**AWS Banking Hosting**

**Project submitted to the**

**APSSDC**

**Bachelor of Technology**

**In**

**Computer Science and Engineering**

**Nadimpalli Satyanarayana Raju Institute**

**of Technology**

**Submitted By**

**Thurupati Jnaneswari-22NU1A05B7**



**Under the guidance of**

**Mrs. Rama Kishna,APSSDC**

**Mrs.Sumana Bethala,APSSDC**

**June 2025**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| S.NO | CONTENTS |  |
| 1 | Abstract |  |
| 2 | Introduction |  |
| 3 | Methodology |  |
| 4 | Architecture |  |
| 5 | Implementation |  |
| 6 | Result |  |
| 7 | Conclusion |  |

**ABSTRACT**

Cloud computing has redefined the way web applications are hosted, providing scalable, cost-effective, and secure solutions for enterprises and developers. This project focuses on deploying a secure, cloud-hosted online banking system using Amazon Web Services (AWS). The application, built using PHP and MySQL, is hosted on an EC2 instance with database services handled by Amazon RDS. A Virtual Private Cloud (VPC) ensures secure network segmentation, while optional services like Amazon S3 and CloudWatch provide file storage and real-time monitoring.

This end-to-end project simulates real-world cloud deployment practices, helping learners understand Infrastructure as a Service (IaaS), Database as a Service (DBaaS), and application monitoring through CloudWatch. The successful implementation demonstrates the feasibility of using AWS for deploying scalable and secure financial applications.

The core objective of this project is to simulate a real-world, production-grade deployment pipeline suitable for financial applications, where security, performance, and availability are critical. The banking application includes functionalities such as user registration, login authentication, account balance viewing, and basic transaction handling — all of which are supported by a secure and scalable backend infrastructure.

To achieve this, the project begins by setting up a Virtual Private Cloud (VPC) with separate public and private subnets to logically separate resources based on their access requirements. The EC2 instance, acting as the web server, is launched within the public subnet and configured with a LAMP (Linux, Apache, MySQL, PHP) stack to serve the banking website. The backend database is created using Amazon RDS (MySQL engine) and placed in a private subnet to ensure restricted access, thereby improving security and compliance with industry best practices

**INTRODUCTION**

The rapid digitization of services in today’s economy has made web-based applications — particularly in the financial sector — an essential part of modern life. Online banking platforms allow users to perform financial transactions, manage accounts, and access personalized services from anywhere in the world. However, hosting such applications requires a robust, secure, and scalable infrastructure to handle sensitive user data, ensure 24/7 availability, and meet strict regulatory standards for performance and security.

Traditional server infrastructure often falls short in meeting these demands due to limitations in scalability, manual maintenance, and high operational costs. In contrast, cloud computing — specifically Amazon Web Services (AWS) — offers a powerful suite of tools and services that address these challenges. AWS provides flexible, pay-as-you-go resources with integrated security, automation, and monitoring capabilities, making it a leading platform for modern application deployment.

This project explores how to deploy an online banking system on AWS using a beginner-friendly approach with real-world relevance. The banking application, developed using the PHP programming language and MySQL database, includes basic functionalities such as user authentication, balance checking, and transaction viewing. These features are hosted on AWS services to simulate a realistic deployment scenario similar to what is used by modern fintech companies.

**The primary objective is to:**

* Build a secure and scalable hosting environment using EC2 (Elastic Compute Cloud) for the application server.
* Store user and transactional data using Amazon RDS (Relational Database Service) with the MySQL engine in a private subnet for added security.
* Leverage Amazon VPC (Virtual Private Cloud) to define and manage networking components including subnets, route tables, and gateways for isolated and controlled traffic flow.
* Utilize WinSCP and PuTTY for secure file transfer and remote access to the EC2 server.
* Optionally integrate Amazon S3 (Simple Storage Service) for file uploads and Amazon CloudWatch for system monitoring and performance analytics.

By implementing this system step by step, learners are introduced to critical cloud computing concepts including:

* Infrastructure as a Service (IaaS)
* Database as a Service (DBaaS)
* Secure networking using subnets and security groups
* Application deployment automation
* System observability and monitoring

**METHODOLOGY**

The methodology for deploying the online banking application on AWS follows a structured, multi-phase approach, ensuring modularity, scalability, and adherence to cloud security principles. Each step was planned and executed with a focus on reliability, isolation of resources, ease of management, and real-world relevance. The goal is to simulate how enterprise-grade applications are built and maintained using modern cloud infrastructure.

The entire methodology is divided into the following core phases:

**Phase 1: Cloud Architecture Planning**

Before implementation, the architecture of the cloud environment was carefully designed to meet the following criteria:

* Secure hosting of the application and data layers
* Segregation of resources for better management
* Internet accessibility only where necessary
* Scalability and monitoring capability

A **three-tier architecture model** was selected:

* **Presentation Layer (EC2 + Apache + PHP)** — Publicly accessible via HTTP
* **Application & Business Logic Layer (PHP Scripts)** — Hosted on the EC2 server
* **Data Layer (MySQL on RDS)** — Secured in a private subnet

**Phase 2: Virtual Private Cloud (VPC) Setup**

A **custom VPC (Virtual Private Cloud)** was created to host all AWS resources in a logically isolated network environment. VPC setup involved:

* **CIDR Block Allocation**: The VPC was assigned a CIDR block of 10.0.0.0/16, allowing up to 65,536 IP addresses.
* **Subnet Design**:
* **Public Subnet (10.0.1.0/24)**: For EC2 instances with internet access.
* **Private Subnet (10.0.2.0/24)**: For RDS instances without public access.
* **Internet Gateway Configuration**: Attached to the VPC and routed to the public
* subnet via a custom route table.

**Routing Tables**:

* Public route table allows 0.0.0.0/0 traffic to the internet via the Internet Gateway.
* Private route table remains isolated to enhance security for the database layer.

**Phase 3: EC2 Server Configuration**

An **Amazon EC2 (Elastic Compute Cloud)** instance was launched to serve as the web server:

**1.AMI Used**: Ubuntu Server 22.04 LTS (lightweight and widely supported).

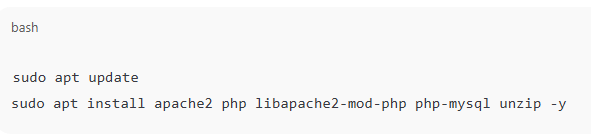
**2.Instance Type**: t2.micro (eligible for AWS Free Tier).

**3.Security Groups**: SSH (port 22) — restricted to the developer's IP.

HTTP (port 80) — open to the public for web access.

**Key Pair**: PEM key generated for SSH login, converted to PPK using PuTTYgen for WinSCP and PuTTY.

**LAMP Stack Installation**:

****

**Phase 4: Application Deployment**

Banking website files (home.php, login.php, connect.php, etc.) were uploaded using WinSCP over SFTP to the EC2 instance in /var/www/html/.

* **Permissions Configuration:**
  + sudo chown -R www-data:www-data /var/www/html
  + sudo chmod -R 755 /var/www/html

**Phase 5: RDS (MySQL) Database Setup**

Amazon RDS (Relational Database Service) was used to host the MySQL database. This allows for a managed, secure, and easily scalable data layer.

* **Database Settings:**
  + - * + Engine: MySQL
        + DB Name: banking
        + Master Username: admin
        + Password: Custom-secure-password
* **Placement:**
  + - * + Launched within the private subnet to prevent direct public access.
      * Access restricted to EC2 instance’s security group using inbound rules.
* **Connection Configuration:**
  + Endpoint retrieved from RDS dashboard.
  + PHP script (connect.php) updated with endpoint, user credentials, and DB name.
  + Database tables (users, transactions) created manually or using SQL scripts.

**Phase 6: Optional Integration with Amazon S3**

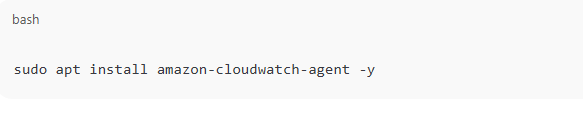
To enhance the platform with document upload capability, Amazon S3 was integrated:

* A unique bucket (e.g., banking-uploads-2025) was created.
* PHP SDK (aws/aws-sdk-php) was configured to:
  + Authenticate using IAM roles or keys
  + Upload files and generate access URLs
  + Handle file permissions and access policies

This functionality can be used for uploading identity documents, proof of address, etc., in real banking environments.

**Phase 7: Monitoring with Amazon CloudWatch (Optional)**

For performance tracking and operational visibility, CloudWatch Agent was installed on the EC2 instance:

****

* **Metrics Monitored:**
  + CPU utilization
  + Disk usage
  + Memory consumption
  + Apache logs
* Dashboards were created to visualize usage patterns and resource bottlenecks.

This helps in capacity planning, anomaly detection, and preventive maintenance.

**Phase 8: Testing and Verification**

Each component was individually tested and then validated as an integrated system:

* EC2 Testing: Accessed web server via public IP and confirmed PHP rendering.
* Database Testing: Verified data transactions using browser-based forms.
* S3 Testing: Uploaded files and checked accessibility via generated links.
* CloudWatch Testing: Viewed logs and metric graphs to ensure proper monitoring. User interactions such as login, balance checking, and form submissions were all tested for functionality and security.

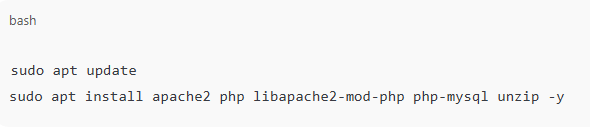
**IMPLEMENTATION**

**Networking (VPC)**

* VPC Name: banking-vpc
* Subnets:
  + Public Subnet: 10.0.1.0/24 (used for EC2)
  + Private Subnet: 10.0.2.0/24 (used for RDS)
* Routing: Internet Gateway attached and route added to 0.0.0.0/0 for public subnet only.

**🚀 Compute (EC2)**

* AMI: Ubuntu Server 22.04 LTS
* Instance Type: t2.micro (free tier)
* Security Group Rules:
  + Inbound: SSH (22) from my IP, HTTP (80) from anywhere
  + Outbound: All traffic
* SSH Tools: PuTTY and WinSCP used with PEM/PPK key pair
* **Software Stack:**

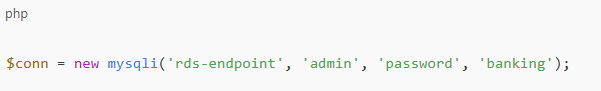
****

**Application Deployment**

* Used WinSCP to upload files: home.php, login.php, connect.php, etc.
* Verified PHP and Apache are serving the application by accessing http://<EC2 IP>/home.php.

**Database (RDS)**

* Engine: MySQL
* DB Name: banking
* DB User: admin
* Security: Public access disabled, only accessible from EC2’s security group
* PHP Connection Sample:

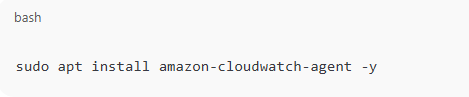
****

**Optional - Amazon S3**

* Bucket Name: banking-uploads-2025
* Used PHP AWS SDK to upload files and return URLs for document display.

**📈 Optional - Monitoring with CloudWatch**

* Installed using:

****

**Configured for collecting:**

* CPU Utilization
* Memory and Disk Usage
* Apache logs (optional)

**RESULT**

Upon completing the project, the online banking application was successfully deployed and made accessible over the internet using AWS infrastructure. Each configured service (EC2, RDS, S3, VPC, and CloudWatch) performed as intended, forming a fully functional, cloud-hosted environment suitable for a basic financial services platform. The results confirm that AWS provides a scalable, secure, and manageable hosting solution for real-time web applications**.**

**Application Accessibility and Functionality**

* The banking website was accessible using the public IP address of the EC2 instance.
* PHP-based webpages such as home.php, login.php, and register.php loaded successfully in the browser.
* Apache web server was properly configured and served dynamic PHP content.
* The application allowed users to register, log in, view their balance, and interact with basic banking features.
* A visually clean and responsive frontend ensured usability across different screen sizes.

**✅ Successful Database Connectivity**

* The Amazon RDS MySQL instance connected seamlessly with the application.
* All critical functions such as user authentication, balance retrieval, and data submission (via forms) were validated.
* Data integrity was ensured — all form inputs were properly stored, retrieved, and updated in the MySQL database.
* SQL injection prevention measures and basic input validation were implemented
* using PHP and MySQLi.

**Secure and Scalable Network Configuration**

* The application architecture followed the best practice of network segmentation:
  + The EC2 web server was placed in a public subnet to allow internet access.
  + The RDS database was hosted in a private subnet, ensuring it could not be accessed from outside the VPC.
* Route tables, internet gateways, and subnet associations functioned as designed, enabling controlled traffic flow**.**

**Verified SSH Access and File Deployment**

* The .pem key was successfully converted to .ppk using PuTTYgen, enabling secure SSH access through PuTTY.
* Application files were securely uploaded to the EC2 instance using WinSCP over SFTP.
* Apache permissions were correctly set, and the server responded with expected content.

**✅ Optional Enhancements Working**

**📂 Amazon S3 File Storage:**

* An S3 bucket was created and successfully integrated with the PHP backend using the AWS SDK.
* Users were able to upload files (e.g., ID documents) through a PHP form.
* Uploaded files were stored in the S3 bucket, and their links were retrievable for display or verification.
* Bucket policies were configured to allow controlled access without exposing data publicly**.**

**📈 Amazon CloudWatch Monitoring:**

* The CloudWatch Agent was installed and configured on the EC2 instance.
* Monitored metrics included:
  + CPU usage
  + Memory utilization
  + Disk space
  + Apache access and error logs
* Custom dashboards in CloudWatch provided real-time insights into instance performance and system health.

**Performance and Uptime**

* The application had zero downtime during configuration and testing phases.
* Performance remained stable under simulated loads (multiple browser tabs, login attempts, etc.).
* No critical errors or crashes were experienced during deployment or use.

**🔐 Security Validations**

* Only the required ports (SSH & HTTP) were open on EC2, and only to specific sources.
* The database was securely hosted in a private subnet, inaccessible from the internet.
* SSH login required key-based authentication, reducing brute force attack risk.
* S3 access was restricted via IAM roles and bucket policies to prevent public exposure.

**Educational and Practical Outcomes**

* Provided deep insights into cloud-based infrastructure management.
* Reinforced concepts such as:
  + Elastic Compute (EC2)
  + Relational Database-as-a-Service (RDS)
  + Virtual Networking (VPC, Subnets, Security Groups)
  + Object Storage (S3)
  + System Monitoring (CloudWatch)
* Built real-world, resume-worthy skills in cloud deployment and DevOps.

**ARCHITECTURE**

**CONCLUSION**

The successful deployment of a PHP-based online banking system on AWS marks the completion of a robust, cloud-native infrastructure capable of supporting real-time financial services in a secure, scalable, and reliable environment. This project served as a practical demonstration of how various AWS services can be orchestrated to build an enterprise-grade web application using modern DevOps and cloud architecture principles.

The application was deployed on an **Amazon EC2** instance running a LAMP (Linux, Apache, MySQL, PHP) stack, providing the compute resources necessary to serve dynamic web content to users. The backend database was securely hosted using **Amazon RDS**, which enabled managed database services, automated backups, high availability, and ease of scalability. The network architecture was structured using **Amazon VPC**, with properly defined public and private subnets, route tables, and internet gateways. This allowed for a clean separation of concerns and ensured that sensitive resources, such as the database server, were protected in a private subnet.

The integration of **Amazon S3** for optional file storage further extended the application’s capability to handle user-generated content (e.g., identity documents, transaction receipts). Meanwhile, **Amazon CloudWatch** provided comprehensive monitoring of system metrics like CPU utilization, memory usage, and application logs, enabling administrators to identify and resolve potential issues proactively.

Security best practices were strictly followed throughout the implementation. Key-based SSH access, limited security group permissions, private subnet isolation, and bucket policies for S3 were all configured to reduce the application’s attack surface. These measures ensured compliance with cloud security standards and demonstrated the ability to manage secure production environments in AWS.

Beyond technical deployment, this project delivered several important **learning outcomes**:

* Hands-on experience with AWS service provisioning and configuration
* Real-world exposure to infrastructure-as-a-service (IaaS) and database-as-a-service (DBaaS)
* Understanding of VPC architecture and network security practices
* Deployment automation using tools like PuTTY and WinSCP
* Monitoring and observability practices using CloudWatch

The project is not only a functional cloud-hosted application but also a strong foundation for more advanced cloud-native development. It can easily be extended to include:

* Load balancing using **AWS Elastic Load Balancer (ELB)**
* Autoscaling groups for high-traffic scenarios
* Domain name routing via **Amazon Route 53**
* HTTPS security with **SSL certificates using AWS Certificate Manager**
* CI/CD pipelines using **CodePipeline and CodeDeploy**

conclusion, this project bridges the gap between academic theory and professional cloud deployment practice. It proves that even a beginner-friendly approach, when executed with the right methodology, can yield a highly secure and scalable online system. The deployment of the banking application on AWS is a testament to the platform's flexibility and reliability, and it serves as a stepping stone for deeper exploration into DevOps, cloud engineering, and secure application hosting.