<table>
<tr><td colspan="4">Name of Student: SNEHAL GAJANAN DIGHORE</td></tr>
<tr><td colspan="2">Roll Number: 07/A</td><td colspan="2">LAB Assignment No.- 01</td></tr>
<tr><td colspan="4">Title of LAB Assignment :
-To understand DevOps: Principles,CI/CD Pipeline, Practices, DevOps Lifecycle, and DevOps Engineer Role and Responsibilities.</td></tr>
<tr><td colspan="2">DOP: 20TH January 2025</td><td colspan="2">DOS: 26th January 2025</td></tr>
<tr><td>CO MAPPED:</td><td>PO MAPPED:</td><td>Signature:</td><td>Marks:</td></tr>
</table>

Aim : To understand DevOps: Principles,CI/CD Pipeline, Practices, DevOps Lifecycle, and DevOps Engineer Role and Responsibilities.

1. DevOps Principles in Practice

In a real-world DevOps environment, the principles translate into specific actions:

- **Collaboration**:
    - **Tools**: Slack, Microsoft Teams, or Zoom for communication. Use Jira or Trello for task tracking, where both developers and operations teams contribute.
    - **Action**: Daily stand-ups, joint planning sessions, and retrospectives to ensure everyone is aligned on goals and progress.
- **Automation**:
    - **Tools**: Jenkins, GitLab CI, CircleCI, or Travis CI for automating builds and tests; Ansible, Terraform, or Chef for infrastructure automation.
    - **Action**: Write scripts to automatically deploy code, provision servers, or manage cloud resources. Use CI/CD pipelines to automate testing and deployment.
- **Continuous Improvement**:
    - **Tools**: Git for version control, JIRA for tracking feedback, GitHub or GitLab for reviewing pull requests.
    - **Action**: Implement a feedback loop where developers continuously improve code quality by integrating feedback from automated tests, peer reviews, and production monitoring.
- **Customer-Centric Action**:
    - **Tools**: Google Analytics, New Relic, or Datadog for gathering insights into user behavior and performance.
    - **Action**: Regularly collect feedback from real users and incorporate it into the development process to ensure the software is evolving in line with customer needs.

2. CI/CD Pipeline in Practice

The CI/CD pipeline is the backbone of DevOps automation. Here's how you can set up and maintain it:

- **Continuous Integration (CI)**:
    - **Tools**: Jenkins, GitHub Actions, GitLab CI, CircleCI.

    - **Action**: Set up a Git repository for version control. Every time a developer pushes code to the repository, the CI tool triggers an automated build and runs unit tests.
    - **Example**: On commit, the pipeline runs mvn clean install (for Java) or npm run build (for Node.js) to build the application and npm run test to run automated tests.
- **Continuous Delivery (CD)**:

    - **Tools**: Kubernetes (for deployment), Helm (for managing Kubernetes deployments), Docker (for containerization), AWS (for hosting).

    - **Action**: After the CI phase passes, the build is automatically deployed to a staging environment for further testing. If the staging environment is approved, the same build is automatically deployed to production.

    - **Example**: Use Jenkins to deploy an app to a Kubernetes cluster. Once tests pass in the staging environment, Jenkins triggers a Helm chart to push the app to production.

- **Continuous Deployment (CD)**:

    - **Tools**: Kubernetes, Docker, GitLab CI/CD, ArgoCD.
    - **Action**: Once the code passes all tests, it's deployed automatically into the production environment without manual intervention.
    - **Example**: After successful tests, the code is deployed into Kubernetes using a continuous deployment strategy like Blue/Green deployment or Canary releases.

3. DevOps Practices in Practice

These are some practical steps and tools used in a DevOps environment:

- **Version Control**:

    - **Tools**: Git (with GitHub, GitLab, or Bitbucket).
    - **Action**: Use Git to commit code in small increments. Use feature branching for new features and pull requests for merging code.
    - **Example**: Use git commit and git push for local development and pushing changes, respectively. Use git merge for integrating code.
- **Infrastructure as Code (IaC)**:

    - **Tools**: Terraform, Ansible, CloudFormation.
    - **Action**: Write configuration files to provision and manage your infrastructure.
    - **Example**: Use Terraform scripts to spin up cloud infrastructure (e.g., EC2 instances, S3 buckets, databases) on AWS.

```
resource "aws_instance" "web" {
  ami = "ami-12345"
  instance_type = "t2.micro"
}
```

- **Automated Testing**:
    - **Tools**: Selenium (for UI), JUnit (for Java), PyTest (for Python), Mocha (for Node.js).
    - **Action**: Set up unit tests, integration tests, and UI tests to run as part of your CI/CD pipeline.
    - **Example**: Write automated tests for your app and integrate them into the pipeline. If the tests fail, the deployment is halted, and the issue is resolved before further deployment.
- **Monitoring and Logging**:
    - **Tools**: Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana), Splunk, Datadog.
    - **Action**: Set up monitoring to track metrics (CPU usage, memory, traffic) and logging to track errors or performance issues.
    - **Example**: Use Prometheus for monitoring app performance and Grafana to visualize the metrics. Set up log aggregation with the ELK Stack.

4. DevOps Lifecycle in Practice

Let's see how the DevOps lifecycle works practically with tools at each stage:

1. **Plan**:
    - **Tools**: Jira, Trello, Confluence.
    - **Action**: Developers and operations teams plan new features, identify risks, and align on goals. Use Kanban boards in Jira to manage workflows and sprints.
2. **Develop**:
    - **Tools**: Git, IDEs like VS Code, IntelliJ, or PyCharm.
    - **Action**: Developers work on code locally and commit changes to a shared repository. Code reviews can be done via pull requests in GitHub or GitLab.
3. **Build**:
    - **Tools**: Jenkins, CircleCI, Maven, Gradle.
    - **Action**: The code is built into an artifact (e.g., JAR, Docker image) using an automated build system. If tests pass, the build is marked as successful.
4. **Test**:
    - **Tools**: Selenium, JUnit, PyTest, Postman (for API testing).

    - **Action**: Automated tests (unit, integration, acceptance tests) are executed to validate the functionality of the application.
5. **Release**:
    - **Tools**: Helm, Kubernetes, Docker.
    - **Action**: Once the tests pass, the code is released to staging, where further manual or automated testing takes place.
6. **Deploy**:
    - **Tools**: Kubernetes, Docker, Terraform.
    - **Action**: Once approved in staging, deploy the code to production. Automate deployment through scripts or using CI/CD tools.
7. **Operate**:
    - **Tools**: Datadog, Prometheus, Nagios.
    - **Action**: The application is monitored continuously to ensure smooth operation. Health checks, uptime monitoring, and performance metrics are tracked.
8. **Monitor**:
    - **Tools**: ELK Stack, Splunk, Grafana.
    - **Action**: Use logging and monitoring tools to collect and analyze logs and performance data. Alerts are set up for issues like high latency or downtime.

5. DevOps Engineer Role and Responsibilities in Practice

A DevOps engineer in practice is the one who:

- **Implements and maintains CI/CD pipelines**: Setting up automated workflows for build, test, and deployment processes.
    - **Tools**: Jenkins, GitLab CI, CircleCI.
- **Manages Infrastructure**: Writing Infrastructure as Code (IaC) scripts to provision and manage cloud resources.
    - **Tools**: Terraform, Ansible, CloudFormation.
- **Automates Cloud Infrastructure**: Working with cloud providers like AWS, Azure, or Google Cloud to deploy applications on scalable infrastructure.
    - **Tools**: AWS CLI, Google Cloud SDK, Kubernetes.
- **Monitors and optimizes systems**: Continuously monitoring infrastructure and application performance to ensure uptime and reliability.
    - **Tools**: Prometheus, Grafana, Datadog.
- **Ensures security**: Integrating security into the development pipeline (DevSecOps).

- **Tools**: Vault, SonarQube for static code analysis, Snyk for vulnerability scanning.
-

Conclusion:

DevOps is a transformative approach that enhances collaboration between development and operations teams, streamlining software delivery processes. By focusing on automation, continuous integration, and continuous delivery (CI/CD), DevOps enables faster, more reliable releases while maintaining high-quality standards.

Through the use of practical tools like Jenkins, GitLab, Kubernetes, Terraform, and Prometheus, organizations can implement a fully automated pipeline, from development to production, with minimal manual intervention. The key practices of infrastructure as code (IaC), automated testing, and continuous monitoring ensure that systems remain scalable, secure, and reliable in production environments.

The role of a **DevOps engineer** is pivotal in setting up, managing, and optimizing these tools and workflows, bridging the gap between development and operations, and ensuring smooth, efficient deployments. By embracing DevOps principles, teams can continuously improve their processes, delivering value to customers quickly and iteratively while reducing the risk of errors and downtime.

Ultimately, DevOps is not just about the tools—it's a cultural shift towards collaboration, transparency, and shared responsibility, enabling teams to deliver software faster and with higher quality, aligned with customer needs.