CSC452/552 - Project

<u>AWS Tennis Court Booking System</u>

Members:
Olga Bienzobas, Erik Claros,
Anousith Keomaly

Project Overview

The AWS Tennis Court Booking System is a cloud-based platform that allows users to reserve tennis courts online. Users can select a court, and see whether or not it is reserved during specific time frames. They can select an available slot and make a reservation through an interactive interface that will confirm the booking if it was successful. In addition, it features a secondary site that provides information on the courts' general hours of operation.

Documentation Preface

Our final result is much different from our original intentions due to time constraints. In terms of implementation, they mostly use the same services but it may be utilized for a different reason. The document first starts off by showing how we **originally planned** to deliver our solution step by step without much implementation details. It will be captured in “**Previous Outline**”. The current solution will be another section and is what we demonstrated **during the presentation**. Any details on what/why steps were changed or implementation details on steps can be found in “**Final Outline**”.

Previous Outline

- Networking using Virtual Private Cloud(VPC)
  The VPC will be set up in the same way as the labs/exercises.

- Public and private subnets
  2 Public Subnets and 2 Private Subnets

- Multiple Availability Zones (AZ).
  2 Availability Zones

- Appropriate routing through NAT and Internet gateways
  Configured the same way as labs/exercises

- Security groups that allowed appropriate traffic

API:	SSH via local network HTTP via local network MYSQL via local network
Bastion:	SSH via via anywhere

Web Server: SSH via local network
HTTP via anywhere

Benchmark: SSH via local network
HTTP via local network

Database: MYSQL/AURORA via local network

- Additional Elastic Block Store (EBS) volumes attached and mounted
  We created an EBS, attached it to Bastion, and mounted it. In the mount, it will contain some .txt files about user information such as name, age, gender from the booking website. It also acts as a secondary source of storage.

- Information stored in and retrieved from S3 Buckets
  Wel created an S3 bucket that:
  Stores tenniscourts.png
  Stores benchmark.yaml (IAC for benchmark EC2 instance)

- EC2 instances for front-end, back-end, security, and stress testing
  EC2 instances:
  API `
  Bastion
  Web Server (Bookmark)
  Web Server (Hours *Also our containerized instance)
  Benchmark

- Deployment through Infrastructure as Code (IaC) and CloudFormation
  We deployed a benchmark server using the yaml file. Cloudformation will use the benchmark.yaml from the S3 bucket.

- Monitoring through EC2 and CloudTrail
  We deployed a cloud trail and created a cloud log to monitor the instances, along with monitoring the event history.

- Encryption with Customer Managed Key using Key Management Service (KMS)
  We created a KMS-SSE key and encrypt the S3 bucket with it.

- Horizontal scaling and stress testing
  For horizontal scaling, we followed lab 3 and used those steps in testing our website. Stress testing used the methods described in exercise 5.

- Load balancing using Elastic Load Balancer (ELB)
  ELB will be used for the web server.

- Containerization using Elastic Container Registry (ECR)
  An image will be made from the API EC2 instance and stored in the ECR.

- Orchestration using Elastic Container Service (ECS)
  Orchestrate by creating an ECS task that uses the API image stored in the ECR.

Final Outline

VPC, Subnets, Availability Zones (AZ), Routing & Security Groups

Because of time constraints, we ended up reusing the VPC we created in Lab 3. The VPC had four subnets: two were public and two were private. For short, they will be referred as public subnet-1a, public subnet-1b, private subnet-1a, and private subnet-1b.

There were 2 availability zones: one in us-east1a and the other was in us-east1b. Each public subnet had an IGW attached and each private subnet had their NAT gateways configured for outbound access. We didn't implement the API mentioned in "**Previous Outline**", so there are some changes in the security groups (as well as in other areas and we may continuously mention it). Because of this, the Web Server (Booking) directly allows traffic from the local database. Our final security groups looked like this.

Bastion:	SSH via anywhere
Web Server (Booking):	SSH via local network HTTP via anywhere MYSQL/AURORA from local network
Web Server (Hours):	SSH via local network HTTP via anywhere
Benchmark:	SSH via local network HTTP via local network

Database: MYSQL/AURORA from
A local network

Compute Resources (EC2 instances and RDS)

In our first attempt, we made a dedicatedAPI instance that sends GET and POST requests to the backend(RDS). Our RDS database contained a table dedicated to monitoring the availability of the Tennis Courts. Our second instance was the actual site. We created it using HTML, CSS, and PHP. We attempted to use javascript in our project but we ended up having many issues with it. Our third instance was our Bastion. Our Bastion acted like a layer of security that was required to gain access to our API, Website, and Database. However, around 9pm we ended up having our lab corrupted which meant we had no way of accessing our EC2.We had one final instance that was dedicated to doing tests but we couldn't fully implement it due constraints on the Lab account.

As said previously, our API is no longer in use so we didn't create an EC2 instance for it. In total, we used a total of four EC2 instances and one RDS. However, to replicate the project we created those instances again. A brief description of instances are:

The first instance is the Bastion Host, which provides secure SSH access from anywhere. It serves as a gateway to access other instances only accepting local traffic. This instance uses an Amazon Machine Image (AMI). It resides on public subnet-1a.

The second instance is the Benchmark Server, an Ubuntu instance created from Cloudformation. Its main goal is to perform stress testing on our Booking Website to ensure its reliability and performance under load. It is on private subnet-1b.

The third instance is the Hours of Operation Website which is an Amazon Machine image within public subnet-1a. The purpose is to show when the tennis court opens and when it closes.

The last EC2 instance is the Booking Website which is hosted on an Amazon Machine Image, within public subnet-1a. This website is designed to display court availability by retrieving information from an RDS database, providing users with up-to-date scheduling information.

The RDS uses MYSQL and is on both private subnet-1a and private subnet-1b. All of the security groups defined previously were used (the naming of each security group corresponding to the compute resource).

Infrastructure as Code (IAC)

Nothing has changed in this area from "**Previous Outline**". Using ex3-stack.yaml from exercise 3 as a template, we edited the yaml file to change the default values to use an Ubuntu image rather than an Amazon image, use the ID of private subnet 1-b, and use the ID of the Benchmark security group. We deployed it using Cloudformation and housed it in an encrypted S3 bucket.

EBS volume

Just like in “**Previous Outline**”, we gave bastion and additional EBS volume. However, we don’t have any .txt files being stored on it due to time constraints. Though if we need to, the EBS volume is already prepared and can be used any time.

S3 Bucket

We used the S3 bucket for deploying the Benchmark.yaml file from and to store the tenniscourts.png image. We followed exercise 1 for this step. As mentioned before, this is the way we can deploy it using CloudFormation. The purpose of storing the tenniscourts.png image is to ensure any of us can retrieve the image since we used that for the Booking Website.

Stress-testing

To evaluate the performance and reliability of our Booking Website under high load conditions, we created the Benchmark Server, as explained previously. We set up the Bastion Host, located in one of our public subnets, to be a secure SSH gateway to access it as well as other instances.

To initiate the stress testing, we used Apache Benchmark, which sent a total of 25,000 requests with a concurrency level of 50, simulating a high utilization on the website. Thanks to this we could observe how reliable our website is and its performance during peak demand.

Containerization & Orchestration

Since the API was no longer in use, we containerized/orchestrated the Hours of operation site. A Docker image was created from the Hours-of-operation EC2 instance. It is stored under ECR, in the private repository “operating-hours”. From the URL of the image in ECR, we created a cluster in ECS called “hours-of-operation”. There, we can run a task so we can connect to the web page hosted by the ECS Service.

KMS Encryption

We used the KMS Encryption keys to encrypt the S3 bucket that housed the image of the Tennis Court, and YAML file for our benchmark server (IAC).

EC2 & Cloudtrail monitoring

A Cloudtrail trail was created in order to track the resource changes involving the KMS key encryption and all interactions with our EC2 instances. In addition, we followed the detailed event history closely, as CloudTrail provides insights into resource utilization and system changes.

Horizontal scaling, Load balancing, & Auto scaling

We first started off by creating an image from the Webserver (Booking) EC2 instance. For load balancing the target CPU utilization was set to 60%. Like lab 3, the minimum number of EC2 instances is 2 and the maximum is 6. This ensures that the load balancer can properly function and be able to use more Webserver (Booking) instances that are created from Auto scaling. For the project, Auto Scaling orchestrates horizontal scaling for us.