**COSC349 Assignment 2**

Name: Bernice Ishumi

Student ID: 1287869

**Introduction**

I have developed an e-commerce platform to offer customers a simple and convenient way to buy electronic devices. The system includes essential features such as customer account management, user authentication, and product browsing, resulting in a robust and user-friendly application.

Behind the scenes, the system is distributed across 2 (VMs) to enhance performance and scalability. This was achieved using the EC2 AWS service, allowing for easy scaling for the web application. This ensures that the platform remains responsive under varying loads. I have implemented Amazon RDS with MySQL for data storage, providing a reliable and scalable database solution. Additionally, I have integrated Amazon CloudWatch to perform real-time monitoring and insights into system performance.

**Application Deployment Process**

1. **EC2 Instance Setup (my-web-server)**

Firstly, I started by provisioning an Amazon EC2 instance to serve as the web server. This instance was configured with Amazon Linux for the Amazon Machine Image. The t2.micro instance type and a security group that allowed SSH from anywhere and HTTPs and HTTP traffic options. This will enable users to access the application through the internet.

1. **Amazon RDS Setup (database-server)**

Then I set up an Amazon RDS DB instance as my database server. This instance was configured with the MySQL engine type alongside the Free Tier template. Additionally, I connected the database with my webserver EC2 instance to allow communication between them. Then I accessed the database using the SSH client of the web server client to set up my relational tables and data.

1. **Backup EC2 Instance Setup (my-backup-server)**

To ensure high availability and redundancy, I have set up a backup EC2 instance with identical features and configurations as the primary web server. This backup instance is also connected to the same RDS database, enabling it to act as a failover option if the primary instance becomes unavailable.

1. **Load Balancer Setup (cosc349-loadbalancer)**

To efficiently manage traffic between the two EC2 instances, I have configured a Load Balancer. This load balancer evenly distributes incoming requests across both instances. Additionally, health checks have been set up to monitor the status of each EC2 instance, ensuring that traffic is directed only to healthy instances.

1. **Deployment**

The application was then deployed. The DNS was set up to point to the load balancer, allowing users to access the platform through a single endpoint. After the launch, I continued to monitor the application using Amazon CloudWatch.

Endpoint: <http://cosc349-loadbalancer-2094742701.us-east-1.elb.amazonaws.com/index.php>

A screenshot of a computer

Description automatically generated

**Accessing the Application in the Cloud**

Users can access the application from any device with an internet connection and a web browser. To do so, simply enter the URL provided above into your web browser's address bar. This URL will take you directly to the homepage of the application, where you can browse products, sign in, and create a new account by clicking on various buttons and links within the application.

**Application Design**

The e-commerce application's design is based on a multi-tier architecture that improves performance, scalability, and maintainability. It consists of three main layers: the web server layer, the database layer, and the load balancing layer.

**Primary EC2 Instance**: This instance serves as the main web server, handling incoming HTTP requests from users and running the application code.

**Backup EC2 Instance**: A second EC2 instance was created as a backup, mirroring the prior instance's configuration. This instance serves as a failover option, ensuring high availability. In case the primary instance fails, the backup instance can take over without interrupting service.

**Database Layer (Amazon RDS):** The application uses Amazon RDS (Relational Database Service) to manage the MySQL database, which stores important data such as user accounts, product information, and other details. Both EC2 instances are connected to the RDS database, allowing them to perform database operations without any issues.

**Load Balancer:** This isconfigured to distribute incoming traffic evenly across both EC2 instances. his helps prevent any single instance from being overloaded with traffic, which in turn improves performance and reliability. The load balancer constantly checks the health of the EC2 instances. If an instance becomes unhealthy or unresponsive, the load balancer automatically redirects traffic to the healthy instance. Additionally, I have enabled sticky sessions which will ensure that requests from the same client always go to the same instance, so the session remains active on that instance. However, the duration of the sessions is only one hour.

When users access the application, their requests are directed to one of the two EC2 instances through the load balancer using load-balancing algorithms. The selected EC2 instance processes the requests, interacting with the RDS database to retrieve or store information as needed. The database processes the request and returns the required data to the requesting EC2 instance, which then formats it for the user interface. Finally, the EC2 instance sends the response back to the user through the Load Balancer, completing the interaction.

**Justifications of Design**

**Amazon RDS (Relational Database Service)**

Amazon RDS is a service that simplifies the setup, operation, and scaling of relational databases in the cloud. With RDS, I can focus on the application rather than managing database infrastructure. This was greatly advantageous as I could not think of a way to create a database using S3 buckets.  Additional benefits of RDS include scalability. RDS allows for easy scaling of database resources ensuring that the database can grow alongside application demands. With high availability, RDS automates routine database tasks such as backups, maintenance, and monitoring, which significantly reduces the administrative tasks associated with managing and setting up a database.

**Usage:**

In my e-commerce application, Amazon RDS is used to manage the MySQL database that stores essential data, including user accounts, product information, and other details. The web server EC2 instances connect to the RDS instance to perform the following operations:

Data Retrieval: When users browse products the EC2 instances query the RDS database for relevant data.

Data Storage: When users create accounts the EC2 instances send information to the RDS database, ensuring that all information is accurately reflected and stored.

**Amazon CloudWatch**

Amazon CloudWatch is a monitoring and management service designed for AWS cloud resources and applications. It provides real-time monitoring of metrics, logs, and events, allowing insights into the performance and operational health of my AWS environment. One of the main reasons I chose this service is its ability for real-time monitoring. CloudWatch provides real-time metrics for AWS resources, enabling me to monitor performance and take quick action if issues arise. In addition, the option of creating custom metrics to track specific data points for applications allowed for a tailored monitoring experience. CloudWatch also allows users to set alarms based on predefined thresholds for various metrics, enabling automatic notifications or actions when specific conditions are met (e.g., high CPU usage or low database connections). Currently, I have set up 3 alarms: 2 to alert if the CPU usage of my EC2 instance is above 75%, and the other to alert if my database write operations exceed a certain threshold. In the future, I could automate scaling, so that if one of the alarms is triggered, the application is scaled upwards to handle the new demand.

**Usage:**

In my e-commerce application, Amazon CloudWatch is utilised to monitor both the EC2 instances and the RDS database. CloudWatch tracks metrics such as CPU utilisation, memory usage, and disk I/O for the EC2 instances, providing insights into resource usage and application performance. Metrics for the RDS instance, including connections, query performance, and read/write latency, are monitored to ensure the database operates efficiently. I configured alarms to notify the owner of potential issues, such as high CPU usage on the web server or increased query latency on the database. This proactive monitoring helps address problems before they impact users.

**Running costs of the Application in the Cloud**

1. **When Idle**

**Two EC2 t3.micro instances:**

Cost per hour per instance: ~$0.0104

Monthly cost ( 2 instances run 24/7 idly): ~$15.06 per instance

Thus $30.12 total.

**Amazon RDS (db.t4g.micro, MySQL):**

Cost per hour: ~$0.0126

Monthly cost (24/7): ~$9.10

Storage: 20 GiB of general-purpose SSD storage

Monthly storage cost: ~$2.30

**Load Balancer:**

Cost per hour: ~$ 0.0225

Monthly cost (24/7 idly): ~$15.12

**Amazon CloudWatch**:

Estimated cost: ~$5.00 per month

Total: ~ $61.64 per month

1. **When Lightly Used.**

**Two EC2 t3.micro instances:**

Monthly cost: ~$31.06 per instance

**Amazon RDS (db.t4g.micro, MySQL):**

Monthly cost (24/7): ~$9.10

Storage: 20 GiB of general-purpose SSD storage

Monthly storage cost: ~$2.30

**Load Balancer:**

Cost per hour: ~$ 0.0225

Data processing cost: ~0.008 per GB

Monthly cost: ~$20.496

**Amazon CloudWatch**:

Estimated cost: ~$6.00 per month

Total: ~ $68.96 per month

**Narration of Demonstration**

The demonstration video has no audio so I will include a textual description of my screen recording instead.

[0:10]

Starting at the homepage of the platform.

“Welcome to my demonstration of the application I created for Assignment 2. In this video, we'll walk through the main functionalities of our application."

[0.13]

The user clicks on the ‘Browse Products’ button.

View products page displays all the products currently in the database.

[0.15]

The user selects some categories. This filters the page to only display products of the chosen category.

[0.20]

The user clicks the ‘All’ category to display all the products regardless of category.

[0:22]

The user clicks the ‘Sign in’ button. The application then displays the Sign-in Page.

[0.30]

Testing the user authentication of the software.

[0.32]

The user clicks the ‘create one’ button. The Create an Account page is displayed.

[0.34]

The user clicks the 'Create Account’ button without filling in the required fields. This demonstrates the user validation and verification of the platform.

[1.03]

After filling in all the required fields correctly the user clicks the ‘Create Account’ button

[1.06]

After the successful creation of the account, the user is transferred to the homepage.

[1.08]

The user clicks the ‘Sign in’ button. The application then displays the Sign-in Page.

[1:15]

Trying to log in with the wrong password. Testing the user authentication of the software.

[1:23]

Trying to log in with the wrong username. Testing the user authentication of the software.

[1:33]

After successfully logging in using the correct username and password, the customer is transferred to the view products page.

[1:40]

The customer clicks the ‘Sign Out’ button. After successfully logging out the customer, the user is transferred to the homepage.

**End of Demonstration**

**Note:** Apologies for the lacklustre application. I could not figure out how to add a CSS script to the webserver.