**Resilient and Scalable Web Application Deployment on AWS**

**A Project submitted**

**in partial fulfilment for the Degree of**

**Bachelor of Technology**

**in**

**Computer & Communication Engineering**

**by**

**PAVNEET THIND**

A logo with a logo and a wreath

Description automatically generated with medium confidence

**Department of Computer and Communication Engineering**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**MANIPAL UNIVERSITY JAIPUR**

**Table of Contents**

1.**Introduction**1.1 Purpose of the Guide  
1.2 Scope and Audience  
1.3 Benefits of AWS-based Deployment

2. **Prerequisites**2.1 Technical Skills Required  
2.2 Tools and Accounts Needed  
2.3 Networking and Security Basics

3.**Architecture Overview**3.1 High-Level Design  
3.2 Components and Their Interactions  
3.3 Availability and Scalability Considerations

4.**AWS Services Configuration**4.1 Virtual Private Cloud (VPC) Setup  
4.2 Subnets and Route Tables  
4.3 Internet Gateway and NAT Gateway  
4.4 Application Load Balancer (ALB)  
4.5 Route 53 (DNS Configuration)  
4.6 RDS Database Setup  
4.7 IAM Roles and Policies

5. **Application Deployment**  
5.1 Code Deployment Strategies

6.**Security Configuration**6.1 Security Groups and NACLs  
6.2 IAM Best Practices  
6.3 Bastion Host Setup  
6.4 Encryption and Key Management

7. References

**1. Introduction**

**1.1 Purpose**

The purpose of this guide is to act as a practical roadmap for deploying a web application on AWS that is highly available, resilient, and scalable. Rather than staying only at a theoretical level, it connects cloud architecture concepts with real-world implementation steps.

By following this guide, readers will learn how to:

* Design AWS environments that can handle failures and sudden spikes in traffic.
* Put in place strategies to minimize downtime and keep applications running smoothly.
* Operate cloud systems that are secure, cost-efficient, and easy to manage.
* Take advantage of AWS-native services to scale automatically, balance traffic effectively, and protect user access.

This guide is especially useful for IT professionals, cloud engineers, and students who want a clear, step-by-step approach to building real-world, resilient cloud systems.

**1.2 Scope**

This document focuses on the end-to-end deployment and management of a web application on AWS. It takes readers from initial design through testing and optimization, covering every major aspect of cloud infrastructure.

* Cloud Architecture Design  
  You’ll learn how to design a Virtual Private Cloud (VPC) with both public and private subnets for security and isolation. It also explains how to set up networking essentials such as Internet Gateways, NAT Gateways, and Route Tables so that traffic flows securely and efficiently.
* Core AWS Services  
  The guide walks through deploying core services like EC2 for hosting, EFS for shared file storage, Auto Scaling Groups to automatically manage resources, an Application Load Balancer to distribute traffic across instances, and Route 53 for domain management and reliable user access.
* High Availability & Resilience  
  Steps are provided for deploying resources across multiple Availability Zones so the application can survive failures and remain online even under stress.
* Scalability & Performance  
  Auto Scaling policies are explained in detail so your infrastructure can grow or shrink with demand. Load testing and optimization practices are also included to ensure performance under both normal and peak traffic.
* Security & Compliance  
  Security is built into the design with Security Groups, IAM roles, and access controls. The guide also shows how to secure communication with HTTPS using SSL/TLS certificates and how to follow best practices for handling sensitive data.
* Testing, Monitoring, and Optimization  
  The final section covers how to test the application setup, monitor it with AWS tools like CloudWatch and CloudTrail, and optimize for cost efficiency by identifying and adjusting underutilized resources.

**2. Prerequisites**

Before beginning the implementation of a highly available and resilient AWS architecture, ensure the following prerequisites are met:

**2.1 AWS Account**

* Active AWS Account:
  + Ensure you have a valid and active AWS account with billing enabled.
  + Root user should only be used for initial setup (like creating the first IAM admin). Daily operations must be performed using IAM users/roles.
* Account Setup Best Practices:
  + Configure AWS CLI and AWS SDKs with Access Key and Secret Key.
  + Set up Billing Alerts & Budgets to monitor spending.
  + Enable Multi-Factor Authentication (MFA) for added security.
  + (Recommended) Use AWS Organizations for multi-account management (separating dev, test, and production).

**2.2 IAM Roles and Policies**

Proper Identity and Access Management (IAM) is critical for secure deployment:

1. IAM Best Practices:

* Follow the principle of least privilege (grant only necessary permissions).
* Use IAM roles instead of long-term access keys for applications.
* Enable password policies for users (complexity, rotation, MFA).

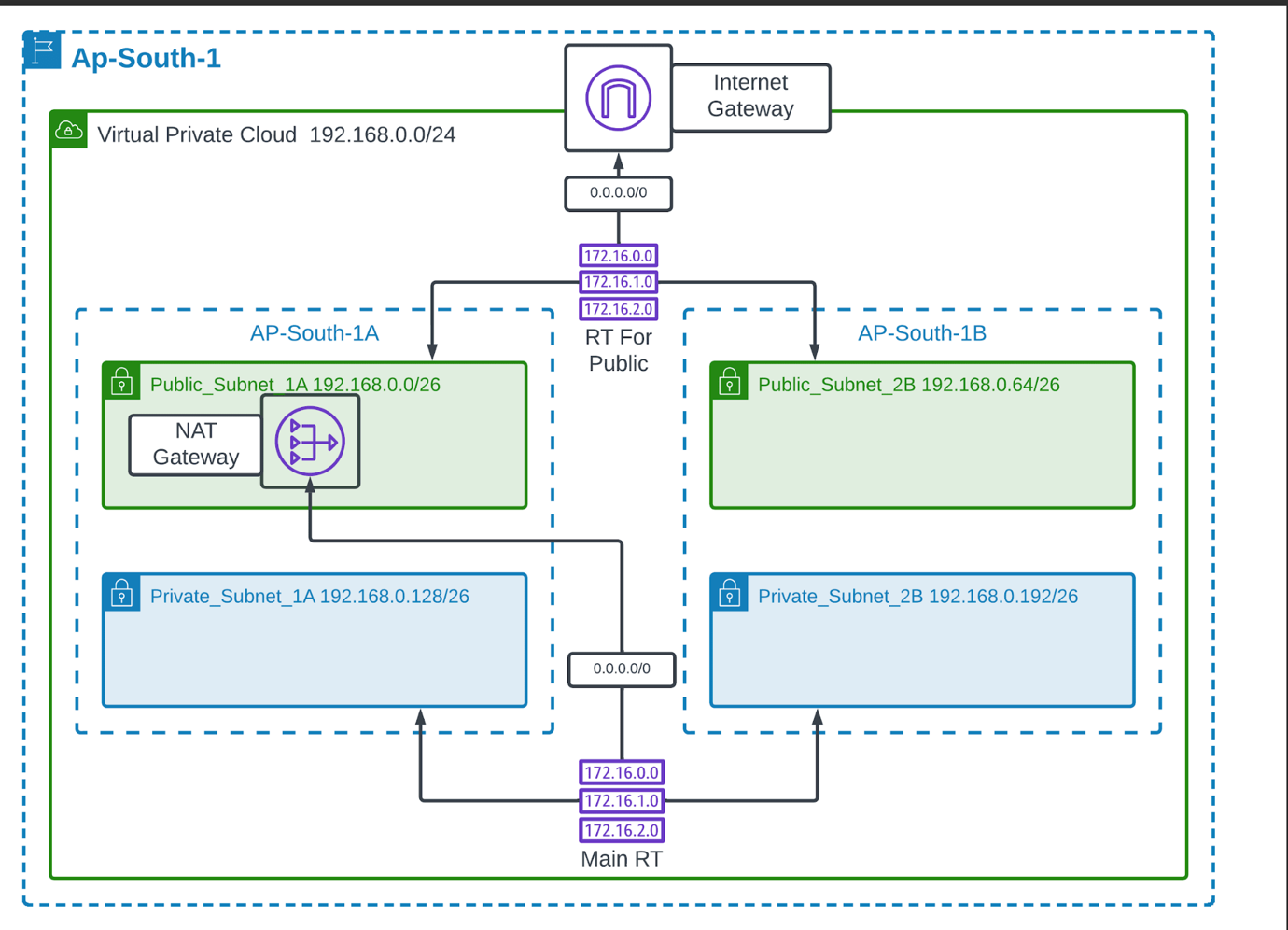
1. Required Roles & Policies (Examples):

* EC2 Role – grants EC2 instances access to S3 (for storing logs or backups), CloudWatch (for monitoring), and Systems Manager.
* ECS/EKS Role – allows containerized workloads to interact with AWS services (e.g., pulling images from ECR).
* RDS Role – provides database instances controlled access to S3 for import/export operations.
* Lambda Execution Role – allows AWS Lambda functions to write to CloudWatch Logs, access S3, or interact with DynamoDB.
* Deployment/CI-CD Role – used by Code Pipeline to manage deployments.
* Administrator Role – for system architects/admins with full access (restricted to a small group).

1. **Architecture Overview**

**3.1 Architecture Diagram**

The high-level architecture consists of the following components (a typical highly available web application setup):



**3.2 High-Level Architecture Description**

This architecture ensures high availability, scalability, and fault tolerance for a web application on AWS.

1. User Access (Entry Point)
   * Users access the application via the internet.
   * Requests are routed through Amazon Route 53, which provides DNS resolution and global traffic management.
2. Edge & CDN Layer
   * Amazon CloudFront (CDN) distributes static assets (images, scripts, CSS, videos) across global edge locations.
   * Improves performance and reduces latency for end users.
3. Load Balancing Layer
   * Elastic Load Balancer (ALB/NLB) distributes incoming traffic evenly across application servers in multiple Availability Zones (AZs).
   * Provides SSL termination and health checks to ensure only healthy instances receive traffic.
4. Application Layer
   * Amazon EC2 instances (or container services like ECS/EKS) run in Auto Scaling Groups across multiple AZs.
   * Auto Scaling automatically adds/removes servers based on demand.
   * EC2s run the web server (e.g., Nginx, Apache) and application backend (Node.js, Python Flask, Java Spring Boot, etc.).
5. Database Layer
   * Amazon RDS (Relational Database Service) deployed in Multi-AZ mode for fault tolerance.
   * Alternatively, Amazon DynamoDB can be used for a serverless, highly scalable NoSQL database.
   * Database instances are placed in private subnets for security.
6. Storage Layer
   * Amazon S3 is used for static asset storage, application backups, and logging.
   * Integrated with CloudFront for content delivery.
7. Networking & Security
   * The architecture runs inside an Amazon VPC with multiple public and private subnets across at least 2–3 Availability Zones.
   * Security Groups and NACLs enforce network security.
   * NAT Gateways allow private subnet instances to access the internet securely.
8. Monitoring & Logging
   * Amazon CloudWatch monitors infrastructure metrics (CPU, memory, requests, latency).
   * AWS CloudTrail tracks API activity for security auditing.
   * Logs are stored in S3 or OpenSearch for analysis.
9. Scalability & Resilience
   * Multi-AZ deployment ensures high availability.
   * Auto Scaling adapts to traffic spikes.
   * Load balancing ensures resilience against server failures.

**4. AWS Services Configuration**

**4.1 Virtual Private Cloud (VPC)**

Amazon Virtual Private Cloud (Amazon VPC) is a foundational networking service provided by AWS that allows you to provision a logically isolated section of the AWS Cloud where you can launch AWS resources within a virtual network that you define. It essentially gives you the power to design and control a customizable virtual data center in the cloud. In a traditional on-premises setup, organizations would manage physical switches, routers, firewalls, and subnets. Amazon VPC replicates these networking functions in a software-defined environment, giving you the flexibility to configure your network topology as per your application’s needs. You get to decide the IP address ranges through Classless Inter-Domain Routing (CIDR) blocks, create public and private subnets for segregating workloads, and configure routing policies that determine how data flows inside and outside the VPC.

One of the most important aspects of VPC is security and isolation. By default, every VPC is logically isolated from other VPCs in AWS, meaning your resources remain protected. Security is enforced through multiple layers: Security Groups, which act as virtual firewalls at the instance level, and Network Access Control Lists (NACLs), which provide subnet-level traffic filtering. Together, these controls allow fine-grained management of inbound and outbound traffic. Additionally, VPC supports features like flow logs to capture network traffic metadata for monitoring and auditing, which further strengthens security and compliance.

Another strength of Amazon VPC lies in its connectivity options. If you want your resources to communicate with the internet, you can attach an Internet Gateway (IGW), which enables inbound and outbound connectivity for public subnets. For private resources that should not be exposed to the internet, a NAT Gateway or NAT Instance allows them to initiate outbound traffic (for tasks like downloading software updates) without being directly accessible from outside. Organizations that want to extend their on-premises networks to the cloud can establish secure connections using AWS Site-to-Site VPN or AWS Direct Connect, ensuring seamless hybrid cloud environments. For more complex multi-VPC topologies, AWS offers VPC Peering and AWS Transit Gateway, which simplify inter-VPC communication at scale.

In terms of scalability and availability, Amazon VPC is designed to span across multiple Availability Zones (AZs) within a region. This enables you to build highly available architectures by distributing subnets and workloads across different AZs, protecting your applications from localized failures. You can start small with just one VPC and a couple of subnets, and later expand by adding more subnets, adjusting route tables, or attaching new gateways as your requirements grow. Importantly, Amazon VPC integrates seamlessly with other AWS services such as Amazon EC2 (for compute), Amazon RDS (for managed databases), Elastic Load Balancing (ELB), AWS Lambda, and many more, making it the backbone of almost every AWS-based solution.

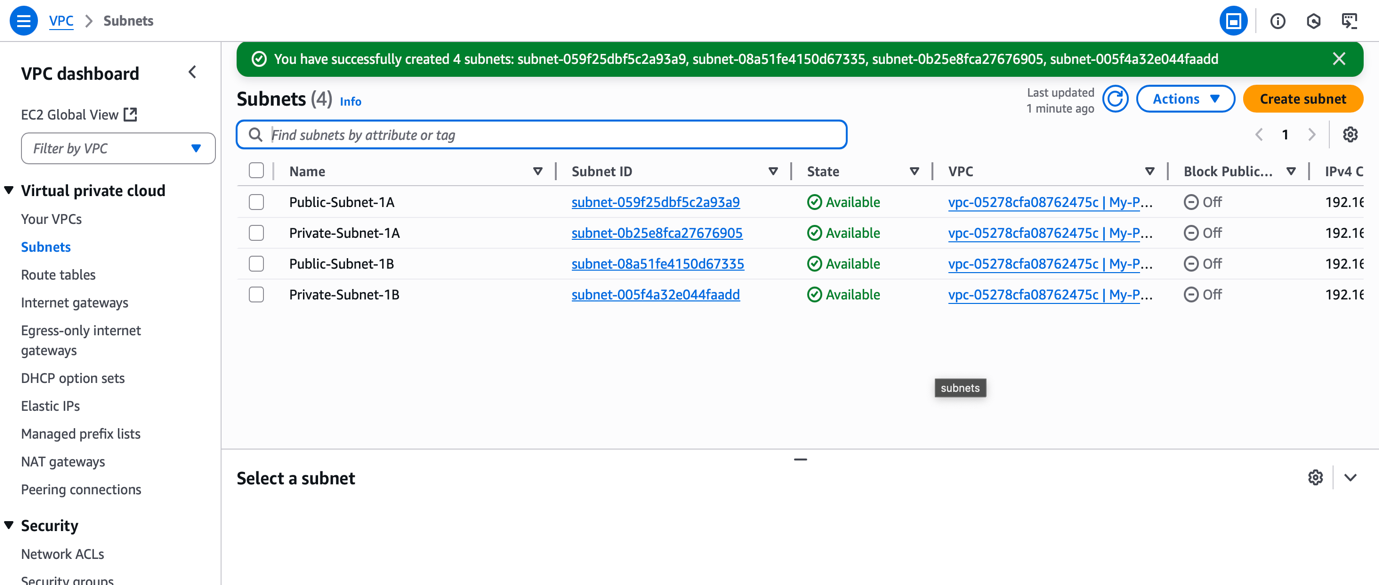
In our project’s architecture, the Amazon VPC serves as the foundation layer upon which all other resources are deployed. Public subnets are used to host the Application Load Balancer and a Bastion Host for secure administrative access. Private subnets are designated for EC2 instances, databases, and application servers, ensuring that sensitive workloads remain isolated from direct internet exposure. An Internet Gateway is attached to the VPC to allow public-facing components to communicate with the outside world, while a NAT Gateway provides controlled outbound access for private instances. Finally, layered security is implemented through Security Groups and Network ACLs, ensuring that only authorized traffic is permitted at both the subnet and instance levels. This architecture leverages the flexibility, security, and scalability of Amazon VPC to deliver a robust, highly available, and secure environment for our application.

Step 1: Create the VPC

1. Navigate to VPC Dashboard → Create VPC.
2. Configure the following:
   * Name tag: Ap-South-1.
   * IPv4 CIDR block: 192.168.0.0/24.
   * IPv6 CIDR: Skip (unless required).
   * Tenancy: Default (No).
3. Click Create VPC.

Step 2: Create Subnets  
Create four subnets across two Availability Zones for HA:

* Public\_Subnet\_1A (AZ: ap-south-1a, CIDR: 192.168.0.0/26).
* Private\_Subnet\_1A (AZ: ap-south-1a, CIDR: 192.168.0.128/26).
* Public\_Subnet\_2B (AZ: ap-south-1b, CIDR: 192.168.0.64/26).
* Private\_Subnet\_2B (AZ: ap-south-1b, CIDR: 192.168.0.192/26).



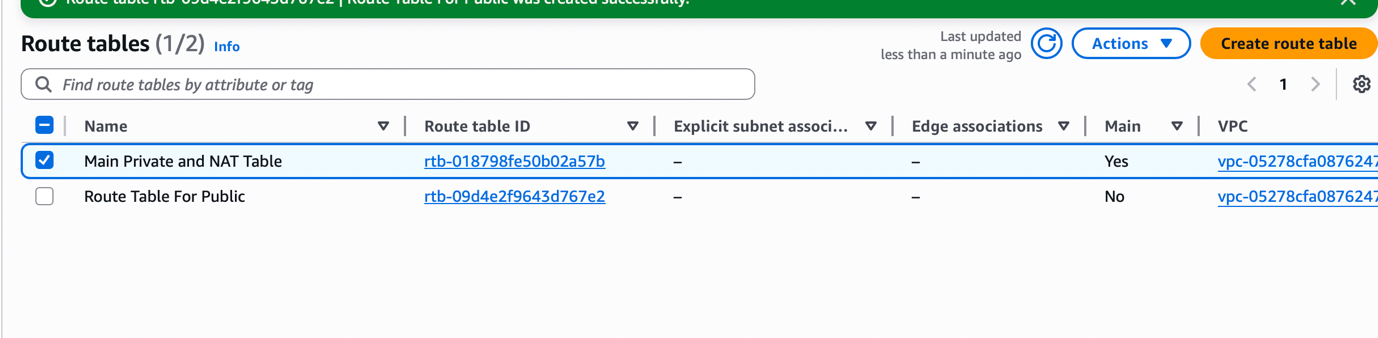
Step 3: Create Internet Gateway (IGW)

1. Go to Internet Gateways → Create Internet Gateway.
2. Name: IGW-Ap-South-1.
3. Attach to Ap-South-1 VPC.A screenshot of a computer

   AI-generated content may be incorrect.

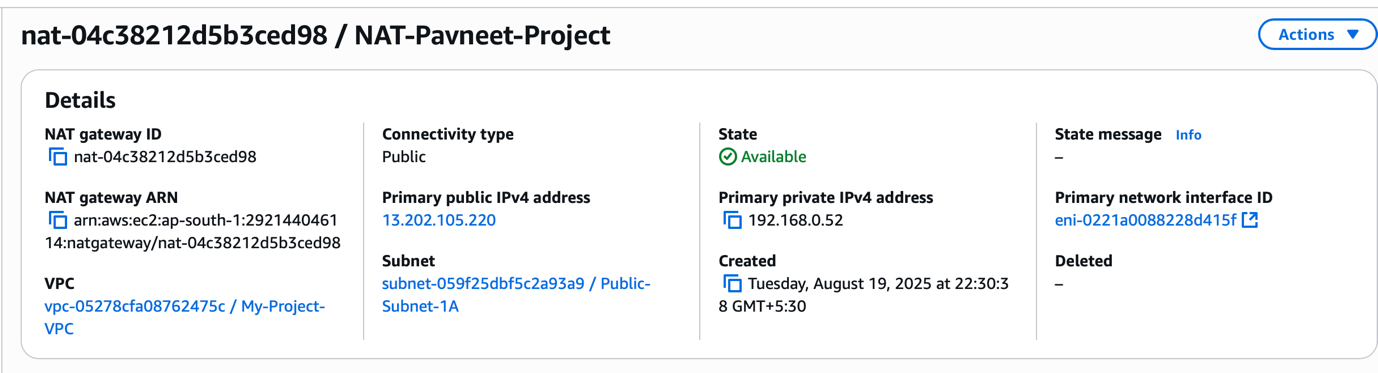
Step 4: Configure Route Tables

1. Create a Public Route Table:
   * Name: Public-RT.
   * Add route: 0.0.0.0/0 → Target: IGW.
   * Associate with public subnets (Public\_Subnet\_1A, Public\_Subnet\_2B).
2. Create a Private Route Table:
   * Name: Private-RT.
   * Add route: 0.0.0.0/0 → Target: NAT Gateway (to be created).
   * Associate with private subnets (Private\_Subnet\_1A, Private\_Subnet\_2B).



Step 5: Create NAT Gateway

1. Go to NAT Gateways → Create NAT Gateway.
2. Subnet: Public\_Subnet\_1A.
3. Elastic IP: Allocate a new one.
4. Update Private Route Table to route 0.0.0.0/0 → NAT Gateway.



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

According to AWS, **Amazon Elastic Compute Cloud (Amazon EC2)** is a web service that provides secure, resizable compute capacity in the cloud, designed to make web-scale cloud computing easier for developers and enterprises. With EC2, you can launch virtual servers—called instances—in minutes, choose from a wide variety of operating systems (such as Amazon Linux, Ubuntu, or Windows), and select instance types optimized for different workloads like general-purpose applications, memory-intensive databases, or compute-heavy analytics. EC2 provides complete control over your instances: you can start, stop, and configure them at will, attach additional storage, assign elastic IPs, and customize networking through Amazon VPC.

One of the major benefits is **flexibility**—instead of investing in physical hardware, EC2 lets you pay only for the compute capacity you use, and scale up or down depending on demand. It also offers **security and reliability** by running instances within Amazon VPC, where you control access using Security Groups and Network ACLs. Additionally, EC2 supports **different purchasing models** such as On-Demand Instances (pay-as-you-go), Reserved Instances (long-term cost savings), and Spot Instances (take advantage of unused capacity at lower prices).

In the context of our architecture, EC2 forms the **compute backbone** of the system. Application servers and backend services run on EC2 instances hosted in private subnets, ensuring security and isolation. Bastion hosts, if required, may be deployed in public subnets to provide secure administrative access. By integrating EC2 with Elastic Load Balancing and Auto Scaling, the application achieves high availability, fault tolerance, and the ability to automatically handle traffic surges.

Instance Setup:

1. Launch EC2 instances in private subnets (for app backend) and optionally public subnets (for bastion/jump server).
   * AMI: Amazon Linux 2 / Ubuntu LTS.
   * Instance type: t3.medium (for testing), scale later.
   * Key Pair: Create or use existing SSH key.
   * Security Group: Allow only required ports (see Section 6).

Auto Scaling:

1. Create a Launch Template with instance configuration (AMI, type, SGs).
2. Go to Auto Scaling Groups → Create.
3. Place across Private\_Subnet\_1A and Private\_Subnet\_2B.
4. Scaling Policies:
   * Scale out at 70% CPU.
   * Scale in at 30% CPU.
   1. **Elastic File System (EFS)**

According to AWS, **Amazon Elastic File System (Amazon EFS)** is a fully managed, scalable, and elastic file storage service that can be mounted simultaneously on multiple Amazon EC2 instances, Amazon ECS tasks, and AWS Lambda functions. It provides a simple, serverless, and elastic NFS (Network File System) solution that automatically grows and shrinks as files are added or removed, without requiring provisioning or capacity management. With EFS, applications can access shared data with high availability and durability across multiple Availability Zones in an AWS Region, making it suitable for use cases such as content management systems, development environments, data analytics, media processing, and backups.

EFS is designed for **scalability and performance**—it can automatically scale to petabytes of data and support thousands of concurrent connections. It also offers **two storage classes** (Standard and Infrequent Access) to optimize cost, with lifecycle management policies that automatically move infrequently used files to lower-cost storage. Security is built in with support for VPC security groups, IAM policies, POSIX-compliant permissions, and encryption at rest and in transit.

In our architecture, Amazon EFS provides a**shared, persistent storage layer** for EC2 instances running in private subnets, ensuring that application data, logs, or configuration files are consistently accessible across multiple instances. This allows for **stateless application design**, where compute resources can scale in and out while maintaining a common data repository.

Steps:

1. Navigate to EFS Dashboard → Create File System.
2. Name: App-EFS.
3. Attach to the same VPC (Ap-South-1).
4. Configure mount targets in ‘Private\_Subnet\_1A’ and ‘Private\_Subnet\_2B’.
5. Attach EFS to EC2 instances using the NFS client.
   1. **Application Load Balancer (ALB)**

The **Application Load Balancer (ALB)** is a Layer 7 load balancer in AWS designed to handle HTTP and HTTPS traffic with advanced routing capabilities. It intelligently distributes incoming requests across multiple targets such as EC2 instances, containers, IP addresses, and even AWS Lambda functions, ensuring high availability and scalability. Unlike the Classic Load Balancer, ALB operates at the application layer, allowing content-based routing based on factors like URL path, hostname, query parameters, or HTTP headers. It integrates seamlessly with microservices and containerized applications through Amazon ECS and EKS, while also supporting modern protocols like HTTP/2 and WebSocket. ALB enhances security by working with AWS WAF and SSL/TLS certificates from AWS Certificate Manager. It is ideal for use cases such as routing API requests to different microservices, hosting multiple applications behind a single load balancer using host or path-based rules, and efficiently managing dynamic traffic for web applications.

Setup:

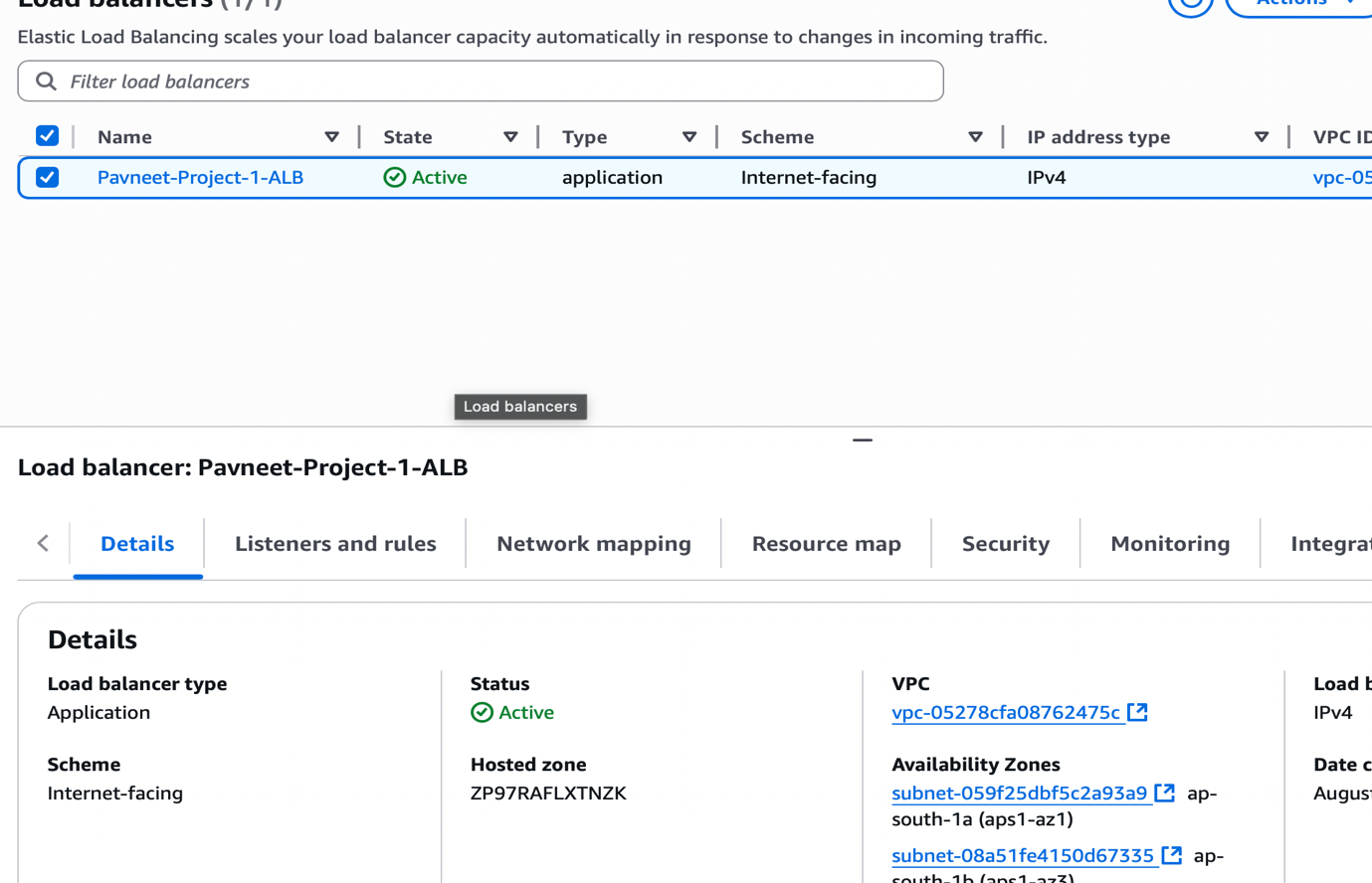
1. Navigate to EC2 → Load Balancers → Create Load Balancer → Application Load Balancer.
2. Name: App-ALB.
3. Scheme: Internet-facing.
4. Network: Select VPC (Ap-South-1).
5. Subnets: Select Public\_Subnet\_1A and \*\*Public\_Subnet\_2B`.
6. Security Group: Allow HTTP (80)/HTTPS (443).
7. Target Group:
   * Create target group with EC2 instances (private subnets).
   * Use health checks (e.g., /health).
8. Register EC2 instances / ASG to target group.
   1. **Route 53 (DNS Configuration)**

Amazon Route 53 is a highly available, scalable, and authoritative Domain Name System (DNS) web service offered by AWS. It is designed to provide developers and organizations with a reliable and cost-effective way to route end-user requests to Internet applications by translating friendly domain names (like *www.example.com*) into the numeric IP addresses (like 192.0.2.1) that computers use to communicate with each other. Route 53 seamlessly connects user requests to resources running inside AWS—such as EC2 instances, Elastic Load Balancers, Amazon S3 buckets, or CloudFront distributions—and can also direct traffic to on-premises infrastructure.

In addition to basic DNS functionality, Route 53 provides domain registration services, enabling customers to purchase and manage domain names directly from AWS. It also includes advanced features like health checks and monitoring, which automatically route traffic away from unhealthy endpoints to improve application availability. Route 53 supports different routing policies such as simple routing, weighted routing, latency-based routing, failover routing, and geolocation/ geo-proximity routing, allowing organizations to control how traffic is distributed globally. Moreover, it integrates tightly with AWS services, enabling dynamic scaling, fault tolerance, and low-latency user experiences. With its global DNS servers and built-in redundancy, Route 53 ensures high performance and reliability, making it a foundational service for hosting, scaling, and managing web applications on AWS.

Steps:

1. Navigate to Route 53 → Hosted Zones.
2. Create a hosted zone for your domain (e.g., example.com).
3. Add record sets:
   * A record: Point to ALB DNS name.
   * CNAME: For subdomains (e.g., app.example.com → ALB).
4. Update domain registrar (if external) with Route 53 name servers.



**5. Application Deployment**

Application deployment on AWS can be achieved through both manual and automated methods, but for production environments, automation is highly recommended to ensure consistency, scalability, and minimal downtime. A basic approach is manual deployment, where developers connect to Amazon EC2 instances over SSH, transfer code, install dependencies, and restart services. While simple, this method is not scalable and is prone to human errors. A more reliable approach is to use AWS’s managed CI/CD services. The application code can be stored in a version control system such as AWS GitHub, which then triggers AWS CodePipeline to automate the release process. CodePipeline integrates with AWS CodeBuild to compile the source code, run tests, and prepare build artifacts. The artifacts are then deployed using AWS CodeDeploy to EC2 instances within an Auto Scaling Group (ASG), enabling rolling updates and minimizing downtime. This CI/CD pipeline ensures rapid, repeatable, and secure deployments across development, staging, and production environments.

In addition to code deployment, proper configuration management is critical. AWS documentation advises against hardcoding secrets and credentials into application code. Instead, environment-specific parameters such as database connection strings, API keys, and app secrets should be stored securely. AWS Systems Manager Parameter Store allows storage of configuration values, while AWS Secrets Manager provides secure storage and automatic rotation of sensitive credentials like database passwords. Both services integrate with AWS Identity and Access Management (IAM) for controlled access, ensuring that only authorized resources, such as EC2 instances, can retrieve these values at runtime. By leveraging Parameter Store, and Secrets Manager, the deployment process becomes secure, automated, and resilient, aligning with AWS best practices for modern cloud-native applications.

**6.Security Configuration**

Security in AWS is implemented using a layered approach, and one of the foundational components is the use of **Security Groups (SGs)** to control inbound and outbound traffic at the instance and service level. For the **Application Load Balancer (ALB)**, the security group should be configured to allow incoming HTTP (port 80) and HTTPS (port 443) traffic from all IP addresses (0.0.0.0/0), ensuring that end-users can reach the application globally. The **application EC2 instances** that host backend services should never be exposed directly to the internet; instead, their security group is configured to accept inbound traffic only from the ALB security group, typically on port 80. This ensures that all requests pass through the load balancer, providing both centralized access control and the ability to scale securely. For the **database layer (Amazon RDS),** the database security group should allow traffic exclusively from the application EC2 security group on the database port (commonly 3306 for MySQL, 5432 for PostgreSQL, etc.), which ensures that only the application layer can interact with the database, blocking any direct external access. If a **bastion host** (jump server) is used for secure administrative access, its security group should allow SSH (port 22) connections only from a predefined set of trusted administrator IP addresses, thereby reducing the attack surface.

Beyond network-level controls, **IAM (Identity and Access Management)** is crucial for managing permissions. AWS documentation recommends following the **principle of least privilege**, meaning every user, role, or service should only have the minimum permissions required to perform its task. Instead of relying on long-term static credentials, IAM roles should be assigned to EC2 instances, ECS tasks, or Lambda functions, allowing them to securely access AWS services (e.g., S3, DynamoDB, Secrets Manager) without embedding keys in code. Credentials and API keys must be rotated regularly, and where possible, replaced with short-lived credentials provided by IAM roles or AWS STS (Security Token Service). By combining properly scoped IAM policies with tightly controlled security groups, the architecture ensures that the application is both **secure by design** and compliant with AWS security best practices.

**Security Groups:**

* **ALB SG:** Allow 80/443 from 0.0.0.0/0.
* **App EC2 SG:** Allow traffic only from ALB SG on port 80.
* **DB SG (RDS):** Allow traffic only from App EC2 SG on port 3306 (MySQL) or relevant DB port.
* **Bastion Host SG:** Allow 22 (SSH) from admin IP only.

**IAM Policies:**

* Follow **least privilege principle**.
* Use **role-based access** for EC2, ECS, and Lambda.
* Rotate credentials regularly.

**References**

1. Amazon VPC – *What is Amazon VPC?*  
   https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html
2. Amazon EC2 – *Amazon Elastic Compute Cloud Documentation*  
   https://docs.aws.amazon.com/ec2/index.html
3. Amazon Elastic File System (EFS) – *Amazon EFS Documentation*  
   https://docs.aws.amazon.com/efs/index.html
4. Elastic Load Balancing – *Application Load Balancers*  
   https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html
5. AWS CodeDeploy – *Deploy Applications to Amazon EC2 Instances*  
   https://docs.aws.amazon.com/codedeploy/latest/userguide/what-is-codedeploy.html
6. AWS Systems Manager – *Parameter Store*  
   https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-parameter-store.html
7. AWS Secrets Manager – *Store and Retrieve Secrets Securely*  
   https://docs.aws.amazon.com/secretsmanager/latest/userguide/intro.html
8. Amazon VPC Security Groups – *Security Group Rules and Best Practices*  
   <https://docs.aws.amazon.com/vpc/latest/userguide/vpc-security-groups.html>
9. Amazon EC2 Security Groups – *Rules Reference*  
   <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/security-group-rules-reference.html>
10. AWS IAM – *Best Practices for Security*  
    <https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html>