Continuous Deployment

SE234 Advance Software Development

CAMT
College of Arts, Media and Technology
Chiang Mai University

Continuous Integration

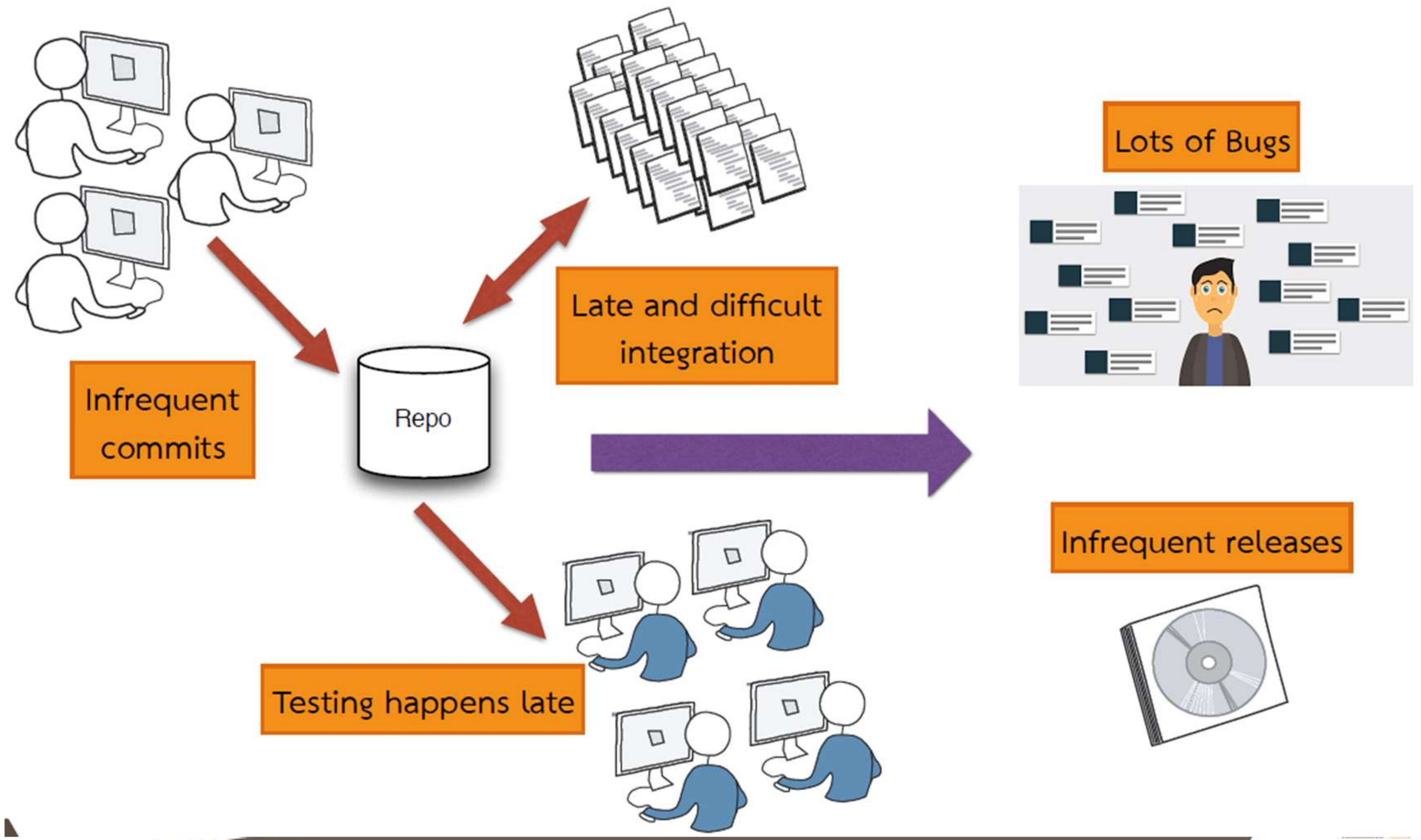
- CI is a **software development practice** where members of a team integrate their work **frequently**; usually each person integrates **at least daily** – leading to multiple integrations per day.”

  — Martin Fowler

Continuous Integration

- Developers practicing CI merge their changes back to the main branch as **early** and **often** as possible.
- The changes are validated by **creating a build** and **running automated tests** against the build.
- CI puts a great emphasis on **testing automation** to check that the application is not broken whenever new commits are integrated into the main branch.
- This will prevent the **integration hell**, i.e., usually happens when people wait for release day to merge their changes into the release branch.

Deployment without CI



Deployment without CI

- Insufficient testing
- Issue raised later are harder to fix
- Slow release process

- Integration hell
- Project delays
- Unhappy clients
- Higher maintenance costs

Lots of Bugs

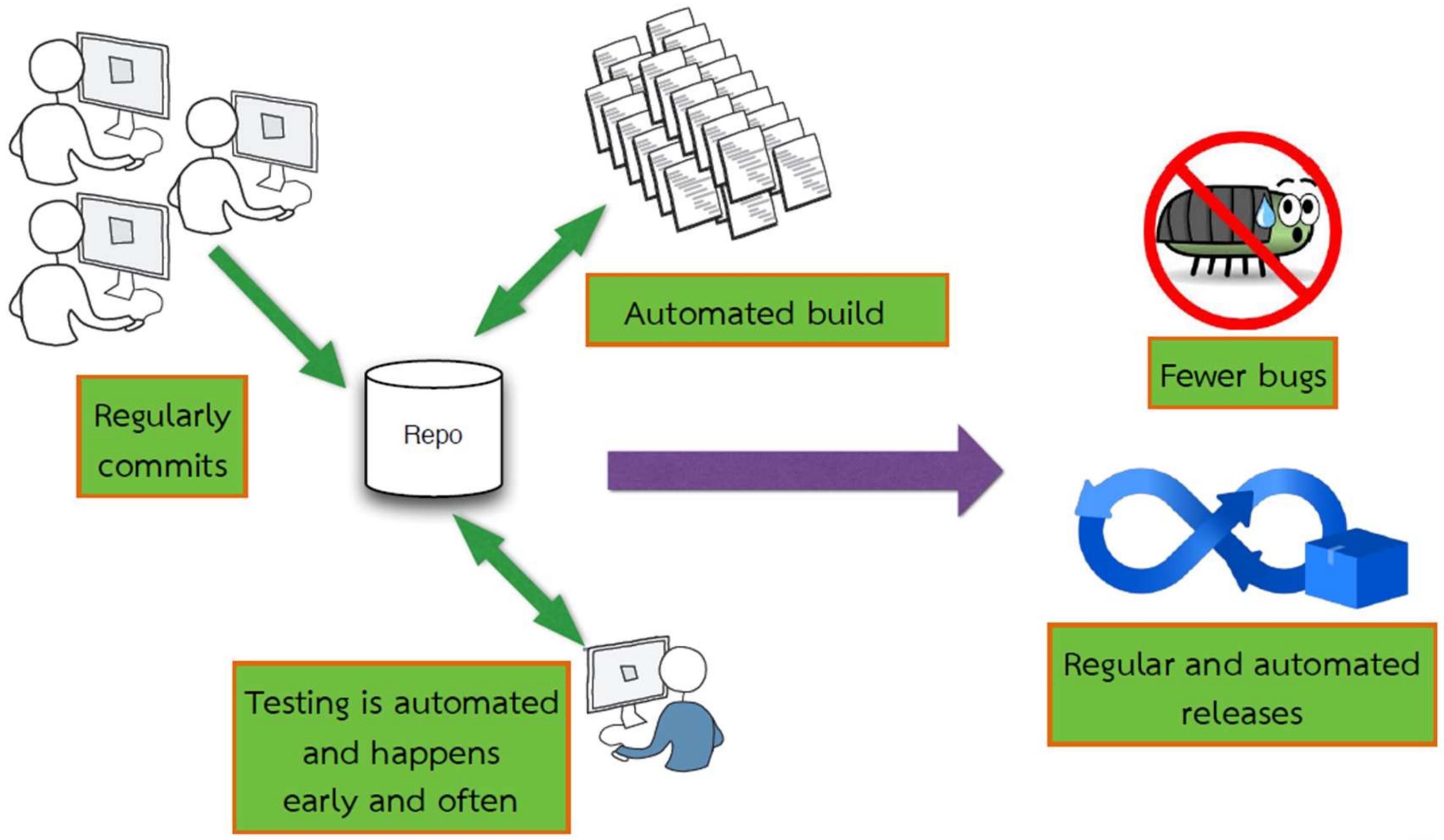
Infrequent releases

CAMT
College of Arts, Media and Technology
Chiang Mai University

Starter Kits

- A source code repository, e.g., Git
- An automated build, i.e., build scripts
- An automated testing suite
- An automated code quality measurement
- A continuous build service or server.

Development with CI


Regularly commits
Repo
Automated build
Testing is automated and happens early and often
Fewer bugs
Regular and automated releases

CAMT
College of Arts, Media and Technology
Chiang Mai University

Deployment with CI

- Immediate bug detection
- Reduce risk of cost schedule and budget
- Measurable & visible code quality
- Record of evolution of the project

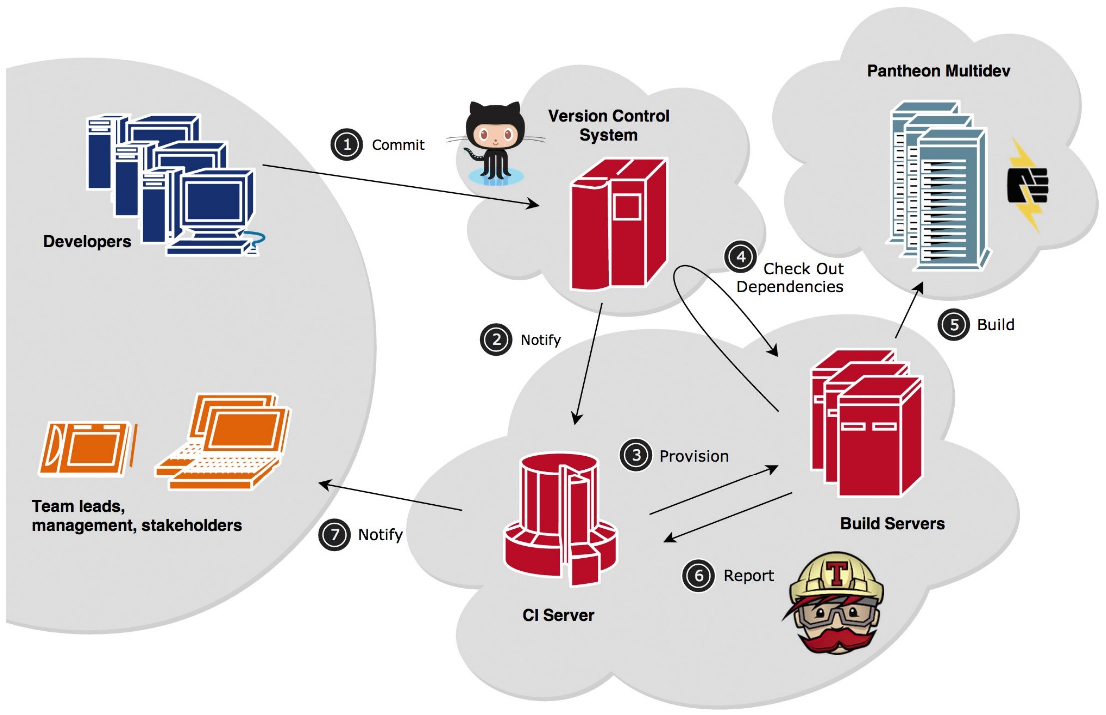
Continuous Deliver

- Extension of continuous integration
- On top of having automated your testing
  - automating your release process
  - Deploying your application at any point of time by clicking on a button
- In theory
  - Software can be release daily, weeking fortnightly, or whatevers
  - The productions should be deploy as early as possible
    - Easy to troubleshoot in case of problem

Continuous Deployment

- One step further than continuous delivery.
- Every change that passes all stages of production pipeline is released to your customers
  - Production pipeline
    - Sequence of activities that guarantee that the software contains the certains quality
- No human intervention
- Only a failed test will prevent a new change to be deployed to production
- Accelerating the feedback loop with customer

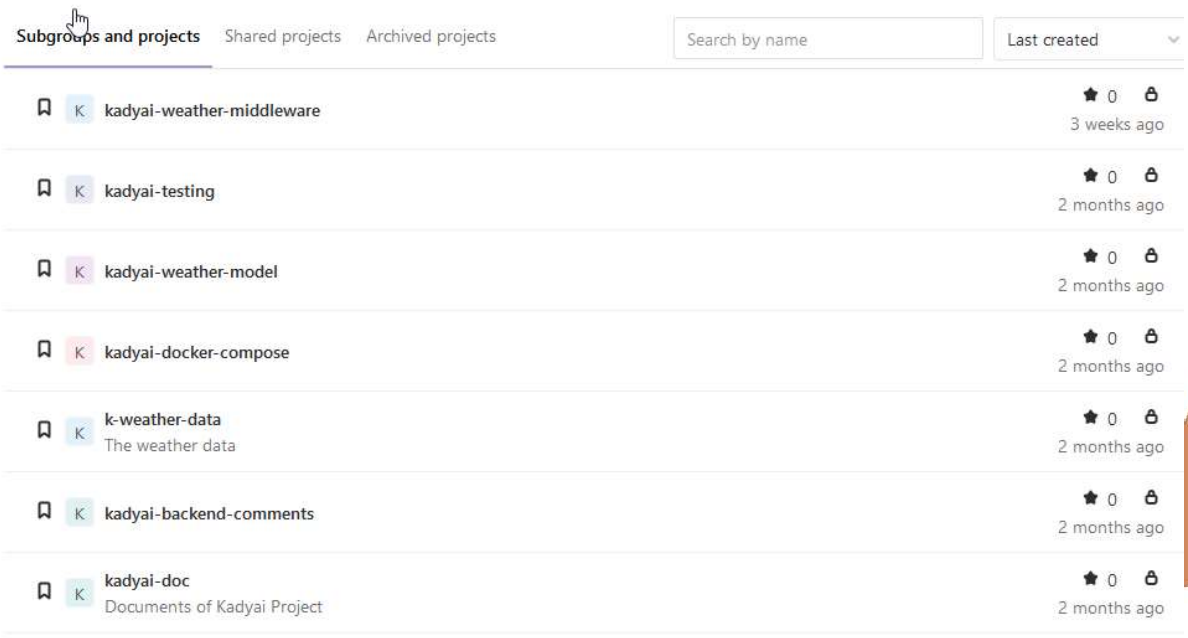
How the CD is working?



CAMT
College of Arts, Media and Technology
Chiang Mai University

Why Using CD tools?

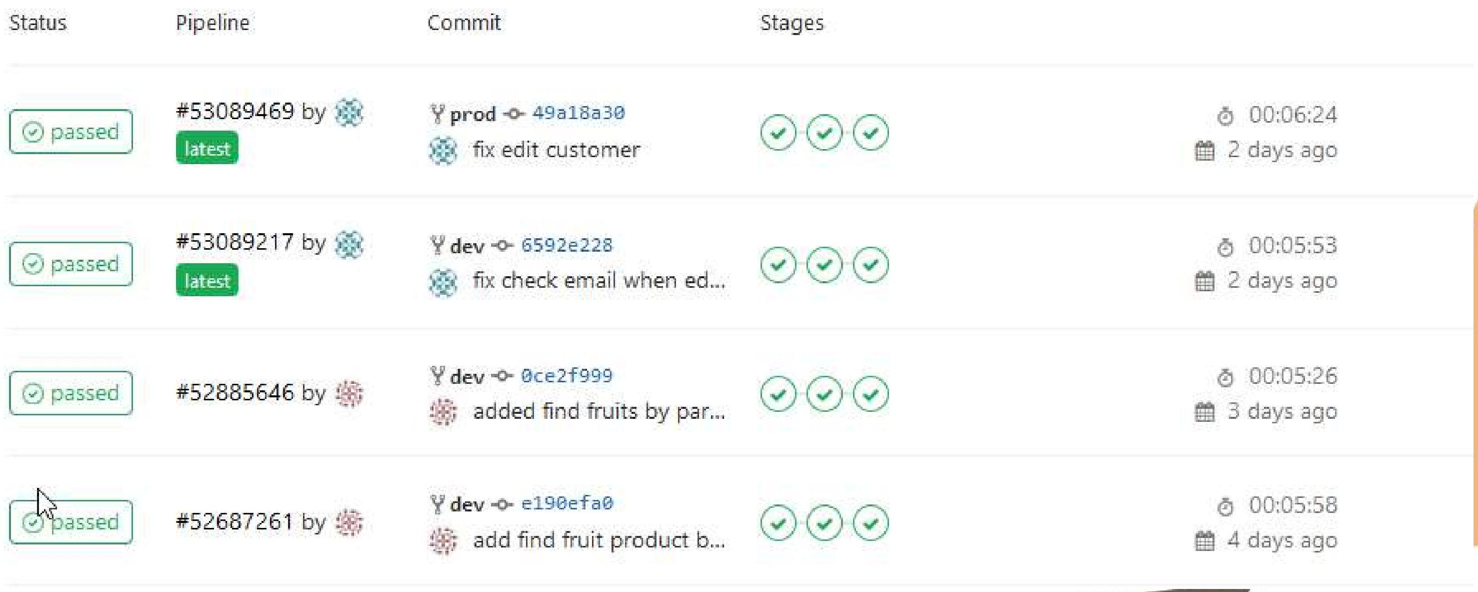
- Multiple project



K	kadyai-weather-middleware	0 · 3 weeks ago
K	kadyai-testing	0 · 2 months ago
K	kadyai-weather-model	0 · 2 months ago
K	kadyai-docker-compose	0 · 2 months ago
K	k-weather-data The weather data	0 · 2 months ago
K	kadyai-backend-comments	0 · 2 months ago
K	kadyai-doc Documents of Kadyai Project	0 · 2 months ago

Why Using CD tools?

- Build History



Status	Pipeline	Commit	Stages	
passed	#53089469 by latest	prod 49a18a30 fix edit customer	✓ ✓ ✓	00:06:24 2 days ago
passed	#53089217 by latest	dev 6592e228 fix check email when ed...	✓ ✓ ✓	00:05:53 2 days ago
passed	#52885646 by	dev 0ce2f999 added find fruits by par...	✓ ✓ ✓	00:05:26 3 days ago
passed	#52687261 by	dev e190efa0 add find fruit product b...	✓ ✓ ✓	00:05:58 4 days ago

Why Using CD tools?

- Branch management
  - Select branch to be deploy

CD Tools?

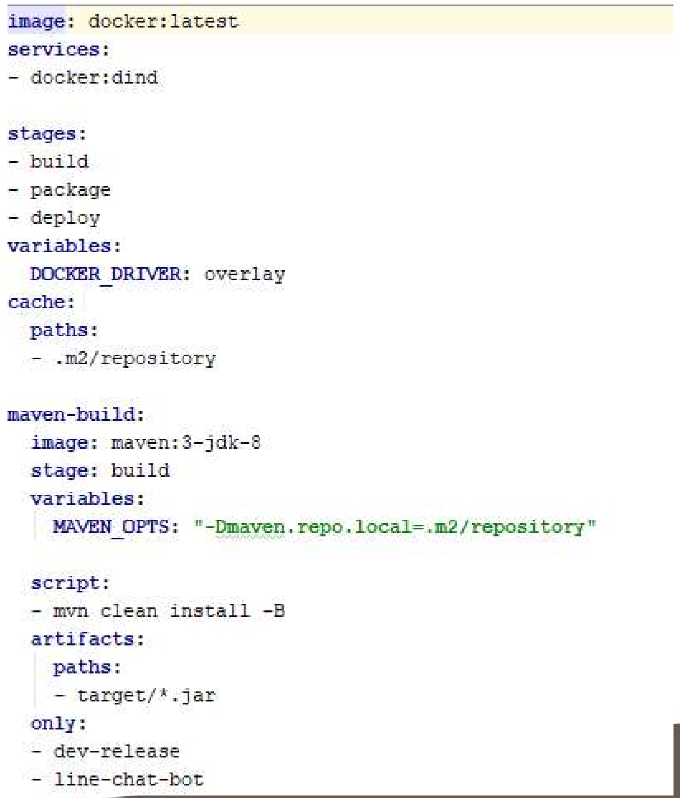
- Give the names

Gitlab-ci

- Less server-side configuration
- Free
  - For 2000 mins build time
- Simple configuration file
  - Used similar format in many
- Cons
  - Can be used with git-lab only

The configuration files

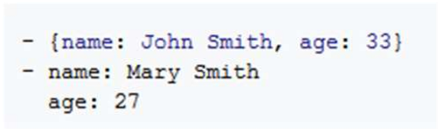
- Use the YAML format



```yaml
image: docker:latest
services:
- docker:dind

stages:
- build
- package
- deploy
variables:
  DOCKER_DRIVER: overlay
cache:
  paths:
  - .m2/repository

maven-build:
  image: maven:3-jdk-8
  stage: build
  variables:
    MAVEN_OPTS: "-Dmaven.repo.local=.m2/repository"

  script:
  - mvn clean install -B
  artifacts:
    paths:
    - target/*.jar
  only:
  - dev-release
  - line-chat-bot
```

YAML

- YAML is a human friendly data serialization standard for all programming languages



```
- {name: John Smith, age: 33}
- name: Mary Smith
  age: 27
```

YAML

- White space indention
  - Set up the structure
- - => list member
- # => comment
- --- => new documents provider
- : => the key-value notation

Data::Denter and Inline

```
md5 : cc9b569052f4daa5b343b1dcb94dd2bc
name : e_cc9b
language : C
date_compiled : Wed Jun 12 12:48:00 2002
inline_version : 0.43
ILSM : %
    module : Inline::C
    suffix : bundle
    type : compiled
Config : %
    apiversion : 5.005
    archname : darwin
    ccflags : -g -pipe -pipe -fno-common
    osname : darwin
    osvers : 1.4
    so : dylib
    version : 5.6.0
```

No tab character, space only

Influences

```
---
scripting languages:
  - Perl
  - Python
  - C
  - Java
standards:
  - RFC0822 (MAIL)
  - RFC1866 (HTML)
  - RFC2045 (MIME)
  - RFC2396 (URI)
protocols:
  - SAX
  - SOAP
  - XML
```

YAML Syntax Basics

- Mappings
- Sequences
- Streams and Documents
- Comments
- Scalars
  - Simple
  - Quoted
  - Block
  - Folded
    - Wiki
  - Escaping
- Anchors & Aliases

College of Arts, Media and Technology
Chiang Mai University

Mappings

- A YAML mapping is like a Perl hash
- Unordered Key/Value pairs
- Separated by ': ' (space is mandatory)

```
---
name: Benjamin
rank: Private
serial number: 1234567890
12:34 PM: My favorite time
```

Sequences

- A YAML sequence is like a Perl array
- An ordered collection of data
- YAML has a bullet like syntax '- '

```
---
- red
- white
- blue
- pinko
```

A YAML Grocery List

```
---
Fruits:
  - Apples
  - Tomatoes
Veggies:
  - Spinach
  - Broccoli
Meats:
  - Burgers
  - Shrimp
Household:
  - Candles
  - Incense
  - Toilet Duck
```

The Matrix

```
---
-
  - 3
  - 5
  - 7
-
  - 0
  - 0
  - 7
-
  - 9
  - 1
  - 1
```

Outline

```
- Intro
-
  Part 1:
    - Up
    - Down
    - Side to Side
- Part 2:
    - Here
    - There
    - Underwear
- Part 3:
    - The Good
    - The Bad
    - The Ingy
```

Comments

- Comments/blank lines can go almost anywhere
- Must not be ambiguous with content
- Comments begin '# ' (almost like Perl)

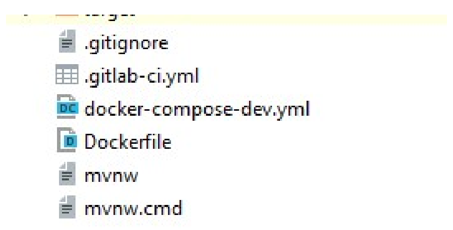
```
# comment before document
--- #DIRECTIVE # comment
foo: bar # inline comment

phone: number #555-1234
   ### Comment
fact: fiction
---
blue: bird
# Comment
```

Create your information in YAML

- Name, surname, the book you have

How to use Gitlab-CI

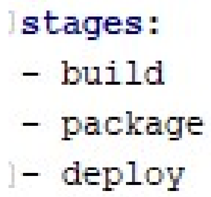
- Create file .gitlab-ci.yml in the root of your repository



```
.gitignore
.gitlab-ci.yml
docker-compose-dev.yml
Dockerfile
mvnw
mvnw.cmd
```

The Stages

- Provide blocks of operation
  - Can be used for selected branch later



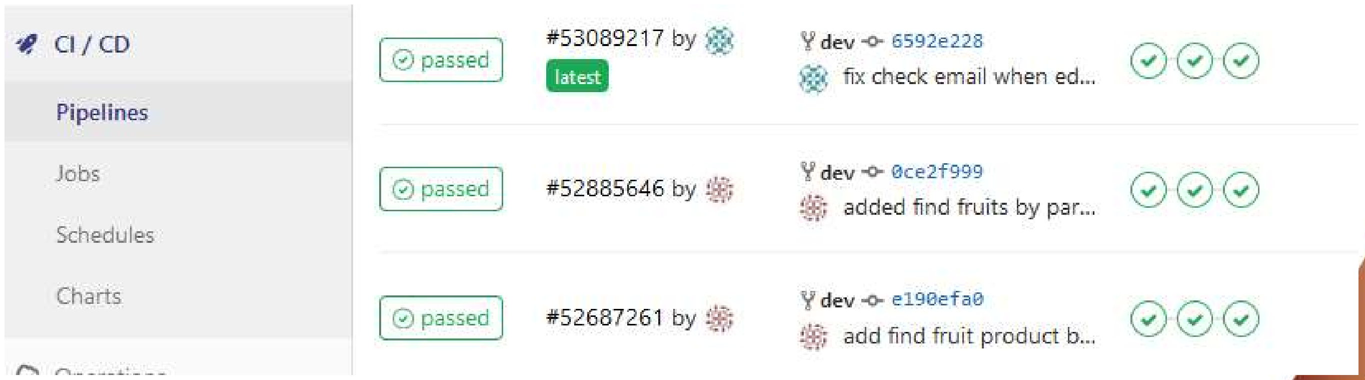
```
stages:
 - build
 - package
 - deploy
```

Stages

- Each Stages the docker is used to run the execution

The lifecycle

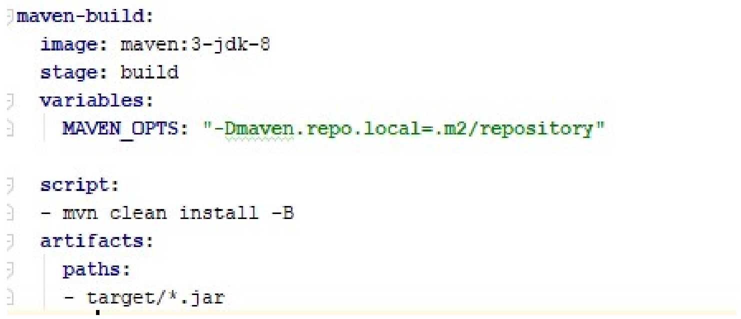
- Each jobs is run due to the stages
- All jobs run are called as pipeline



passed	#53089217 by latest	dev 6592e228 fix check email when ed...	✓ ✓ ✓
passed	#52885646 by	dev 0ce2f999 added find fruits by par...	✓ ✓ ✓
passed	#52687261 by	dev e190efa0 add find fruit product b...	✓ ✓ ✓

Life cycle

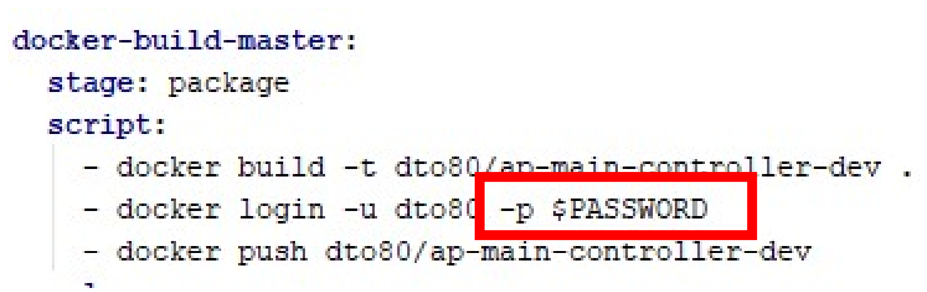
- Can use the Linux shell script files



```
maven-build:
  image: maven:3-jdk-8
  stage: build
  variables:
    MAVEN_OPTS: "-Dmaven.repo.local=.m2/repository"

  script:
  - mvn clean install -B
  artifacts:
    paths:
    - target/*.jar
```

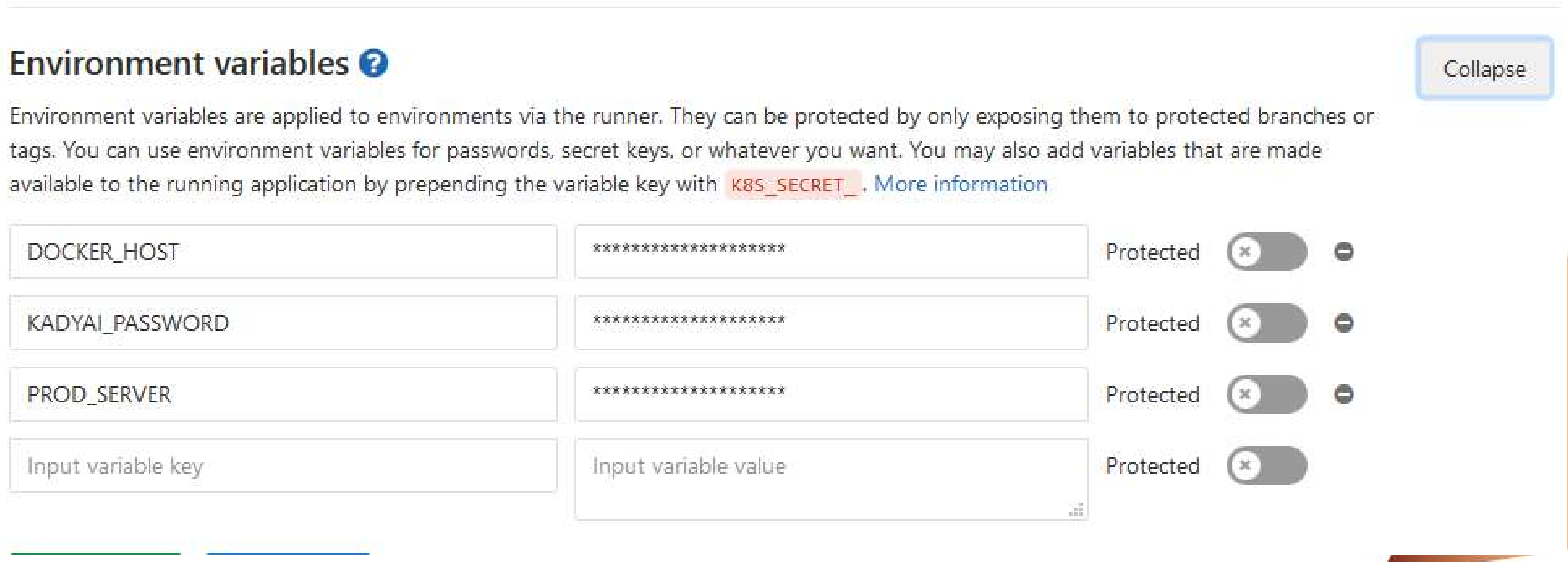
Using Variable

- Hide some secret information



```
docker-build-master:
  stage: package
  script:
    - docker build -t dto80/ap-main-controller-dev .
    - docker login -u dto80 -p $PASSWORD
    - docker push dto80/ap-main-controller-dev
```

Setting up the password



Environment variables

Collapse

Environment variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. You can use environment variables for passwords, secret keys, or whatever you want. You may also add variables that are made available to the running application by prepending the variable key with `K8S_SECRET_`. More information

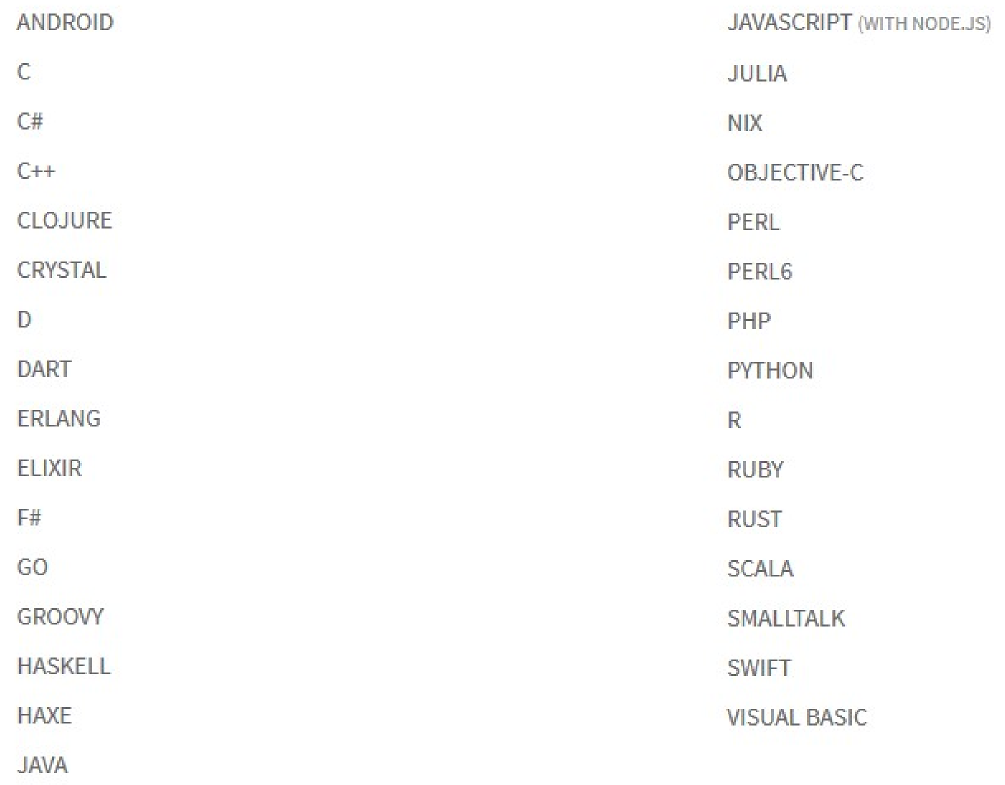
DOCKER_HOST	********************	Protected
KADYAI_PASSWORD	********************	Protected
PROD_SERVER	********************	Protected
Input variable key	Input variable value	Protected

Using the simple Linux commands

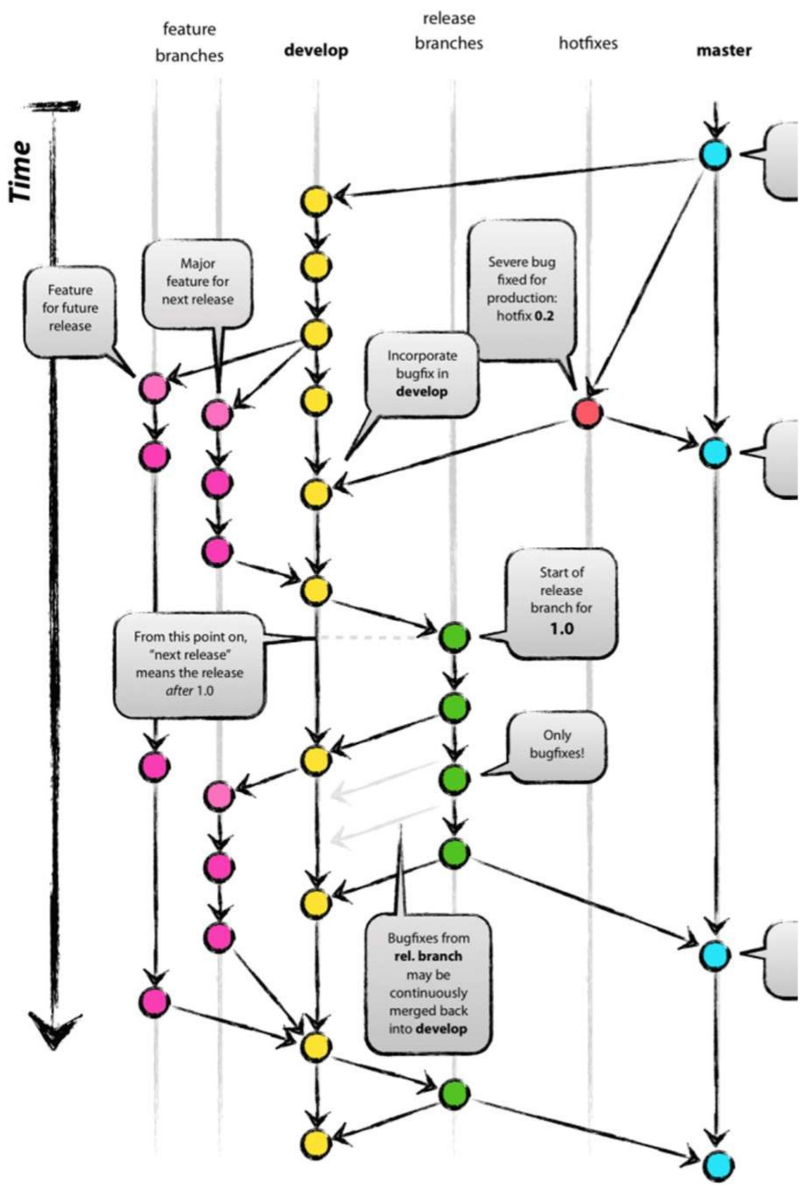
- In the build, and script tag
- Easy
  - For the people who can use Linux

Supports Languages

- Build management
  for each programming language



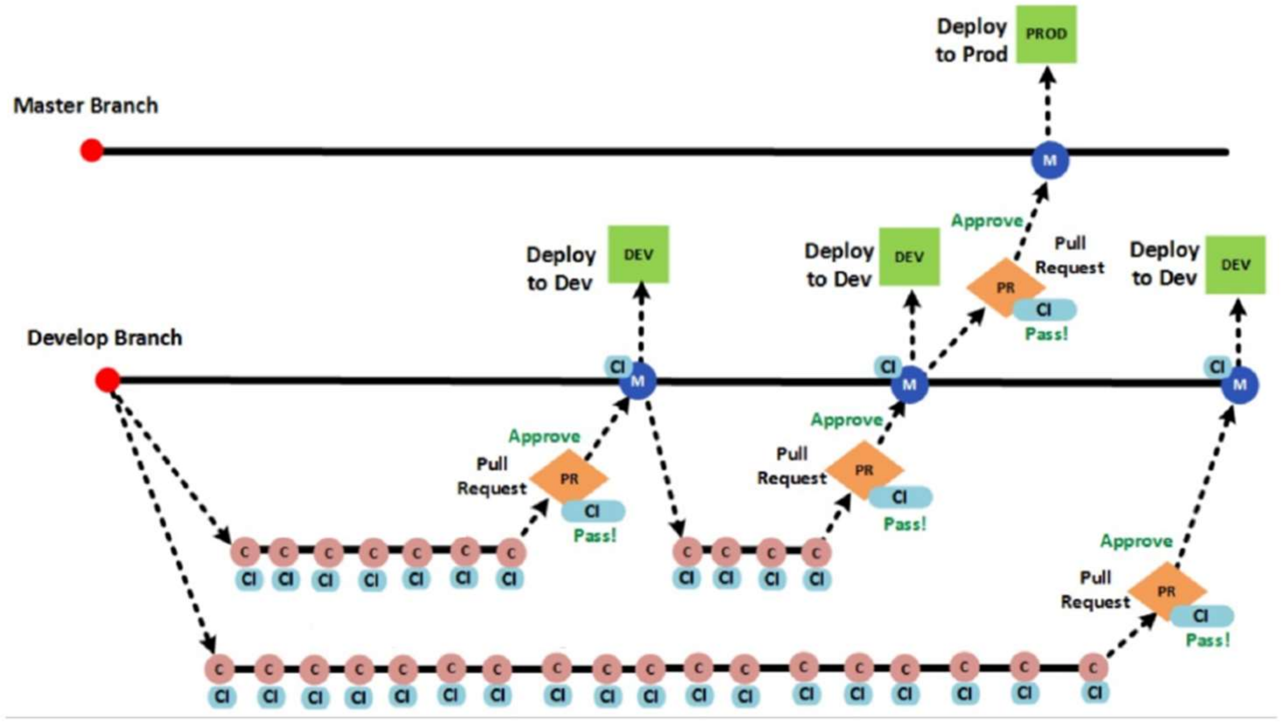
ANDROID
C
C#
C++
CLOJURE
CRYSTAL
D
DART
ERLANG
ELIXIR
F#
GO
GROOVY
HASKELL
HAXE
JAVA
JAVASCRIPT (WITH NODE.JS)
JULIA
NIX
OBJECTIVE-C
PERL
PERL6
PHP
PYTHON
R
RUBY
RUST
SCALA
SMALLTALK
SWIFT
VISUAL BASIC


feature branches
develop
release branches
hotfixes
master
Time
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop

Branch selection

- Git workflow
- Deploy only some branches

CD flows



CI = Continuous integration, C = commit, PR = pull request, M = merge

Gitlab CD

```
stages:
  - deploy
```

```
deploy_app:
  stage: deploy
  script:
    - ssh ubuntu@$DEPLOY_SERVER "rm -rf /var/www/html/*"
    - scp html/* ubuntu@$DEPLOY_SERVER:/var/www/html/
```

Gitlab CD/CI

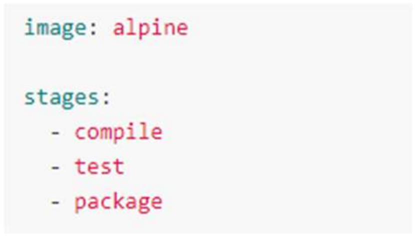
- Multiple stages

```yaml
image: alpine

stages:
  - compile
  - test
  - package
```

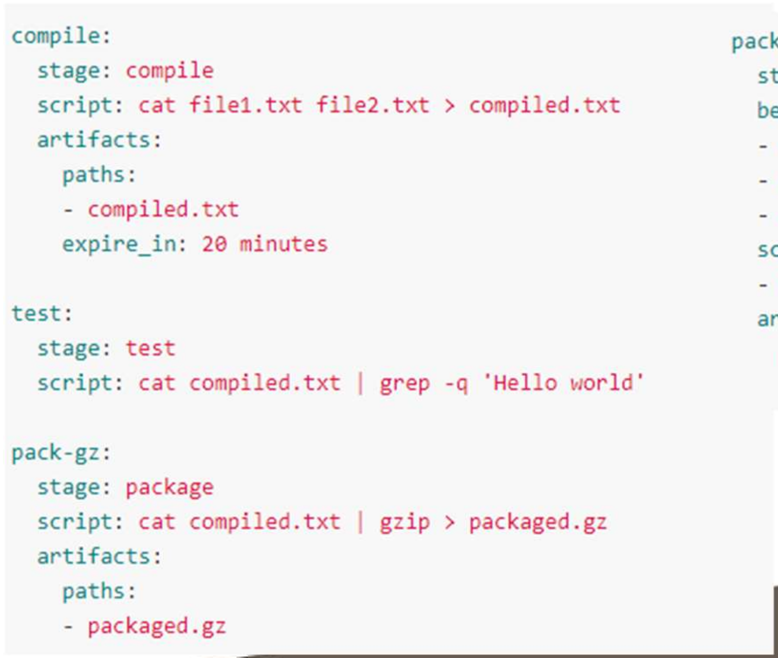
```yaml
compile:
  stage: compile
  script: cat file1.txt file2.txt > compiled.txt
  artifacts:
    paths:
    - compiled.txt
    expire_in: 20 minutes

test:
  stage: test
  script: cat compiled.txt | grep -q 'Hello world'

pack-gz:
  stage: package
  script: cat compiled.txt | gzip > packaged.gz
  artifacts:
    paths:
    - packaged.gz
```

Gitlab CD/CI

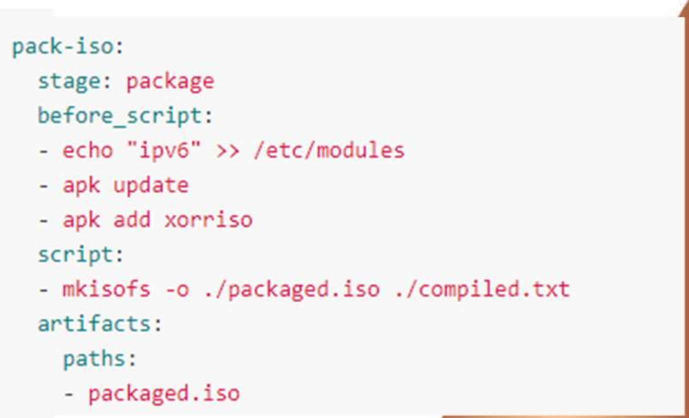
- Multiple parallel stages



```yaml
image: alpine

stages:
  - compile
  - test
  - package
```



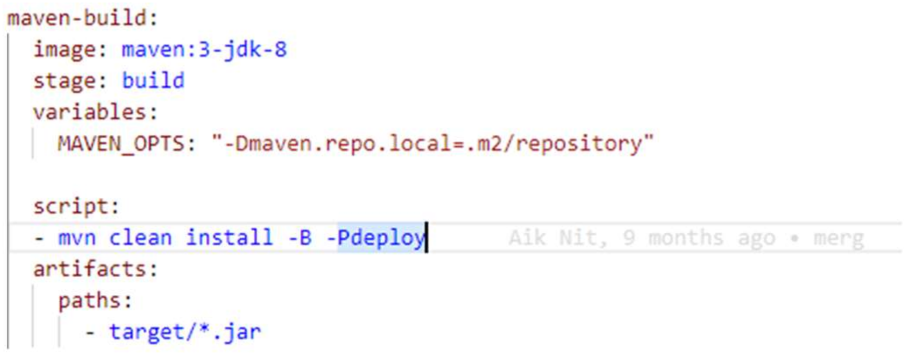
```yaml
compile:
  stage: compile
  script: cat file1.txt file2.txt > compiled.txt
  artifacts:
    paths:
    - compiled.txt
    expire_in: 20 minutes

test:
  stage: test
  script: cat compiled.txt | grep -q 'Hello world'

pack-gz:
  stage: package
  script: cat compiled.txt | gzip > packaged.gz
  artifacts:
    paths:
    - packaged.gz
```



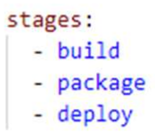
```yaml
pack-iso:
  stage: package
  before_script:
  - echo "ipv6" >> /etc/modules
  - apk update
  - apk add xorriso
  script:
  - mkisofs -o ./packaged.iso ./compiled.txt
  artifacts:
    paths:
    - packaged.iso
```

Path selection

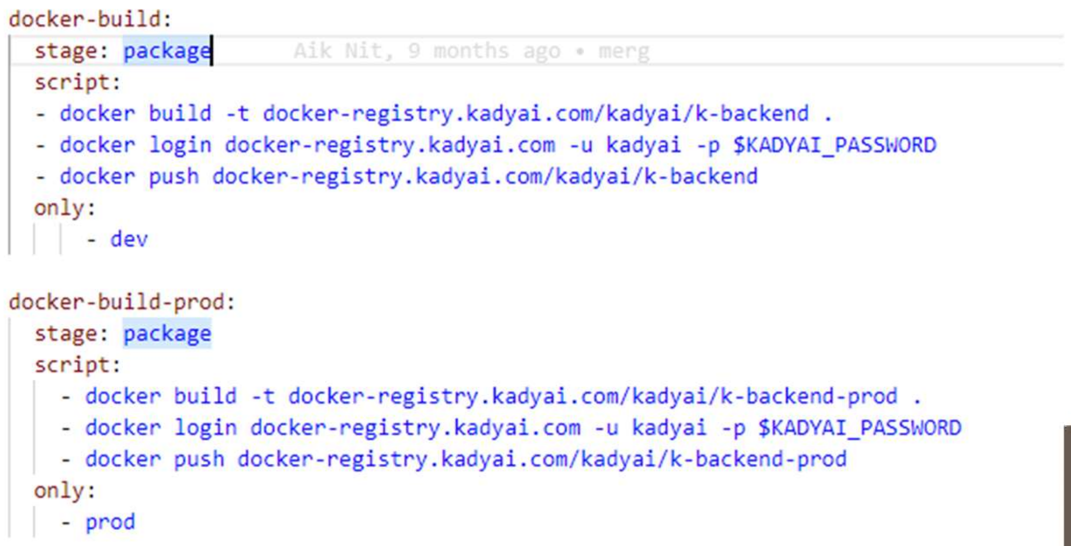
- only
- except



```
maven-build:
  image: maven:3-jdk-8
  stage: build
  variables:
    MAVEN_OPTS: "-Dmaven.repo.local=.m2/repository"

  script:
  - mvn clean install -B -Pdeploy        Aik Nit, 9 months ago • merg
  artifacts:
    paths:
      - target/*.jar
```



```
stages:
  - build
  - package
  - deploy
```



```
docker-build:
  stage: package        Aik Nit, 9 months ago • merg
  script:
  - docker build -t docker-registry.kadyai.com/kadyai/k-backend .
  - docker login docker-registry.kadyai.com -u kadyai -p $KADYAI_PASSWORD
  - docker push docker-registry.kadyai.com/kadyai/k-backend
  only:
      - dev

docker-build-prod:
  stage: package
  script:
    - docker build -t docker-registry.kadyai.com/kadyai/k-backend-prod .
    - docker login docker-registry.kadyai.com -u kadyai -p $KADYAI_PASSWORD
    - docker push docker-registry.kadyai.com/kadyai/k-backend-prod
  only:
    - prod
```

Q & A