**Deploying Three Tier Architecture Using AWS Services**

**About this project:**

Throughout the workshop, you'll focus on ensuring **availability** (the system is operational and accessible) and **scalability** (the system can handle increased loads) by implementing best practices like load balancing, auto-scaling, and security configurations. By the end of the project, you will have built and deployed a resilient and scalable web architecture in AWS, gaining practical knowledge in networking, security, and cloud infrastructure management.

**AWS Services used:**

1. **EC2 (Elastic Compute Cloud)**

* **Use:** Provides scalable virtual servers to run applications in the cloud. You can launch instances with different operating systems, storage, and networking capacities, which can be easily scaled up or down.

1. **ASG (Auto Scaling Group)**

* **Use:** Automatically adjusts the number of EC2 instances to meet demand, maintaining availability and scaling the instances up or down based on traffic or performance requirements.

1. **ALB (Application Load Balancer)**

* **Use:** Distributes incoming traffic to multiple targets (e.g., EC2 instances, containers, Lambda functions) based on application-layer (HTTP/HTTPS) protocols. Ideal for microservices and containerized applications.

1. **IAM (Identity and Access Management)**

* **Use:** Provides secure control over access to AWS services and resources. Enables you to create and manage users, groups, and permissions for secure access to AWS resources.

1. **S3 (Simple Storage Service)**

* **Use:** Object storage service that offers highly durable, scalable, and low-latency storage for data, such as backups, archives, and big data analytics. Data can be accessed from anywhere and is stored in buckets.

1. **EFS (Elastic File System)**

* **Use:** Provides scalable file storage that can be mounted across multiple EC2 instances. It is fully managed and grows automatically as files are added.

1. **RDS (Relational Database Service)**

* **Use:** Manages relational databases (like MySQL, PostgreSQL, MariaDB, Oracle, and SQL Server) in the cloud. It automates backups, patching, and scaling while maintaining high availability.

1. **VPC (Virtual Private Cloud)**

* **Use:** Provides a logically isolated section of the AWS cloud where you can launch AWS resources in a virtual network that you define, offering full control over networking and security.

1. **CloudWatch**

* **Use:** Monitors AWS resources and applications, providing real-time data, logs, and alarms to track metrics, analyze logs, and detect anomalies in cloud infrastructure.

1. **SNS (Simple Notification Service)**

* **Use:** Fully managed messaging service for sending notifications to different platforms (email, SMS, mobile push notifications) or triggering automated workflows through AWS Lambda or SQS.

1. **CloudTrail**

* **Use:** Provides logging, monitoring, and retention of AWS account activity across the AWS infrastructure. Helps with governance, compliance, and operational or risk auditing of AWS accounts.

1. **Route 53**

* **Use:** Scalable domain name system (DNS) web service that provides reliable and cost-effective domain name registration, DNS routing, and health checking of resources.

1. **CloudFront**

* **Use:** Content delivery network (CDN) service that speeds up the distribution of static and dynamic web content, like HTML, CSS, JavaScript, and media files, by caching them in global edge locations.

1. **WAF (Web Application Firewall)**

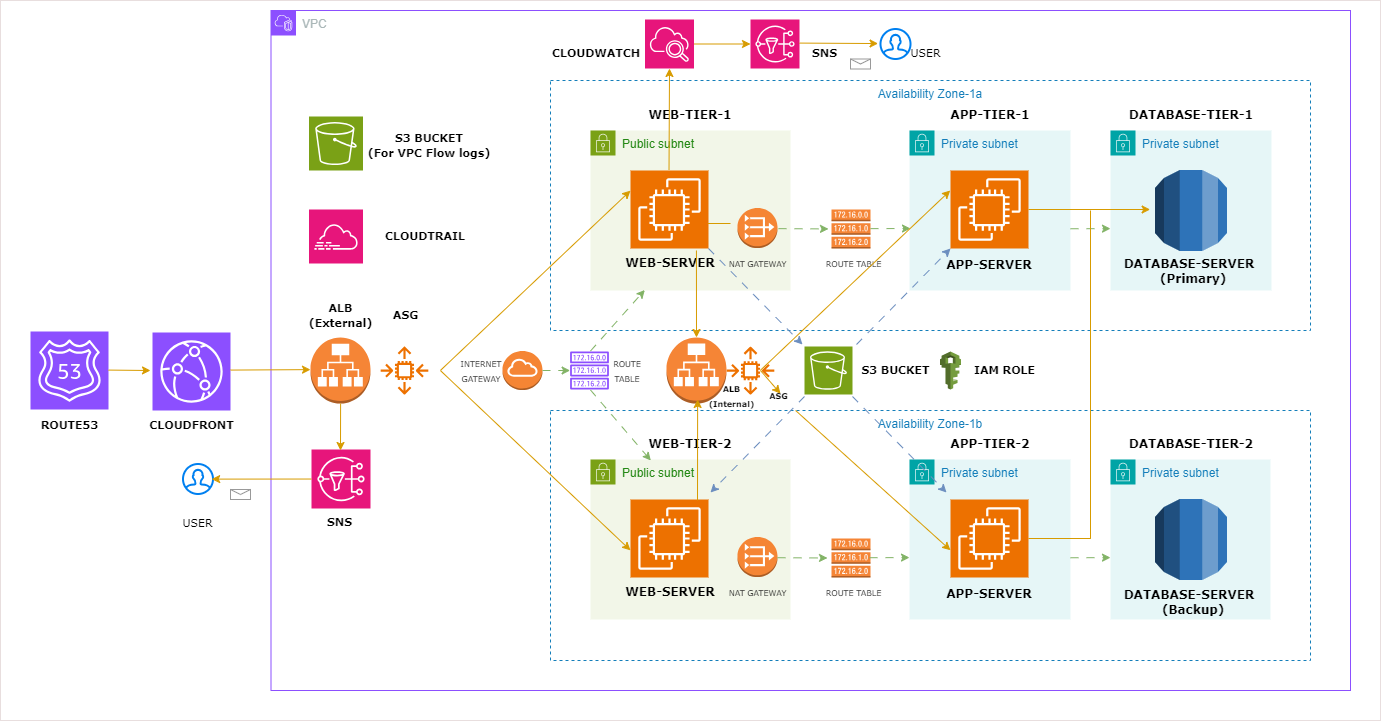
* **Use:** Protects web applications from common web threats, such as SQL injection and cross-site scripting (XSS). It monitors and controls incoming and outgoing HTTP(S) requests.

1. **Shield**

* **Use:** Managed Distributed Denial of Service (DDoS) protection service. It provides safeguards against DDoS attacks for applications running on AWS, with different levels of protection.

**Architecture:**

In this architecture, a public-facing Application Load Balancer forwards client traffic to our web tier EC2 instances. The web tier is running Nginx webservers that are configured to serve a React.js website and redirects our API calls to the application tier’s internal facing load balancer. The internal facing load balancer then forwards that traffic to the application tier, which is written in Node.js. The application tier manipulates data in an Aurora MySQL multi-AZ database and returns it to our web tier. Load balancing, health checks and autoscaling groups are created at each layer to maintain the availability of this architecture.

****



* **Presentation tier**: Component that the user directly interacts with (for example, webpages and mobile app UIs).
* **Logic tier**: Code required to translate user actions to application functionality (for example, CRUD database operations and data processing).
* **Data tier**: Storage media (for example, databases, object stores, caches and file systems) that hold the data relevant to the application.
* **Creating 3 Tier Architecture & Integrating Other AWS Resources**

**Step 1: Download Code from GitHub in Your Local System**

**Step 2: Create Two S3 Buckets**

* Create one S3 bucket for storing web-server & app-server code.
* Upload the code to your S3 from your local system.
* Create another S3 bucket for VPC flow logs.

**Step 3: Create IAM Role with Policies**

* S3 read only.
* SSM managed instance core.

**Step 4: Create VPC, Subnets, IGW, NAT-GW, RT**

* Enable auto-assign public IP for web-tier public subnets.
* Create flow logs for VPC & use the S3 bucket created above.

**Step 5: Create Security Groups**

1. **External-Load-Balancer-SG** --> HTTP (80): 0.0.0.0/0.
2. **Web-Tier-SG** --> HTTP --> Ext-LB-SG.
3. **Internal-Load-Balancer-SG** --> HTTP --> Web-Tier-SG.
4. **App-Tier-SG** --> Port 4000 --> Internal-LB-SG.
5. **DB-Tier-SG** --> MySQL (3306) --> App-Tier-SG.

**Step 6: Create DB Subnet Group & RDS**

* Create DB subnet group.
* Create RDS - Multi-AZ.
* Place them in DB subnet group created above.

**Step 7: Create Test App Server, Install Packages, Test Connections**

* [Test App-Server Commands](https://github.com/pandacloud1/AWS_Project1/blob/main/app-server-commands)
* Create AMI.
* Create launch template using AMI.
* Create target group.
* Create internal load balancer.
* Create autoscaling group.
* Edit nginx.conf file in local system by adding Internal-LB-DNS & upload the file in S3.

**Step 8: Create Test Web Server, Install Packages (Nginx, Node.js (React)), Test Connections**

* [Test Web-Server Commands](https://github.com/pandacloud1/AWS_Project1/blob/main/web-server-commands)
* Create AMI.
* Create launch template using AMI.
* Create target group.
* Create external load balancer.
* Create autoscaling group.

**Step 9: Add External-ALB-DNS Record in Route 53**

**Step 10: Create CloudWatch Alarms Along with SNS**

**Step 11: Create CloudTrail**

**Step 12: Deleting the Entire Infrastructure**

* Delete CloudFront.
* Delete CloudWatch alarms.
* Delete records from Route 53.
* Delete load balancers, target groups, ASG, launch templates.
* Delete security group.
* Delete NAT gateway (it will take 5 mins).
* Release elastic IP.
* Delete VPC.
* Delete RDS subnet group, RDS.

**==============================================================**

This AWS project involves building a **three-tier web architecture** using Amazon Web Services (AWS). In this hands-on workshop, participants will manually create and configure the essential components needed for a fully functional web application. The architecture consists of three layers:

1. **Network Layer**: Setting up Virtual Private Cloud (VPC), subnets, and security groups to ensure secure communication between different parts of the application.

2. **Application Layer:** Deploying the application servers, likely using EC2 instances or AWS services like Elastic Beanstalk, to handle the core business logic.

3. **Database Layer**: Configuring a scalable and highly available database using services like Amazon RDS or DynamoDB.

----------------------------------------------------------------------------------------------------------------

**DESCRIPTION:**

This 3-tier infrastructure is built within an AWS VPC, featuring separate tiers for the web, application, and database layers. Traffic flows from users through Route 53 to CloudFront, then to an external Application Load Balancer (ALB). The web tier consists of EC2 web servers in public subnets, auto-scaled for availability. The internal ALB directs requests to the app servers in private subnets, located in the application tier. App servers interact with an Aurora MySQL database, utilizing multi-AZ deployment for fault tolerance. Monitoring is done via CloudWatch, while security logs are stored in S3 buckets. NAT Gateways ensure secure internet access.

