AKADEMI
CLOUD CONNECT

KURSUS TREK KEPAKARAN: LATIHAN PROSES MODEN

- Hari Kedua

Dibawakan kepada anda oleh:

cloud connect

Pautan Latihan

https://code.cloud-connect.asia/Hanafiah/proses-moden

AKADEMI CLOUD CONNECT

cloud connect

Agenda – Hari 02

HARI KEDUA	
Masa / Sesi	Topik
Sesi 1	Ringkasan Latihan Hari Pertama
Latihan Proses Moden	
Sesi 2	• Pengenalan kepada Git
Sesi 3	• Latihan 3: Pemasangan & Konfigurasi Git
Sesi 4	• Pengenalan kepada code repository
Sesi 5	• Latihan 4: Penggunaan Git & Code Repository
Sesi 6	• Git Flow & Branching
Sesi 7	• Latihan 5: Git Flow & Branching
Sesi 8	• Pengenalan kepada CI (Continuous Integration)
Sesi 9	• Latihan 6: Stress Test Laman Web
Sesi 10	• Latihan 7: Stress Test Laman Web
Sesi 11	• Latihan 8: Pengujian Keselamatan

Ringkasan Hari 01

HARI PERTAMA	
Masa / Sesi	Topik
Sesi 1	Pengenalan Latihan
Sesi 2	• Ringkasan kepada pengenalan Cloud Computing, MyDigital, 4IR & Strategi Cloud-First
Latihan Proses Moden	
Sesi 3	• Pengenalan kepada DevOps, Proses Agile, Git, CI / CD & Continuous Testing
Sesi 4	• Pengenalan kepada GitLab
Sesi 5	• Latihan 1: Pendaftaran dan Konfigurasi GitLab
Sesi 6	• Agile Project Management menggunakan GitLab ○ GitLab Projects, Boards, Issues, Milestones
Sesi 7	• Latihan 2: Agile Project Management menggunakan GitLab
Sesi 8	• Pengenalan kepada Git

Apa itu *Git*?

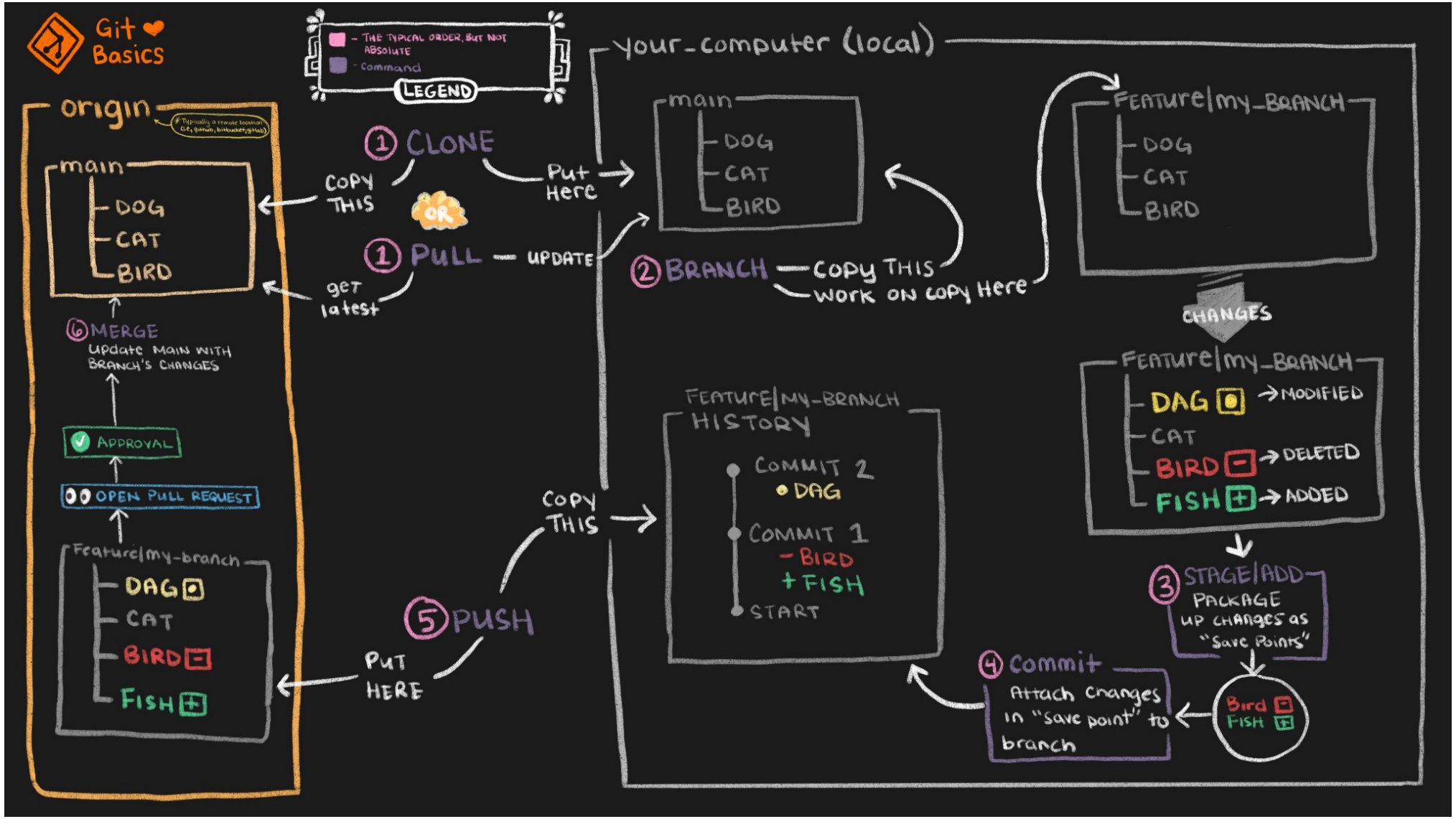
- *Git - is a **distributed version-control system** for tracking changes in source code during software development*
- *Code Repository – is an archive with the code as well as the hosting facility for these software archives, where you can also have the project's technical documentation, web pages, snippets, patches, etc. which can be accessed publicly (open-source) or privately.*

Bitbucket GitHub GitLab Azure DevOps

Kelebihan *Git*?

- *Keeps your* ***code safe***
- *Offers* ***version control*** *option to make sure all changes that were done to your code are tracked, and you know who did something to your code. You can also revert back to the previous version of the code before "Everything went bad."*
- ***Simplifies the process*** *of unifying changes from developers' collaboration*
- *Provides and promotes teamwork principles since several developers can work together on the same projects, modules, and even code lines*
- *Prepares your code for* ***release*** *to production*
- *Keep the* ***statistics*** *and* ***analytics*** *of the changes in the code*

Proses Git


Git ❤ Basics
- THE TYPICAL ORDER, BUT NOT ABSOLUTE
- Command
LEGEND
origin
Typically a remote location (Git, github, bitbucket, gitlab)
main
DOG
CAT
BIRD
1 CLONE
COPY THIS
Put Here
OR
1 PULL — UPDATE
get latest
your-computer (local)
main
DOG
CAT
BIRD
2 BRANCH — COPY THIS
work on copy Here
feature/my-BRANCH
DOG
CAT
BIRD
CHANGES
feature/my-BRANCH
DAG → MODIFIED
CAT
BIRD → DELETED
FISH → ADDED
3 STAGE/ADD
PACKAGE UP CHANGES as "Save Points"
Bird
FISH
4 Commit
Attach changes in "save point" to branch
FEATURE/MY-BRANCH HISTORY
COMMIT 2
• DAG
COMMIT 1
- BIRD
+ FISH
START
COPY THIS
5 PUSH
PUT HERE
6 MERGE
Update MAIN WITH BRANCH'S CHANGES
APPROVAL
OPEN PULL REQUEST
feature/my-branch
DAG
CAT
BIRD
FISH

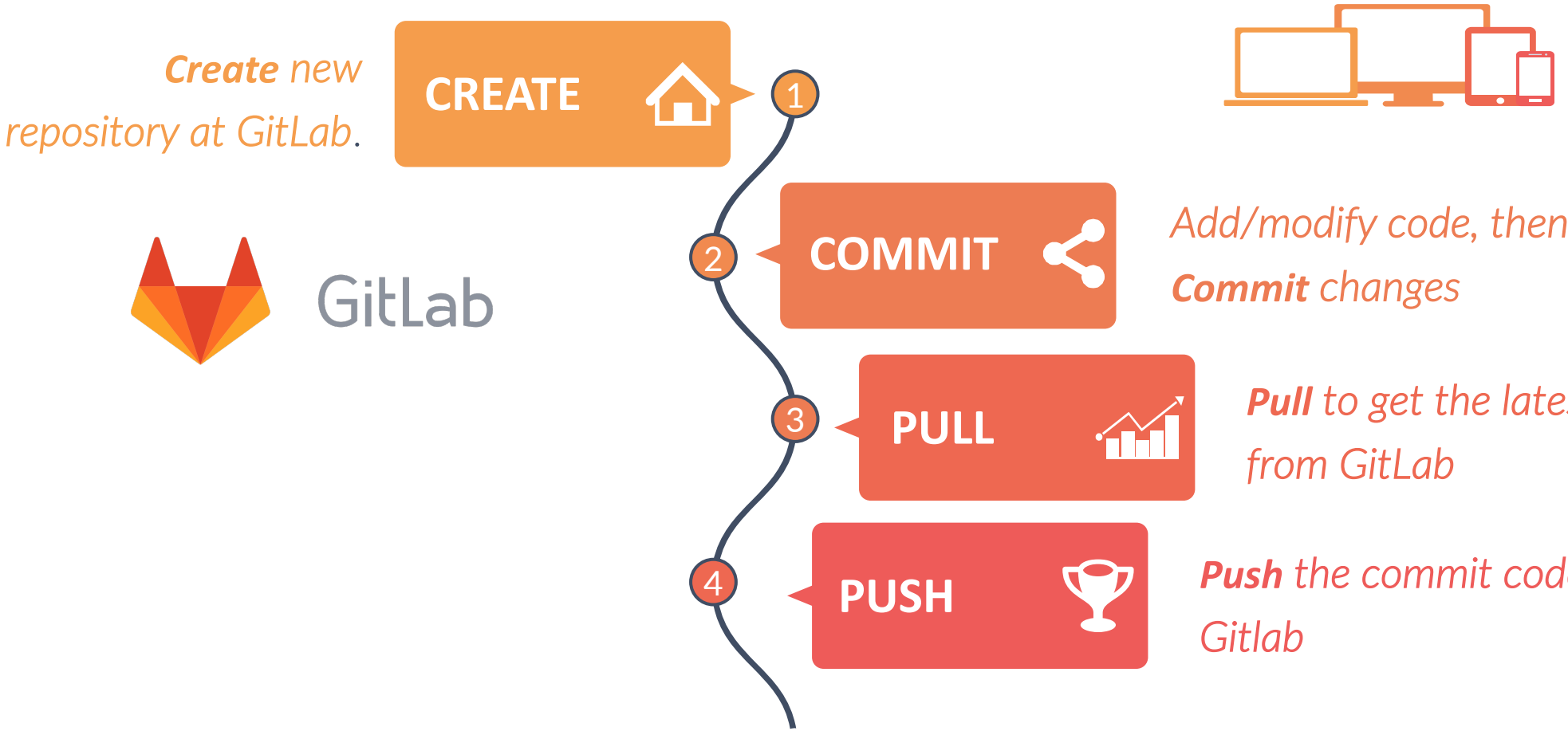
Git *Commands*

- Setup & Config
  - git --version
  - git --help
  - git config
- Getting and Creating Projects
  - git init
  - git clone
  - git add
  - git commit
  - git status
- Branching and Merging
  - git branch
  - git checkout
  - git merge
  - git log
- Sharing and Updating Projects
  - git fetch
  - git pull
  - git push
  - git remote

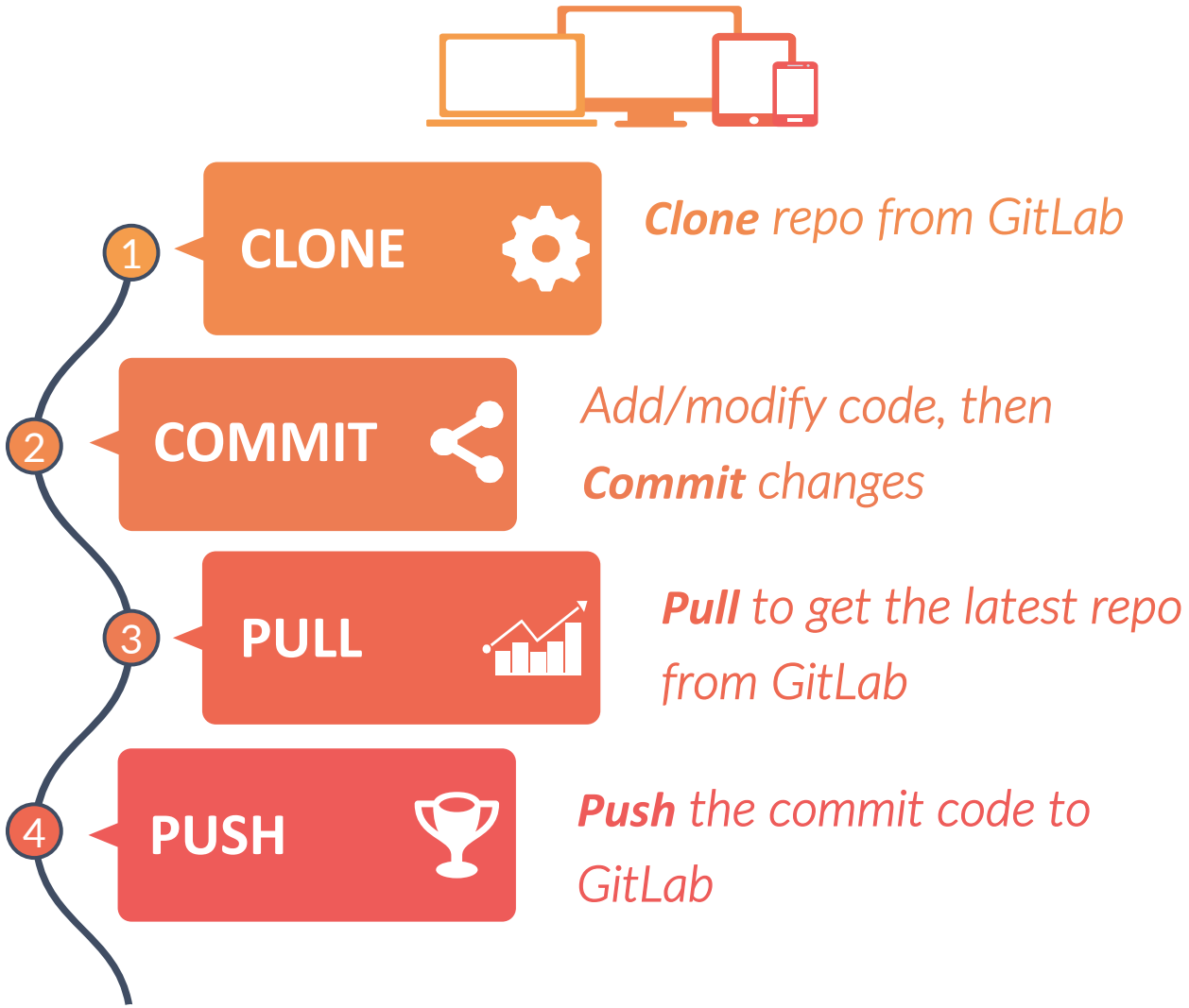
{ Latihan 3: Pemasangan & Konfigurasi Git }

{ Pengenalan kepada *code repository* }

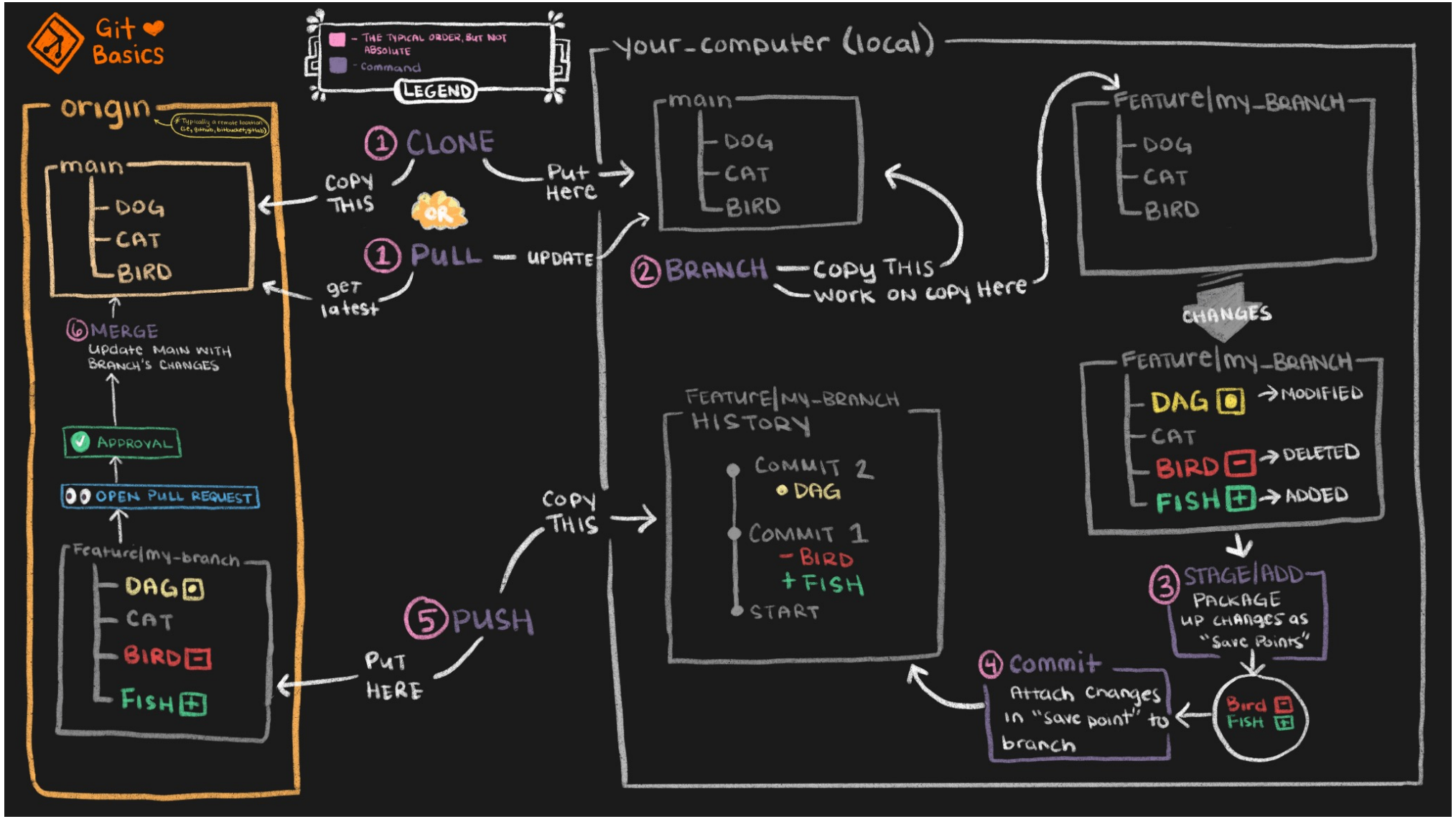
Proses Git – Projek Baru



Proses Git - *Clone*



Proses Git


Git ❤ Basics
- THE TYPICAL ORDER, BUT NOT ABSOLUTE
- Command
LEGEND
origin
Typically a remote location (Git, github, bitbucket, gitlab)
main
DOG
CAT
BIRD
1 CLONE
COPY THIS
Put Here
OR
1 PULL — UPDATE
get latest
your-computer (local)
main
DOG
CAT
BIRD
2 BRANCH — COPY THIS
work on copy Here
feature/my-BRANCH
DOG
CAT
BIRD
CHANGES
feature/my-BRANCH
DAG → MODIFIED
CAT
BIRD → DELETED
FISH → ADDED
3 STAGE/ADD
PACKAGE UP CHANGES as "Save Points"
Bird
FISH
4 Commit
Attach changes in "save point" to branch
FEATURE/MY-BRANCH HISTORY
COMMIT 2
• DAG
COMMIT 1
- BIRD
+ FISH
START
COPY THIS
5 PUSH
PUT HERE
6 MERGE
Update MAIN WITH BRANCH'S CHANGES
APPROVAL
OPEN PULL REQUEST
feature/my-branch
DAG
CAT
BIRD
FISH

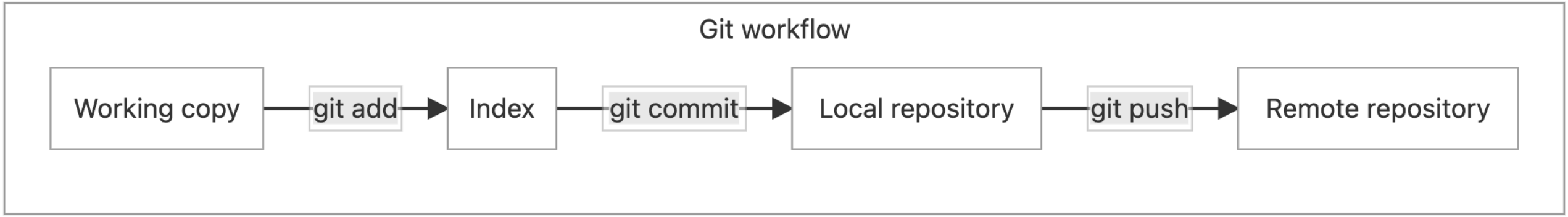
{ Latihan 4: Penggunaan Git & *Code Repository* }

{ Git Branch Management }

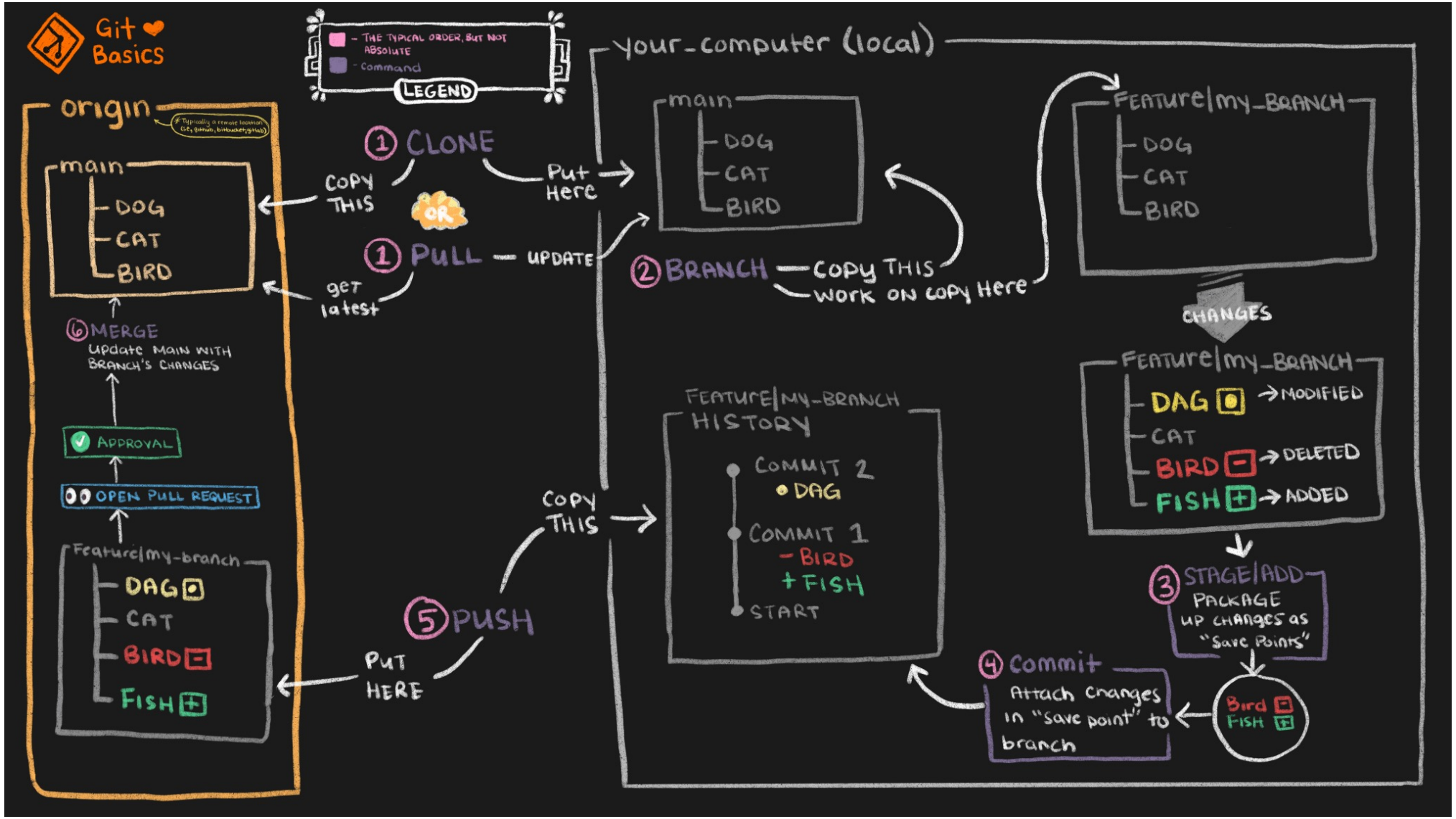
Git *Branching*

- *Branching means you diverge from the main line of development and continue to do work without messing with that main line.*
- *Git branches are effectively a pointer to a snapshot of your changes.*
- *A branch is essentially is a unique set of code changes with a unique name.*
- *The main branch — the one where all changes eventually get merged back into, and is called master.*

Git Workflow



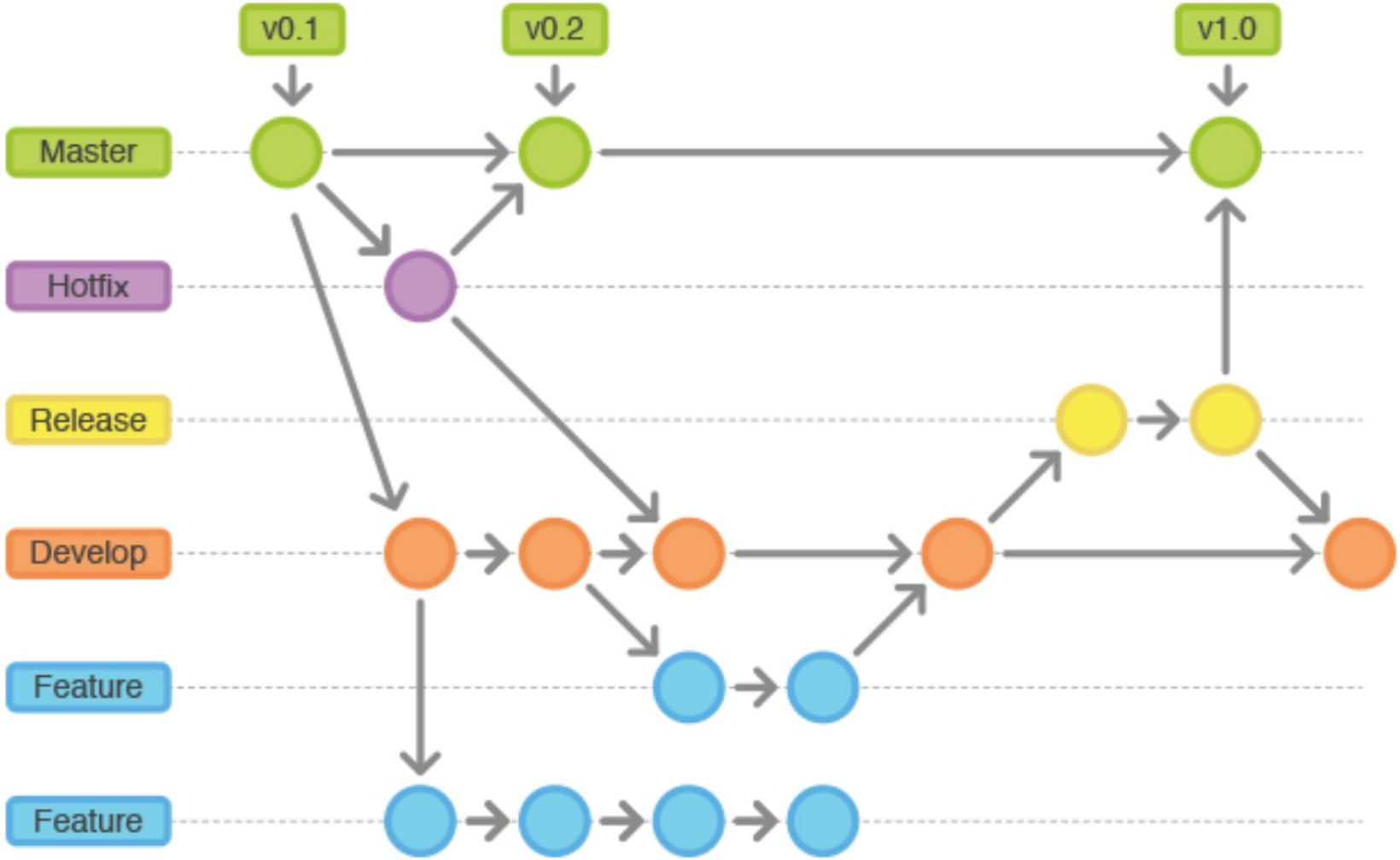
Proses Git


Git ❤ Basics
- THE TYPICAL ORDER, BUT NOT ABSOLUTE
- Command
LEGEND
origin
Typically a remote location (Git, github, bitbucket, gitlab)
main
DOG
CAT
BIRD
1 CLONE
COPY THIS
Put Here
OR
1 PULL — UPDATE
get latest
your-computer (local)
main
DOG
CAT
BIRD
2 BRANCH — COPY THIS
work on copy Here
feature/my-BRANCH
DOG
CAT
BIRD
CHANGES
feature/my-BRANCH
DAG → MODIFIED
CAT
BIRD → DELETED
FISH → ADDED
3 STAGE/ADD
PACKAGE UP CHANGES as "Save Points"
Bird
FISH
4 Commit
Attach changes in "save point" to branch
FEATURE/MY-BRANCH HISTORY
COMMIT 2
• DAG
COMMIT 1
- BIRD
+ FISH
START
COPY THIS
5 PUSH
PUT HERE
6 MERGE
Update MAIN WITH BRANCH'S CHANGES
APPROVAL
OPEN PULL REQUEST
feature/my-branch
DAG
CAT
BIRD
FISH

Jenis-jenis *Git workflow*

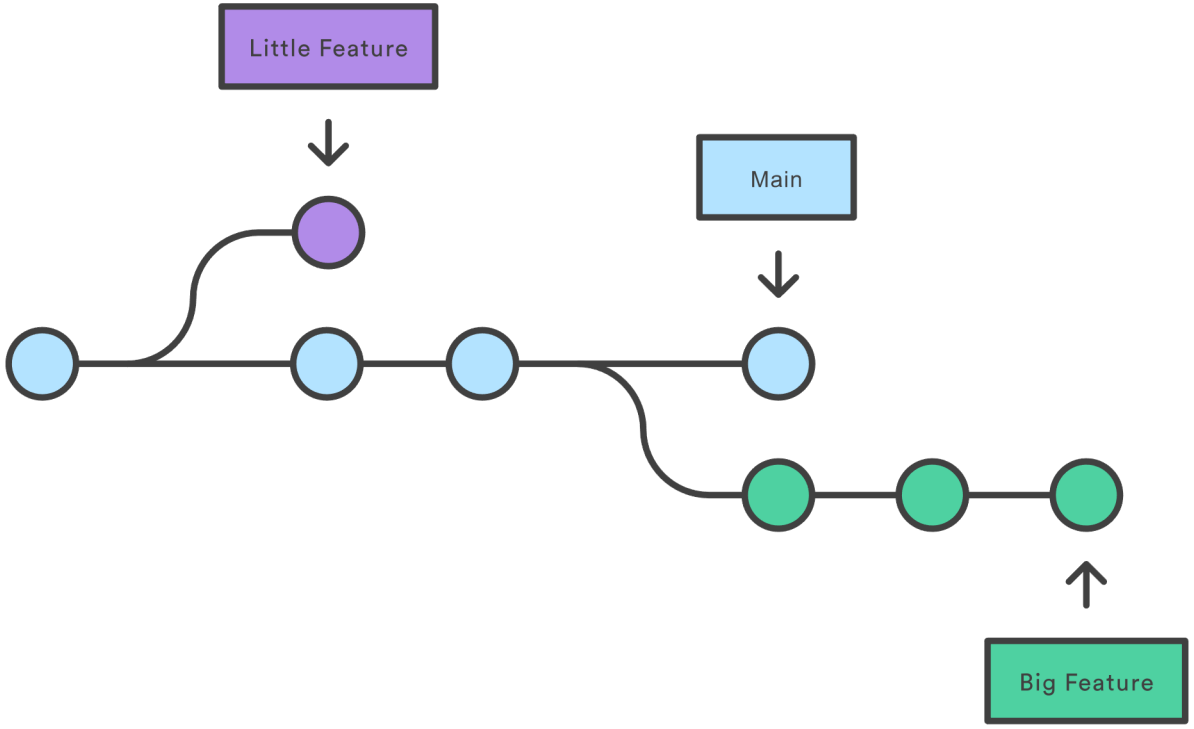
- *Git Flow*
- *GitHub Flow*
- *GitLab Flow*

Git Flow



GitHub Flow

- GitHub flow is a lightweight, branch-based workflow.

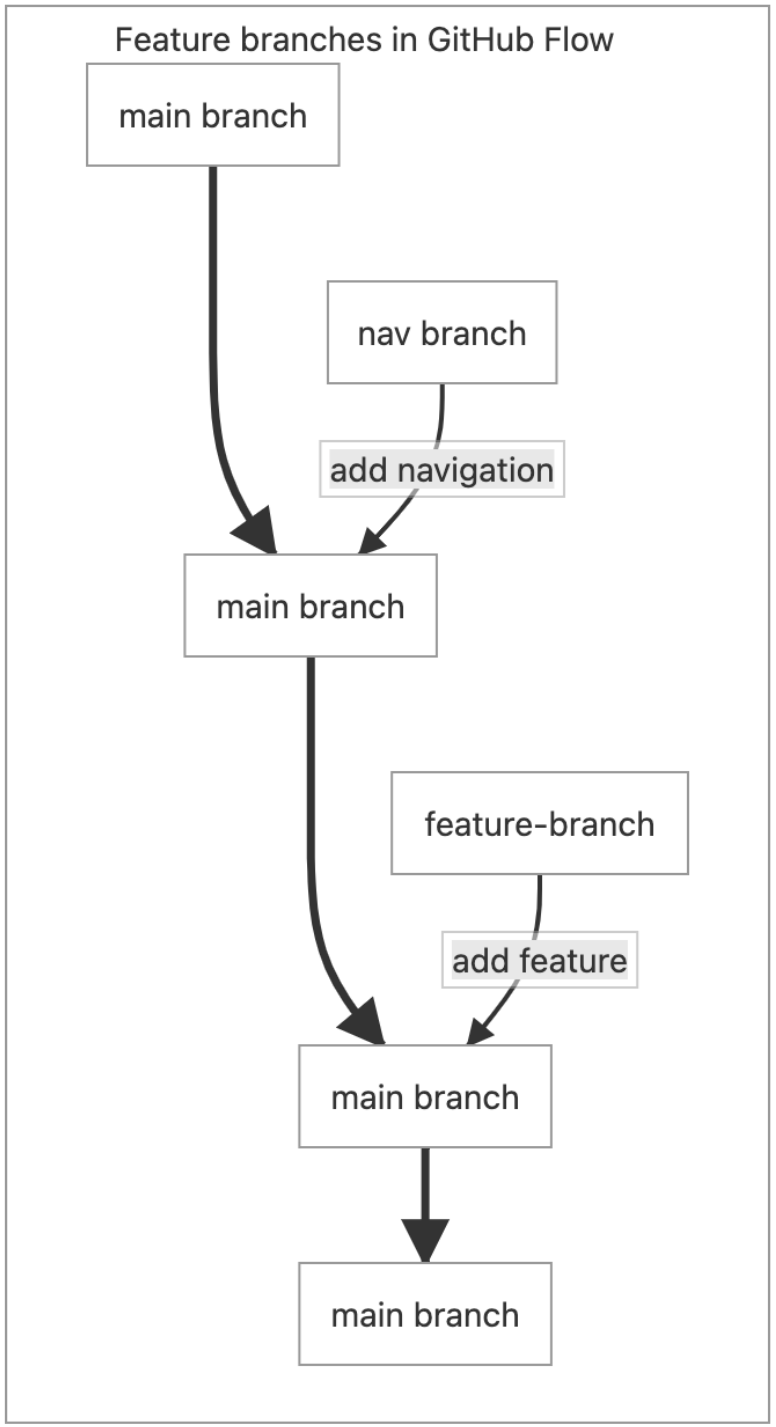


AKADEMI
CLOUD CONNECT

cloud connect

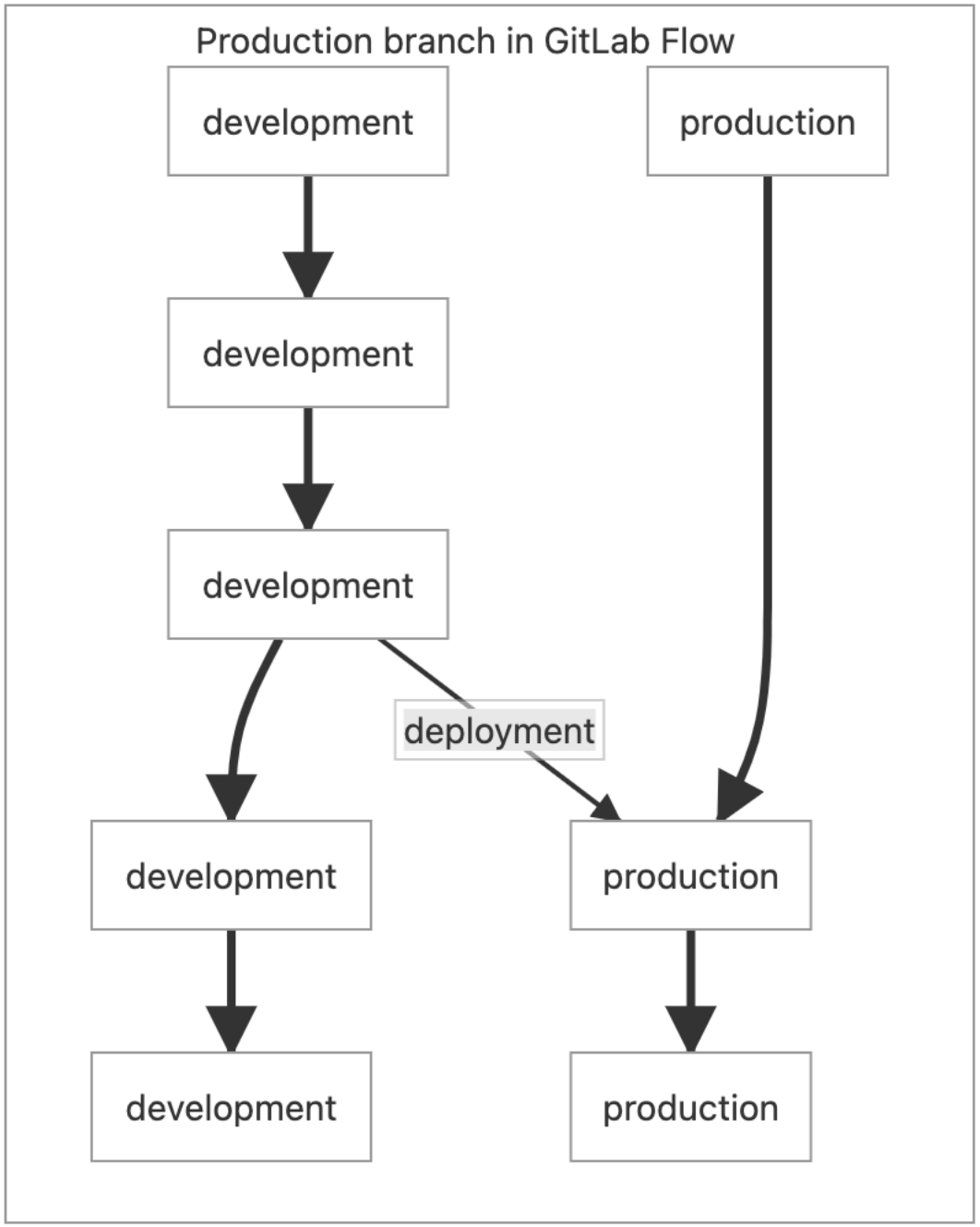
Main

Feature



GitLab Flow

- *GitHub flow assumes you can deploy to production every time you merge a feature branch*
- *there are some cases where this is not possible, such as:*
  - *You don't control the timing of a release. For example, an iOS application that is released when it passes App Store validation.*
  - *You have deployment windows - for example, workdays from 10 AM to 4 PM when the operations team is at full capacity - but you also merge code at other times.*
- *make a production branch that reflects the deployed code.*
- *You can deploy a new version by merging development into the production branch*


Production branch in GitLab Flow
development
production
development
development
deployment
development
production
development
production

{ Latihan 5: *Git Flow & Branching* }

{ Pengenalan kepada CI *(Continuous Integration)* }

What is CI/CD?

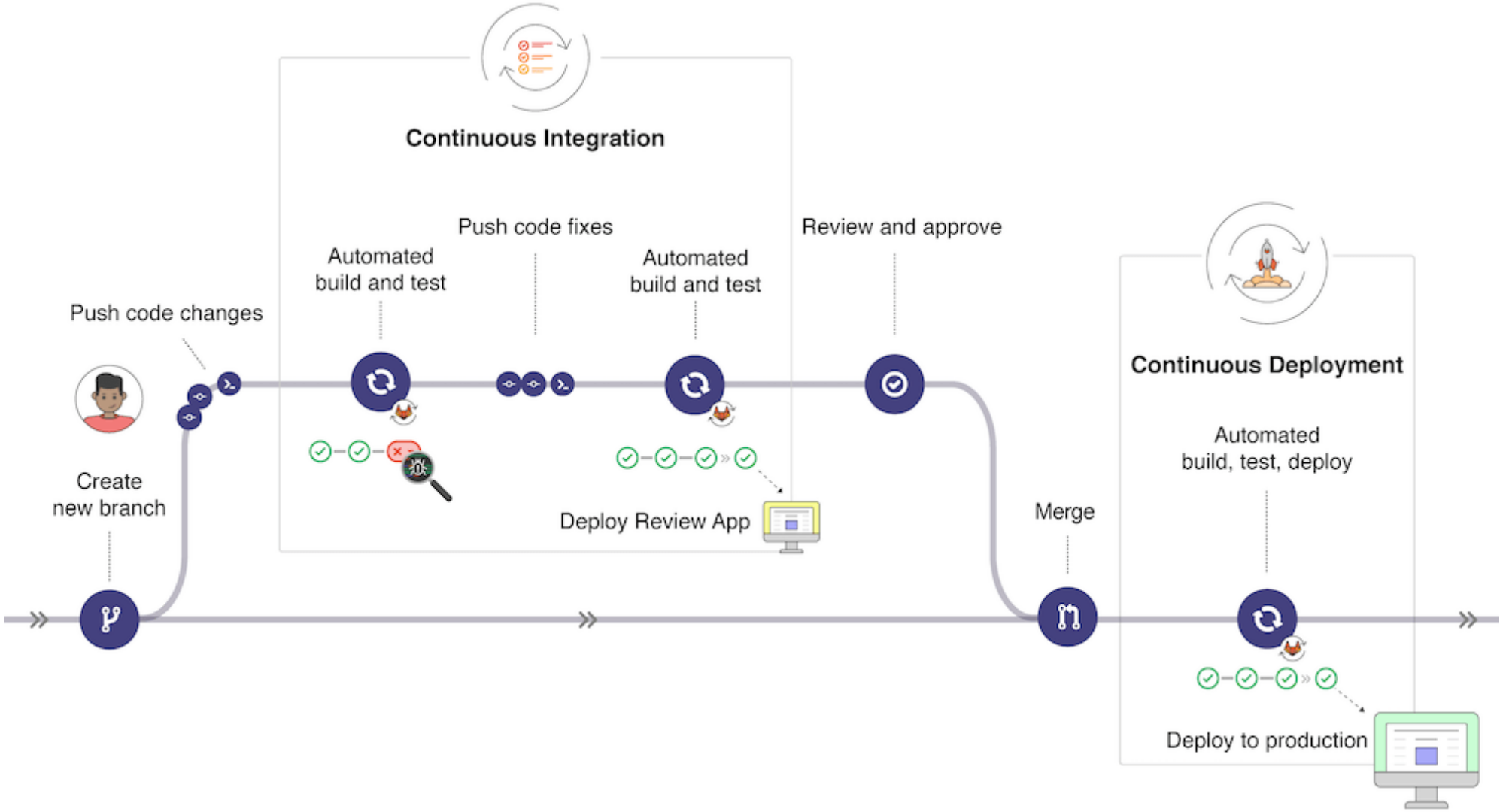
- In software engineering, CI/CD or CICD is the combined practices of continuous integration and either continuous delivery or continuous deployment.
- CI/CD bridges the gaps between development and operation activities and teams by enforcing automation in building, testing and deployment of applications.
- Modern day DevOps practices involve continuous development, continuous testing, continuous integration, continuous deployment and continuous monitoring of software applications throughout its agile development life cycle.
- The CI/CD practice, or CI/CD pipeline, forms the backbone of modern day DevOps operations.

- Wikipedia

Continuous integration (CI)

- Software development practice in which small adjustments to the underlying code in an application are tested every time a team member makes changes.
- The practice to build & test the application

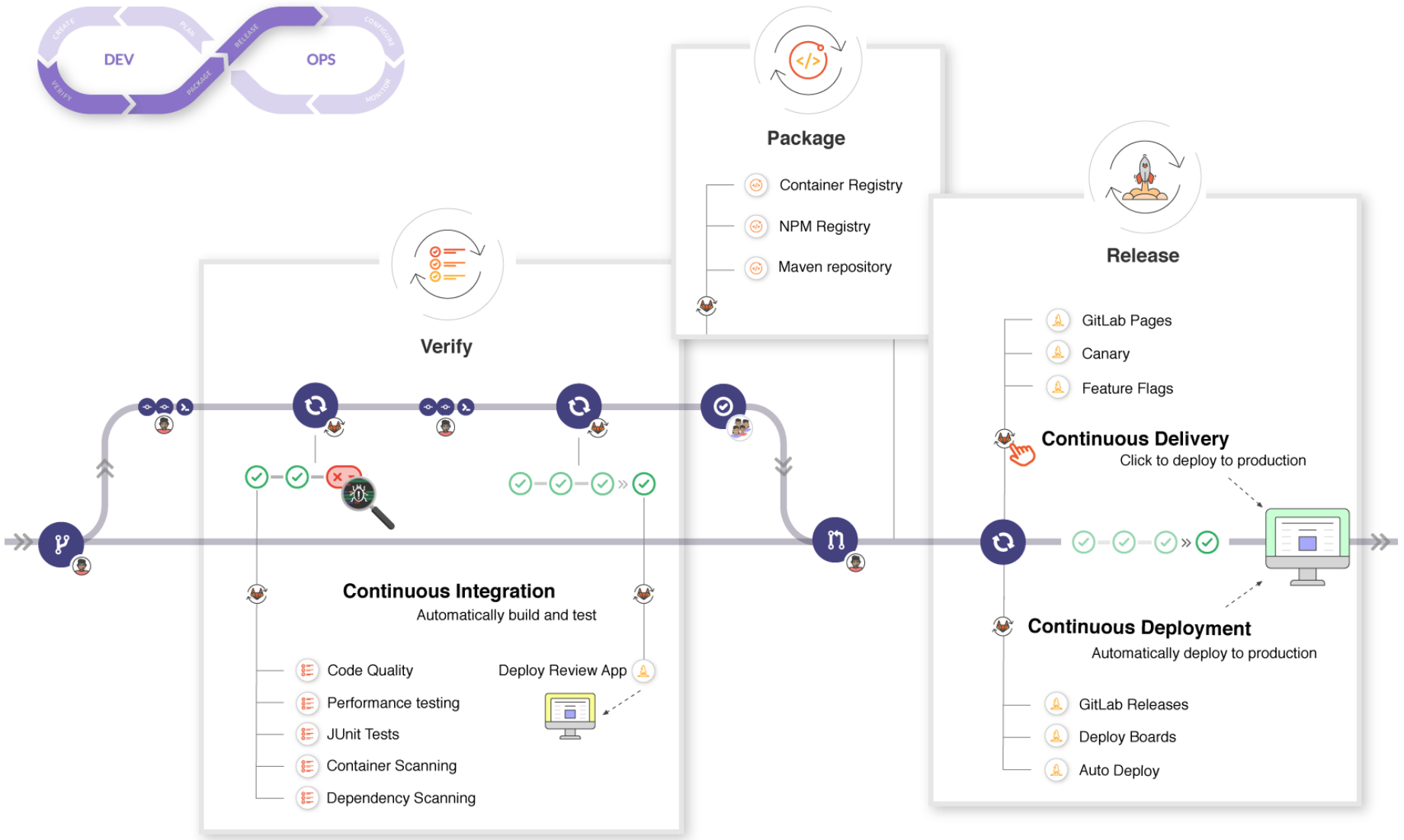
GitLab CI/CD


Continuous Integration
Push code fixes
Automated
build and test
Automated
build and test
Review and approve
Push code changes
Create
new branch
Deploy Review App
Merge
Continuous Deployment
Automated
build, test, deploy
Deploy to production

AKADEMI
CLOUD CONNECT

cloud connect

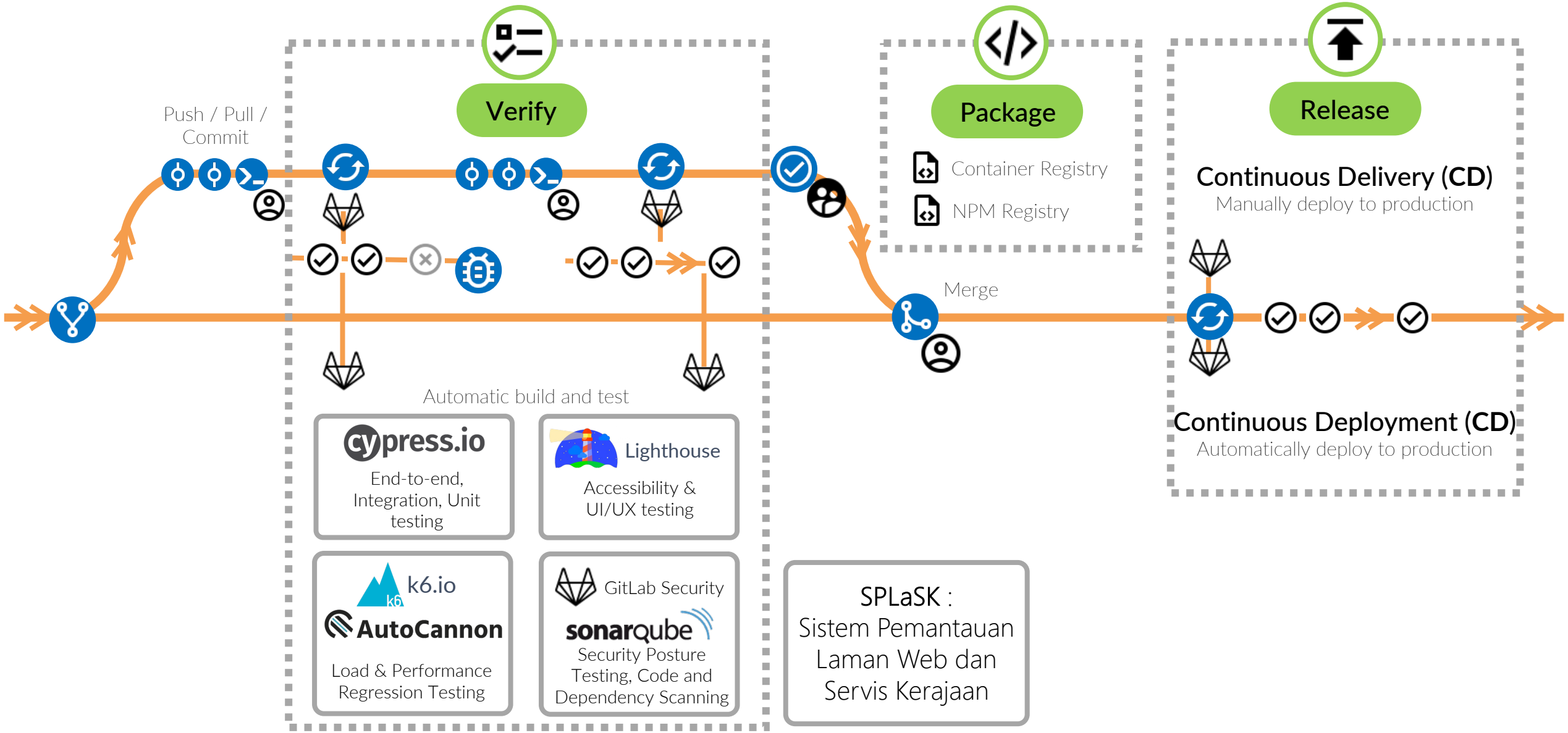
GitLab CI/CD



AKADEMI CLOUD CONNECT

cloud connect

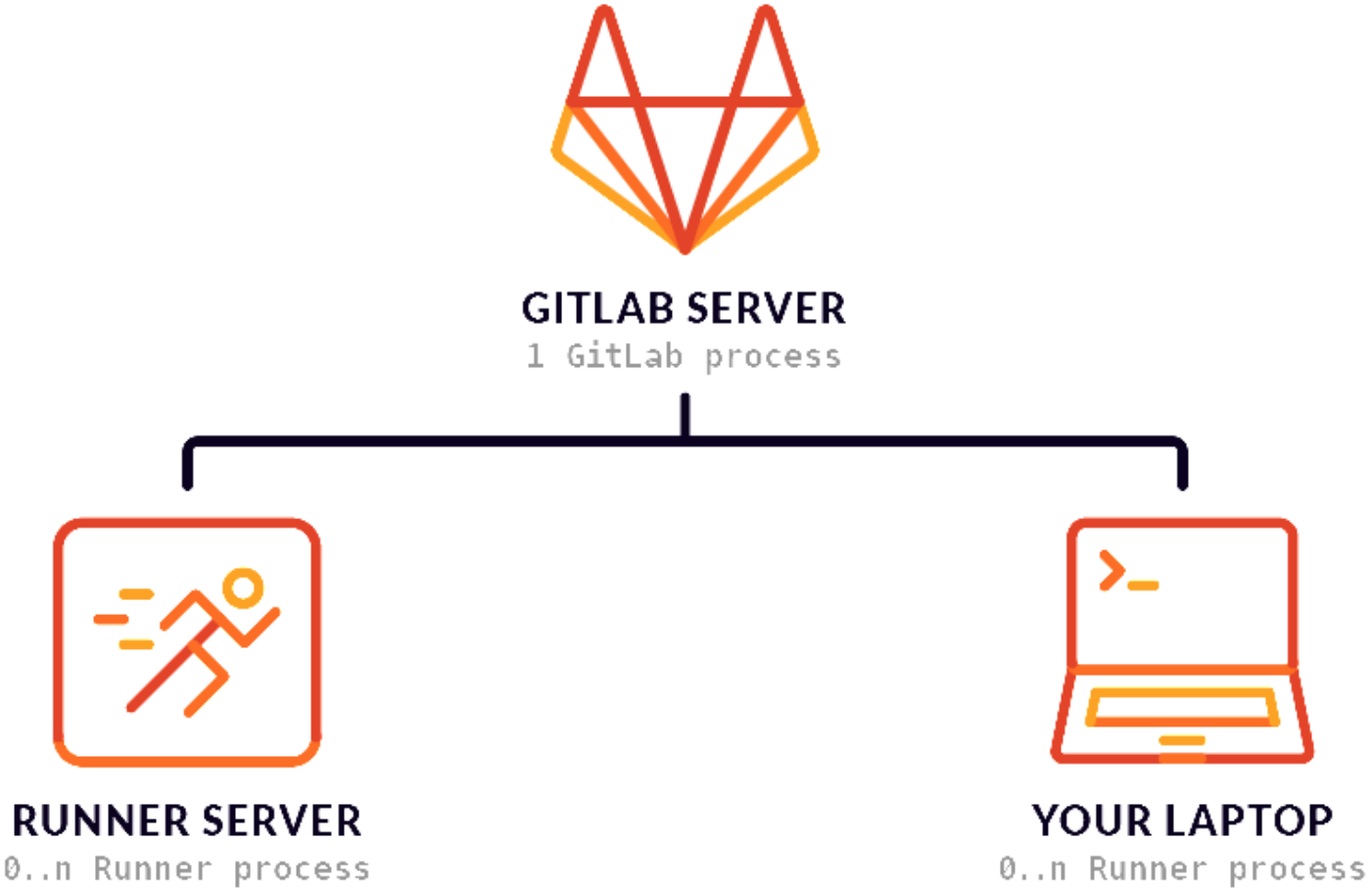
Modern Process using GitLab



GitLab Ci/CD

- To use GitLab CI/CD:
  1. Runners available to run your jobs.
     - GitLab SaaS provides runners
     - Self-hosted GitLab – you need to install runner
  2. Create a .gitlab-ci.yml file at the root of your repository. This file is where you define your CI/CD jobs.

GitLab Runners



.gitlab-ci.yml file

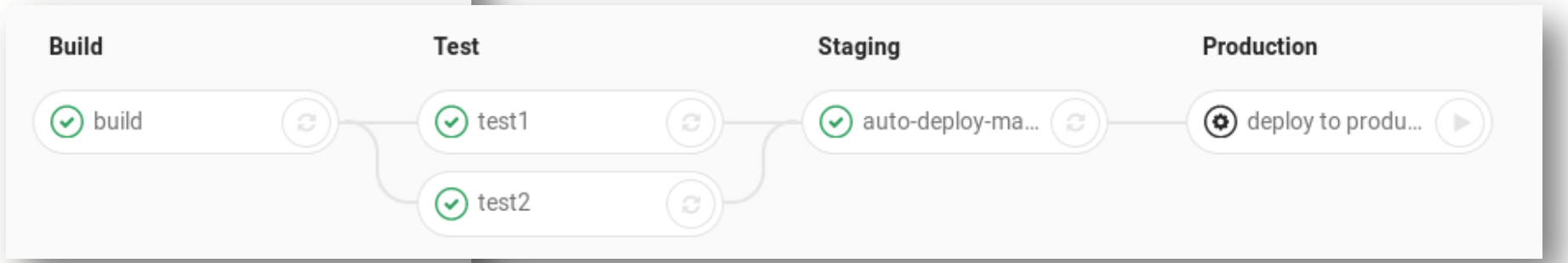
- is a **YAML** file where you configure specific instructions for GitLab CI/CD:
  - The structure and order of jobs that the runner should execute.
  - The decisions the runner should make when specific conditions are encountered

CI/CD Pipelines

- Pipelines are the top-level component of continuous integration, delivery, and deployment.
- Pipelines comprise:
  - Jobs (executed by runners), which define what to do. For example, jobs that compile or test code.
  - Stages, which define when to run the jobs.
    - A **build** stage, with a job called compile.
    - A **test** stage, with two jobs called test1 and test2.
    - A **staging** stage, with a job called deploy-to-stage.
    - A **production** stage, with a job called deploy-to-prod.

CI/CD Pipelines

```yaml
stages:
- build
- test
- stage
- deploy
build-job:
  stage: build
  script:
  - echo "Compiling the code..."
  - echo "Compile complete."
unit-test-job:
  stage: test
  script:
  - echo "Running unit tests... This will take about 60 seconds."
  - sleep 60
  - echo "Code coverage is 90%"
stage-job:
  stage: stage
  script:
  - echo "Staging the code ..."
  - echo "Staging complete."
deploy-job:
  stage: deploy
  script:
  - echo "Deploying application..."
  - echo "Application successfully deployed."
```



{ Latihan 6: *GitLab CI/CD* }

Performance Testing with K6

- *Performance Testing is the process of analyzing the quality and capability of a product.*
- *Load testing is a subset of performance testing where we analyze the behavior of the application under normal or peak load conditions.*
- ***k6** is an open-source load testing tool for testing the performance of APIs, microservices, and websites.*

K6 Features

1

Load testing

Verify that your systems can handle the expected volume of visitors. Traffic is rarely consistent. Quickly adapt your scripts to run various types of testing: stress tests, peak tests, soak tests, and more.

2

Chaos and reliability testing

Simulate real-world traffic in your chaos experiments. Ensure that your environment is able to withstand traffic surges without outages, gracefully recovering as components fail.

3

Performance and synthetic monitoring

For ambitious applications, ping testing is not enough anymore. Reuse your tests that mimic user traffic to further monitor the availability and performance of your systems.

{ Latihan 7: *Stress Test* Laman Web }

GitLab CI - Security

- Static Application Security Testing (SAST),
- Dynamic Application Security Testing (DAST),
- Container Scanning,
- Dependency Scanning,
- Code Quality,
- Secret Scanning,
- Fuzz Testing

{ Latihan 8: *Pengujian Keselamatan* }