups are made as restrictive as possible.

A possible vulnerability would be the AWS account itself having admin access to all resources in the infrastructure, hackers can access the account and use those credentials to access the infrastructure and obtain administrative privileges. This can be patched using MFA (Multi Factor Authentication) for a two-step authentication process.

Other potential vulnerabilities include DDoS attacks that deny service to the infrastructure and Code injection attacks such as XSS, this can be mitigated by employing a combination of AWS WAF (Web Application Firewall) and the AWS Shield Advanced security standard at the Application Load Balancer level to achieve higher security. At the instance level, the use of a NAT gateway enables outbound-only internet access to the instances in the private subnet, this allows the instance to get the latest security patches and address CVEs at the OS level.

Which security mechanisms are used to achieve the data security described in the previous question? List them, and explain any choices you made for each mechanism (technology you used, algorithm, cloud provider service, etc.)

The following security mechanisms/services ensure security:

1. AWS KMS (Key Management Service): KMS is used to encrypt data at rest at the EBS volume level in EC2 instances, the DynamoDB table for table data and the secrets manager for storing database and the discord bot API token.
2. AWS IAM (Identity Access Management): AWS IAM offers management of access permissions for users through roles, in this case, AWS Academy has provisioned a preset role “LabRole” which is used throughout the project, further examination of the LabRole reveals that it is similar to the PowerUser predefined role offered by AWS.
3. ACM (AWS Certificate Manager): ACM offers a secure data store for an SSL certificate that also doubles up as a certificate authority. ACM is used at the Application Load Balancer level to encrypt traffic in transit from the internet via the HTTPS listener.
4. Security groups: The security groups at the instance level and the Application Load Balancer tiers act as a stateless virtual firewall to the elastic network interfaces, ensuring only specific ports and sources allow traffic.
5. Network Access Control Lists: The stateless ACL at the VPC level ensures that only specific IP addresses are allowed to access the application residing in the VPC.
6. Public/Private subnet split: The VPC infrastructure is split into the public subnet and the private subnet, traffic from the internet is allowed to access the instances only through the Application Load Balancer and not directly as the instances are in the private subnet without public internet access.
7. AWS Secrets Manager: Secrets Manager offers a secure key store with direct integration with AWS CloudFront, this ensures easy provisioning and an encrypted data store to store the database keys and the discord API token.
8. AES-256 standard: The AES 256 standard is employed by AWS KMS to encrypt data using managed encryption keys, this algorithm is practically immune to brute-force attempts.
9. DNSSec protocol for DNS queries: AWS Route53 secures DNS queries against Man in the Middle attacks.

What would your organization have to purchase to reproduce your architecture in a private cloud while providing relatively the same level of availability as your cloud implementation?

While the 99.99% availability provided by AWS cannot be perfectly met in a private cloud, a complex architecture consisting of Compute servers, networking appliances and storage devices can be setup to provide a highly available architecture.

Compute servers must be placed in different clusters or data centres located a few hundred meters apart so that physical conditions don't cause downtime for the whole application. This involves replicating infrastructure at multiple locations and connecting them via a highly resilient network connection. Additionally, Specialized hardware and third-party software can also be used to ensure that the traffic in the network is secure at all levels.

The cost breakdown for each layer in the on-premises infrastructure would be as follows (All hardware and software costs are taken from newegg.ca for reference):

1. Compute layer: This layer would have hardware with a server-grade processor such as the Ryzen EPYC series or an Intel Xeon series processor to enable advanced virtualization and scaling technology. This would allow the provisioning of multiple instances and scaling them individually. A Host virtualization solution would be recommended to host the application instances. Avg Cost: **4000 $ X 2 = 8000 $**
2. Storage Layer: This would consist of Solid-State Drives or Hard Drives setup in a RAID0 configuration for the best performance, server grade hardware is expensive, so a highly durable storage solution would necessitate purchasing an enterprise SSD since the storage layer would house the instance Level Operating systems and the DynamoDB tables.

   Cost Per GB: 3.69 $
   Avg 4TB SSD cost: 12000 X2 = 24000 $

   Additionally, the compute hardware would also require memory to run tasks, so adequate DRAM needs to be purchased to meet scaling and performance demands. ECC enabled memory modules is essential to prevent data corruption in our data centres.

   Cost per GB: 7 $
   64 GB ECC memory cast: 500 X2 = 1000 $

3. Network Layer: Network layer involves hardware like gateways, Hubs, switches and cables.

   Avg cost of Networking switch: 300 x2 = 600 $
   Avg Cost of high-performance network Card: 250 x 2 = 500 $
   Avg Cost of Cables and installation: 400$ x2 = 800$

4. Software and other third-party solutions: Our cloud infrastructure employs a variety of services to ensure that data at all levels is secure through encryption. This involves purchasing licenses to third-party encryption software and procuring SSL certificates provided by a certifying authority.

   SSL certificate cost: 30 $
   License for Encryption software: 64 $ x 2 = 128 S
   Cost of Hosted Zones through a registrar: 50 $ Annually.
   Cost of Operating System license: $ 0 as open-source LINUX distros are used.
   Nginx would be used to provision a free Load Balancer acting as a proxy: 0 $ cost.
   Virtualization Software such as VMWare VSphere License: 4000 x2 = 8000 $
   Tools like Chef and Automate would be used to standardize server configuration: 0 $ cost.

5. Essential Maintenance, Space and Power costs: Maintaining a data centre in multiple locations can be very expensive, appropriate space with adequate cooling, power supply and maintenance staff can easily add several thousand dollars to the existing system.

Total expected On-premises infrastructure cost: **43078 $**, this excludes costs of power supply, cooling infrastructure, space constraints and maintenance staff. Which could easily add up to a large amount.

Which cloud mechanism would be most important for you to add monitoring to to make sure costs do not escalate out of budget unexpectedly? In other words, which of your cloud mechanisms has the most potential to cost the most money?

In a broad scope, services like the AWS Cost Explorer and AWS Budgets can be used to visualize, forecast, and monitor costs to ensure that the expense of cloud infrastructure is within an acceptable threshold. Other mechanisms are:

1. AWS Budgets User-generated tags: User generated tags can be used to appropriately tag resources and perform remediation actions based on usage; AWS Budgets also has integration with AWS SNS topics for alerting administrators in case the expenses exceed specified limits.
2. AWS Compute Optimizer: Compute Optimizer uses data from CloudWatch metrics to make right-sizing recommendations for the EC2 instances used, this could potentially save instance costs by avoiding over-provisioning.
3. AWS Trusted Advisor: Trusted Advisor can make cost recommendations by analyzing current AWS infrastructure and workload patterns.
4. AWS SNS Topics: SNS topics are used in case of notifications for alerts and budget events.
5. AWS IAM: IAM users and roles should be configured on the basis of the principles of least privilege, ensuring attackers can't launch expensive instance types or over-provision DynamoDB Capacities.

In terms of expenditure, DynamoDB has the potential to generate a high bill in case of any DDoS [9] attack or poorly written code when there are several concurrent requests to DynamoDB.

How would your application evolve if you were to continue development? What features might you add next and which cloud mechanisms would you use to implement those features?

With continued development and popularity, additional matchmaking and balance features like network lobbies and Power-ups can make the game more fun. Refactoring current architecture to move to a managed containerized solution to use AWS ECS on Fargate [10] Clusters instead of EC2 instances would allow the application to adapt to increased traffic in a cost-effective manner.

The following features are planned in future releases.

1. Win Rate calculations in matchmaking – using EMR clusters for matchmaking and a fully functional ELO System.
2. Authentication integration with most Identity providers using the AWS Identity center.
3. More dynamic, Responsive UI.
4. Lower latency using CloudFront and Elasticache as a caching layer.

References:

1. *Stomp over websocket*. [Online]. Available: http://jmesnil.net/stomp-websocket/doc/. [Accessed: 09-Apr-2023].
2. “ACM,” *Amazon*. [Online]. Available: https://docs.aws.amazon.com/acm/latest/userguide/data-protection.html. [Accessed: 09-Apr-2023].
3. “Using the dynamodb enhanced client in the AWS SDK for Java 2.X.” [Online]. Available: https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/dynamodb-enhanced-client.html. [Accessed: 10-Apr-2023].
4. “Tutorial: Build a hello world rest api with lambda proxy integration.” [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-create-api-as-simple-proxy-for-lambda.html. [Accessed: 10-Apr-2023].
5. A. W. S. Official, “Attach EC2 instances with private IP addresses to an internet-facing elb load balancer: AWS re:post,” *Amazon Web Services, Inc.*, 22-Dec-2022. [Online]. Available: https://repost.aws/knowledge-center/public-load-balancer-private-ec2. [Accessed: 09-Apr-2023].
6. “AWS:Secretsmanager::Secret - AWS cloudformation.” [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-secretsmanager-secret.html. [Accessed: 10-Apr-2023].
7. A. W. S. Official, “Associate an ACM/SSL certificate with a load balancer: AWS re:post,” *Amazon Web Services, Inc.*, 12-Oct-2021. [Online]. Available: https://repost.aws/knowledge-center/associate-acm-certificate-alb-nlb. [Accessed: 09-Apr-2023].
8. CSCI 5409 Lecture Notes.
9. P. Anderson, “Shield,” *Amazon*, 1982. [Online]. Available: https://aws.amazon.com/shield/ddos-attack-protection/. [Accessed: 09-Apr-2023].
10. Bläsi Denise and S. Enggist, “Branding through Cult Marketing: Fargate,” *Amazon*, 2003. [Online]. Available: https://aws.amazon.com/fargate/. [Accessed: 09-Apr-2023].

Report By

Adithya Hadadi

ad368540@dal.ca

B00886916