BMS COLLEGE OF ENGINEERING

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

LAB CYCLE – AY2024-2025 [ODD]

CIE Exam Date: 4th Jan 2025 (Saturday), 10 AM - 12 PM

COURSE: DevOps **SECTION: A, B, C & D**

COURSE CODE: 21IS7PEDVR **TOTAL CREDITS: 30 Marks**

Instructions and Syllabus for CIE and Semester End Examination

- **Instructions**
  - You must demonstrate a working solution either on your laptop or on a designated lab system.
  - Share / Copy-Paste **screenshots** as evidence of your work in a document.
  - Feel free to use any **internet resources** (e.g., Google, GPT, etc) for assistance during the exam.
- Choose and answer **only one** of the provided questions.

Question Paper # 148

Q 148.1: Minikube scaling with ReplicaSet
Steps:
- Install Minikube and start a local Kubernetes cluster.
- Write a Python Flask application that serves a simple API endpoint at /wise-cobra-50, returning the message: {"message": "Everything is OK"}.
- Create a Docker image for the Flask application and push it to a local or public container registry.
- Deploy the Flask application to the Kubernetes cluster using a ReplicaSet with the following specifications:
- The ReplicaSet should run 5 replicas of the Flask application.
- Use the label app=flask-app to manage the pods.
- Expose the application on port 5093 within the cluster.
- Test the deployment by accessing the /wise-cobra-50 endpoint through a Kubernetes Service of type NodePort.
- Scale the ReplicaSet to 7 replicas and demonstrate that the new pods are successfully created.
- Delete one of the pods manually and verify that the ReplicaSet automatically recreates it to maintain the desired state.

-- OR --

Q 148.2: Jenkins Freestyle Job with Docker Image Pull Included
Steps:
- Create a new GitHub repository called 'nebula-runner'.
- Generate a fine-grained personal access token on GitHub for secure access to the repository.
- Write a Python Flask application that serves a simple API endpoint at /system/check, returning the message: {"state": "Healthy"}.
- Commit and push the code to the GitHub repository.
- Install Jenkins on your local or cloud environment and ensure it is running.
- Set up a freestyle Jenkins job.
- Configure the job to execute successfully with a build output log.
- Schedule the job to run daily at 4:30 PM.
- Integrate the job with Docker to pull the image `mariadb` as part of the build process.
- Extend the job to run the pulled Docker image and verify its execution.
- Write a script to log the output of the `docker run` command into a file named `build_log.txt` and store it in a shared workspace.
- Parameterize the Jenkins job to accept a custom Docker image name as input and use it to dynamically pull and run the specified Docker image.
- Implement a post-build action to clean up unused Docker images and containers on the Jenkins host after the build completes.