CSCI 5409 ADV TOPIC IN CLOUD COMPUTING

TERM ASSIGNMENT

BannerId: B00971102

Name: Shrey Devendrabhai Monka

Table of Contents

Table of Figures

WHAT DID I BUILD?

Base Plan - Travel Companion

The application provides a platform where users can create detailed travel plans, including flights, accommodations, activities, and transportation. The users can share their stories via a mini tweet like posts on the web application.

Codebase

Backend and Frontend Technologies

- Spring Boot Backend: The backend is built using Spring Boot, providing a robust and scalable architecture.
- Angular Frontend: The frontend is developed using Angular, offering a dynamic and responsive user interface.

Features

1. User Authentication and Authorization

- Sign Up and Login: Secure user registration and login functionality, ensuring only authorized access to the application.
- Password Encryption: User passwords are encrypted for enhanced security.

2. Itinerary Planning

- Detailed Itinerary Creation: Users can create itineraries that include flights, accommodations, activities, and transportation.
- Category-Based Input: Depending on the category selected (Flights, Accommodations, Activities, Transportation), the relevant input fields are dynamically displayed.

3. Travel Journal and Social Sharing

- Journal Entries**:** Users can document their travels with notes, and experiences.
- Create Posts: Users can create posts to share information about their travel experiences.
- View Posts: All posts are displayed on a user-friendly dashboard, with key information prominently shown.
- Rating System: Users can rate posts using a 5-star rating system.

4. Notifications

- User will get notification when they signup where I am using sns for sending email notifications.

What it is supposed to do?

So, it is basically a virtual journal or which will let users build and plan their future travel and share the user stories related to a particular place and even rate that place.

HOW DID I MET THE MENU REQUIREMENTS?

In the development of the Travel Companion project, several AWS services were utilized to meet various requirements. Below, the selected services are listed along with a comparison of alternative services and the rationale for choosing the specific services in the system.

List of used services

Compute

1. AWS EC2 [1]

   Selected Service: AWS EC2 (Elastic Compute Cloud) is used to host the web application on virtual machines. So, I wanted to deploy my whole frontend and backend which I deploy on my local machine directly to a virtual machine that was the main reason of selecting EC2. So, I can make docker image of my frontend and backed and directly run it on containers on virtual machines.

   Alternatives:

- AWS Elastic Beanstalk [2]: Provides a managed service for deploying and scaling web applications.
- AWS Lambda [3]: Allows running code without provisioning servers. I have used lambda in my project but what I mean here is deploying the whole backend on multiple lambdas for complete backend.
- AWS EKS [4]: So, I also thought of using Kubernetes for deploying the code and it is actually a good idea if the application is big enough.

   Rationale for Choice: AWS EC2 offers full control over the environment, which is beneficial for custom configurations and handling specific requirements of the Travel Companion application. It provides a flexible and scalable solution suitable for the backend services.

2. Lambda [3]

   Selected Service: AWS Lambda is used for serverless functions to send notifications upon user signup.

   Alternatives:
   AWS EC2 [1]: Running notification services on EC2 instances.
   AWS Fargate [5]: Running containerized functions without managing the servers. So this option is available if I was using EKS so I could deploy the notification service on Fargate instead of lambda.

   Rationale for Choice: AWS Lambda allows running code in response to events without provisioning or managing servers. This is ideal for the notification service as it only runs when a user signs up, providing a cost-effective and scalable solution. Main reason was it is on click service so it only triggers if user sign's up.

Storage

- ### AWS Relational Database Service (RDS) [6]
  Selected Service: AWS RDS is used to host the application's backend database.
  Alternatives:
  Amazon DynamoDB [7]: A NoSQL database service.
  Rationale for Choice: AWS RDS provides a fully managed relational database service, making it easier to set up, operate, and scale a relational database in the cloud. It supports MySQL, which was used for the application's existing database schema. And as I was using Spring Boot in backend it was really easy to integrate a mysql like database with the code. That was the main reason for using RDS because I wanted to store structured data in table format.

Network

- ### AWS Virtual Private Network [8]
  Selected Service: AWS VPC is used to define a virtual network for the application. Where my whole application lives. Everything including backend, frontend and database.

  Rationale for Choice: AWS VPC provides complete control over the virtual networking environment, including selection of IP address range, creation of subnets, and configuration of route tables and network gateways. It is essential for securely isolating the application resources. The reason for using this was security like if I create a boundary for the users who can access my cloud and who cannot. So, I chose this based on my deployment model.

- ### AWS API Gateway [9]
  Selected Service: AWS API Gateway is used to create and manage APIs that interface with the Lambda functions and backend services.

  Rationale for Choice: So, the main reason was I was using Lambda and I need an endpoint for the lambda to be called that is why I used API Gateway. AWS API Gateway provides a managed service for creating, publishing, maintaining, monitoring, and securing APIs. It integrates seamlessly with AWS Lambda and other AWS services, providing a unified solution for API management.

General

- ### AWS SNS [10]
  Selected Service: AWS SNS is used to send notifications to users upon successful signup.

  Alternatives:
  Sending the notifications through java mail sender, i.e., SMTP. Which is not an AWS service but it can still be used.

  Rationale for Choice: AWS SNS provides a fully managed messaging service for both application-to-application (A2A) and application-to-person (A2P) communication. It is ideal for sending real-time notifications to users.

- AWS CloudFormation (Infrastructure as code) [11]
  Selected Service: AWS CloudFormation is used to define and provision the infrastructure through code.

  Alternatives:
  Terraform: An open-source infrastructure as code software tool.
  AWS CDK (Cloud Development Kit): A software development framework for defining cloud infrastructure.
  Rationale for Choice: One reason for using this was easy integration with CI/CD and code pipeline. AWS CloudFormation provides a simple way to manage infrastructure through code. It integrates tightly with AWS services and allows for the provisioning of all necessary resources in a consistent and repeatable manner.

Deployment Model

Virtual Private Cloud (VPC)

Description

In the Travel Companion project, the deployment model chosen is the Virtual Private Cloud (VPC) [8]. AWS VPC is a service that allows you to provision a logically isolated section of the AWS cloud where you can launch AWS resources in a virtual network that you define. This virtual network closely resembles a traditional network that you might operate in your own data center but with the benefits of using the scalable infrastructure of AWS. And it gives high flexibility what I want to do inside the VPC. I control that part of cloud which can easily enhance my security if I do it right.

Components of the my VPC Deployment

Subnets

- Public Subnet: A subnet that is accessible from the internet. Typically, I used this for hosting my EC2 of backend and frontend. So, I kept my frontend and backend accessible from the internet by hosting an ec2 inside the public subnet.

- Private Subnet: A subnet that is not accessible from the internet. I used this for storing my database so no one from the internet can access the database from internet only those with permission from access this subnet and can make a request and get a response from resources inside this subnet.

Internet Gateway (IGW)

A gateway that allows communication between instances in your VPC and the internet. So I made the virtual private cloud inside the public cloud but obviously need internet to connect with this cloud or else it's just an block of cloud with no access.

Route Tables

Determines how traffic is directed within the VPC. Configured to route traffic to and from the internet gateway for the public subnet and for the private subnet. Even the routes where public and private subnet communicate are also mentioned in route tables.

Security Groups

Acts as a virtual firewall to control the inbound and outbound traffic to your instances. So I have two EC2 and one RDS so I need to put boundaries on them to allow incoming and outgoing traffic from those instances.

Deployment Flow

Web Application (Frontend and Backend) Deployment

- Frontend: Deployed on AWS EC2 instances within the public subnet. These instances are accessible over the internet to allow users to interact with the web application.
- Backend: Deployed on AWS EC2 instances within the public subnet, accessible over the internet to handle API requests and other external communications.

Database Deployment

- AWS RDS: Deployed within the private subnet to ensure it is not directly accessible from the internet, enhancing security. The database is only accessible from the backend instances in the public subnet.

Serverless Functions

- AWS Lambda: Utilized for specific functionalities such as sending notifications. Integrated with AWS API Gateway to provide secure access to these functions.

Communication and API Management

- AWS API Gateway: Manages the APIs used by the frontend to communicate with the backend and other serverless functions.

Notification Services

- AWS SNS: Used to send notifications, such as signup confirmation emails.

Reasoning for Choosing VPC Deployment Model

Isolation and Security

VPC allows for logical isolation of resources, ensuring that sensitive components (e.g., databases) are kept in private subnets with no direct internet access. Security groups provide total control over inbound and outbound traffic.

Scalability

VPC supports the scaling needs of the Travel Companion application. It allows for adding more instances in both public and private subnets as demand increases.

Flexibility

VPC provides the flexibility to run a wide range of AWS resources within a virtual network that closely resembles a traditional network. This makes it easier to manage and configure resources as needed.

Control

Full control over the virtual networking environment, including IP address ranges, subnets, route tables, and network gateways, allowing for detailed customization to meet specific application requirements.

Integration with AWS Services

VPC integrates seamlessly with other AWS services such as EC2, RDS, Lambda, API Gateway, and SNS, providing a flexible infrastructure environment.

Compliance and Security Standards

Using VPC allows adherence to various compliance and security standards by controlling the data flow and access within the network, ensuring data is handled securely and efficiently.

By deploying the Travel Companion application within a VPC, we ensure a secure, scalable, and flexible environment that meets the application's requirements while leveraging the full suite of AWS services.

Delivery Model Description

Infrastructure-as-a-Service (IaaS)

Description

In the Travel Companion project, the chosen delivery model is Infrastructure-as-a-Service (IaaS). IaaS is a cloud computing model that provides virtualized computing resources over the internet. It offers fundamental infrastructure, including computing power, storage, and networking capabilities, and flexible infrastructure without having to manage the underlying physical hardware.

As I have created the whole virtual cloud and provisioned all the resources inside that cloud the delivery model can easily be recognized as Infrastructure as a Service.

Key Components of my IaaS Delivery Model

1. Compute Resources

- AWS EC2 (Elastic Compute Cloud)

2. Storage

- AWS RDS (Relational Database Service)

3. Networking

- AWS VPC (Virtual Private Cloud)

4. Security

- Security Groups

So basically, I can give my CloudFormation file to any one and they will have all this resources provisioned ready to deploy their frontend, backend and database in just few minutes with flexibility and security. This is what make my delivery model as IaaS.

Reasoning for Choosing the IaaS Delivery Model

Full Control and Customization

This model provides complete control over the virtualized infrastructure, allowing for detailed customization of the compute, storage, and networking resources. As my application is a complete backend frontend and database this is very essential.

Scalability

So, let’s say I deployed my backend on EC2 which is t2.micro and I deployed my frontend on another EC2 which is t3.small. Now if the codebase becomes huge, I might need to upgrade the EC2’s or I can add more EC2 so this flexibility is available in Iaas.

Cost-Effectiveness

IaaS follows a pay-as-you-go pricing model, allowing for cost optimization by only paying for the resources used. If I need more compute power in future, why would I pay for it now, keeping that in mind I choose IaaS.

Security

IaaS provides robust security features, including VPC, security groups, and NACLs, ensuring that the application meets security and compliance requirements. Data is securely stored and transmitted within the isolated virtual network.

High Availability

IaaS providers like AWS offer high availability and reliability through features such as multiple Availability Zones (AZs), automated backups, and disaster recovery options. This ensures minimal downtime and robust data protection for the application.

By choosing the IaaS delivery model, the Travel Companion project benefits from a flexible, scalable, and secure infrastructure that meets the application's requirements while providing full control over the virtualized resources.

ARCHITECTURE DIAGRAM

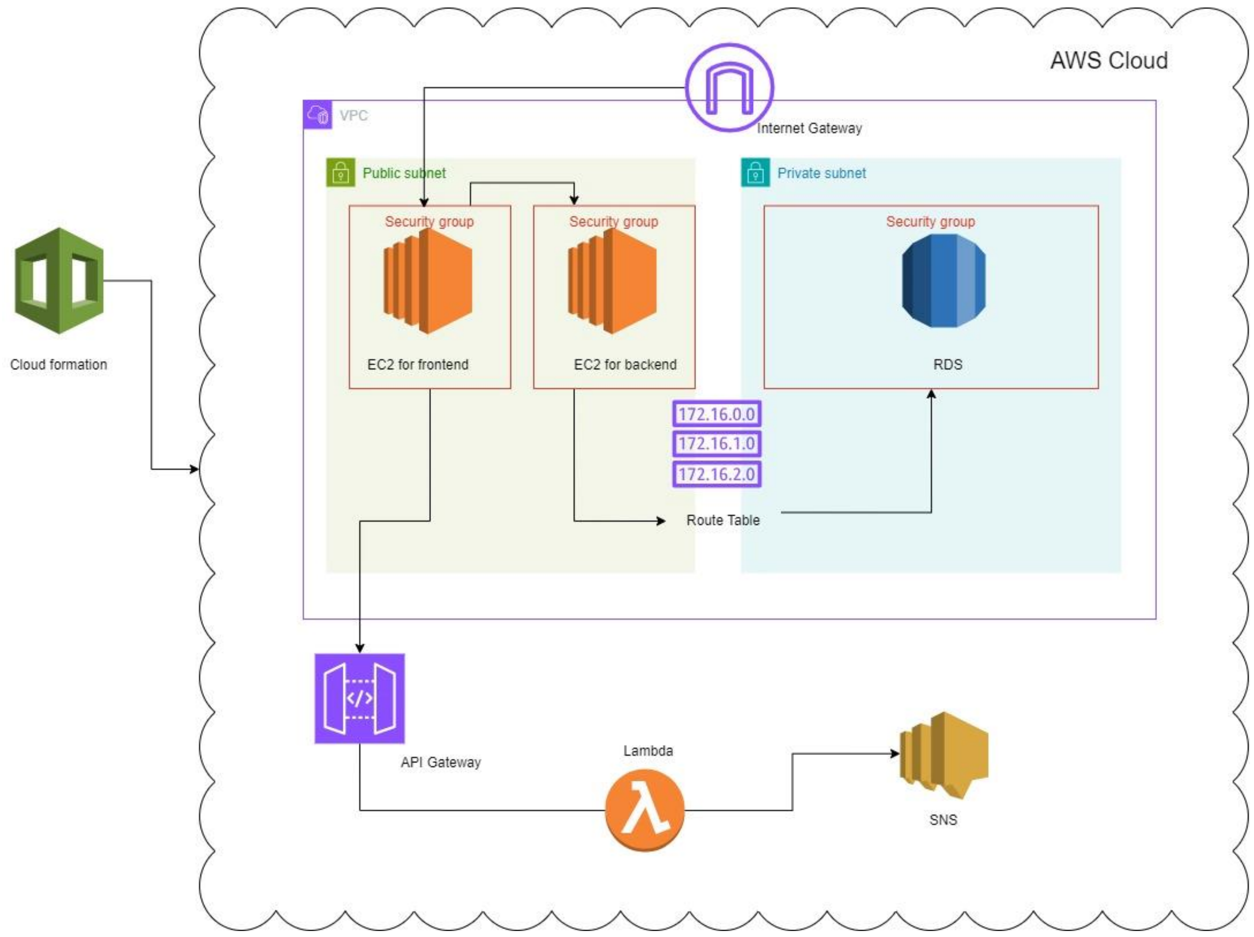

Figure 1: Architecture Diagram [12]

Analysis of Security Approach

Security Measures

Data at rest refers to inactive data stored on physical media, whether on-premises or in cloud storage. To secure data at rest, the following measures are implemented.

- **Security Groups:** Security groups act as virtual firewalls, controlling inbound and outbound traffic to AWS resources. They are configured to allow only necessary traffic to the RDS instance and other services.
- **VPC Configuration:** The application is deployed within a Virtual Private Cloud (VPC) with subnet isolation. Public subnets host the web servers, while private subnets are used for databases, enhancing security by limiting exposure to the internet.

And as my delivery model is IaaS I have provisioned all the resources and set the security groups for all the important instances. The very next thing is I am using VPC which only allows the allowed users to access the cloud. So that's the first layer of security the next layer of security is Security groups and the data becomes in transit if there is a route association between the components else the data stays in active it doesn't transit.

So, choosing IaaS gave me a lot of benefits and security too.

Analysis of Cost Metrics for Operating the Travel Companion System

Cost Metrics Analysis

Compute

1.) AWS EC2

- Instance Type: t3.small (suitable for small-scale applications)
- Cost: ~$0.01 per hour
- Monthly Cost (assuming 24/7 operation): 0.01 * 24 * 30 * number of instances = $7.20 per instance per month

2.) AWS Lambda:

- Invocation Cost: First 1 million requests per month are free; $0.20 per 1 million requests thereafter
- Monthly Cost: Depends on the number of requests; assuming 2 million requests, the cost would be $0.20

Storage

1.) AWS RDS

- Instance Type: db.t3.micro
- Cost: ~$0.017 per hour
- Monthly Cost: 0.017 * 24 * 30 = ~$12 per month

Network

1.) AWS VPC

- No direct costs, but there may be data transfer costs.

2.) AWS API Gateway:

- Cost: $3.50 per million API calls received plus data transfer charges
- Monthly Cost: Assuming 1 million API calls, the cost would be $3.50

General

1.) AWS SNS

- Cost: $0.50 per million publishes, $0.60 per million deliveries over HTTP, $2.00 per million deliveries over email
- Monthly Cost: Assuming 1 million notifications, the cost would be around $3

2.) AWS CloudFormation:

- No direct costs; charges are for the AWS resources created.

Additional Costs

1.) Monitoring and Logging:

- AWS CloudWatch: $0.30 per metric per month

Total Estimated Monthly Cost

- EC2 Instances (2 instances): $7.20 * 2 = $14.40
- AWS Lambda: $0.20
- RDS Instance: $12.00
- API Gateway: $3.50
- SNS Notifications: $3.00
- CloudWatch Monitoring (10 metrics): $3.00

Total Monthly Cost: $36.10

Alternative Approaches and Cost-Saving Strategies

1.) Auto-Scaling:

- Implementing auto-scaling for EC2 instances can help reduce costs by scaling down during off-peak times.

2.) Serverless Architecture:

- A fully serverless architecture using AWS Lambda, DynamoDB, and API Gateway can reduce costs significantly by charging only for actual usage.

3.) S3 Storage Classes

- Utilizing different S3 storage classes such as S3 Infrequent Access or S3 Glacier for data that is accessed less frequently can reduce storage costs.

4.) Free Tier Usage

- Maximizing the use of AWS Free Tier services, which provide a limited amount of usage for free for the first 12 months.

Justification for the Chosen Solution

Scalability: The chosen deployment and delivery models provide high scalability, allowing the application to handle varying loads efficiently.

Flexibility: Using a mix of EC2 for persistent workloads and Lambda for event-driven tasks offers flexibility in resource management.

Managed Services: Leveraging managed services like RDS reduces the operational overhead of managing infrastructure, allowing the team to focus on application development.

Security: The use of VPC, and managed services ensures a secure environment, adhering to best practices for data protection and compliance.

In conclusion, the chosen architecture provides a balanced approach to performance, cost, and security, ensuring a flexible and scalable application while keeping operational costs reasonable.

References

[1] "What is Amazon EC2?," AWS, [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html. [Accessed 29 06 2024].

[2] "What is AWS Elastic Beanstalk?," AWS, [Online]. Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html. [Accessed 30 06 2024].

[3] "AWS Lambda The Ultimate Guide," Serverless, [Online]. Available: https://www.serverless.com/aws-lambda. [Accessed 1 07 2024].

[4] "EKS Workshop," eksworkshop, [Online]. Available: https://www.eksworkshop.com/. [Accessed 26 06 2024].

[5] "Simplify compute management with AWS Fargate," AWS, [Online]. Available: https://docs.aws.amazon.com/eks/latest/userguide/fargate.html. [Accessed 25 07 2024].

[6] "What is Amazon Relational Database Service (Amazon RDS)?," AWS, [Online]. Available: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html. [Accessed 7 07 2024].

[7] "What is DynamoDB?," scylla, [Online]. Available: https://www.scylladb.com/learn/dynamodb/introduction-to-dynamodb/. [Accessed 7 07 2024].

[8] "What is Amazon VPC?," AWS, [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html. [Accessed 15 06 2024].

[9] "What is Amazon API Gateway?," AWS, [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html. [Accessed 26 07 2024].

[10] "Amazon Simple Notification Service Examples," AWS, [Online]. Available: https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sns-examples.html. [Accessed 25 07 2024].

[11] "What is AWS CloudFormation?," AWS, [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html. [Accessed 28 05 2024].

[12] "Draw.io," Diagram.net, [Online]. Available: https://app.diagrams.net/. [Accessed 07 08 2024].