Student Handout: GitLab CI/CD & GitLab Runner Deep Dive

# Part 1 – GitLab CI/CD

## 1. What is CI/CD?

Continuous Integration (CI): Automating build and test of code changes as soon as they are pushed to a shared repo.  
Continuous Delivery (CD): Automatically preparing code for deployment to staging or production once CI succeeds.  
Continuous Deployment (CD): Fully automating release to production without manual approvals.  
  
Benefits:  
- Faster feedback  
- Improved code quality  
- Reduced manual effort  
- Quicker release cycles

## 2. Tools for CI/CD (Comparison)

- Jenkins: Flexible, plugin-rich, but requires heavy maintenance.  
- GitLab CI/CD: Integrated with GitLab SCM, YAML-based config, strong container support.  
- CircleCI: Cloud-native, quick setup, limited free usage.  
- Bamboo: Atlassian product, works well with Jira, paid.  
- TeamCity: JetBrains product, strong IDE integration, paid.

## 3. Why GitLab CI/CD?

- Integrated with GitLab (no external setup).  
- YAML-based pipelines (.gitlab-ci.yml).  
- Parallel job execution for faster pipelines.  
- Supports containerized development & deployment.  
- Built-in security & secrets management.

## 4. Key Components

- Pipeline: Defines stages and job execution order.  
- Jobs: Smallest execution units in CI/CD.  
- Runners: Agents that execute jobs.  
- Artifacts: Files generated by jobs (e.g., test reports).  
- CI/CD Variables: Environment variables for configuration.  
- Triggers: Used for pipeline automation (e.g., webhooks).

## 5. GitLab CI/CD Features

- Automated builds, tests, and deployments.  
- Integration with Docker/Kubernetes.  
- Infrastructure as Code support (Terraform, Ansible, etc.).  
- Built-in container registry.  
- Advanced caching & parallelization.

## 6. Use Cases & Best Practices

Use Cases:  
- Web app CI/CD  
- Infrastructure automation  
- Data pipelines  
- Containerized applications  
  
Best Practices:  
- Secure secrets (use GitLab CI/CD variables, avoid hardcoding).  
- Use caching for faster builds.  
- Test pipelines thoroughly before production.  
- Apply Infrastructure as Code principles.

# Part 2 – GitLab Runners

## 1. What is a GitLab Runner?

- An agent that executes jobs defined in .gitlab-ci.yml.  
- Open-source application.  
- Can run on local machines, VMs, cloud instances, or containers.  
- Communicates with GitLab to receive jobs and send results.

## 2. Why Runners are Important

- Perform the actual work in pipelines.  
- Provide execution environments (Docker, Shell, Kubernetes).  
- Enable fast, automated delivery.

## 3. Runner Architecture & Workflow

1. Registration – Runner registers with GitLab instance.  
2. Job Assignment – GitLab assigns jobs from pipeline.  
3. Environment Launch – Executor prepares the environment.  
4. Execution & Reporting – Runs job, streams logs, sends results back.

## 4. Types of Runners

- Shared Runners: Available to all projects in an instance.  
- Specific Runners: Dedicated to one project/group.  
- Group Runners: Shared across a group and subgroups.

## 5. Executors

- Shell Executor: Runs directly on host machine, low isolation, good for simple builds.  
- Docker Executor: Runs each job in a fresh container, high isolation, consistent environments.  
- Kubernetes Executor: Schedules each job as a Pod, highly scalable, best for cloud-native teams.  
  
Comparison:  
| Executor | Isolation | Use Case | Complexity |  
|------------|-----------|-------------------------|------------|  
| Shell | Low | Simple local builds | Low |  
| Docker | High | Most common, consistent | Medium |  
| Kubernetes | Very High | Large, scalable workloads | High |

## 6. Registering a Runner

Shell Example:  
sudo gitlab-runner register  
  
Docker Example:  
sudo gitlab-runner register \  
 --url https://gitlab.com/ \  
 --registration-token <TOKEN> \  
 --executor docker \  
 --description "docker-runner" \  
 --docker-image "alpine:latest"  
  
Kubernetes Example (Helm):  
helm repo add gitlab https://charts.gitlab.io  
helm install gitlab-runner gitlab/gitlab-runner \  
 --set gitlabUrl=https://gitlab.com/ \  
 --set runnerRegistrationToken=<TOKEN>

## 7. Managing Runners

- Check from GitLab UI → Project > Settings > CI/CD > Runners  
- Assign tags for job routing.  
- Enable/disable runners as needed.

## 8. Best Practices

- Use tags for job targeting.  
- Use Docker/Kubernetes executors for isolation.  
- Avoid shared runners for sensitive workloads.  
- Keep Runners updated.  
- Monitor CPU/memory usage.  
- Use autoscaling runners in cloud environments.

## 9. Troubleshooting

- Check logs at /var/log/gitlab-runner/.  
- Useful commands:  
 - gitlab-runner list – List registered runners  
 - gitlab-runner verify – Check connectivity  
 - gitlab-runner restart – Restart runner service

## 10. Key Takeaways

- Runners are the backbone of GitLab CI/CD.  
- Choosing the right executor is critical.  
- Keep them secure, updated, and monitored.  
- Use private runners for sensitive data.

# Further Learning

- Official GitLab CI/CD Documentation: https://docs.gitlab.com/ee/ci/  
- GitLab Runner Documentation: https://docs.gitlab.com/runner/