Student Handout: GitLab CI/CD – Pipelines, Variables & Secrets

# Part 1 – Building Your First Pipeline

## 1. What is a CI/CD Pipeline?

A pipeline is an automated sequence of steps that takes code from development → testing → deployment. It ensures faster and reliable delivery with less manual intervention.  
  
Flow:  
1. Development: Code is committed to GitLab repository.  
2. Testing: Automated tests run.  
3. Deployment: Validated code is deployed to environments (staging/prod).  
  
In GitLab, pipelines are defined in `.gitlab-ci.yml`.

## 2. Anatomy of .gitlab-ci.yml

- YAML Format: Human-readable configuration.  
- Stages: Logical sequence of execution (build, test, deploy).  
- Jobs: Tasks within a stage (e.g., run tests, build image).  
- Scripts: Commands executed by a job.

Example:

stages:  
 - build  
 - test  
 - deploy  
  
build-job:  
 stage: build  
 script:  
 - echo "Building the application"  
  
test-job:  
 stage: test  
 script:  
 - echo "Running tests"  
  
deploy-job:  
 stage: deploy  
 script:  
 - echo "Deploying application"

## 3. Stages – Best Practices

- Common stages: build, test, deploy, lint, review.  
- Sequential flow: Stages run top-to-bottom in order.  
- Parallel execution: Jobs in the same stage run concurrently → faster pipelines.

## 4. Job Configuration Options

Jobs can be customized with multiple options:

test-job:  
 stage: test  
 tags:  
 - python  
 script:  
 - npm install  
 - npm test  
 artifacts:  
 paths:  
 - coverage/

- script: Actual commands run in the job.  
- only/except: Run jobs only on certain branches/tags.  
- tags: Match jobs with Runners that have required environments.  
- artifacts: Save job outputs (logs, coverage reports, build binaries).

## 5. Hands-On: Your First Pipeline

1. Create a file `.gitlab-ci.yml` in project root.  
2. Add:

stages:  
 - build  
 - test  
  
build-job:  
 stage: build  
 script:  
 - echo "Compiling code..."  
  
test-job:  
 stage: test  
 script:  
 - echo "Running unit tests..."

3. Commit & push → GitLab auto-triggers pipeline.

# Part 2 – Managing Variables and Secrets

## 1. Why Variables & Secrets?

- Variables: Store non-sensitive config (paths, URLs, build flags).  
- Secrets: Handle sensitive info (passwords, API keys, tokens).  
  
Risks of exposed secrets:  
- Data breaches  
- Unauthorized access  
- Compliance violations  
- Customer trust damage

## 2. Types of Variables

1. Standard Variables – Non-sensitive config like paths, flags, URLs.  
2. Secret Variables – Sensitive info like API keys, DB passwords.

Example:

variables:  
 APP\_ENV: "dev"  
 DB\_USER: "appuser"  
 DB\_PASS: $DB\_PASSWORD # pulled from GitLab CI/CD variables

## 3. Protected Variables

- Accessible only in pipelines running on protected branches/tags.  
- Ideal for production secrets (e.g., AWS\_PROD\_KEY).  
- Prevents leaks from feature branches.

## 4. Masked Variables

- Hide values in logs → shown as [MASKED].  
- Must be single-line, ≥8 chars, limited charset.  
- Not 100% secure (values still accessible by job).  
- Use external secret managers for high-security needs.

## 5. Project vs Group Variables

- Project-level: Available only within one repo.  
- Group-level: Inherited across projects under a group.  
  
Use Case:  
- Group-level: Shared DB host, cloud credentials.  
- Project-level: Specific DB password for that app.

## 6. Best Practices for Variables & Secrets

- Automate secret management (Vault, AWS Secrets Manager, Azure Key Vault).  
- Rotate secrets every 30–90 days.  
- Apply least privilege principle.  
- Combine protection + masking.  
- Regular audits of variables/secrets.

## 7. Example: Using Secrets in GitLab Pipeline

stages:  
 - deploy  
  
deploy-job:  
 stage: deploy  
 script:  
 - echo "Deploying with API key..."  
 - curl -H "Authorization: Bearer $DEPLOY\_API\_KEY" https://myapp.com/deploy

Here `$DEPLOY\_API\_KEY` is a masked & protected variable stored in GitLab → not hardcoded in repo.

# Key Takeaways

- Pipelines (`.gitlab-ci.yml`) define automation flow: stages → jobs → scripts.  
- Variables & secrets enable config management without exposing sensitive info.  
- Use protected & masked variables with secret managers for production safety.  
- Following best practices ensures secure, maintainable, and efficient pipelines.