Here’s a **detailed breakdown for all experiments (13 to 25)** as requested:

---

13. Set up a CI/CD Pipeline to Automate the Building, Testing, and Deployment of a Containerized Application

AIM

To create and automate the building, testing, and deployment process of a containerized application using Jenkins.

TOOLS USED

- **Jenkins**
- **Docker**
- **GitHub**

PROCEDURE

1. **Install Jenkins**:
   - Install Jenkins on your local machine or server.
   - Add plugins for Docker, Pipelines, and GitHub integration.
2. **Write the Application**:
   - Create a sample application (e.g., a Flask app or static website).
   - Add a `Dockerfile` to containerize the application:
   ```
   FROM python:3.9
   WORKDIR /app
   COPY . /app
   RUN pip install flask
   CMD ["python", "app.py"]
   ```
3. **Push Code to GitHub**:
   - Create a repository for the application.
   - Push all code, including the `Dockerfile`.
4. **Write a Jenkinsfile**:
   - Create a `Jenkinsfile` in the repository for pipeline automation:
   ```
   pipeline {
       agent any
       stages {
           stage('Build') {
               steps {
                   sh 'docker build -t my-app .'
               }
           }
           stage('Test') {
               steps {
                   sh 'echo "Testing the application"'
               }
           }
           stage('Deploy') {
               steps {
                   sh 'docker run -d -p 5000:5000 my-app'
               }
           }
   ```

```
        }
    }
```

5. **Configure Jenkins Pipeline**:
   - Set up a pipeline project in Jenkins.
   - Link it to your GitHub repository.
   - Trigger builds on every commit.
6. **Run and Validate**:
   - Push changes to GitHub.
   - Jenkins will automatically build, test, and deploy the application.

RESULT

A fully functional CI/CD pipeline is established in Jenkins for a containerized application.

---

14. Implement Continuous Deployment Using GitHub Actions

AIM

To automate the deployment of a Dockerized application using GitHub Actions.

TOOLS USED

- **GitHub Actions**
- **Docker**
- **Cloud Platforms (AWS, Azure, or GCP)**

PROCEDURE

1. **Push Dockerized App to GitHub**:
   - Write a simple application (e.g., Flask app).
   - Add a `Dockerfile` and push the code to GitHub.
2. **Create a Workflow**:
   - Create a `.github/workflows/deploy.yml` file in the repository:

   ```
   name: CI/CD Pipeline
   on:
     push:
       branches:
         - main
   jobs:
     build-and-deploy:
       runs-on: ubuntu-latest
       steps:
       - name: Checkout Code
         uses: actions/checkout@v2
       - name: Build Docker Image
         run: docker build -t my-app .
       - name: Push to Docker Hub
         env:
           DOCKER_USERNAME: ${{ secrets.DOCKER_USERNAME }}
           DOCKER_PASSWORD: ${{ secrets.DOCKER_PASSWORD }}
         run: |
   ```

```
          echo $DOCKER_PASSWORD | docker login -u
  $DOCKER_USERNAME --password-stdin
          docker push my-app
      - name: Deploy to AWS ECS
        run: aws ecs deploy my-cluster my-service
```

3. **Test Workflow**:
   - Push code to the repository.
   - Verify the automated build and deployment process.

RESULT

GitHub Actions successfully automates deployment to a cloud platform.

---

15. Create a GitHub Repository and Implement Version Control

AIM

To create a GitHub repository and use version control for collaborative development.

TOOLS USED

- **Git**
- **GitHub**

PROCEDURE

1. **Create Repository**:
   - Log in to GitHub and create a new repository with a README file.
2. **Clone Repository**:
   - Clone it locally: `git clone <repo_url>`.
3. **Add Features**:
   - Create feature branches: `git checkout -b feature-login`.
   - Develop features and commit changes:
     ```
     git add .
     git commit -m "Added login feature"
     ```
4. **Merge Branches**:
   - Push changes to GitHub: `git push origin feature-login`.
   - Create pull requests and resolve conflicts.
5. **Document Workflow**:
   - Update the README with project details and workflow.

RESULT

Version control is successfully implemented for the team project.

---

16. Containerize a Python Flask Application Using Docker

AIM

To build and run a Python Flask app inside a Docker container.

TOOLS USED

- **Flask**
- **Docker**

PROCEDURE

1. **Write the Flask App**:
   - Create a simple Flask app (`app.py`).
2. **Create a Dockerfile**:
   - Write a `Dockerfile` to containerize the app.
3. **Build Docker Image**:
   - Run: `docker build -t flask-app .`
4. **Run and Test**:
   - Start the container: `docker run -p 5000:5000 flask-app`.
   - Access the app in the browser.

RESULT

The Flask app runs successfully in a Docker container.

---

17. Push and Pull Docker Images Using Docker Hub

AIM

To push and pull Docker images using Docker Hub.

TOOLS USED

- **Docker**
- **Docker Hub**

PROCEDURE

1. **Build an Image**:
   - Create and tag the Docker image:
     `docker build -t username/image-name .`
2. **Push Image**:
   - Login to Docker Hub: `docker login`.
   - Push: `docker push username/image-name`.
3. **Pull and Run Image**:
   - Pull on another machine:
     `docker pull username/image-name`.

RESULT

The Docker image is successfully pushed, pulled, and tested.

---

18. Deploy a Multi-Container Application Using Kubernetes

AIM

To deploy a multi-container application using Kubernetes.

TOOLS USED

- **Kubernetes**
- **Docker**

PROCEDURE

1. **Write Kubernetes YAML Files**:
   - Define deployments and services for frontend, backend, and database.
2. **Deploy with `kubectl`**:
   - Run: `kubectl apply -f deployment.yaml`.
3. **Monitor and Scale**:
   - Monitor pods: `kubectl get pods`.
   - Scale services: `kubectl scale deployment frontend --replicas=3`.

RESULT

The multi-container application is deployed and scaled successfully.

---

19 to 25: GitHub & Collaboration Experiments

19. Create a CI/CD Pipeline Using GitHub Actions

- Write a GitHub Actions workflow for testing, building, and deploying the application.

20. Push Changes to GitHub

- Create a repo, edit a file, and push changes.

21. Clone and Modify a Repository

- Clone a public repository, make edits, and commit changes.

22. Commit and Push Changes

- Modify a local file, stage the changes, commit, and push.

23. Pull Changes from a Collaborator

- Pull the latest changes using `git pull` and verify updates.

24. Feature Branch and Pull Request

- Create a feature branch, develop a feature, and merge via a pull request.

25. Fork, Modify, and Submit a Pull Request

- Fork a repository, implement changes in a branch, and submit a pull request.

Each experiment focuses on practical implementation, ensuring you gain hands-on experience. Let me know if you'd like to focus on any specific experiment for even more details!

Here's a **detailed breakdown of experiments 19 to 25**, with an emphasis on aim, tools used, procedure, results, and steps for practice:

---

19. Create a CI/CD Pipeline Using GitHub Actions

AIM

To create and automate a CI/CD pipeline using GitHub Actions for building, testing, and deploying a containerized application.

TOOLS USED

- **GitHub Actions**
- **Docker**
- **Cloud Platform (Heroku, AWS, GCP, or Azure)**

PROCEDURE

1. **Set up GitHub Repository**:
   - Create a new repository for the application.
   - Add the application code (e.g., a Flask API or static website).
2. **Add a Workflow File**:
   - Create a `.github/workflows/ci-cd.yml` file in the repository.
   - Define the stages for CI/CD:

```
name: CI/CD Pipeline
on:
  push:
    branches:
      - main
jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout Code
      uses: actions/checkout@v2
    - name: Build Docker Image
      run: docker build -t my-app .
    - name: Push to Docker Hub
      env:
        DOCKER_USERNAME: ${{ secrets.DOCKER_USERNAME }}
        DOCKER_PASSWORD: ${{ secrets.DOCKER_PASSWORD }}
      run: |
        echo $DOCKER_PASSWORD | docker login -u
$DOCKER_USERNAME --password-stdin
        docker push my-app
    - name: Deploy to Heroku
      env:
        HEROKU_API_KEY: ${{ secrets.HEROKU_API_KEY }}
      run: |
        heroku container:login
        heroku container:push web --app my-heroku-app
        heroku container:release web --app my-heroku-app
```

3. **Configure Secrets**:
   - Add Docker Hub and Heroku credentials in the repository's GitHub Secrets.
4. **Trigger the Workflow**:
   - Push code to the repository. GitHub Actions will automatically build, test, and deploy the application.

RESULT

A fully functional CI/CD pipeline is implemented and verified using GitHub Actions.

---

20. Create a New GitHub Repository and Push Changes

AIM

To create a GitHub repository, make changes locally, and push updates to the repository.

TOOLS USED

- **Git**
- **GitHub**

PROCEDURE

1. **Create a New Repository**:
   - Go to GitHub and create a repository with a README file.

2. **Clone the Repository**:
   - Clone the repository to your local machine using:
   - `git clone <repository_url>`
3. **Edit the README File**:
   - Open the README file in a text editor.
   - Add a project description or additional information.
4. **Stage and Commit Changes**:
   - Use the following commands to save changes:
   - `git add README.md`
   - `git commit -m "Updated project description"`
5. **Push Changes to GitHub**:
   - Push the changes to the remote repository:
   - `git push origin main`

RESULT

The updated changes are successfully pushed to GitHub.

---

21. Clone an Existing GitHub Repository and Modify a File

AIM

To clone an existing GitHub repository, modify a file, and commit the changes.

TOOLS USED

- **Git**
- **GitHub**

PROCEDURE

1. **Choose a Repository**:
   - Select a public or private repository on GitHub.
2. **Clone the Repository**:
   - Clone it to your local machine:
   - `git clone <repository_url>`
3. **Modify a File**:
   - Navigate to the repository folder and edit a file (e.g., README.md or code).
4. **Stage and Commit Changes**:
   - Stage the modified file:
   - `git add <filename>`
   - Commit the changes:
   - `git commit -m "Updated file with new changes"`
5. **Push Changes**:
   - Push the updates to the repository:
   - `git push origin main`

RESULT

The file modifications are committed and pushed to the repository.

22. Commit and Push Changes After Modifying a File

AIM

To modify a file in a GitHub repository, commit the changes locally, and push them to the remote repository.

TOOLS USED

- **Git**
- **GitHub**

PROCEDURE

1. **Clone the Repository**:
   - Clone the repository to your local machine.
2. **Modify a File**:
   - Open and edit a file, such as fixing a typo in the README.
3. **Stage and Commit Changes**:
   - Add the changes to staging:
   - `git add <filename>`
   - Commit the changes:
   - `git commit -m "Fixed typo in README"`
4. **Push Changes to GitHub**:
   - Push the committed changes:
   - `git push origin main`

RESULT

The modifications are successfully pushed to the GitHub repository.

---

23. Pull Latest Changes from a Collaborator

AIM

To update your local repository with the latest changes made by collaborators.

TOOLS USED

- **Git**
- **GitHub**

PROCEDURE

1. **Clone a Repository**:
   - Clone a shared repository:
   - `git clone <repository_url>`

2. **Collaborator Makes Changes**:
   - Ask a collaborator to update the repository and push changes.
3. **Pull the Latest Changes**:
   - Update your local repository:
   - `git pull origin main`
4. **Verify Updates**:
   - Check if the collaborator's changes are reflected in your local repository.

RESULT

Your local repository is updated with the collaborator's changes.

---

24. Create a Feature Branch and Merge via Pull Request

AIM

To create a new branch for a feature, develop the feature, and merge it into the main branch via a pull request.

TOOLS USED

- **Git**
- **GitHub**

PROCEDURE

1. **Create a Branch**:
   - Create a new branch for the feature:
   - `git checkout -b feature-login`
2. **Develop the Feature**:
   - Add a file (e.g., `login.py`) with a simple login function.
3. **Stage and Commit Changes**:
   - Stage and commit the changes:
   - `git add login.py`
   - `git commit -m "Added login feature"`
4. **Push Branch to GitHub**:
   - Push the branch to the remote repository:
   - `git push origin feature-login`
5. **Create Pull Request**:
   - Go to GitHub and create a pull request to merge `feature-login` into `main`.
6. **Merge Pull Request**:
   - Review and merge the pull request.

RESULT

The feature branch is successfully merged into the main branch via a pull request.

---

25. Fork, Modify, and Submit a Pull Request

AIM

To fork a repository, make modifications, and submit a pull request to the original repository.

TOOLS USED

- **Git**
- **GitHub**

PROCEDURE

1. **Fork a Repository**:
   - Fork the original repository into your GitHub account.
2. **Clone the Forked Repository**:
   - Clone it locally:
   - `git clone <forked_repo_url>`
3. **Create a Branch**:
   - Create a new branch for your feature:
   - `git checkout -b feature-enhancement`
4. **Make Changes**:
   - Implement the feature or fix an issue in the branch.
5. **Stage and Commit Changes**:
   - Add and commit changes:
   - `git add .`
   - `git commit -m "Enhanced feature"`
6. **Push Changes**:
   - Push the branch to your forked repository:
   - `git push origin feature-enhancement`
7. **Submit a Pull Request**:
   - Go to the original repository and submit a pull request to merge your changes.

RESULT

The pull request is successfully submitted to the original repository.

This detailed breakdown covers everything you need to perform and understand each experiment. Let me know if you'd like additional examples or troubleshooting steps!