



# LAB WORKBOOK

**23SDCS13A / 23SDCS13R**

**CI/CD AND CLOUD DEVOPS**

TEAM  
CI/CD AND CLOUD DEVOPS  
K L UNIVERSITY | VADDESWARAM



# LAB WORKBOOK

**23SDCS13A / 23SDCS13R**

**CI/CD AND CLOUD DEVOPS**

<b>STUDENT NAME</b>	
<b>STUDENT ID</b>	
<b>YEAR</b>	
<b>SEMESTER</b>	
<b>SECTION</b>	
<b>FACULTY NAME</b>	

## **DEPARTMENT VISION AND MISSION**

### **Vision**

To be a leader in Information Technology education and research, driving innovation in emerging technologies and empowering students to develop ethical, sustainable, and impactful IT solutions for a digitally connected world.

### **Mission**

- To provide high-quality, tool-based hands-on education that integrates theory with industry-driven learning.
- To foster a research ecosystem that promotes innovation and practical applications in IT.
- To equip students with leadership skills, entrepreneurship mindset, and interdisciplinary adaptability.
- To instill ethical and responsible computing practices in emerging domains.
- To prepare graduates to excel in the IT industry with technical expertise and continuous learning.

### **Program Educational Objectives**

1. Demonstrate proficiency in emerging tools, methodologies, and emerging technologies, enabling them to contribute effectively to industry-driven projects and real-world problem-solving.
2. Engage in research and innovation, applying analytical and critical thinking skills to develop novel solutions that address contemporary challenges.
3. Exhibit leadership qualities, an entrepreneurial mindset, and the ability to adapt to interdisciplinary environments, fostering career growth and business ventures in the IT sector.
4. Uphold ethical computing practices, social responsibility, and professional integrity while continuously updating their knowledge to stay relevant in the evolving landscape.

## 23SDCS13A / 23SDCS13R – CI/CD AND CLOUD DEVOPS LAB WORKBOOK

PROGRAM OUTCOMES		
PO	Graduate Attributes	Program Outcome Description
1	Engineering Knowledge	To impart mathematics, science, & engineering knowledge to develop skills to solve complex engineering problems.
2	Problem Analysis	Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3	Design/ development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4	Conduct investigations of complex problems	An ability to use research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.
5	Modern tool usage	Ability to create, select and apply appropriate techniques, resources and modern engineering activities, while understanding its limitations.
6	The engineer and society	Ability to apply reasoning and the contextual knowledge to assess social & health safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practices.
7	Environment and sustainability	Ability to demonstrate the engineering knowledge to find solutions to contemporary issues by understanding their impact on societal and environmental contexts, towards sustainable development
8	Ethics	An ability to apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.
9	Individual and teamwork	To inculcate abilities to be able to act as a leader as well as team player effectively in multi-disciplinary settings
10	Communication	To develop oral and written communication skills to articulate the complex engineering activities with the engineering community and and society effectively through reports and design documentation, make effective presentations, and give and receive clear instructions.
11	Project management and finance	To develop working knowledge and understanding of the engineering and management principles to manage projects in multi-disciplinary environments.
12	Lifelong learning	To inculcate the habit of constant knowledge upgrading habit to meet the ever-changing technology and industry needs.
PROGRAM SPECIFIC OUTCOMES		
PSO1	An ability to design and develop software projects as well as to analyze and test user requirements.	
PSO2	Working knowledge on emerging technologies as per the industry requirements	

## LAB EXPERIMENTS

<b>Exp. No.</b>	<b>Experiment Name</b>	<b>Page No.</b>
LAB – 1	➔ Git Branching, Merging & Conflict Resolution	1
LAB – 2	➔ Fullstack CI: Jenkins Pipeline Script for Code Integration	9
LAB – 3	➔ Integrate Jenkins CI/CD for a fullstack project in AWS Linux environment	17
LAB – 4	➔ Clone, Build & Run Frontend App in Docker	24
LAB – 5	➔ Deploy fullstack app using Docker Compose	31
LAB – 6	➔ CI/CD for Dockerized Apps Using GitHub Actions	38
LAB – 7	➔ Deploy Frontend & Backend on Kubernetes with Services	46
LAB – 8	➔ Helm-Based Fullstack Deployment with Ingress & Autoscaling	53
LAB – 9	➔ CI/CD Deployment to Cloud Kubernetes Cluster	61
LAB – 10	➔ Containerize & Deploy Fullstack Apps Using Ansible	68
LAB – 11	➔ Automate AWS Provisioning with Jenkins & Ansible	75
LAB – 12	➔ Deploy Dockerized Fullstack App on AWS with Terraform & CI/CD	82
LAB – 13	➔ Source Control Management with AWS CodeCommit	89
LAB – 14	➔ Automated Build & Test with AWS CodeBuild	96
LAB – 15	➔ End-to-End Fullstack CI/CD with AWS CodePipeline	103

**2025-26 ODD SEMESTER CI/CD AND CLOUD DEVOPS LAB CONTINUOUS EVALUATION**

S. No	Date	Experiment Name	Pre-Lab (10M)	In-Lab (25M)			Post-Lab (10M)	Viva Voce (5M)	Total (50M)	Faculty Signature
				Program/Procedure (5M)	Data and Results (10M)	Analysis & Inference (10M)				
1		Git Branching, Merging & Conflict Resolution								
2		Fullstack CI: Jenkins Pipeline Script for Code Integration								
3		Integrate Jenkins CI/CD for a fullstack project in AWS Linux environment								
4		Clone, Build & Run Frontend App in Docker								
5		Deploy fullstack app using Docker Compose								
6		CI/CD for Dockerized Apps Using GitHub Actions								
7		Deploy Frontend & Backend on Kubernetes with Services								
8		Helm-Based Fullstack Deployment with Ingress & Autoscaling								
9		CI/CD Deployment to Cloud Kubernetes Cluster								
10		Containerize & Deploy Fullstack Apps Using Ansible								

**23SDCS13A / 23SDCS13R – CI/CD AND CLOUD DEVOPS LAB WORKBOOK**

<b>S. No</b>	<b>Date</b>	<b>Experiment Name</b>	<b>Pre- Lab (10M)</b>	<b>In-Lab (25M)</b>			<b>Post- Lab (10M)</b>	<b>Viva Voce (5M)</b>	<b>Total (50M)</b>	<b>Faculty Signature</b>
				<b>Program/ Procedure (5M)</b>	<b>Data and Results (10M)</b>	<b>Analysis &amp; Inference (10M)</b>				
11		Automate AWS Provisioning with Jenkins & Ansible								
12		Deploy Dockerized Fullstack App on AWS with Terraform & CI/CD								
13		Source Control Management with AWS CodeCommit								
14		Automated Build & Test with AWS CodeBuild								
15		End-to-End Fullstack CI/CD with AWS CodePipeline								

Date of the Session: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Time of The Session: \_\_\_\_ to \_\_\_\_

## LAB – 1 Git Branching, Merging & Conflict Resolution

### Aim / Objective:

To understand the process of creating branches in Git, making changes in isolated environments, and merging them back while handling merge conflicts if they arise.

### Description:

In software development, branching in Git allows developers to work on new features or bug fixes in isolation from the main production code. This lab demonstrates how to:

- Create and switch between branches
- Commit changes to specific branches
- Merge branches into the main branch
- Resolve merge conflicts manually when multiple branches modify the same file

### Prerequisites:

- Have Git installed on your system.
- Understand basic Git commands (init, add, commit, status, log).
- Have a Git repository initialized (git init or a cloned repo).
- Understand how to edit and save files in a text/code editor (like VS Code, Sublime, or Vim).
- Know how to use the command line (CLI) for Git operations.

### Pre-Lab:

1. What is version control and why is it important in software development?
2. How does Git differ from other version control systems like SVN?

3. What is the difference between git add and git commit?
4. What does the git status command show you?
5. Why is it important to use a separate branch for feature development?

### **In Lab:**

You're working on a web application. You want to add a new feature without affecting the main (production-ready) code. You will use branching to isolate your work, then merge it back, and finally resolve a conflict when two branches edit the same file.

- Initialize a Git repo and create a base project.
- Create a new branch feature/navbar.
- Add a new file (navbar.html) and commit it in the feature/navbar branch.
- Switch to main branch, make a change in the same file (or another one).
- Try to merge feature/navbar into main.
- If there's a conflict, manually resolve it and complete the merge.

### **Project Structure (Example):**

git-branching-demo/

```
|  
├── index.html  
├── style.css  
└── README.md
```

**After feature work:**

git-branching-demo/

|

|— index.html <-- modified in main

|— navbar.html <-- added in feature/navbar

|— style.css

|— README.md

**Data and Results:**

**Analysis and Inferences:**

**Sample VIVA-VOCE Questions (In-Lab):**

1. What is the purpose of branching in Git?
2. How do you create and switch to a new branch in Git?
3. What is a merge conflict and when does it occur?
4. How do you resolve a merge conflict in Git?
5. What command is used to merge branches in Git?

**Post-Lab Task:**

You are part of a development team working on a web application stored in a Git repository. After completing the basic feature development in the lab, your team lead assigns you the following task:

"Create a new branch to implement a footer section in the website. While you work on the footer in one branch, another teammate is making unrelated changes to the same index.html file in the main branch. Once both changes are ready, you are responsible for merging the branches and resolving any conflicts that arise."

**Data and Results:**

**Analysis and Inferences:**

*(For Evaluator's use only)*

<u>Comment of the Evaluator (if Any)</u>          	<u>Evaluator's Observation</u> Marks Secured: _____ out of 50  Full Name of the Evaluator:   Signature of the Evaluator Date of Evaluation:
--	---

Date of the Session:   /  /  

Time of The Session:        to       

## LAB – 2 Fullstack CI: Jenkins Pipeline Script for Code Integration

### Aim / Objective:

To automate the build and integration process of a fullstack web application using Jenkins pipeline scripting, covering both frontend (React) and backend (Spring Boot) components, and preparing it for deployment.

### Description:

In this lab, you will implement a Jenkins pipeline to automate the integration of a fullstack application comprising a React frontend, Spring Boot backend, and MySQL database. The Jenkins pipeline will:

- Clone code from a Git repository.
- Build the React application.
- Package the Spring Boot application using Maven.
- Integrate both components and prepare them for deployment.
- Optionally deploy into Docker containers or a local environment.

### Prerequisites:

- A **Spring Boot backend**, **React frontend**, and **MySQL database**.
- Source code hosted on **GitHub** or another Git repository.
- **Jenkins installed** on your system or server.
- **Jenkins Plugins**:
  - Git plugin
  - Pipeline plugin
  - NodeJS plugin (for React builds)
- Jenkins server has:
  - **Java (JDK)** and **Maven** for Spring Boot
  - **Node.js** and **npm/yarn** for React
  - **MySQL running locally** or accessible remotely

**Tools Installed on Jenkins Agent:**

- Java 11 or later
- Maven
- Node.js (16+)
- MySQL client (for DB migrations or testing)

**Pre-Lab:**

1. What are the key differences between declarative and scripted Jenkins pipelines?
2. Why do we need separate build steps for frontend and backend in CI pipelines?
3. What is the purpose of npm run build in a React project?
4. How does Maven help in building Spring Boot applications in a Jenkins pipeline?
5. What Jenkins plugins are required to support a fullstack CI process, and what does each one do?

### **In Lab:**

You're part of a development team building a fullstack web application which allows users to manage personal tasks.

You are developing a fullstack web application with:

- A React frontend (frontend/)
- A Spring Boot REST API backend (backend/)
- A MySQL database

You want Jenkin pipeline to:

- Clone the full project
- Build the React app into static files
- Build the Spring Boot app
- Integrate the frontend with the backend
- Configure the full stack application using jenkins pipeline script
- Deploy both apps (locally or into Docker containers)

**Data and Results:**

**Analysis and Inferences:**

**Sample VIVA-VOCE Questions (In-Lab):**

1. How do you handle both frontend and backend builds in a Jenkins pipeline?
2. How is the React build integrated with Spring Boot for deployment?
3. What is the advantage of automating code integration using Jenkins?
4. How can Jenkins manage environment variables like DB credentials?
5. What challenges might arise when integrating React and Spring Boot in CI/CD?

### **Post-Lab Task:**

Your team recently reported intermittent build failures in the CI pipeline for a task management fullstack application. The Jenkins pipeline is set up to build both the React frontend and Spring Boot backend. After cloning the project, the frontend sometimes fails due to missing Node.js environment configuration, and the backend build fails if the database connection is not established. You are assigned to stabilize and improve the Jenkins pipeline to make it more reliable and environment-aware.

### **Data and Results:**

**Analysis and Inferences:**

*(For Evaluator's use only)*

<u>Comment of the Evaluator (if Any)</u>          	<u>Evaluator's Observation</u> Marks Secured: _____ out of 50  Full Name of the Evaluator:   Signature of the Evaluator Date of Evaluation:
--	---

Date of the Session: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Time of The Session: \_\_\_\_ to \_\_\_\_

### **LAB – 3 Integrate Jenkins CI/CD for a fullstack project in AWS Linux environment**

#### **Aim / Objective:**

To automate the CI/CD process for a fullstack web application using Jenkins installed on an AWS EC2 Linux instance, enabling continuous build, test, and deployment from a Git repository.

#### **Description:**

In this lab, you will integrate Jenkins on an AWS EC2 instance to automate the CI/CD process of a fullstack Java-based application. The project includes:

- A React frontend and a Spring Boot backend
- A GitHub repository hosting the source code
- An EC2 Linux instance as the Jenkins host and optionally as the deployment server

#### **Prerequisites:**

- AWS EC2 instance (Amazon Linux 2 or Ubuntu) with public IP
- Java, Git, and Maven/Node.js installed (based on tech stack)
- Jenkins installed and running
- GitHub/GitLab repository for your fullstack project
- Proper security group rules (open port 8080 for Jenkins UI)
- SSH access to deployment servers (if separate)

#### **Pre- Lab:**

1. What are the basic steps to launch and connect to an AWS EC2 instance?
2. What is the purpose of a Git webhook in CI/CD workflows?

3. How do security groups in AWS affect access to Jenkins and deployed services?
4. What role does chmod or chown play in configuring Jenkins on Linux servers?
5. Why is it important to separate build, test, and deploy stages in a pipeline?

**In Lab:**

You have a fullstack Java-based project (Spring Boot backend and React frontend) hosted in GitHub. You want to automate the build, test, and deployment process using Jenkins on an AWS Linux EC2 instance.

- Install Jenkins on AWS EC2 (Linux)
- Configure Jenkins
- Set up Jenkins Pipeline
- Configure Git webhook

**Data and Results:**

**Analysis and Inferences:**

**Sample VIVA-VOCE Questions (In-Lab):**

1. What are the key stages in deploying a Spring Boot app via Jenkins?
2. How can you trigger a Jenkins build automatically when code is pushed?
3. Why do we use `nohup java -jar ...` & in deployment?
4. How do you define a pipeline in Jenkins for deployment?
5. What is the difference between build and deploy in CI/CD?

### **Post-Lab Task**

Your team follows a two-stage deployment process: changes are first deployed to a staging environment for internal testing, and then to a production environment after approval. Both environments are hosted on separate AWS EC2 Linux instances. You are responsible for extending the Jenkins CI/CD pipeline to support multi-environment deployment and include manual approval for production release.

### **Data and Results:**

**Analysis and Inferences:**

*(For Evaluator's use only)*

<u>Comment of the Evaluator (if Any)</u>          	<u>Evaluator's Observation</u> Marks Secured: _____ out of 50  Full Name of the Evaluator:   Signature of the Evaluator Date of Evaluation:
--	---

Date of the Session: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Time of The Session: \_\_\_\_ to \_\_\_\_

## LAB – 4 Clone, Build & Run Frontend App in Docker

### Aim / Objective

To containerize a frontend web application using Docker by cloning the source code from a Git repository, building a Docker image, and running it in a container accessible via a local web server.

### Description

This lab introduces Docker-based deployment of a frontend application, focusing on:

- Cloning a project (e.g., React app) from GitHub
- Writing a Dockerfile to define the container environment
- Building and tagging a Docker image
- Running the image as a container and exposing it on port 3000

### Prerequisites:

- Git – To clone the repository
- Docker – To build and run containers
- Internet access – To pull base Docker images and clone the repository
- Basic knowledge of terminal/command-line usage
- A GitHub repository URL with frontend source code (React, Angular, etc.)

### Pre Lab:

1. What is the difference between a virtual machine and a Docker container?
2. What is the purpose of using COPY and RUN instructions in a Dockerfile?

3. Why is port mapping required when running a Docker container?
4. How does a container differ from an image in Docker?
5. What happens if you don't tag your Docker image explicitly?

**In Lab:**

You are working as a DevOps intern. Your team has developed a frontend web application using React. The source code is hosted on GitHub. You are assigned the task of deploying this application in a Docker container using a custom Dockerfile.

- Clone the frontend source code using Git
- Write or use a custom Dockerfile
- Build a Docker image from the Dockerfile
- Run the application inside a Docker container
- Make the app accessible at <http://localhost:3000>

**Data and Results:**

**Analysis and Inferences:**

**Sample VIVA-VOCE Questions (In-Lab):**

1. What is the role of a Dockerfile in containerization?
2. What does the npm run build command do in a React project?
3. Why do we use the serve package in this Dockerfile?
4. What's the difference between docker build and docker run?
5. What does the -p 3000:3000 option do in the docker run command?

**Post-Lab Task:**

You have just joined a DevOps team working on microfrontend architecture. Your first task is to deploy one of the frontend modules independently using Docker. The code is hosted in a public GitHub repository. You are expected to build a Docker image, run the app inside a container, and document your work for future interns.

**Data and Results:**



Date of the Session: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Time of The Session: \_\_\_\_ to \_\_\_\_

**LAB – 5 Deploy fullstack app using Docker Compose****Aim / Objective:**

To deploy a fullstack web application (React frontend + Spring Boot backend) using Docker Compose by defining services in a single docker-compose.yml file. The goal is to orchestrate both containers, set up networking, environment variables, and volume management, while exposing only the frontend service to the host system.

**Description:**

In this lab, you will simulate a real-world DevOps workflow where a fullstack application (React + Spring Boot) needs to be deployed using **Docker Compose** instead of running containers manually. With Docker Compose:

- You'll define services in a single YAML file.
- Both frontend and backend services will run in a shared custom network.
- The frontend will forward API calls to the backend service by service name (not localhost).
- Only the frontend will be exposed to the host system (e.g., <http://localhost:3000>).

By the end, you will have a fully functional multi-container setup with a single docker-compose up command.

**Prerequisites:**

- Git installed and configured
- Docker & Docker Compose installed and running
- Basic knowledge of YAML files and Docker CLI
- Access to frontend and backend Git repositories
- Backend runs on **port 5000**, frontend on **port 3000**

### Pre Lab:

1. What is Docker Compose, and how is it different from Docker CLI commands?
2. What is the role of the docker-compose.yml file?
3. What is the default networking mode used by Docker Compose?
4. How does Docker Compose handle container restarts if a service crashes?
5. Why is it recommended to use service names instead of IP addresses in Compose setups?

### In Lab:

You are working as a DevOps engineer. Your team's frontend (React) and backend (Spring Boot) apps are already containerized. Instead of running containers manually, you are asked to use **Docker Compose** for easier orchestration.

Your task is to:

- Clone both frontend and backend repositories.
- Ensure both repos have working Dockerfiles.
- Write a docker-compose.yml file to:
  - Define **frontend** service (React, port 3000)
  - Define **backend** service (Spring Boot, port 5000)
  - Configure **networking** so frontend communicates with backend using `http://backend:5000/api`

- Mount volumes for hot-reload (optional for dev)
  - Use environment variables for database configs if required
- Run the setup with:
- `docker-compose up --build`
- Access the frontend at `http://localhost:3000`.

**Data and Results:**

**Analysis and Inferences:**

**Sample VIVA-VOCE Questions (In-Lab):**

1. Why do we use Docker Compose instead of running docker run commands manually?
2. What happens if you don't expose the backend service in Compose? Can frontend still reach it?
3. How does Docker Compose manage container networking automatically?
4. What's the difference between docker-compose up and docker-compose up -d?
5. How can you scale a service in Docker Compose (e.g., multiple backend replicas)?

### Post-Lab Task

Your team is preparing to deploy the fullstack app in a staging environment using Docker Compose. Your task is to:

- Write a **production-ready docker-compose.yml** file that:
  - Builds both frontend and backend from GitHub repos
  - Uses **environment variables** from a .env file
  - Persists backend logs or database data using **volumes**
  - Uses a **custom bridge network** for service communication
  - Exposes **only the frontend** to the host machine

Run the application locally with Docker Compose and confirm:

- Frontend is accessible at <http://localhost:3000>
- API requests are correctly proxied to the backend

### Data and Results:



Date of the Session: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Time of The Session: \_\_\_\_ to \_\_\_\_

**LAB – 6 CI/CD for Dockerized Apps Using GitHub Actions****Aim / Objective:**

To implement a complete CI/CD pipeline using GitHub Actions for a Dockerized fullstack application with a React frontend and Spring Boot backend, ensuring code is tested, built, containerized, pushed to DockerHub, and optionally deployed to a production server automatically on every code push.

**Description:**

In this lab, you will:

- Set up **GitHub Actions workflows** for two separate repositories: frontend and backend.
- Automate code checkout, unit testing (npm test and mvn test), Docker image build, and pushing images to DockerHub or GitHub Container Registry (GHCR).
- Optionally, configure **deployment to a production server** (via SSH or cloud scripts) after the Docker image is pushed.

**Prerequisites:**

- **GitHub repositories** for both frontend and backend
- **Dockerfiles** created for both projects
- **Basic test scripts** included in each project (npm test or similar)
- **DockerHub or GitHub Container Registry (GHCR)** account for pushing images
- **GitHub Actions enabled** in your repositories
- Optional: **Production server access** (e.g., via SSH or cloud provider) if deploying beyond build stage

### Pre Lab:

1. What is the difference between GitHub Actions and Jenkins in CI/CD workflows?
2. Why is it beneficial to run tests before building Docker images?
3. What is the use of secrets.DOCKER\_USERNAME and secrets.DOCKER\_PASSWORD in GitHub Actions?
4. What are the pros and cons of using DockerHub vs GHCR?
5. Why should CI/CD pipelines be triggered on a push to the main branch?

### In Lab

You are working as a DevOps engineer for a company that has developed a fullstack web application. The **frontend** is built with **React (Node.js)** and the **backend** is developed using **Spring Boot (Java)**. Both are containerized using Docker and maintained in separate GitHub repositories.

Your task is to implement **full CI/CD automation using GitHub Actions** that performs:

- Code checkout on push to the main branch
- Builds the frontend and backend using their respective build tools (npm and Maven)

- Runs tests
- Builds Docker images
- Pushes images to DockerHub
- Deploys them to a production server

**Data and Results:**

**Analysis and Inferences:**



**Sample VIVA-VOCE Questions (In-Lab):**

1. How does GitHub Actions support multi-language CI pipelines like Node.js + Java?
2. How is the Spring Boot application built before Dockerizing it?
3. Why is DockerHub login required in the CI/CD pipeline?
4. What happens if npm test or mvn test fails during the workflow?
5. How can you deploy both containers in production after pushing images?

### **Post-Lab Task**

Your company has onboarded a CI/CD-first policy. Every code push to the main branch of either the frontend or backend repo must automatically:

- Run tests
- Build the project
- Package it as a Docker image
- Push the image to DockerHub
- Optionally, deploy it to a production server using SSH

You are assigned to implement and validate this process for both the React frontend and Spring Boot backend, hosted in separate GitHub repositories.

### **Data and Results:**



Date of the Session: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Time of The Session: \_\_\_\_ to \_\_\_\_

**LAB – 7 Deploy Frontend & Backend on Kubernetes with Services****Aim / Objective:**

To deploy a fullstack web application—consisting of a React frontend, Spring Boot backend, and MySQL database—on a Kubernetes cluster using Docker images, and expose the frontend using Kubernetes Services for both internal and external communication.

**Description:**

In this lab, you'll simulate a production-ready Kubernetes deployment of a containerized fullstack application. You will:

- Use pre-built Docker images for the frontend and backend
- Deploy MySQL with persistent storage and secure credentials
- Define **Deployments** and **Services** for each component
- Use **ClusterIP** services for internal backend-DB communication
- Use a **NodePort** service to expose the frontend

**Prerequisites:**

- Docker images for both apps (e.g., on DockerHub)
- A running Kubernetes cluster (e.g., Minikube or KIND)
- kubectl and access to the cluster
- YAML file editor (or terminal)

**Pre-Lab:**

1. What is the purpose of a Deployment object in Kubernetes compared to a Pod?
2. How does a ConfigMap differ from a Secret in Kubernetes?
3. Why is persistent storage important for databases in Kubernetes?
4. What command is used to view logs from a running Pod?
5. What happens if multiple Pods try to access the same PersistentVolumeClaim?

### **In Lab:**

You are working as a junior DevOps engineer in a software company. The development team has completed a full-stack web application consisting of a React frontend and a Spring Boot backend API. The backend also connects to a MySQL database. Your task is to containerize the frontend and backend, and deploy all three components (frontend, backend, MySQL) to a Kubernetes cluster, ensuring proper communication between them using Kubernetes Services.

- Create Docker images for the frontend and backend and push them to Docker Hub.
- Deploy MySQL in Kubernetes with persistent storage and credentials.
- Deploy the Spring Boot backend connected to the MySQL service.
- Deploy the React frontend that calls the backend API.
- Use appropriate Kubernetes Services:
- ClusterIP for internal communication (backend & MySQL)
- NodePort to expose the frontend to the outside

### **Data and Results:**

**Analysis and Inferences:**

**Sample VIVA-VOCE Questions (In-Lab):**

1. What is the difference between a Pod and a Service in Kubernetes?
2. Why do we use a ClusterIP service for the backend?
3. What does a NodePort service do?
4. Why must the React frontend know the backend service name (backend-service)?
5. What happens if you don't set the correct label selectors in the Service YAML?

**Post-Lab Task:**

Your company is transitioning from Docker Compose to Kubernetes for better scalability and orchestration. You are responsible for deploying a **three-tier application** on a local Kubernetes cluster (Minikube or KIND). The application includes a **React frontend**, a **Spring Boot backend API**, and a **MySQL database**. You must ensure that:

- The backend communicates with MySQL internally
- The frontend communicates with the backend via a Kubernetes service
- The frontend is accessible outside the cluster

**Data and Results:**

<u>Comment of the Evaluator (if Any)</u>	<u>Evaluator's Observation</u>
	Marks Secured: _____ out of 50
	Full Name of the Evaluator:
	Signature of the Evaluator Date of Evaluation:

Date of the Session: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Time of The Session: \_\_\_\_ to \_\_\_\_

**LAB – 8 Helm-Based Fullstack Deployment with Ingress & Autoscaling****Aim / Objective:**

To deploy a containerized fullstack web application (React frontend + Spring Boot backend) using Helm, configure Ingress for custom subdomain routing, and apply Horizontal Pod Autoscaling based on CPU usage in a Kubernetes cluster.

**Description:**

This lab introduces the use of Helm to manage complex Kubernetes deployments more efficiently. You'll package the entire fullstack application into a single Helm chart that includes:

- A React frontend served as static files
- A Spring Boot backend exposing APIs
- Kubernetes Services, Deployments, and Autoscaling
- Ingress routing to expose services at subdomains (frontend.local, api.frontend.local)

You'll also configure and verify Horizontal Pod Autoscaling (HPA) for both frontend and backend and ensure proper exposure via NGINX Ingress Controller.

**Prerequisites:**

Before deploying, ensure you have:

- A Kubernetes cluster (Minikube, Kind, EKS, GKE, etc.)
- kubectl configured to access your cluster
- helm (v3+) installed
- Docker images for both frontend and backend (React + Spring Boot) pushed to a container registry (e.g., DockerHub)
- Horizontal Pod Autoscaler enabled in your cluster (use metrics-server)
- Ingress Controller installed (e.g., NGINX Ingress)
- Ingress DNS or localhost setup (e.g., via /etc/hosts or Minikube tunnel)

**Pre-Lab:**

1. What is the difference between a Helm chart and a Kubernetes manifest file?
2. Why do we use values.yaml in Helm charts?
3. How does a Service object enable communication between Pods?
4. What are the minimum requirements for HPA to function correctly?
5. What's the purpose of using minikube tunnel or updating /etc/hosts when using Ingress locally?

## In Lab

You are a DevOps engineer in a cloud-native team tasked with deploying a fullstack web application to Kubernetes using best practices.

The app consists of:

- A **React frontend** (served as static files)
- A **Spring Boot backend** exposing REST APIs

Your objectives are:

1. **Package** the frontend and backend deployments and services into a single **Helm chart**.
2. Configure **Kubernetes Ingress** to expose the frontend at frontend.local and the backend API at api.frontend.local.
3. Implement **Horizontal Pod Autoscaling (HPA)** for both frontend and backend to scale pods based on CPU usage.
4. Deploy the Helm chart to your Kubernetes cluster and verify:
  - Both apps are accessible via their respective subdomains.
  - Autoscaling works as expected when CPU load increases.

## Data and Results:



**Analysis and Inferences:**

**Sample VIVA-VOCE Questions (In-Lab):**

1. What is Helm and why is it useful in Kubernetes?
2. What is the role of an Ingress in Kubernetes?
3. How does the Horizontal Pod Autoscaler work?
4. Why use separate subdomains for frontend and backend in Ingress?
5. What happens if you forget to define targetPort in a service?

**Post-Lab Task:**

You have been assigned the responsibility of making the deployment of a fullstack application more manageable and scalable in Kubernetes. The goal is to use Helm to encapsulate frontend and backend configurations into reusable charts, expose them using custom subdomains via Ingress, and configure autoscaling to respond to changing traffic patterns.

**Data and Results:**



Date of the Session: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Time of The Session: \_\_\_\_ to \_\_\_\_

## LAB – 9 CI/CD Deployment to Cloud Kubernetes Cluster

### Aim / Objective

To automate the deployment of a fullstack web application (React + Spring Boot + MySQL) to a cloud-hosted Kubernetes cluster using a CI/CD pipeline with GitHub Actions, leveraging Docker, Kubernetes manifests, and continuous integration workflows.

### Description

This lab guides you through a **production-level DevOps workflow**, from development to deployment, using:

- Docker for containerizing a fullstack application
- Kubernetes for orchestration and scalability
- GitHub Actions for CI/CD automation
- A cloud Kubernetes provider (GKE, EKS, AKS, or DigitalOcean)

### Prerequisites:

- Git & GitHub account
- Docker installed locally
- Kubernetes cluster (local with Minikube or cloud: GKE/EKS/AKS/DigitalOcean)
- kubectl configured
- Node.js & npm
- Java + Maven (or Gradle)
- MySQL (local or as a container)

**Pre-Lab:**

1. What is the purpose of a Deployment file in Kubernetes compared to a Pod definition?
2. How do secrets and configmaps differ in Kubernetes?
3. Why is it important to tag Docker images during CI builds?
4. How does a container registry (e.g., DockerHub) fit into the CI/CD pipeline?
5. What are the risks of applying Kubernetes manifests directly from CI/CD without review?

**In Lab:**

You are part of a DevOps team responsible for deploying a web-based student management system built with:

- **Frontend:** React
- **Backend:** Spring Boot (REST API)
- **Database:** MySQL

Your objective is to:

- Containerize each component (React, Spring Boot, MySQL)
- Write Kubernetes deployment and service YAML files
- Push Docker images to a registry
- Set up a CI/CD pipeline (using GitHub Actions)

- Deploy the full application automatically to a **cloud-hosted Kubernetes cluster**

**Data and Results:**

**Analysis and Inferences:**

**Sample VIVA-VOCE Questions (In-Lab):**

1. What is the role of CI/CD in DevOps pipelines?
2. How does Kubernetes manage high availability and scaling?
3. Why is Docker used before deploying to Kubernetes?
4. How does GitHub Actions automate deployment workflows?
5. What would you do if a CI/CD deployment breaks production?

### **Post-Lab Task:**

You've been assigned to automate the deployment of a Student Management System consisting of:

- A React frontend for student interaction
- A Spring Boot backend REST API
- A MySQL database

The app should be automatically built, tested, packaged, and deployed to a cloud Kubernetes cluster using GitHub Actions. You are expected to:

- Use separate Dockerfiles for each component
- Push built images to DockerHub
- Deploy to the cloud cluster using kubectl apply from GitHub Actions

### **Data and Results:**



Date of the Session: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Time of The Session: \_\_\_\_ to \_\_\_\_

## LAB – 10 Containerize & Deploy Fullstack Apps Using Ansible

### Aim / Objective

To automate the containerization and deployment of a fullstack React and Spring Boot application using Ansible, by writing and executing playbooks that build Docker images and deploy containers on a target host.

### Description

This lab demonstrates how to use Ansible to automate the deployment process of a modern web application consisting of:

- A React frontend (built with Node.js)
- A Spring Boot backend (built with Java and Maven)

The lab focuses on:

- Automating code cloning
- Building production artifacts (React build & Spring Boot JAR)
- Creating and building Docker images
- Deploying containers using Ansible playbooks
- Ensuring both services run correctly on a local or remote Linux host

### Prerequisites:

- A Linux machine or VM (Ubuntu recommended)
- Ansible installed (sudo apt install ansible)
- Docker installed and configured (sudo apt install docker.io)
- Git
- Node.js & npm for React
- Java 17+ and Maven for Spring Boot
- A sample React and Spring Boot app (or GitHub repository)

**Pre-Lab:**

1. What is the difference between an Ansible ad-hoc command and a playbook?
2. How can you use Ansible to manage dependencies (like Node.js or Maven)?
3. What is the role of handlers in an Ansible playbook?
4. How does Ansible ensure idempotency during configuration?
5. Why is it important to separate tasks into different roles in larger Ansible projects?

**In Lab:**

You are a DevOps engineer tasked with automating the deployment of a React + Spring Boot full stack application using Ansible. The goal is to:

- Clone React and Spring Boot app code.
- Build React frontend (npm build).
- Create Dockerfiles for both apps.
- Build Docker images locally.
- Write Ansible playbook to automate build and deployment.
- Run playbook on target host to deploy containers.
- Verify both frontend and backend are running.

**Data and Results:**

**Analysis and Inferences:**

**Sample VIVA-VOCE Questions (In-Lab):**

1. What is the purpose of Ansible in deployment automation?
2. How does Ansible interact with Docker during this process?
3. Why do we containerize applications before deploying them?
4. What is the advantage of using roles in Ansible?
5. What changes would be required to deploy this setup to a remote server?

**Post-Lab Task:**

You're working in a team that manages deployments of containerized applications across development environments. Your goal is to use Ansible to automate the build and deployment of a React frontend and Spring Boot backend on a Linux host. Both components are Dockerized, and the playbook must handle everything from source code cloning to container startup.

**Data and Results:**



Date of the Session: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Time of The Session: \_\_\_\_ to \_\_\_\_

**LAB – 11 Automate AWS Provisioning with Jenkins & Ansible****Aim / Objective**

To automate the provisioning of AWS infrastructure—specifically EC2 instances—using **Ansible**, and integrate the process with **Jenkins** to trigger provisioning jobs based on **code push or schedule**.

**Description**

This lab introduces Infrastructure as Code (IaC) using Ansible for provisioning AWS EC2 instances and uses Jenkins pipelines to automate the process. You will:

- Define infrastructure using Ansible playbooks
- Use Jenkins to run Ansible automation jobs
- Trigger AWS provisioning from Jenkins using a webhook or cron schedule
- Deploy a simple application (e.g., a static website or script) onto the newly provisioned EC2 instance

By the end of this lab, you'll have an end-to-end automation setup that provisions infrastructure, configures it, and runs an app—all managed by Jenkins CI/CD.

**Prerequisites:**

- AWS Account with permissions to create EC2 instances
- Jenkins installed (locally or on EC2)
- Ansible installed on Jenkins or controller machine
- IAM credentials (Access Key & Secret) configured
- AWS CLI configured (aws configure)
- SSH key pair for EC2 access

### **Pre-Lab Questions**

1. What is the difference between declarative and imperative pipelines in Jenkins?
2. How does Ansible handle remote execution over SSH on EC2?
3. Why do we use dynamic inventories when working with AWS in Ansible?
4. What is the function of the boto3 and botocore Python libraries in AWS automation?
5. What considerations are needed when using scheduled (cron) triggers in Jenkins pipelines?

**In Lab:**

You are working as a DevOps engineer. Your task is to automate the provisioning of AWS infrastructure using Ansible and integrate it with Jenkins to trigger provisioning jobs automatically upon code push or schedule.

- Use Ansible to define and provision EC2 instances on AWS.
- Configure Jenkins to run Ansible playbooks.
- Create a Jenkins pipeline to trigger playbooks on code push or on a schedule.
- Automate the infrastructure deployment process via a Jenkins pipeline.
- Deploy a simple app on the provisioned EC2.

**Data and Results:**

**Analysis and Inferences:**

**Sample VIVA-VOCE Questions (In-Lab):**

1. What is Ansible used for in AWS provisioning?
2. How does Jenkins trigger Ansible playbooks?
3. What is a Jenkins pipeline?
4. How do you secure AWS credentials in Jenkins?
5. What is an Ansible inventory?

### Post-Lab Task:

You are assigned to build a **CI/CD-driven infrastructure provisioning pipeline**. Whenever code is pushed to a GitHub repository (or based on a schedule), **Jenkins** should trigger an **Ansible playbook** that provisions an **EC2 instance** on AWS. After provisioning:

- The instance should have Docker and Git installed
- A simple static web page or app should be deployed
- The process should be fully automated, without manual intervention

### Data and Results:



Date of the Session: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Time of The Session: \_\_\_\_ to \_\_\_\_

**LAB – 12 Deploy Dockerized Fullstack App on AWS with Terraform & CI/CD****Aim / Objective:**

To provision AWS infrastructure using Terraform, deploy a Dockerized fullstack application (React + Spring Boot + MySQL) on an EC2 instance, and implement CI/CD automation using GitHub Actions or Jenkins.

**Description:**

This lab simulates a real-world DevOps deployment pipeline combining **Infrastructure as Code (IaC)** with **CI/CD automation**. You will:

- Define and deploy AWS infrastructure using **Terraform**
- Deploy Docker containers for a React frontend, Spring Boot backend, and MySQL database on a provisioned EC2 instance
- Use **GitHub Actions** (or Jenkins) to automate deployment when code is pushed

By the end, you'll have an end-to-end automated pipeline from code to running infrastructure and application.

**Prerequisites:**

- AWS account with permissions for EC2, VPC, IAM
- Terraform installed (terraform -v)
- Docker installed locally
- GitHub account (for code and CI/CD)
- Jenkins or GitHub Actions set up
- SSH key pair (for EC2 access)

**Pre-Lab:**

1. What is the difference between Terraform state files (terraform.tfstate) and configuration files (.tf)?

2. Why is it important to manage infrastructure as code rather than provisioning manually?
3. How does a Security Group in AWS differ from a traditional firewall?
4. What are the advantages of using `user_data` in an EC2 Terraform configuration?
5. Why is it useful to separate frontend, backend, and DB containers even if they run on one EC2 instance?

**Scenario:**

You're a DevOps engineer responsible for deploying a Dockerized full-stack application using Terraform and applying CI/CD practices. Your stack includes:

- **Frontend:** React
- **Backend:** Spring Boot (Java)
- **Database:** MySQL

Your goal is to:

1. Use **Terraform** to provision AWS infrastructure (EC2, Security Groups, etc.)
2. Deploy **Dockerized containers** for frontend, backend, and MySQL on the EC2 instance
3. Set up a **CI/CD pipeline** using GitHub Actions (or Jenkins) to automate deployment

**Data and Results:**

**Analysis and Inferences:**

**Sample VIVA-VOCE Questions (In-Lab):**

1. What is the role of Terraform in infrastructure provisioning?
2. How does Docker help in deploying full-stack applications?
3. What is the difference between terraform apply and terraform destroy?
4. How does CI/CD improve the software release process in cloud deployments?
5. How would you make this deployment more scalable and fault-tolerant?

**Post-Lab Task:**

You are responsible for deploying a fullstack student management system with:

- A **React frontend**
- A **Spring Boot REST API backend**
- A **MySQL database**

Your company requires the infrastructure to be provisioned on **AWS EC2** using **Terraform** and the deployment automated using **GitHub Actions**. You must:

- Create Terraform scripts to provision the infrastructure (EC2, SG, IAM)
- Use Docker Compose or scripts to start all three services on the instance
- Configure GitHub Actions to automate Docker image build, push, and remote deployment

**Data and Results:**



Date of the Session: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Time of The Session: \_\_\_\_ to \_\_\_\_

**LAB – 13 Source Control Management with AWS CodeCommit****Aim / Objective**

To manage source code collaboratively and securely by using **AWS CodeCommit** as the central Git repository for a fullstack application, and perform essential Git operations like cloning, pushing, branching, and merging.

**Description**

AWS CodeCommit is a **fully managed source control service** that hosts private Git repositories. In this lab, you will:

- Set up a CodeCommit repository
- Push a complete fullstack application (React frontend, Spring Boot backend, MySQL support files) to the repository
- Practice basic Git operations such as cloning, branching, committing, and merging
- Understand how to prepare your project repository for future CI/CD workflows using AWS services like **CodeBuild** and **CodePipeline**

**Prerequisites:**

- AWS account with IAM user having CodeCommit permissions
- Git installed (git --version)
- AWS CLI installed and configured (aws configure)
- React + Spring Boot + MySQL sample project
- SSH key pair (or HTTPS credential helper)

**Pre-Lab:**

1. What is the purpose of configuring SSH or HTTPS credential helper when using Git with CodeCommit?

2. What is the role of aws configure in enabling AWS CLI operations?
3. Why is it important to include .gitignore in a project before pushing it to a remote repository?
4. What are the differences between git merge and git rebase?
5. How can Git branches help in collaborative fullstack application development?

**In Lab:**

You are part of a development team building a full-stack web application using React (frontend), Spring Boot (backend), and MySQL (database). To manage the source code collaboratively and securely, your team has decided to use AWS CodeCommit as the central Git repository. Your task is to create a CodeCommit repository, push the full project to it, and demonstrate basic Git operations like branching, committing, and merging. This setup will serve as the foundation for future CI/CD integration using AWS Developer Tools such as CodePipeline and CodeBuild. Your task is to:

1. Create a CodeCommit repository
2. Push the full-stack app into it using Git
3. Demonstrate basic Git operations (clone, commit, push, branch, merge)

**Data and Results:**

**Analysis and Inferences:**

**Sample VIVA-VOCE Questions (In-Lab):**

1. What is AWS CodeCommit and what are its main features?
2. How do you set up Git credentials for accessing CodeCommit?
3. What are the benefits of using CodeCommit over other version control platforms?
4. How would you organize and manage a full-stack project (React + Spring Boot + MySQL) in a single CodeCommit repository?
5. How can you integrate AWS CodeCommit with other AWS services for CI/CD?

### Post-Lab Task:

You are working on a team project that includes a **React frontend**, a **Spring Boot backend**, and a **MySQL database config**. Your organization uses **AWS CodeCommit** for secure, centralized source control. Your task is to:

- Set up a **new CodeCommit repository**
- Push the entire fullstack codebase into it
- Create a **feature branch** for a new module (e.g., search functionality)
- Perform changes, commit them, and **merge** the feature branch into main
- Document Git operations for collaborative team usage

### Data and Results:



Date of the Session: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Time of The Session: \_\_\_\_ to \_\_\_\_

**LAB – 14 Automated Build & Test with AWS CodeBuild****Aim / Objective:**

To configure AWS CodeBuild for automating the build and test process of a full-stack application (React + Spring Boot + MySQL) by defining a buildspec.yml file, connecting it to a code repository, and ensuring builds are automatically triggered on code changes.

**Description:**

In this lab, you will learn how to use AWS CodeBuild, a fully managed build service, to automate the compilation, testing, and packaging of a multi-language full-stack application. You'll create a buildspec.yml file that defines how CodeBuild should handle frontend and backend builds and test scripts. Then, you'll set up an AWS CodeBuild project, connect it to your repository (GitHub, CodeCommit, or Bitbucket), and verify build success.

**Prerequisites:**

- AWS Account
- Source Code Repository (e.g., AWS CodeCommit, GitHub, Bitbucket)
- AWS CLI installed and configured
- Project structure with:
  - frontend/ (React)
  - backend/ (Spring Boot)
  - database/ (MySQL schema or config)
- buildspec.yml file for AWS CodeBuild configuration

**Pre-Lab:**

1. Why is it important to separate build commands for frontend and backend in a CI system?
2. How does the `buildspec.yml` file help in defining language-specific dependencies?
3. What is the difference between CodeBuild's **phases**: `install`, `pre_build`, `build`, and `post_build`?
4. Why should test scripts be included as part of the CI build step?
5. What permissions are required in the **IAM role** used by CodeBuild to access the repository and build resources?

**In Lab:**

You're developing a full-stack application where the frontend (React), backend (Spring Boot), and database (MySQL) are tightly integrated. You want to ensure every code push triggers an automated build and test process to catch issues early. You'll use AWS CodeBuild to compile and test your code automatically.

- Organize code into frontend, backend, and database folders.
- Create a `buildspec.yml` for build and test commands.
- Set up an AWS CodeBuild project linked to your repo.
- Configure automatic triggers on code push.
- Run and verify builds/tests in CodeBuild.
- Check logs and fix any issues.

**Data and Results:**

**Analysis and Inferences:**

**Sample VIVA-VOCE Questions (In-Lab):**

1. What is AWS CodeBuild and how does it differ from CodeDeploy and CodePipeline?
2. What is the purpose of a buildspec.yml file in CodeBuild?
3. How can you automate builds for a full-stack application with frontend and backend in different languages?
4. What are the common issues faced during build automation and how do you debug them?
5. How can you integrate CodeBuild into a CI/CD pipeline with CodeCommit or GitHub?

### **Post-Lab Task:**

You are working on a full-stack project where the frontend (React), backend (Spring Boot), and MySQL configuration are organized into different directories within the same repository. Your task is to:

- Create a buildspec.yml file that:
  - Installs dependencies for both React and Java
  - Builds both the frontend and backend
  - Runs unit tests for each
- Create an AWS CodeBuild project linked to this repository
- Set it to trigger builds automatically on each push to the main branch
- Fix any issues during build and verify that logs show success

### **Data and Results:**

<u>Comment of the Evaluator (if Any)</u>	<u>Evaluator's Observation</u> Marks Secured: _____ out of 50  Full Name of the Evaluator:   Signature of the Evaluator Date of Evaluation:
--	---

Date of the Session: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Time of The Session: \_\_\_\_ to \_\_\_\_

**LAB – 15 End-to-End Fullstack CI/CD with AWS CodePipeline****Aim / Objective**

To implement a complete CI/CD pipeline using AWS CodePipeline that automatically builds and deploys a fullstack application (React + Spring Boot + MySQL) from source code repository to a running infrastructure (EC2 or Elastic Beanstalk) on every code update.

**Description**

In this lab, you will configure AWS CodePipeline to automate every stage of your application lifecycle—from source control to deployment. This includes:

- Storing code in AWS CodeCommit or GitHub
- Building it using AWS CodeBuild with a buildspec.yml file
- Deploying to a target environment using AWS CodeDeploy (on EC2) or Elastic Beanstalk
- Managing the entire pipeline in one seamless flow

This enables continuous delivery (CD) and significantly reduces the risk and overhead of manual deployments.

**Prerequisites**

1. AWS Account with access to CodeCommit, CodeBuild, CodeDeploy, CodePipeline, S3, EC2, IAM.
2. Full-stack application with:
  - frontend/ (React)
  - backend/ (Spring Boot)
  - database/ (MySQL schema/config)
3. Source code repository (e.g., AWS CodeCommit or GitHub)
4. buildspec.yml file for AWS CodeBuild
5. EC2 instance or Elastic Beanstalk for deployment

**Pre-Lab:**

1. What is the difference between continuous integration and continuous delivery?
2. Why is it important to separate build and deploy phases in a CI/CD pipeline?
3. What is the role of appspec.yml in CodeDeploy, and where is it located in the project?
4. How does CodePipeline detect a change in the source repository?
5. What are the benefits of using a managed build service like CodeBuild instead of building locally?

**In Lab:**

You are working on a cloud-based full-stack project using React, Spring Boot, and MySQL. Your goal is to implement an automated CI/CD pipeline using AWS CodePipeline. The pipeline should fetch code from a repository, build the app using AWS CodeBuild, and deploy it to an EC2 instance or Elastic Beanstalk, ensuring continuous delivery on every code push.

- Create a CodeCommit repository (or use GitHub) and push your project.
- Set up AWS CodeBuild project linked to your repo.
- Launch an EC2 instance with Java, Node.js, and MySQL client installed.

- Install CodeDeploy agent on EC2
- Create appspec.yml with deployment hooks
- Create deployment group and assign EC2 & IAM roles
- Create AWS CodePipeline with stages:
  - Source (CodeCommit/GitHub)
  - Build (CodeBuild)
  - Deploy (CodeDeploy or Elastic Beanstalk)
- Test the CI/CD workflow by pushing code changes and observing automated build and deployment.

**Data and Results:**

**Analysis and Inferences:**

**Sample VIVA-VOCE Questions (In-Lab):**

1. What are the main components of AWS CodePipeline and their roles?
  
  
  
  
  
  
  
  
  
  
2. How does CodePipeline integrate with CodeCommit, CodeBuild, and CodeDeploy?
  
  
  
  
  
  
  
  
  
  
3. What is the difference between deploying to EC2 and Elastic Beanstalk using CodePipeline?
  
  
  
  
  
  
  
  
  
  
4. How does buildspec.yml help in automating the build process in CodePipeline?
  
  
  
  
  
  
  
  
  
  
5. What are the security considerations when creating IAM roles for CI/CD pipelines?

### Post-Lab Task:

Your team is building a student management system with:

- **Frontend:** React
- **Backend:** Spring Boot
- **Database:** MySQL

You've been assigned the task of implementing a **fully automated CI/CD pipeline** using AWS Developer Tools. Your deliverables:

1. Push the fullstack project (frontend, backend, database) to **CodeCommit** or **GitHub**
2. Create a **buildspec.yml** for CodeBuild to build both frontend and backend
3. Install **CodeDeploy agent** on an EC2 instance and prepare it for deployment
4. Write an **appspec.yml** file with deployment hooks (e.g., for stopping old app, running new JAR)
5. Create and configure **AWS CodePipeline** to:
  - Pull from the source repo
  - Build the project
  - Deploy to EC2 or Elastic Beanstalk automatically
6. Make a change (e.g., UI text or backend route) and verify that CodePipeline picks it up and updates the app

### Data and Results:

<p><u>Comment of the Evaluator (if Any)</u></p>          	<p align="center"><u>Evaluator's Observation</u></p> <p>Marks Secured: _____out of   50</p> <p>Full Name of the Evaluator:</p>   <p>Signature of the Evaluator Date of Evaluation:</p>
---	--