Room Booking

James Liu 566269 Ben Nicholson 2956890

October 9, 2024

1. Introduction

In the current age of cloud computing, application deployment strategies play a pivotal role in ensuring seamless user experience, scalability, and cost optimization. Our project, the Online Room Booking System, embodies these principles while being deployed on AWS.

2. Architecture and Application Design

The Online Room Booking System is a web-based application, providing a platform where users can book rooms, and administrators manage the available shared rooms. The application is divided into two components: a user-friendly frontend and a robust backend. It leverages the power of PHP for backend logic, with MySQL (hosted on Amazon RDS) serving as the data storage. The application involves CRUD operations that the user can perform.

Our system currently consists of a login page where a user can choose to log in as an admin or a regular user. The user can view and book available rooms. Once a room is booked, it is marked as unavailable. Administrators have access to a separate interface to add, delete, and manage rooms. The "admin" and "user" services are hosted by their respective EC2 instances. Amazon RDS was implemented to host the database in the cloud.

3. Deployment Strategy

3.1 Virtual Machines - EC2 Instances

Two EC2 instances were employed:

- **WebServer**: Serves as the EC2 instance hosting the user-facing web pages.
- **AdminServer**: Serves as the EC2 instance hosting the admin interface for managing rooms.

Both EC2 instances run **PHP** to serve the backend and frontend.

3.2 Storage Service - Amazon RDS

Amazon RDS is optimized for MySQL database hosting. It contains structured data about rooms, users, and administrators.

4. Deployment Procedure

EC2 Setup: Two EC2 instances were set up: one for the frontend and another for the admin interface.

RDS Configuration: We created a MySQL database in Amazon RDS. Security groups were configured to ensure only our EC2 instances could communicate with the database.

Manual Deployment Steps: The deployment process is documented step-by-step in the repository, making it easy for new developers to join the project. Key steps include the creation of a key-pair, aws configuring & the population of addresses within our EC2 instances for db connection.

admin email: admin@example.com
admin pw: root

- clone the repo: https://github.com/BenCNicholson/COSC349_Assignment_2
- cd ~/../COSC349_Assignment2
- run: vagrant up
- run vagrant ssh
- copy your AWS CLI into ~/.aws/credentials
- run: aws configure (set region to "us-east-1"
- Gather a keypair from AWS
- Set the setup-webservers.tf keypair names to your designated keypair
- run: terraform init
- run terraform plan
- run terraform apply
- gather the printed addresses.
- ssh into both the Admin-Server and Web-Server
- Configure /var/www/main/dbconnect.php by passing in the addresses
- run: sudo chmod +x /var/www/main/database/setupDatabase.sh
- run: /var/www/main/database/setupDatabase.sh
- Use the WebServer address in the browser, explore the webpage.

5. User Interaction Guide

Users can browse available rooms, view details, and book them. Administrators have a separate portal to add, modify, or remove rooms. The intuitive interface ensures that even non-tech-savvy users can interact with the system.

- **Frontend (User)**: Users can log in, browse available rooms, and book them. Once booked, the room is marked unavailable.
- **Admin (Administrator)**: Admins can log in and manage rooms, adding new ones or removing those no longer available.

6. Cost Estimation

We have estimated the cost of running the Shared Resource Booking System on AWS in the Sydney region (ap-southeast-2) based on the following services:

6.1. Idle Usage Costs

In an idle state, infrastructure is running but has minimal activity. Costs are primarily for uptime without significant data transfer or processing.

- **EC2 (Elastic Compute Cloud)**: *t3.micro instance* (2 vCPU, 1 GB Memory)
  - On-Demand Price: $0.0124 per Hour.
  - Monthly: $0.0124 x 24 hours x 30 days = $8.93
- **RDS (Relational Database Service)** for MySQL: *db.t3.micro instance*
  - On-Demand Price: $0.030 per Hour.
  - Monthly: $0.030 x 24 hours x 30 days = $21.60
- **Snapshot Storage for Database Backups**: 20GB Snapshot: $2.50 per month.
- **Data Transfer Costs**: If idle, no significant data transfer. Assuming minimal data transfer: $0.00.

Total Monthly Idle Costs: $33.03 per month.

6.2. Light Usage Costs

Light usage reflects occasional traffic with low data transfer and periodic database updates.

- **EC2 (Elastic Compute Cloud)**: *t3.micro instance* (2 vCPU, 1 GB Memory)
  - On-Demand Price: $0.0124 per Hour.
  - Monthly: $0.0124 x 24 hours x 30 days = $8.93
- **RDS (Relational Database Service)** for MySQL: *db.t3.micro instance*

  - On-Demand Price: $0.030 per Hour.
  - Monthly: $0.030 x 24 hours x 30 days = $21.60
- **Snapshot Storage for Database Backups**: 20GB Snapshot: $2.50 per month.
- **Data Transfer Costs**: Light usage data transfer estimated at 5GB per month (1GB free):
  - 4GB x $0.114 = $0.456

Total Monthly Light Usage Costs: $33.49 per month.

6.3. Heavy Usage Costs

Heavy usage assumes significant traffic, database queries, and large data transfer volumes.

- **EC2 (Elastic Compute Cloud)**: *t3.micro instance* (2 vCPU, 1 GB Memory)
  - On-Demand Price: $0.0124 per Hour.
  - Monthly: $0.0124 x 24 hours x 30 days = $8.93
- **RDS (Relational Database Service)** for MySQL: *db.t3.micro instance*
  - On-Demand Price: $0.030 per Hour.
  - Monthly: $0.030 x 24 hours x 30 days = $21.60
- **Snapshot Storage for Database Backups**: 20GB Snapshot: $2.50 per month.
- **Data Transfer Costs**: Heavy data transfer estimated at 100GB per month (1GB free):
  - 99GB x $0.114 = $11.29

Total Monthly Heavy Usage Costs: $44.32 per month.

7. Development Process and Git Insights

7.1 Development Milestones

We can categorize our development process into three main phases:

- **Database Design**: Planning the database was our top priority before coding. This structured approach ensured we had a solid foundation for resource management.
- **Backend Development**: After setting up the database, we focused on the backend using PHP handling business logic and database interaction.
- **Frontend Development**: Finally, we built a user-friendly interface.

7.2 Git Repository Analysis

Our Git usage has room for improvement. While we tracked changes, there were periods of inactivity, and some commits lacked detailed descriptions. We primarily worked on one computer logged into Ben's GitHub account, which may cause confusion during grading.

7.3 Challenges and Problem-Solving

One key problem faced was the non-EC2 service. We simply couldn't get this to work. Each time trying different services, Lambda and Cloudfront we kept getting these access errors. Therefore unfortunately. We were unable to get this key part of our assignment running.

8. Conclusion

In conclusion, the Shared Resource Booking System demonstrates the power of cloud computing for application deployment. While we encountered challenges, the project showcases our commitment to creating a functional and cost-effective solution. Continuous learning and improvement will further enhance the capabilities of this system and the utilization of AWS services.