**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.

The 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.

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**

goes for low traffic, reducing the number of resources a VM needs without affecting the whole system.

Thirdly, bugs can be isolated to specific VMs, making identifying and resolving problems easier without impacting other parts of the system. I found this very helpful during the assignment. In a single server, trying to find one specific small bug would be an arduous task of filtering thousands of lines of code.

Lastly, to add to the points above, if one VM fails (e.g., webserver), the others (e.g., database) continue to operate. This helps maintain overall system availability. However, in a single server architecture, if the server fails, so does the whole system.

Thus, having a separate VM architecture helps reduce vulnerability while also increasing security, performance, and availability.

**Developer Suggestions:**

1. Shopping Cart and Checkout

Implementing a shopping cart and checkout system will allow customers to add items to a cart, review their purchases, and complete transactions. This can be achieved by adding new PHP pages to manage the shopping cart and handle the checkout process. Modify existing pages such as view-products.php to include options for adding products to the cart and linking to cart.php. Additionally, changing the database setup to have more tables in the database to store the order and its details.

To rebuild the application, enter the command "vagrant reload" in the command line.

2. More Informational Pages:

Enhance the website by adding additional web pages to improve user interaction and provide more information.

For example:

- help.php: for user assistance

- contact.php: for contact information

- about.php: for details about the company

To rebuild the application, enter the command "vagrant reload" in the command line.

**Build Process**

1. Clean Build

**Components to download:**

1. The base Vagrant box: ubuntu/focal64 is about 500MB in size.
2. Apache2 and its related packages for the webserver are about 5MB in size.
3. PHP and related packages about 10MB.
4. libapache2-mod-php component that allows the Apache web server to interpret PHP scripts and serve them as web pages.
5. MySQL Server with an approximate size of 25 MB for each database server.

Therefore, the approximate volume for a clean build of the platform is about 600 MB.

1. Redeployment

**Components to download:**

1. The base Vagrant box: ubuntu/focal64 is downloaded and cached. Thus, no download volume.
2. Apache2 and its related packages for the webserver are already installed.
3. PHP and related packages are already installed.
4. libapache2-mod-php component that allows the Apache web server to interpret PHP scripts and serve them as web pages. Already installed.
5. MySQL Server is already installed on each database server.

Therefore, the approximate volume for a redeployment build of the platform is close to 0MB. The volume would change if any of the components required updates or there were changes in the component version.

**Narration of Demonstration**

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

[0:00]

Starting at the homepage of the platform.

“Welcome to my demonstration of the application I created for Assignment 1. In this video, we'll walk through the main functionalities of our application."

[0.08]

The user clicks on the ‘Browse Products’ button.

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

[0.12]

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

[0.17]

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

[0.31]

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

[0.36]

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

[1.06]

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

[1.07]

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

[1.09]

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.

[1:30]

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

[1:37]

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

**End of Demonstration**

**Unsuccessful Attempts:** The plan was to have 4 VMS with 2 web servers and 2 database servers. One would have been used for the pages that require user input to isolate the system from malicious attacks that occur from user input. While the other web server would host the other pages. I decided to forgo the idea when I was struggling with session connections between the 2 web servers. Thus, the final platform of only 3 VMS was finalised.