**Secure AWS Architecture Project**

This project demonstrates how I built and secured a basic web application architecture on AWS using best practices. The setup includes a public EC2 web server, a private RDS database, IAM roles, network isolation, WAF, CloudTrail, GuardDuty, VPC Flow Logs, and more.

⚡ **Real-world Focus**: This project simulates how a small business could securely host their website on AWS. It balances simplicity and security, showing how to build a cost-effective, production-ready environment without overcomplicating the setup.

**🧱 Architecture Overview**

I created a simple 2-tier architecture with full network and application-layer security:

* **Public EC2 Web Server (Amazon Linux 2023)** with NGINX
* **Private RDS MySQL Database**, encrypted with KMS
* **Custom VPC** with public and private subnets
* **IAM roles** instead of access keys
* **WAF** for web-layer protection
* **Monitoring and logging** using GuardDuty, VPC Flow Logs, and CloudTrail

A screenshot of a computer

AI-generated content may be incorrect.

**🔧 Tech Stack**

* Amazon EC2 (Web Server)
* Amazon RDS (MySQL, encrypted)
* Amazon VPC (with IGW, custom subnets, route tables)
* AWS CloudTrail
* VPC Flow Logs
* AWS GuardDuty
* AWS WAF
* IAM Roles
* Docker (to connect to MySQL from EC2)

**🔐 Security Principles Used**

* **Least privilege** (EC2 can only access RDS)
* **No public access** to database
* **KMS encryption** for data at rest
* **IAM roles** instead of hardcoded secrets
* **Logging and threat detection** across layers
* **WAF protection** against SQLi/XSS/known bad inputs

**⚙️ Step-by-Step Setup**

**1. VPC + Subnet Setup**

I created a custom VPC called secure-webapp-vpc with two public subnets and two private subnets. I configured route tables, added an internet gateway, and made sure each subnet was properly associated.

A screenshot of a computer

AI-generated content may be incorrect.

* *Possible Upgrade: Add Network ACLs for layered control*

**2. EC2 Web Tier**

I launched an EC2 instance using Amazon Linux 2023 and installed NGINX using DNF. I allowed HTTP and SSH traffic only from my IP. I also attached an IAM role for potential future use with S3.

A screenshot of a computer

AI-generated content may be incorrect.

* *Possible Upgrade: Add CloudFront for global distribution + HTTPS*

**3. RDS Private Database (MySQL)**

I deployed an RDS instance in a private subnet, encrypted with KMS and with public access disabled. I configured the security group to allow traffic only from the EC2 instance’s SG.

I ran into multiple issues here:

* MySQL client wasn’t available on Amazon Linux 2023
* I used a Docker container with the official MySQL image to connect instead
* Initially, the RDS security group didn’t allow traffic from EC2 — I fixed this by explicitly adding the EC2’s SG

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

* *Possible Upgrade: Store credentials in Secrets Manager and use IAM auth*

**4. IAM Roles (for EC2)**

I created an IAM role with basic S3 access and attached it to the EC2 instance. This let me avoid using access keys or hardcoded credentials.

* *Possible Upgrade: Use least-privilege custom policies instead of managed ones*

**5. Monitoring & Logging**

I enabled CloudTrail in all regions and sent logs to a dedicated S3 bucket. I enabled VPC Flow Logs to a CloudWatch log group and turned on GuardDuty for threat detection.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

* *Possible Upgrade: Add SNS alerts or event-driven remediation with Lambda*

**6. AWS WAF**

Even though I didn’t use ALB or CloudFront, I created a WAF Web ACL in CloudFront (global) scope to demonstrate my understanding. I added AWS-managed rule groups:

* Core Rule Set
* SQL Database
* Known Bad Inputs I also enabled logging to CloudWatch.

A screenshot of a computer

AI-generated content may be incorrect.

* *Possible Upgrade: Attach WAF to ALB or API Gateway for production use*

**💡 Lessons Learned**

* EC2 couldn’t connect to RDS at first — I learned how important SG pairing is
* Amazon Linux 2023 didn’t include MySQL CLI, so I had to find an alternative (Docker)
* Password input in MySQL client doesn’t show — caused confusion but now I’ll never forget it
* WAF rule names change slightly but their core function is the same
* Most real issues are fixed by checking SGs, VPC routes, and logs

**📈 Future Improvements**

* Store DB creds in Secrets Manager
* Use ALB + attach WAF for real HTTP protection
* Add CloudFront for CDN and caching
* Enable CloudWatch alarms + SNS alerts for threat detection

**✅ Cleanup**

I deleted all the major resources after finishing the project:

* EC2 terminated
* RDS deleted (skipped final snapshot)
* CloudTrail trail + logs deleted
* WAF, log groups, GuardDuty disabled
* Custom VPC left intact (no cost)

**📸 Screenshots**

1. VPC setup + route tables
2. EC2 NGINX page
3. RDS config + security group
4. Successful Docker MySQL connection
5. CloudTrail + Flow Log console
6. WAF rule config
7. GuardDuty enabled

**🧠 Final Thoughts**

This project took me from theory to real-world cloud security problem solving. I built this after passing my AWS Certified Security – Specialty (SCS-C02), and it helped me turn that knowledge into hands-on skills.

Working through real issues like network access, encryption, logging, and app-layer defense gave me the confidence that I can build and troubleshoot secure AWS environments — not just talk about them.