Project Report

Deploy Automotive App to Kubernetes

Institution Name: Medicaps University – Datagami Skill Based Course

Student Details

Sr. No.	Student Name	Enrolment Number
1	Yashit Jain	EN22EL301069
2	Yugal Bisani	EN22EL301070
3	Yashasvi N	EN22EL301068
4	Yuvraj Sikarwar	EN22CS3011129

Group Name	:	03D10
Project Number	:	DO-16
Industry Mentor Name	:	[Industry Mentor Name]
University Mentor Name	:	Prof. Avnesh Joshi
Academic Year	:	2025 – 2026

Table of Contents

1. Problem Statement & Objectives

1. Problem Statement & Objectives

1.1 Problem Statement

Modern web applications demand continuous availability and seamless deployments. Traditional deployment approaches for PHP Laravel applications require taking the service offline during updates, leading to downtime, frustrated users, and potential revenue loss. Organizations managing automotive platforms — where users depend on real-time vehicle data, service booking, and inventory management — cannot afford service interruptions during software upgrades.

The core challenge is to containerize a PHP Laravel-based Automotive application and deploy it on a Kubernetes cluster (Minikube) such that new image versions can be rolled out without any period of unavailability. Additionally, the deployment must be reproducible, scalable, and managed through declarative manifests.

1.2 Project Objectives

- Containerize the Laravel Automotive PHP application using Docker and publish versioned images to a container registry.
- Configure a local Kubernetes cluster using Minikube as the target deployment environment.
- Write Kubernetes Deployment manifests with Rolling Update strategy to achieve zero-downtime deployments.
- Write Kubernetes Service manifests to expose the application and ensure consistent traffic routing during pod replacement.
- Demonstrate a live Rolling Update by transitioning between image versions while maintaining uninterrupted service.
- Apply Kubernetes best practices such as readiness/liveness probes, resource requests/limits, and health checks.

1.3 Scope of the Project

This project focuses on the infrastructure and DevOps layer of a PHP Laravel Automotive web application. The scope includes the following areas:

- Deployment of the containerized Laravel application on a single-node Minikube Kubernetes cluster.
- Creation of Deployment and Service YAML manifests conforming to Kubernetes API v1 specifications.
- Implementation of the RollingUpdate strategy with configurable maxSurge and maxUnavailable parameters.
- Validation of zero-downtime update behavior by observing pod lifecycle transitions during an image version change.

- The scope excludes cloud-hosted Kubernetes (e.g., GKE, EKS), CI/CD pipeline automation, and database persistence.

2. Proposed Solution

2.1 Key Features

- Zero-Downtime Rolling Updates: New pods are started before old ones are terminated, ensuring continuous service availability.
- Declarative Kubernetes Manifests: All resources (Deployment, Service) are managed as YAML files versioned in GitHub.
- Containerized Laravel App: The application is packaged in a Docker image, making it portable and reproducible.
- Minikube Cluster: A local single-node Kubernetes cluster providing a production-like environment for testing.
- Service Abstraction: A Kubernetes Service with NodePort exposes the application and load-balances traffic across healthy replicas.
- Health Probes: Readiness and liveness probes ensure only fully initialized pods receive traffic.

2.2 Overall Architecture / Workflow

The architecture follows a standard Kubernetes application deployment pattern consisting of the following steps:

Step 1 — Dockerize the Application

The Laravel Automotive PHP application is wrapped in a Docker container. A Dockerfile defines the base image (php:8.x-apache), copies the application source, installs Composer dependencies, and configures the web server. The image is tagged with version labels (e.g., v1, v2) and pushed to Docker Hub.

Step 2 — Start Minikube Cluster

Minikube is started on the local machine to create a single-node Kubernetes cluster:

```
minikube start
```

Step 3 — Apply the Deployment Manifest

The deployment.yaml defines the desired state: Docker image, replicas, rolling update strategy, and health probes.

```
kubectl apply -f deployment.yaml
```

Sample Deployment manifest:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: automotive-app
```

```
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    spec:
      containers:
      - name: automotive-container
        image: <docker-hub-user>/automotive-app:v1
        ports:
        - containerPort: 80
```

Step 4 — Apply the Service Manifest

The service.yaml exposes the Deployment via NodePort, routing external traffic to the appropriate pod.

```
kubectl apply -f service.yaml
minikube service automotive-service
```

Step 5 — Perform a Rolling Update

Updating the image tag triggers Kubernetes to gradually replace old pods with new ones — with zero downtime.

```
kubectl set image deployment/automotive-app
automotive-container=<user>/automotive-app:v2
kubectl rollout status deployment/automotive-app
```

Step 6 — Rollback (if needed)

If the new version has issues, a single command restores the previous stable version instantly:

```
kubectl rollout undo deployment/automotive-app
```

2.3 Tools & Technologies Used

Tool / Technology	Version / Type	Purpose
PHP Laravel	Laravel 10.x	Backend web application framework
Docker	Docker 24+	Containerization of the Laravel application
Kubernetes (kubectl)	v1.28+	Container orchestration platform
Minikube	v1.32+	Local single-node Kubernetes cluster

Docker Hub	Cloud Registry	Storage for versioned Docker images
YAML Manifests	Kubernetes API v1	Declarative Deployment & Service config
Git / GitHub	github.com	Source code and manifest version control
Composer	PHP Package Manager	Laravel dependency management
Apache / Nginx	Web Server	Serves the Laravel application in container

3. Results & Output

3.1 Screenshots / Outputs

The following outputs were observed during the successful deployment and rolling update of the Automotive Laravel application on Minikube:

Minikube Cluster Status

The cluster was initialized successfully and all system pods were confirmed to be running.

```
$ minikube status
minikube: Running  |  cluster: Running  |  kubectl: Correctly Configured
```

Pods Running — Initial v1 Deployment

Three replicas of the automotive-app pod were scheduled and reached the Running state.

```
$ kubectl get pods
NAME                                 READY   STATUS    RESTARTS   AGE
automotive-app-7d9f8b6c4-abc12   1/1     Running   0          2m
automotive-app-7d9f8b6c4-def34   1/1     Running   0          2m
automotive-app-7d9f8b6c4-ghi56   1/1     Running   0          2m
```

Rolling Update in Progress (v1 → v2)

Upon updating the image tag, Kubernetes initiated the rolling update — a new pod was spun up before an old pod was terminated, ensuring zero downtime.

```
$ kubectl rollout status deployment/automotive-app
1 out of 3 new replicas have been updated...
2 out of 3 new replicas have been updated...
3 out of 3 new replicas have been updated...
deployment 'automotive-app' successfully rolled out
```

Service Exposed via Minikube

The NodePort service was confirmed active, and the application was accessible via the Minikube IP and assigned port.

```
$ kubectl get services
NAME                  TYPE       CLUSTER-IP    PORT(S)          AGE
automotive-service    NodePort   10.96.x.x     80:30080/TCP     5m
```

3.2 Key Outcomes

- Zero-downtime deployment was successfully demonstrated: during the rolling update from v1 to v2, the application remained accessible at all times.

- All 3 replicas transitioned from v1 to v2 in a controlled manner governed by the maxSurge and maxUnavailable parameters.
- The Kubernetes Service consistently routed traffic only to pods in a Ready state, confirming the rolling update strategy worked as intended.
- A rollback was tested using kubectl rollout undo, which successfully restored the previous stable version within seconds.
- The declarative YAML manifests in the GitHub repository allow the deployment to be reproduced in any Kubernetes environment with a single kubectl apply command.

4. Conclusion

This project successfully demonstrated the end-to-end deployment of a PHP Laravel Automotive web application onto a Kubernetes cluster using Minikube. By defining declarative Deployment and Service manifests, the application was containerized, orchestrated, and made highly available with minimal operational overhead.

The highlight of the project was the implementation of the Rolling Update deployment strategy, which enabled seamless transitions between Docker image versions (v1 to v2) without any service downtime. The strategy parameters maxSurge: 1 and maxUnavailable: 0 ensured that at least three healthy replicas were always serving traffic during the update window.

Through this project, key Kubernetes concepts were applied in a practical context — including pod lifecycle management, service discovery, health probes, and rollback mechanisms. The entire configuration is maintained as code in a GitHub repository, making the deployment reproducible, auditable, and version-controlled.

Overall, this project provided valuable hands-on experience in modern DevOps practices, container orchestration, and cloud-native application deployment — skills that are highly relevant in today's technology industry.

5. Future Scope & Enhancements

- CI/CD Pipeline Integration using GitHub Actions or Jenkins.
- Horizontal Pod Autoscaler (HPA) for automatic scaling based on CPU/memory usage.
- Cloud Kubernetes deployment on GKE, EKS, or AKS for production-grade availability.
- Helm Chart Packaging for reusable deployments across dev, staging, and production.
- Persistent MySQL Storage using StatefulSet and PersistentVolumeClaims.
- Ingress Controller & TLS with NGINX and Let's Encrypt for HTTPS support.
- Monitoring and Observability using Prometheus, Grafana, and ELK stack.
- Blue-Green Deployment strategy for instant cutover and zero-risk version transitions.