Patient Insight Project: Kubernetes EKS Deployment Report

Overview

The **Patient Insight** project is a cloud-based AI platform designed to improve patient-doctor interaction, automate initial data collection, and manage the treatment lifecycle. Our deployment leverages **Amazon Elastic Kubernetes Service (EKS)** to ensure scalability, reliability, and security. This report provides a comprehensive guide for replicating the deployment process, including environment setup, deployment automation, monitoring, and retraining mechanisms.

Deployment Strategy

Cloud Deployment on AWS

The project is deployed on AWS, utilizing the following services:

- **Kubernetes (EKS)**: To orchestrate containerized workloads, including frontend, backend, ML pipeline, and data pipeline.
- **RDS (PostgreSQL)**: For scalable and secure database management.
- **ECR (Elastic Container Registry)**: To host Docker images for the frontend and backend.

Components and Deployment Details

1. Frontend

- Built with **React**, serving multiple users through a user-friendly interface.
- Dockerized and pushed to **ECR**.
- Connected to a **PostgreSQL database** in RDS.
- Deployed on Kubernetes pods in a private subnet with scaling managed by **Horizontal Pod Autoscaler (HPA)**.

2. Backend

- **ML Pipeline**:
    - Provides predictions, responses, and acts as a virtual nurse.
    - Utilizes GPU-enabled pods for intensive tasks.

- **Data Pipeline**:
    - Handles ETL processes, orchestrated by **Apache Airflow**, running as Kubernetes jobs.
    - Includes mechanisms to process incoming data efficiently and prepare it for ML tasks.
- Backend Docker image is hosted in **ECR** and deployed on EKS.

3. EKS Cluster Configuration

- **Nodes**: Two nodes deployed for illustration purposes across private subnets in different availability zones for fault tolerance.
- **Pods**: Dedicated pods for frontend, backend, ML pipeline, and Airflow, managed by Kubernetes.
- **Load Balancer**: An **AWS Application Load Balancer (ALB)** integrated with an Ingress Controller manages traffic routing and SSL termination.
- **Security Groups and Subnets**: Secure access with fine-grained control and isolation of sensitive components.

Deployment Workflow

Environment Setup

1. **Dependencies**: Install Kubernetes tools (kubectl, eksctl, Helm), AWS CLI, and Docker.
2. **Cluster Initialization**: Use eksctl to create an EKS cluster with appropriate node groups.

Deployment Automation

1. **Dockerization**:
    - Separate Dockerfiles for frontend and backend services.
    - Docker images pushed to **Amazon ECR**.
2. **Kubernetes Configuration**:
    - Use **Helm charts** to define Kubernetes resources for frontend, backend, ML pipeline, and Airflow.
    - Deployments include ConfigMaps, Secrets, Services, and Ingress rules.
3. **CI/CD Integration**:
    - Automate the deployment process using **GitHub Actions**.
    - Trigger deployments automatically when new changes are pushed to the repository.
4. **Replication Steps**:
    - Clone the repository and configure AWS CLI with appropriate permissions.
    - Set up EKS using the provided Helm charts and deployment scripts.
    - Validate the deployment by accessing the endpoints through the ALB.

Monitoring and Retraining

Monitoring

1. **Metrics Collection**:
    - Use **Prometheus** and **Grafana** for cluster and application monitoring.
    - Collect metrics like CPU, memory usage, and API response times.
2. **Logging**:
    - Centralized logging using **Amazon CloudWatch** for troubleshooting and tracking.

Retraining Workflow

1. **Model Decay and Data Drift Detection**:
    - Monitor key performance metrics using Evidently AI or TensorFlow Data Validation (TFDV).
    - Detect changes in input data distribution.
2. **Automated Retraining**:
    - On detecting decay or drift, automatically pull new data, retrain the model, and redeploy it.
    - Trigger the pipeline through CI/CD.
3. **Notifications**:
    - Configure alerts via email or Slack to notify stakeholders when retraining is triggered or a new model is deployed.

Validation and Demonstration

Validation Steps

1. Access the frontend application through the load balancer’s public endpoint.
2. Test backend API responses and model predictions.
3. Verify data processing through the Airflow DAG UI.

Video Demonstration

A video recording showcasing the deployment process on a fresh environment has been created. It includes:

- Environment setup and installation of dependencies.

- Running automated deployment scripts.
- Verifying the deployment through application and API testing.

Submission and Code Details

Code Repository

The repository contains:

- Deployment scripts for Kubernetes resources.
- Dockerfiles for frontend and backend.
- CI/CD configurations for GitHub Actions.

Files Provided

1. **Deployment Scripts**: Helm charts and YAML files for Kubernetes resources.
2. **Automation Code**: GitHub Actions pipeline configuration.
3. **Monitoring and Retraining Code**: Scripts for detecting data drift and automating retraining.

Conclusion

The **Patient Insight** project demonstrates an efficient, scalable, and secure cloud-based deployment strategy using AWS EKS. By leveraging Kubernetes' orchestration capabilities and integrating robust monitoring and retraining workflows, the platform is optimized for real-world healthcare demands. This deployment ensures a streamlined process that is replicable, automated, and ready for production-grade applications.