DEVOPS

LAB-01

Git and GitHub

Theory:

Git:

→ It is a distributed version control system.

→ It allows developers to track changes in their codebase over time.

→ It enables collaboration by letting multiple developers work on the same project simultaneously without overwriting each others work.

→ Git works by storing data in snapshots, not differences.

→ Every time you commit, it takes a snapshot of your file.

→ Branching and merging in Git helps in developing features, fixing bugs and experimenting safely.

GitHub:

→ It is a cloud-based hosting services for Git repositories.

→ It provides a web-based interface and additional tools that enhance collaboration and project management.

→ It allows for pull requests, which are proposed changes to the code.

→ They can be reviewed and discussed before being merged.

→ It supports issue tracking, CI/CD integrations, and project boards for agile development.

→ GitHub integrates with Jenkins for CI/CD, Docker for container builds, Snyk for vulnerability scanning and IDEs like VS code.

→ In cyber security, GitHub helps monitor code changes, implement secure workflows, and track vulnerability fixes through audit trails.

→ Together, Git and GitHub streamline collaborative software development by offering version control, tracking and deployment tools.

Lab:

→ Ensure Git is installed.

> git --version
→ Displays installed git version.

→ Go to vs code terminal.

> mkdir gitnid → make a directory

> cd gitnid → go to that directory

> git config --global user.name <username>

→ provide your username for your GitHub account or any name.

> git config --global user.email <email>
→ provide your email.

> git init
→ initializes a new git repository in the current directory.

> git status
→ shows the status of changes (staged, unstaged and untracked files) in the repository.

> git add <filename>
→ stages a specific file for the next commit.

> git commit -m "<message/comment>"
→ creates a commit which describing the changes in a message.
(> git status to show changes).

> git branch <branchname1>
→ creates a new branch with the specified name. (> ~~git add <>, > git commit -m <msg>, etc~~)

> git checkout <branchname1>
→ switches to the specified branch (perform > git add <f>, git commit -m <msg>, git status.. etc)

> git checkout master
→ switches back to the main branch.

> git fetch
→ downloads changes from a repository without merging.

→ create another branch <branchname2> and perform the common commands like add, commit.. etc and go back to main branch

> git merge <branchname2>
→ merges the specified branch to the current branch.

> git log
→ displays the commit history in detail.

> git log --oneline
→ shows commit history in a condensed one-line format.

> git branch -d <branchname2>
→ deletes the specified branch.

> git checkout <hash1>
→ switches to a specific branch using its hash.

> git clone <link of repo>
→ clones a remote repo to your local machine.

> git add . → stages all changes.

> git push origin main → pushes local commits to main branch.

LAB-02

Implement, Build, Test, Configure and Monitor a Software App using Flask

Theory:

Flask:

→ It is a lightweight, python-based micro web framework used for developing web applications.

→ It is known for its simplicity and flexibility.

→ Allows developers to build scalable apps with minimal boilerplate code.

Usage:

Build:

→ Flask allows routing of URLs to python functions using ~~decorator~~ decorators.

→ Templates are rendered using Jinja2.

→ Its modular structure makes it easy to build APIs and web apps.

Test:

→ Flask supports unit testing using Python's built-in "unittest" or "pytest".

Configure:

→ Flask uses configuration files or environment variables for managing development, testing and production settings.

Monitor:

→ Integrations like Prometheus, Sentry or Flask-logging can be used to monitor logs, errors and performance.

Additional Features:

→ Supports RESTful APIs using Flask-RESTful.
→ Easy integration with ORMs like SQLAlchemy.
→ Blueprinting allows the application to be split into reusable components.

Lab:

Implementation: (In Ubuntu)

```
> sudo apt update
> python3 --version
> sudo apt install python3-pip -y
> pip --version
> sudo apt install python3-venv -y
> mkdir devops2
> cd devops2
>python3 -m venv flaskenv
>source flaskenv/bin/activate
>pip install flask
>flask --version
> nano app.py
```

→ Copy/write the following program.

Program:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def home:
    return "Hello, DevOps!"
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000)
```

>python3 app.py
→ Click on the link or copy/paste in browser.

Build:

>nano build.sh

```
#!/bin/bash
echo "setting up environment..."
python3 -m venv flaskenv
source ./flask/bin/activate
pip install flask.
echo "Envirohment setup complete. Run the app with: python
```

>bash build.sh

Test:

>nano test.py

```
import unittest
from app import app
class TestApp(unittest.TestCase):
    def test_home(self):
        tester = app.test_client()
        response = tester.get('/')
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data, b"Hello, DevOps!")
if __name__ == "__main__":
    unittest.main()
```

>python3 test.py

Configure:

>nano .env

FLASK_PORT=5000

>pip install python-dotenv

>nano app.py
(modify the program)

```
import os
from flask import Flask
app = Flask(__name__)
@app.route('/')
def home:
    return "Hello, DevOps!"
if __name__ == "__main__":
    port = int(os.getenv("FLASK_PORT", 5000))
    app.run(port=port)
```

>python3 app.py

Monitor:

>nano app.py (modify the prog!!)

Add funcn:

```
@app.route('/health')
def health():
    return {"status": "up"}, 200
```

>python3 app.py
(open link) (also append /health in url and check)

LAB-03

Selenium

Theory:

Selenium:

→ It is a powerful open-source tool used for automated web browsers. ~~It allows testers~~

→ It allows testers to write scripts in multiple languages like Java, Python, or JavaScript to simulate user interaction with web-pages.

Key Features:

(i) Cross Browser Testing:

→ Selenium WebDriver interacts with Chrome, Firefox, Safari..etc for consistent behavior across browsers.

(ii) Multi-language Support:

→ Works with Java, Python, Ruby, C#..etc

(iii) Automated Form Filling and Navigation:

→ Automates clicks, form entries, scrolls and navigation.

(iv) Element Inspection:

→ Identifies elements by ID, name, XPath, CSS selectors ..etc.

Usage:

(i) Functional Testing:

→ Ensures the features of a web application work as expected

(ii) Regression Testing:

→ Re-runs test cases to confirm that new code changes haven't broken existing functionality.

(iii) Integration Testing:

→ Verifies the interaction between components.

→ Selenium also supports TestNG or JUnit for test management.

→ It can be integrated with CI/CD tools like Jenkins for

Lab:

→ Install python
>python3 --version
>pip install selenium
>selenium --version (or) >pip show selenium

Program-1

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time
browser = webdriver.Firefox()
browser.get('http://www.yahoo.com')
assert 'Yahoo' in browser.title
elem = browser.find_element(By.NAME, 'p')
elem.send_keys('selenium' + Keys.RETURN)
time.sleep(10)
browser.quit()
```

Program-2

```
import unittest
from selenium import webdriver
class GoogleTestCase(unittest.TestCase):
    def setUp(self):
        self.browser = webdriver.Firefox()
        self.addCleanup(self.browser.quit)
    def test_page_title(self):
        self.browser.get('http://www.google.com')
        self.assertIn('Google', self.browser.title)
if __name__ == "__main__":
    unittest.main(verbosity=2)
```

Program-3: (Install chrome driver, disable device security)

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
import time
chrome_driver_path = "C:\Driver\chromedriver.exe"
service = Service(executable_path=chrome_driver_path)
driver = webdriver.Chrome(service=service)
driver.get("https://www.google.com")
assert "Google" in driver.title
time.sleep(5)
driver.quit()
```

LAB-04

Gradle

Theory:

Gradle:

→ It is an advanced build automation tool used primarily for Java-based projects, though it supports other languages too.

→ It allows developers to automate tasks like compiling code, running tests, packaging applications and deploying them.

→ Gradle uses a "direct acyclic graph" (DAG) to determine task dependencies and execution order.

→ Gradle supports incremental builds, where only modified files are recompiled.

Groovy:

→ It is the scripting language used to write Gradle build scripts.

→ It's a dynamic language for the Java platform.

→ Gradle build scripts written in Groovy are concise and readable.

Eg:
```
task hi {
    doLast {
        println "Hello World!"
    }
}
```

Key Features:

→ Plugin-based architecture.

→ Dependency management through Maven or Ivy repositories.

→ Easy CI/CD Integration.

→ Gradle and Groovy together provide a flexible, high performance build system suitable for large enterprise applications and mobile apps.

Gradle automates the following tasks:

i) Compiling
ii) Running tests
iii) Creating Jar files
iv) Creating Tasks.

Lab:

Step 1: Open VSCode

Step 2: Go to Extensions

Step 3: → Install Gradle for Java
→ Install Extension Pack for Java
→ Install Groovy

Step 4: Open new folder and choose new file in the main window.

Step 5: Choose new Java project and press select.

Step 6: Choose Groovy and press Enter (don't create any folder)

Step 7: Once the Gradle file is opened, and created, go to:
app → src → main → java → App.java.

→ Type in the following program and run it:

```
package gradleproj;(or the name of the folder you created)
import java.awt.Desktop;
import java.net.URI;
public class App{
    public static void main(String[] args){
        try{
            String url="https://www.google.com";
            if(Desktop.isDesktopSupported()){
                Desktop desktop=Desktop.getDesktop();
                desktop.browse(new URI(url));
            }
            else{
                System.out.println("Desktop is not
                supported on this platform.");
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Manual Gradle Installation:

Step 1: Search "Gragle download" in browser.

Step 2: Download the "binary-only" zip file.

Step 3: Extract the zip file and copy it into C: Drive

Step 4: Open the file, open "bin" folder and copy the path.

Step 5: Open "Edit System Environment Variables" and edit "path" and add this path (copied) to it.

CMD:

> gradle --version
> mkdir gradleproject
> cd gradleproject
> gradle init

→ Choose the following:

Application
↓
Groovy
↓
click Enter
↓
click Enter
↓
Single Application Project
↓
Groovy

> gradlew task

→ Go to "app" in gradleproject in C:Drive → Users → CYSG → gradleproject, open "build.gradle" in VSCode and type the following after the last block of code.

```
task hello{
    doLast{
        println ("Hellogradle");
    }
}
```

> gradlew task
> gradle hello

LAB-05

Snyk

Theory:

Snyk:

→ Snyk is a developer-first security tool designed to find and fix vulnerabilities in code, dependencies, containers ..etc.

Core Functions:

(i) Code Scanning:
→ Analyzes source code for security flaws using SnykCode.

(ii) Dependency Scanning:
→ Detects known vulnerabilities in open-source packages via Snyk Open Source.

(iii) Container Security:
→ Scans Docker images for vulnerabilities.

(iv) IaC Scanning:
→ Checks Kubernetes, Terraform ..etc. for insecure code configurations.

Advantages:

→ Easy integration with GitHub, GitLab, Bitbucket, and CI/CD tools.
→ Offers automated pull requests to fix vulnerable dependencies.

Workflow:

→ Snyk scans a project and identifies security issues.
→ It links issues to known CVEs.
→ Provides remediation steps like upgrading a package or patching a vulnerability.

PART-01

Step ①: Google "snyk" and choose the website "snyk.io".

Step ②: Create an account and link your github account to it.

Step ③: Choose a repository and click on "import and scan".

Step ④: The tool will display varied levels of vulnerabilities and their count on the right in boxes.

C → Critical
H → High
M → Moderate
L → Low

(It's assigned based on a vulnerability score)

Step ⑤: Click on any of the boxes and it will display the files within the project that are vulnerable.

Step ⑥: Click on any project and that will display ~~the files within the project that are vulnerable~~ the vulnerabilities in detail with options to ignore and "Fix this vulnerability".

Step ⑦: Click on "Fix the vulnerability". It will display a list or series of actions recommended to be taken to fix that vulnerability.

PART-02: Using command line interface:

→Install nodejs
(Go to nodejs website and install Windows installer (.msi file). Make sure to check the boxes to include "npm".
Open cmd and type >node -v and >npm -v to confirm)

```
>node
>npm install -g snyk
>snyk auth
```

→This will open a webpage asking permission to link github account.
→Click on "Grant App Access".
→Go to web browser and search "pygoat".
→Click on the first link, click on "Code" and copy HTTPs link.

```
>mkdir snykscan
>cd snykscan
>git clone <copied link>
>cd pygoat
>snyk test
>snyk monitor
```

PART-03 : Using VS Code:

Step 1: Open the "pygoat" folder.

Step 2: Open "Extensions" and install "Snyk Security"

Step 3: Go to the installed extension.

Step 4: Click on "Code Quality" (or any option)

Step 5: Click on any file. It will provide a dropdown list with all existing vulnerabilities.

Step 6: Click on any vulnerability to get an analysis and solution window.

LAB-06
Docker

Theory:

Docker:

→ It is a containerization platform that allows developers to package applications and their dependencies into lightweight, portable containers.

Core Concepts:

(i) Containers:

→ Run the same regardless of the environment, ensuring consistency

(ii) Images:

→ Read only templates used to create containers.

(iii) Dockerfile:

→ Script containing instructions to build Docker images.

Why use Docker?:

(i) Portability:

→ Runs on any system with Docker Engine.

(ii) Isolation:

→ Apps run in seperate containers, preventing conflict.

(iii) Efficiency:

→ Uses fewer resources than virtual machines.

Usage:

(i) Build:

→ Developers create reproductible environments.

(ii) Test:

→ Automated tests run inside isolated containers

(iii) **Deploy:**
→ Containers are deployed on servers or orchestrated via Kubernetes.

(iv) **Monitor:**
→ Integration with tools like Prometheus and Grafana for monitoring ~~for monitoring~~ container performance.

Lab:

PART-01: Downloading and Executing an already existing Project from GitHub

Step 1: Go to Chrome and search "docker desktop".

Step 2: Go to "docker.com", click on "Download for Windows AMD64".

Step 3: Check for WSL in the system.
(If not installed, >wsl --install)

Step 4: Double-click on the downloaded file, click on "Accept", "Skip" and choose "Student", click on "Continue".

Step 5: Click on "What is a container?" and keep clicking "Next" until you reach the end.

Step 6: If there is no link, then manually search "github.com/docker/welcom-to-docker"

Step 7: Copy the HTTP code link.

Step 8: Go to cmd, execute the following commands:

```
>mkdir dockproj
>cd dockproj
>git clone <copied link>
>cd welcome-to-docker
>docker build -t welcome-to-docker .
>docker run -p 5000:5000 welcome-to-docker
```

Step 9: Go to Docker and click on the link given for the directory/project you just created.
→ It will display a window with the message "Congratulations!"

PART-2: Manually Creating a Project (Flask):

In cmd,

```
>mkdir dockerproject
>cd dockerproject
>notepad app.py
```

→ Type the following code in app.py:

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello_docker():
    return "Hello Docker!"
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=3000)
```

→ Create another file "requirements.txt" and type the following:

```
Flask==2.0.1
Werkzeug==2.0.1
```

→ Create another file Dockerfile (file type: all) and type the following:

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 3000
CMD ["python", "app.py"]
```

(All these files should be in the same directory)

→ Go to cmd,

>ren Dockerfile.txt Dockerfile

>docker build -t dockerproject

>docker run -p 3000:3000 dockerproject.

LAB-07

Jenkins

Theory:

Jenkins:

→ It is an open-source automation server widely used for implementing CI/CD in software development.
→ It helps automate building, testing and deploying applications, making the development process faster and reliable.

Key Features:

(i) Plugins:
→ It supports over 1000 plugins to integrate with tools like Git, Maven, Docker.. etc.

(ii) Pipeline as Code:
→ Developers can define their CI/CD pipelines using Groovy-based "Jenkinsfile".

(iii) Distributed Builds:
→ Jenkins can manage builds across multiple machines.

Workflow:

→ Developer pushes code to GitHub.
→ Jenkins detects the change and starts a build.
→ Code is compiled, tested and packaged.
→ If successful, it may be deployed to staging or production.

Benefits:

→ Early detection of bugs.
→ Faster release cycles
→ Reduced manual interventions
→ Supports integration with Slack, JIRA, Snyk.. etc

Lab:

Step 1: Installation of JDK:

→ Go to chrome and type "prebuilt openJDK binaries for free" and select "adoptium.net".
→ Click on "Latest LTS releases".
→ Download version 21.0.7 (.msi files)
→ Go to Downloads and double click on openJDK and install.
(Choose "all users of this machine" and select "will be installed on Local Harddrive")
→ Go to C: Drive → Program Files → Eclipse Adoptium → JDK → bin.
→ Copy path and add it to Environment Variables.
(Make sure "Java Home" is also a path).
→ To verify, → java --version.

Step 2: Installation of Jenkins:

→ Go to Chrome and type "Jenkins for windows" and click on "jenkins.io" and select "Windows OS" and download.
→ Go to Downloads and double click on Jenkins.
→ In "Run service as a local or domain user", since we don't know the password, select "Run service as local system".
→ Click on "Test Port". If failed, give another port (eg. 8001).
→ Click Next → Next → disable "Start Service" feature → select "Entire feature will be unavailable" → Install.

Step 3: → Search "Local Security Policies" and select it.
→ Select "User Rights Assignment" and select "Log on as a service" → add user → type "Administrator" → select and apply.

Step 4: → Search "Services" in Start and find "Jenkins"
→ Right click on it and select "start"
and close the window.

Step 5: → To check if server is running, go to:
C: Drive → Program Files → Jenkins → Check whether "Jenkins.out" exists

Step 6: Go to Chrome and type "localhost:8081".

Step 7: Open "Jenkins.err" file and look for the default password generated.
→ Copy password and paste in the browser window.

Step 8: Go to "Customise Jenkins" and select "Install Suggested Plugins".

Step 9: In Create First Admin User, enter the details and click on "Save and Continue" and then click on "Save and Finish".
→ Click on "Start Using Jenkins".

Step 10: Click on "+New Item", enter Item name, select "Freestyle Project" and click OK.

Step 11: Go to "Build Steps", click on "Execute Windows Batch Command".
→ Type (echo "proj 1 is created" %date% %time%)
and save it.

Step 12: Click on "Build now".
→ If build is successful, click on "Console Output".

LAB-8

Kubernetes

Theory:

Kubernetes:

→ Also called K8s.
→ It is an open-source platform for automating the deployment, scaling and management of containerized applications.
→ Originally developed by Google.
→ Managed by Cloud Native Computing Foundation (CNCF).

Core Concepts:

(i) **Pod:**
→ The smallest deployable unit.
→ Contains one or more containers.

(ii) **Service:**
→ Defines how Pods communicate with each other or the outside world.

(iii) **Deployment:**
→ Manages the desired state of your application.

(iv) **Node:**
→ A server where Pods are deployed.

(v) **Cluster:**
→ A collection of nodes managed by Kubernetes.

Benefits:

(i) **Self-healing:**
→ Automatically replaces failed containers.

(ii) **Scalability:**
→ Easily scale apps up/down based on load.

(iii) **Rolling Updates:**
→ Deploy new versions with zero downtime.

(iv) **Load Balancing:**
→ Distributes traffic efficiently across Pods.

Lab:

Step 1: Install kubectl:

→ Search "kubectl download" in browser.
→ Go to the first link and click on "Install kubectl binary on Windows (via direct download or curl)".
→ Go to cmd, type cd.. until you're in :C directory.
> mkdir kube
> cd kube
→ Copy the first "curl" command with url and paste in cmd.
→ Copy the second link and execute in cmd.
~~→ Follow all the steps on the website.~~
→ Validate kubectl binary against checksum file (paste the next 2 commands from the website in cmd and execute).
→ If the hash values in both command outputs are the same, ~~validation is done~~ continue.

Step 2: Check kubectl version

> kubectl version --client

Step 3: Check the path.

> where kubectl.

Step 4: Add path to Environment Variables.

→ copy path from cmd only.

Step 5: Open Powershell (NOT as Administrator)

→ Type cd.. until in :C directory
>cd kube (the name of directory you created)
→ Copy the command for Powershell in Step 2 in the website and execute it- Powershell.
(To get a True/False output)
→ If you get True, validation is done.

Step 6: Install Minikube:

→ Open Powershell (RUN as Administrator)
→ Navigate back to :C directory by using cd..
> cd kube
→ Search "install minikube" in browser and click the first link.
→ Copy and execute the first two command in the Powershell cmd.
→ Close the Admin Powershell window, make sure Docker is running, and restart PowerShell as admin
→ Then execute the following:

> minikube start

(If it doesn't work, try:

> minikube start --driver=docker)

> minikube version

> minikube status
(everything should be "running")

> kubectl get nodes

→ Scroll down to go to "Services" tab in the website, copy and paste and execute the first two commands.

→ Then,
> kubectl get services hello-minikube
> minikube service hello-minikube.

→ This will open the browser to display a HTTP GET request header format details.

LAB-09

Ansible

Theory:

Ansible:

→ It is an open-source automation tool used for configuration management, application deployment and orchestration.
→ Uses simple YAML (Yet Another Markup Language) files called "playbooks" to define automation tasks.

Basic Components:

(i) Inventory:
→ Contains list of hosts in .ini file.

(ii) Modules:
→ Reusable units of code.

(iii) Playbooks:
→ contains a series of tasks
→ extension is .yml

(iv) Roles:
→ way to organize playbooks and tasks into reusable components

Key Features:

(i) Agentless:
→ It doesn't require installing agents on client machines.

(ii) Idempotent:
→ Ensures that operations only make changes when needed.

(iii) SSH-based:
→ Secure and simple communication with nodes via SSH.

Use Cases:

→ Automating server setup and deployment.
→ Managing configurations across multiple environments.
→ Orchestrating updates and patches.

Lab:

→ In Ubuntu cmd,

> sudo apt update

→ To install Ansible,

> sudo apt install ansible

> ansible --version

→ > nano hosts.ini

Type the following:

```
[local]
localhost ansible_connection=local
```

→ Ctrl X

→ Y

→ Enter

→ > nano setup.yml

Type the following:

```
---
- name: Basic Server Setup
  hosts: local
  become: yes
  tasks:
    - name: Update apt cache
      apt:
        update_cache: yes
    - name: Install curl
      apt:
        name: curl
        state: present
```

→ Ctrl X

→ Y

→ Enter.

→ Open browser, copy the above code, search "yaml validator" and check code validity.

→ To execute the code, > sudo ansible-playbook -i hosts.ini setup.yml

LAB-10

Maven

Theory:

Maven:

→ It is a popular build automation and dependency management tool.

→ Recommended for Java-based projects, though it supports other languages.

→ It uses an XML file called pom.xml (Project Object Model) to manage:

(i) Project Structure
(ii) Dependencies
(iii) Build Lifecycle
(iv) Plugins

Core Concepts:

(i) Dependencies:

→ Automatically downloads required libraries from central repositories.

(ii) Phases:

→ Build lifecycle is divided into phases.

(iii) Plugins:

→ They extend Maven functionality.

Advantages:

(i) Standardization:

→ Every Maven project has a similar structure.

(ii) Easy Integration:

→ Compatible with CI/CD tools (Eg: Jenkins).

(iii) Reusable Artifacts:

→ Built artifacts can be reused across projects.

→ Build contains the following commands:
(i) clean (to delete a folder called "target" if any, before execution of other commands)
(ii) validate
(iii) test (for unit testing)
(iv) compile
(v) package (to run the project)

Lab:

→ In Ubuntu cmd,
>sudo apt update
→ > sudo apt install openjdk-11-jre-headless -y
(Install OpenJDK)

→ To install Maven,
>sudo apt install maven -y
>mvn --version
→ >mkdir mavendir
>cd maven dir
→ > git --version
(If not installed, then,
>sudo apt install git
>git --version)
→ >git clone https://github.com/Krishpluto/Boardgame Listing WebApp.git
>cd Boardgame Listing WebApp
→ Execute all Maven commands:
>mvn clean
>mvn validate
>mvn test
>mvn compile
>mvn package
→ Check if "target" folder has been created in the directory, if yes:
>cd target.
→ Execute the JAR file:
>java -jar database_service_project-0.0.1.jar