**DEVOPS**

**ASSIGNMENT -1**

**1. Preferred Way of Installing Jenkins**

I prefer installing Jenkins using **Docker** because it is fast, easy to manage, and avoids dependency issues. This method provides an isolated environment, making upgrades and rollbacks easy.

✅ **Quick Setup** – No need to manually install Java or other dependencies.  
✅ **Easy Cleanup** – Just remove the container when needed.  
✅ **Portability** – Works the same on different operating systems.  
✅ **No System Pollution** – No extra packages installed on your machine.  
✅ **Easy Upgrades** – Just pull the latest Jenkins image.

**2. Steps to Build, Test, and Deploy a Web App**

**➔ Development Phase: Building the Web App**

**Step 1:** **Requirement Gathering & Planning**

* Define project scope, features, and technology stack (e.g., MERN, Django, etc.).
* Set up a repository on **GitHub, GitLab, or Bitbucket** for version control.
* Create a development workflow using **Kanban, Agile, or Scrum**.

**Step 2:** **Setting Up the Development Environment**

* Install required software like **Node.js, Python, Docker, and databases**.
* Initialize the project using package managers (e.g., npm init, pip install).
* Set up a code editor (VS Code, WebStorm) and frameworks (React, Express, etc.).

**Step 3:** **Writing Code**

* Build the frontend using **React, Angular, Vue, HTML/CSS**.
* Develop the backend using **Node.js, Django, Flask, Spring Boot**.
* Integrate the database (**MongoDB, PostgreSQL, MySQL**).
* Implement authentication (**JWT, OAuth, Firebase**).

**Step 4:** **Version Control**

* Create a Git repository and push code.

bash

CopyEdit

git init

git add .

git commit -m "Initial commit"

git push origin main

**➔ Testing Phase: Ensuring Quality**

**Step 5:** **Unit Testing**

* Write unit tests using **Jest, Mocha, or PyTest**.  
  Example:

javascript

CopyEdit

test('adds 1 + 2 to equal 3', () => {

expect(1 + 2).toBe(3);

});

**Step 6:** **Integration & API Testing**

* Test API endpoints with **Postman, Newman, or Supertest**.
* Automate API tests using **Cypress** or **Selenium**.

**Step 7:** **UI/UX Testing**

* Check cross-browser compatibility on **Chrome, Firefox, and Edge**.
* Test mobile responsiveness.

**Step 8:** **Security Testing**

* Run vulnerability scans using **OWASP ZAP** or **Burp Suite**.
* Implement **SSL, CORS**, and authentication checks.

**➔ Deployment Phase: QA & Production**

**Step 9:** **Deploy to QA Environment**

* Containerize the application using Docker:

bash

CopyEdit

docker build -t myapp .

docker run -d -p 3000:3000 myapp

* Deploy to a QA server (AWS, DigitalOcean, Azure).
* QA team runs manual and automated tests.

**Step 10:** **Deploy to Production**

* Use CI/CD Pipelines (**GitHub Actions, Jenkins, GitLab CI/CD**).
* Deploy using **Kubernetes, Docker Swarm, AWS Elastic Beanstalk**.
* Run load testing using **JMeter**.
* Deploy with zero downtime using **Blue-Green Deployment**.

**➔ Post-Deployment: Monitoring & Maintenance**

**Step 11:** **Monitoring**

* Use **Prometheus, Grafana, Datadog** for app monitoring.
* Set up logging with **ELK Stack** (Elasticsearch, Logstash, Kibana).

**Step 12:** **Bug Fixes & Updates**

* Fix issues based on user feedback.
* Deploy new features using feature flags.
* Maintain rollback plans in case of failure.