SCALABLE AND CRASH-PROOF E-COMMERCE WEBSITE USING AWS CLOUD SERVICES

TEAM MEMBERS :

1) ADITHYA R
2) AROCKIYA ROSHAN J
3) KISHORE S

PROJECT OVERVIEW

Objectives

The main objective of this project is to create a scalable and reliable cloud-based e-commerce website called *Wise Cart* using AWS services. The front-end is built using HTML, CSS, and JavaScript and hosted on an EC2 instance with Apache2. To ensure high availability, the system uses an Elastic Load Balancer and Auto Scaling Group. CloudWatch monitors performance and traffic, while SNS sends real-time alerts to prevent downtime. This setup helps handle heavy traffic efficiently and ensures a smooth shopping experience for users.

Key Features

1) **Scalable Architecture**: Automatically handles increasing traffic using AWS services like EC2 and Auto Scaling Group.

2) **Elastic Load Balancer (ELB)**: Distributes incoming user traffic evenly to avoid overloading a single server.

3) **Real-time Monitoring**: AWS CloudWatch tracks system performance, uptime, and traffic patterns.

4) **Instant Alerts**: Amazon SNS sends notifications if the server is down or traffic spikes unexpectedly.

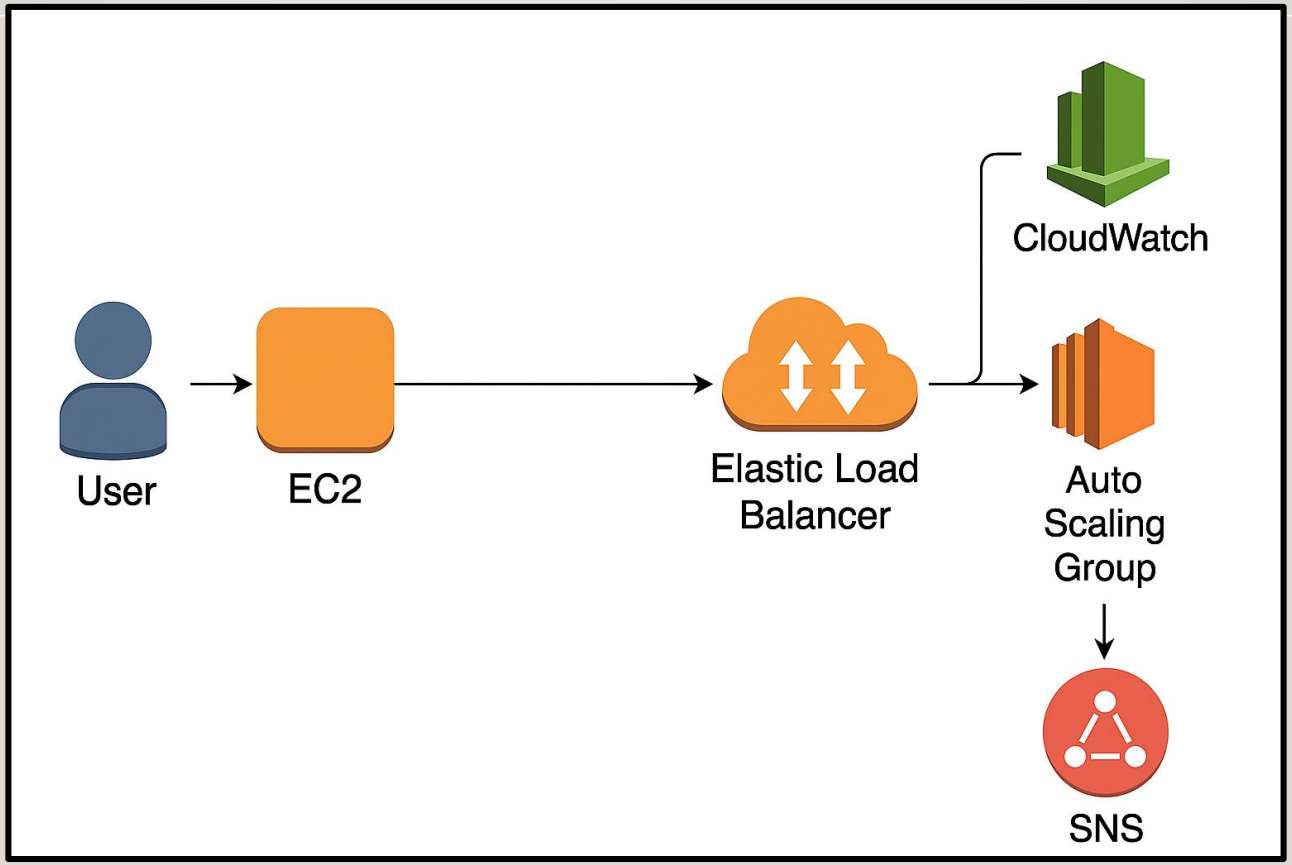
5) **Auto Scaling Group (ASG)**: Automatically increases or decreases the number of EC2 instances based on traffic, ensuring performance and cost-efficiency.

6) **Simple Frontend Website**: A user-friendly shopping interface built with HTML, CSS, and JavaScript.

7) **Cost-Effective Hosting**: Deployed on Ubuntu EC2 instance with Apache2, keeping it lightweight and budget-friendly.

8) **Improved Uptime**: Helps prevent crashes or downtime, ensuring continuous availability.

ARCHITECTURE DIAGRAM



TECHNOLOGIES USED

Frontend Technologies

- **HTML5** – Structure of the web pages
- **CSS3** – Styling and layout
- **JavaScript** – Interactivity and dynamic behavior

Cloud Services (AWS)

- **Amazon EC2** – Hosting the frontend on a virtual Ubuntu server using apache2
- **Elastic Load Balancer (ELB)** – Distributes traffic evenly
- **Auto Scaling Group (ASG)** – Automatically scales EC2 instances
- **Amazon CloudWatch** – Real-time performance monitoring
- **Amazon SNS** – Sends instant alerts/notifications

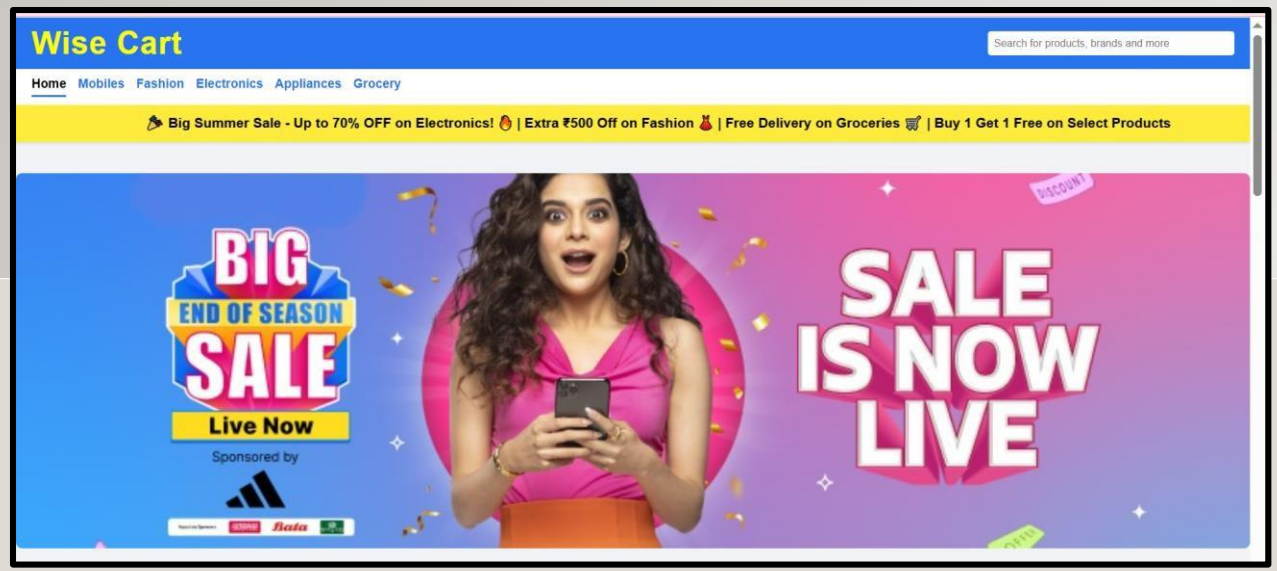
PROGRAMMING LANGUAGE JUSTIFICATION

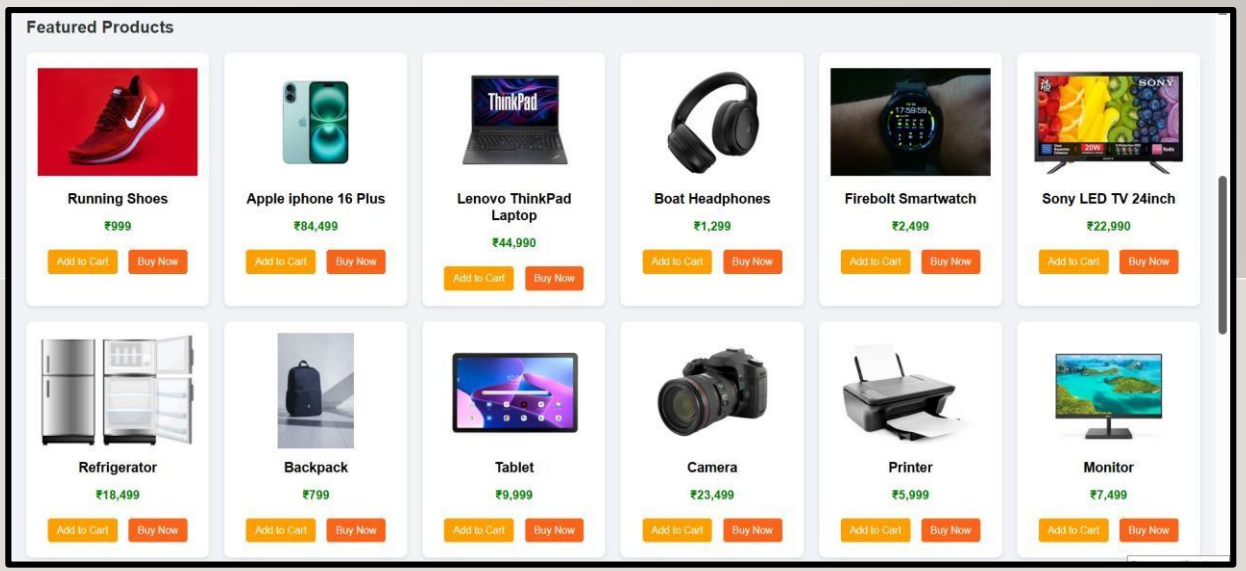
1) **HTML** is used to build the structure of the website.

2) **CSS** is used to design and style the layout, colors, fonts, and overall visual appearance.

There are no backend technologies used in this project. The website is entirely static and was developed solely for illustrative purposes to demonstrate how AWS services like Elastic Load Balancer, CloudWatch, and Auto Scaling Group can handle high traffic and ensure system reliability. The focus is on frontend deployment and cloud infrastructure, not dynamic or database-driven functionalities.

UI SCREENSHOTS & EXPLANATION





The user interface of *WiseCart* is a simple, static e-commerce frontend built using HTML and CSS. The homepage includes a clean navigation bar, a centered banner image, and a featured product section displayed in a grid format. The layout is fully responsive, ensuring accessibility on both desktop and mobile devices. While the site contains no dynamic functionality or backend, it serves as a lightweight and illustrative frontend to simulate real-world traffic and demonstrate the AWS-based scalable hosting architecture.

CODE EXPLANATION

1) **Navigation Bar**

```
<nav>
    <a href="#">Home</a>
    <a href="#">Mobiles</a>
     …
</nav>
```

Explanation

Creates a top menu bar for users to navigate across product categories.

2) **Product Grid Layout**

```
<section class="products">
  <div class="product-card">
    <img src="images/product1.jpg" alt="Product 1">
    <h4>Product Name</h4>
    <p>₹Price</p>
  </div>
  …
</section>
```

Explanation

Shows products in a card format using images, names, and prices.

3) **CSS Responsive Layout**

```
.products {
   display: flex;
   flex-wrap: wrap;
   justify-content: center;
}
```

Explanation:
Makes the product section responsive and neatly arranged across all screen sizes.

4) **Scrolling Text Banner**

```
<marquee behavior="scroll" direction="left">
  Mega Sale: Flat 50% OFF on Mobiles & Electronics!
Shop        Now
</marquee>
```

Explanation:
Displays a scrolling announcement banner to grab user attention with ongoing offers.
It moves the text from right to left continuously using the <marquee> tag (legacy HTML).

FAQS – COMMONLY ASKED QUESTIONS

1)Why did you use AWS instead of other cloud providers?
A) AWS offers a wide range of scalable and reliable services. It's widely adopted in the industry and provides free-tier usage ideal for student projects.

2)Why is there only one EC2 instance in the architecture?
A) Since this is a demo project showcasing frontend only, a single EC2 instance is sufficient. The architecture can be scaled easily in real scenarios using ASG (Auto Scaling Group).

3)What is the purpose of ELB if there's only one EC2?
A) ELB is added to demonstrate how traffic would be distributed in a scalable setup. It prepares the architecture for future scalability with multiple EC2s.

4) Why is there no backend or database used?
A) The focus of this project is to showcase cloud architecture handling traffic. The website is static and meant for demonstration purposes only.

5) What role does AWS CloudWatch play in this project?
A) CloudWatch monitors the EC2 instance's performance. It can trigger alarms based on CPU usage, which can then notify users or trigger scaling actions.

6) Why did you use Apache2?
A) Apache2 is a lightweight and widely-used HTTP server. It's easy to configure and suitable for hosting static websites on Ubuntu EC2 instances.

CONCLUSION & FUTURE ENHANCEMENTS

This project successfully demonstrates the power of cloud computing in solving real-world challenges like website downtime due to traffic surges. By leveraging AWS services such as EC2, Elastic Load Balancer, CloudWatch, and SNS, we created a reliable, scalable, and cost-effective hosting environment for a static e-commerce website. Though the site itself is simple and static, the infrastructure behind it showcases how even basic websites can benefit from enterprise-level architecture, ensuring high availability, uptime, and user satisfaction. This lays a strong foundation for transforming the prototype into a fully functional, production-ready solution in the future.

Future Enhancements

- **Backend:** Add backend functionality using Node.js or Python to support dynamic features like login, product management, and payments.
- **Database:** Integrate a database (e.g., MySQL or MongoDB) to store user, product, and order data for a fully functional e-commerce experience.
- **Auto Scaling Group (ASG):** Add multiple EC2 instances with autoscaling based on traffic spikes.
- **CI/CD Pipeline:** Automate deployments using AWS CodePipeline and GitHub.

THANK YOU