# Overview

This document describes the architecture and design of the infrastructure and application deployment for a web application hosted in a Kubernetes cluster. It includes:

* **Network configuration choices** in Terraform.
* Explanation of **Kubernetes deployment strategies** and resource configurations.
* Security measures at **AWS and Kubernetes levels**.
* How the solution ensures **scalability**, **availability**, and **maintainability**.

# Architecture Diagram

### **1.1 VPC Design**

The network configuration uses Terraform to create an AWS **Virtual Private Cloud (VPC)** with the following components:

* **CIDR block**: 10.0.0.0/16 for a large private address space.
* **Subnets**:
  + **Public Subnets** for internet-facing resources like load balancers (e.g., 10.0.1.0/24, 10.0.2.0/24).
  + **Private Subnets** for backend services and Kubernetes worker nodes (e.g., 10.0.3.0/24, 10.0.4.0/24).
* **NAT Gateway**:
  + A single NAT Gateway is used for private subnets to allow outbound internet access (e.g., for downloading updates or accessing APIs) while maintaining security. However for high availability a NAT Gateway should be deployed on each Availability zone.

### **1.2** Justification of Choices

* **Separation of Public and Private Subnets**:
  + Public subnets are used only for internet-facing services, minimizing exposure of backend services to external threats.
  + Private subnets ensure Kubernetes worker nodes and backend services are not directly accessible from the internet.
* **NAT Gateway**:
  + Provides internet access for private subnets, enabling updates and dependencies for services.
* **Tagging**:
  + Subnets are tagged with kubernetes.io/role/elb for public subnets and kubernetes.io/role/internal-elb for private subnets. These tags ensure proper usage of subnets by Kubernetes when provisioning load balancers.

### **2.1** Deployment Strategies

* **Rolling Updates**:
  + The default strategy is used to ensure zero downtime during application updates.
  + Pods are updated incrementally, maintaining service availability throughout the process. If a deployment has 3 replicas, one replica is updated at a time until all replicas are updated.
* **Replica Management**:
  + Frontend: Configured with 3 replicas for redundancy.
  + Backend: Configured with 2 replicas to ensure fault tolerance.

### **Resource configurations**

* **Requests and Limits**:
  + Defines the minimum and maximum resource allocations for CPU and memory to prevent resource starvation or overcommitment.
* **Probes:**
  + **Liveness Probes**: Ensures the container is running and restarts it if necessary.
  + **Readiness Probes**: Checks if the application is ready to serve traffic before directing requests to it.

### **AWS Level Security**

* **IAM Roles and Policies**
  + **IAM roles are attached to Kubernetes service accounts using IRSA (IAM Roles for Service Accounts).** The EBS CSI Driver uses a specific IAM role with the AmazonEBSCSIDriverPolicy attached, ensuring least privilege.
* **Security Groups**
  + Frontend Security Group: Allows HTTP traffic (port 80) from the internet.
  + Backend Security Group: Restricts access to traffic originating from the frontend.
* **Private Subnets**
  + Worker nodes are deployed in private subnets, making them inaccessible from the internet.

### **Kubernetes Level Security**

* **Namespaces**
  + Applications are isolated into specific namespaces to segregate resources and enforce resource quotas.
* **RBAC (Role-Based Access Control)**
  + Ensures fine-grained permissions for users, applications, and processes. Example: Only specific roles can access or modify certain namespaces or resources.
* **Secrets**:
  + Application secrets (e.g., database credentials) are stored securely in Kubernetes Secrets.