# End-to-End DevOps Pipeline Architecture Design for a Web Application

## Introduction

This document provides a comprehensive architecture design for an end-to-end DevOps pipeline for a Flask-based web application. The design integrates CI/CD, infrastructure as code, monitoring tools, and ingress with Application Load Balancer (ALB) to ensure scalability, resilience, and automation.

## High-Level Architecture

The architecture consists of the following components:  
1. Amazon EKS Cluster:  
 - 3 master nodes (managed by AWS for high availability).  
 - 2 worker nodes for running application pods and additional services.  
2. CI/CD Server:  
 - Jenkins for orchestrating the CI/CD pipeline.  
 - Ansible for configuration management.  
 - Terraform for infrastructure provisioning.  
3. Database Server:  
 - MariaDB NoSQL database for the application backend.  
4. Monitoring Stack:  
 - Prometheus and Grafana for monitoring application and infrastructure performance.  
5. Ingress Controller:  
 - An ALB is used as the ingress controller to handle external HTTP/HTTPS requests to the Kubernetes Cluster.

## Detailed Architecture Components

### 1. Amazon EKS Cluster

The EKS cluster is deployed across multiple Availability Zones (AZs) for high availability. It consists of:  
- \*\*Master Nodes\*\*: Managed by AWS to ensure fault tolerance and scalability.  
- \*\*Worker Nodes\*\*: 2 t3.medium EC2 instances provisioned via Auto Scaling Groups.  
- \*\*Networking\*\*: Utilizes VPC with private and public subnets, NAT gateways, and security groups for secure communication.

### 2. CI/CD Server

A dedicated EC2 instance is used to host the CI/CD server with the following components:  
- \*\*Jenkins\*\*: Orchestrates the CI/CD pipeline, including:  
 - Building and testing the Flask application.  
 - Pushing Docker images to Amazon ECR.  
 - Deploying workloads to the EKS cluster using `kubectl`.  
- \*\*Terraform\*\*: Manages infrastructure provisioning for AWS resources such as VPCs, subnets, and the EKS cluster.  
- \*\*Ansible\*\*: Automates configuration management and application setup.

### 3. Database Server

A separate EC2 instance is provisioned to host the MariaDB database. Key features include:  
- \*\*High Performance\*\*: MariaDB is optimized for handling NoSQL workloads.  
- \*\*Resilience\*\*: Backup strategies are implemented for data recovery.  
- \*\*Security\*\*: The database instance is placed in a private subnet with restricted access.

### 4. Monitoring Stack

Prometheus and Grafana are deployed on the worker nodes in the EKS cluster to monitor the application and infrastructure:  
- \*\*Prometheus\*\*: Collects metrics from the application, database, and Kubernetes cluster.  
- \*\*Grafana\*\*: Visualizes metrics through dashboards for real-time insights.

### 5. Ingress and ALB Configuration

To manage external HTTP/HTTPS traffic, an Application Load Balancer (ALB) is deployed and configured as the ingress controller for the Kubernetes cluster. The configuration includes:  
- \*\*Ingress Controller\*\*: Deployed as a Kubernetes service that interacts with the ALB.  
- \*\*ALB\*\*: Routes external traffic to the appropriate services in the cluster based on ingress rules.

## CI/CD Workflow

1. \*\*Code Commit\*\*:  
 - Developers push code to a Git repository.  
2. \*\*Build and Test\*\*:  
 - Jenkins triggers the CI pipeline to build and test the Flask application.  
3. \*\*Docker Image Creation\*\*:  
 - A Docker image of the application is built and pushed to Amazon ECR.  
4. \*\*Deployment\*\*:  
 - Jenkins triggers the CD pipeline to deploy the application to the EKS cluster using `kubectl`.  
5. \*\*Monitoring\*\*:  
 - Prometheus collects metrics, and Grafana displays insights.

## Auto Scaling and High Availability

1. \*\*EKS Worker Nodes\*\*:  
 - Configured with Auto Scaling Groups to ensure node replacement and dynamic scaling.  
2. \*\*Ingress Controller with ALB\*\*:  
 - Ensures high availability by routing traffic to healthy pods in the cluster.  
3. \*\*Fault Tolerance\*\*:  
 - Nodes are distributed across multiple AZs to ensure high availability.

## Security Considerations

- Use IAM roles and policies to control access to AWS resources.  
- Encrypt sensitive data at rest and in transit.  
- Restrict access to the Jenkins server, database instance, and ALB ingress.

## Conclusion

This architecture design ensures a robust and scalable CI/CD pipeline for the Flask-based web application. The integration of Jenkins, Terraform, Ansible, Kubernetes, and ALB ingress enhances automation and reduces manual intervention. The use of Auto Scaling Groups, Ingress Controller with ALB, and monitoring tools ensures high availability and operational efficiency.