Web 3-Tier Architecture Project | AWS Services

# 1. Introduction

The **Web 3-Tier Architecture Project** showcases the development of a scalable and secure web application using Amazon Web Services (AWS). The project leverages a three-tier architecture, a widely adopted model in enterprise-level web applications, to ensure enhanced scalability, security, and manageability.

**Project Highlights**

* **Frontend**: Hosted on Amazon S3 and distributed via Amazon CloudFront.
* **Backend**: Built using Amazon EC2 instances with Elastic Load Balancing (ELB).
* **Data Management**: Handled by Amazon RDS, ensuring high availability and automated backups.

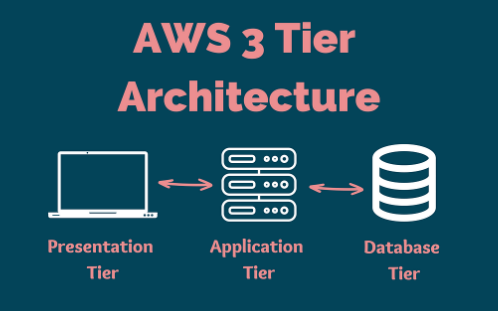
# 2. Project Architecture

**2.1 Overview**

The project is divided into three distinct layers:

* **Presentation Layer (Frontend)**
* **Application Layer (Backend)**
* **Data Layer (Database)**

**2.2 Architecture Diagram-**

****

# 3. Implementation Process

**3.1 Setting Up the Frontend**

**Step 1: Create an S3 Bucket**

* Navigate to the S3 console and create a new bucket with a globally unique name.
* Enable static website hosting by configuring the bucket properties.
* Upload your static files (HTML, CSS, JavaScript) to the S3 bucket.
* Set appropriate permissions to make the content publicly accessible.

**Step 2: Configure CloudFront**

* Go to the CloudFront console and create a new distribution.
* Set your S3 bucket as the origin.
* Enable SSL/TLS for secure data transfer.
* Configure caching settings to optimize content delivery.

**3.2 Configuring the Backend**

**Step 1: Launch EC2 Instances**

* Open the EC2 console and launch a new EC2 instance.
* Select an Amazon Machine Image (AMI) that suits your application (e.g., Amazon Linux 2).
* Choose a t2.micro instance type to stay within the Free Tier.
* Configure security groups to allow inbound traffic on HTTP (port 80) and HTTPS (port 443).
* Install necessary software (e.g., Apache, Nginx, or your preferred application server).

**Step 2: Set Up an Elastic Load Balancer (ELB)**

* Navigate to the ELB console and create a new load balancer.
* Select the EC2 instances that will handle the traffic.
* Configure health checks to monitor the status of your instances.
* Set up listeners for HTTP and HTTPS protocols.

**3.3 Database Management**

**Step 1: Create an RDS Instance**

* Open the RDS console and create a new database instance.
* Choose the database engine (e.g., MySQL, PostgreSQL).
* Configure the instance size (db.t2.micro to stay within Free Tier) and storage.
* Enable Multi-AZ deployment for high availability.
* Set up automated backups and specify a backup retention period.

**Step 2: Connect EC2 to RDS**

* Update your application’s configuration files to connect to the RDS instance using its endpoint.
* Ensure that the EC2 instances can communicate with the RDS instance by configuring the appropriate security groups.

**3.4 Security Best Practices**

**Step 1: Implement IAM Roles and Policies**

* Create IAM roles for your EC2 and RDS instances to manage permissions securely.
* Attach necessary policies that grant the minimum required permissions.

**Step 2: Configure Security Groups**

* Define security groups for each tier (frontend, backend, database) with the principle of least privilege.
* Restrict inbound and outbound traffic based on the application's needs.

**Step 3: Enable SSL/TLS Encryption**

* Use SSL/TLS certificates in CloudFront to secure frontend communication.
* Enable SSL/TLS for your RDS instance to secure data in transit.

**3.5 Performance Optimization**

**Step 1: Optimize Resource Allocation**

* Utilize Free Tier resources effectively by selecting t2.micro instances for EC2 and RDS.
* Monitor resource usage regularly to avoid exceeding Free Tier limits.

**Step 2: Configure CloudFront Caching**

* Optimize caching settings in CloudFront to reduce latency and improve user experience.

**Step 3: Implement Auto Scaling**

* Set up Auto Scaling for EC2 instances to handle varying traffic loads.
* Define scaling policies based on CPU utilization or other metrics.

# 4. Conclusion

This project demonstrates a robust implementation of a web application using AWS's powerful services, following industry best practices for security, scalability, and cost-efficiency. By leveraging AWS’s Free Tier, the project illustrates how to build and deploy a production-ready application without incurring significant costs.

# 5. Appendix

**5.1 AWS Services Used**

* Amazon S3
* Amazon CloudFront
* Amazon EC2
* Elastic Load Balancer (ELB)
* Amazon RDS
* AWS IAM

**5.2 Tools and Technologies**

* HTML/CSS/JavaScript for frontend development.
* Apache/Nginx for the application server.
* MySQL/PostgreSQL for database management.
* IAM for secure access management.

