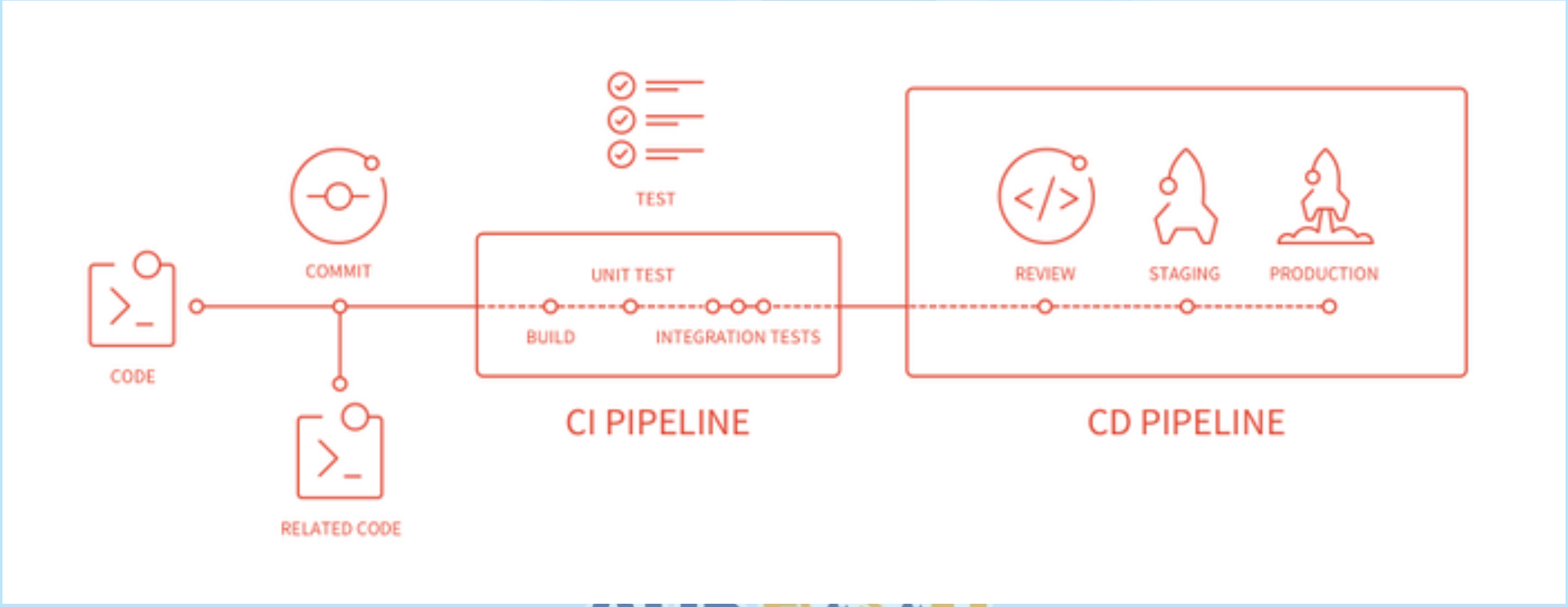
GITLAB-CI

AUKFOOD
Infogérance et conseil en logiciel libre

OBJECTIF

Maitriser les différentes options présentes dans Gitlab-CI



ÉTAPE 2 : AJOUT DU FICHIER .GITLAB-CI.YML

AUKFOOD
Infogérance et conseil en logiciel libre

ÉTAPE 2 : AJOUT DU FICHIER .GITLAB-CI.YML

L'arborescence du projet **avant** la CI :

```
.
├── config
│   └── ...
├── main.tf
├── .gitignore
└── README.md
```

ÉTAPE 2 : AJOUT DU FICHIER .GITLAB-CI.YML

L'arborescence du projet **avant** la CI :

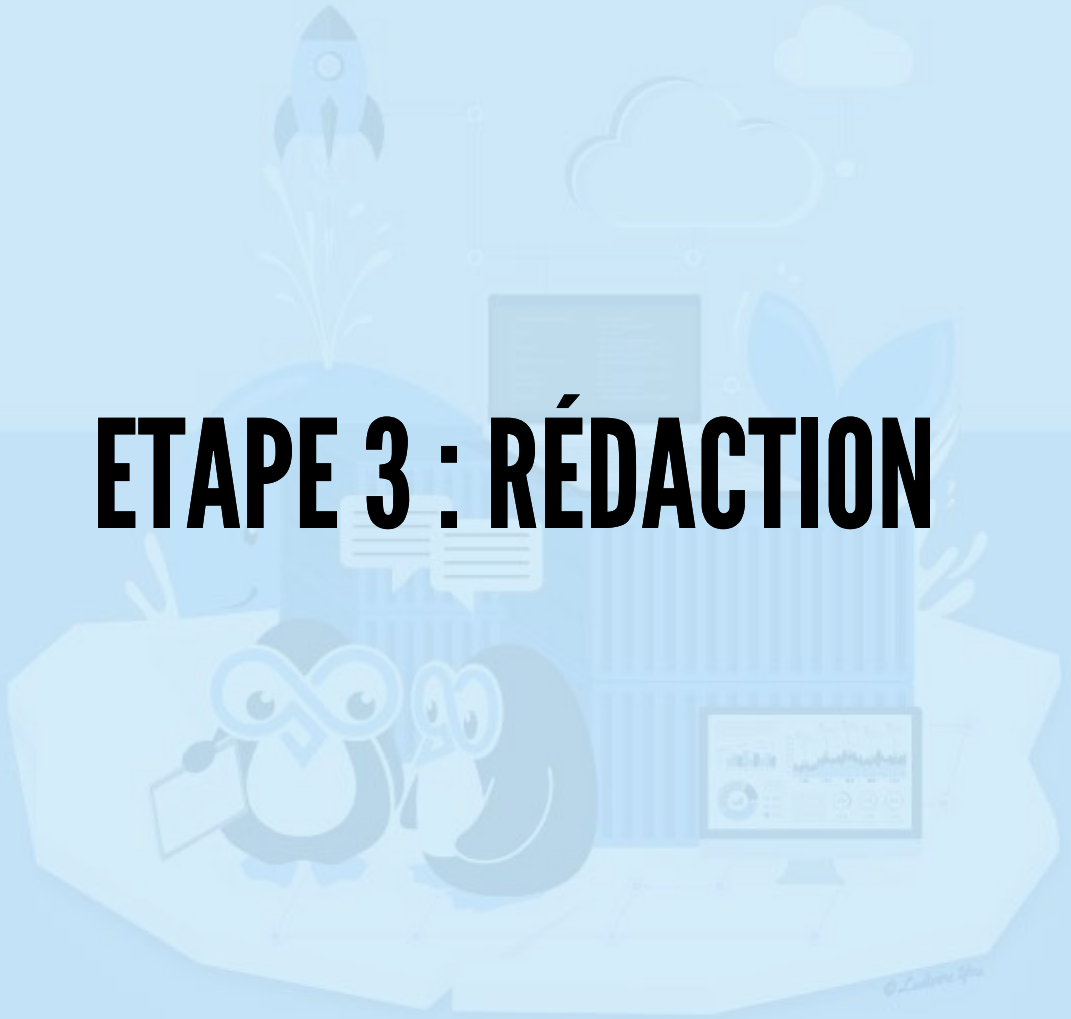
```
.
├── config
│   └── ...
├── main.tf
├── .gitignore
└── README.md
```

l'arborescence du projet **après** la CI :

```
.
├── config
│   └── ...
├── main.tf
├── .gitignore
├── .gitlab-ci.yaml
└── README.md
```

AUKFOOD
Infogérance et conseil en logiciel libre

ETAPE 3 : RÉDACTION



AUKFOOD
Infogérance et conseil en logiciel libre

DÉCLARATION D'UN JOB

AUKFOOD
Infogérance et conseil en logiciel libre

DÉCLARATION D'UN JOB

Voici comment déclarer un job le plus simplement possible :

```
1 nomjob:
2   script: echo 'my first job'
```

AUKFOOD
Infogérance et conseil en logiciel libre

DÉCLARATION D'UN JOB

Voici comment déclarer un job le plus simplement possible :

```
1 nomjob:
2   script: echo 'my first job'
```

Et si vous voulez déclarer plusieurs jobs :

```
1 job1:
2   script: echo 'my first job'
3
4 exemple2:
5   script: echo 'my second job'
```

AUKFOOD
Infogérance et conseil en logiciel libre

DÉCLARATION D'UN JOB

Voici comment déclarer un job le plus simplement possible :

```
1 nomjob:
2   script: echo 'my first job'
```

Et si vous voulez déclarer plusieurs jobs :

```
1 job1:
2   script: echo 'my first job'
3
4 exemple2:
5   script: echo 'my second job'
```

Liste à Bannir :

```
1 image
2 services
3 stages
4 types
5 before_script
6 after_script
7 variables
8 cache
```

AUKFOOD
Infogérance et conseil en logiciel libre

SCRIPT

AUKFOOD
Infogérance et conseil en logiciel libre

SCRIPT

Cette déclaration est le cœur du job car c'est ici que vous indiquerez les actions à effectuer.

AUKFOOD
Infogérance et conseil en logiciel libre

SCRIPT

Cette déclaration est le cœur du job car c'est ici que vous indiquerez les actions à effectuer.

BEFORE_SCRIPT ET AFTER_SCRIPT

```yaml
before_script: # Exécution d'une commande avant chaque `job`
  - echo 'start jobs'

after_script: # Exécution d'une commande après chaque `job`
  - echo 'end jobs'
```

AUKFOOD
Infogérance et conseil en logiciel libre

SCRIPT

Cette déclaration est le cœur du job car c'est ici que vous indiquerez les actions à effectuer.

BEFORE_SCRIPT ET AFTER_SCRIPT

```yaml
before_script: # Exécution d'une commande avant chaque `job`
  - echo 'start jobs'

after_script: # Exécution d'une commande après chaque `job`
  - echo 'end jobs'
```

```yaml
job:no_overwrite: # Ici le job exécutera les action du `before_script` et `after_script` par défaut
  script:
    - echo 'script'

job:overwrite:before_script:
  before_script:
    - echo 'overwrite' # N'exécutera pas l'action définie dans le `before_script` par défaut
  script:
    - echo 'script'
```

IMAGE

Cette déclaration est simplement l'image docker qui sera utilisée lors d'un job ou lors de tous les jobs

```yaml
image: alpine # Image utilisée par tous les `jobs`, ce sera l'image par défaut

job:node: # Job utilisant l'image node
  image: node
  script: yarn install

job:alpine: # Job utilisant l'image par défaut
  script: echo $USER
```

AUKFOOD
Infogérance et conseil en logiciel libre

STAGE

Cette déclaration permet de grouper des jobs en étapes. Par exemple on peut faire une étape de build, de codestyling, de test, de code coverage, de deployment, ….

```yaml
stages: # Ici on déclare toutes nos étapes
  - build
  - test
  - deploy

job:build:
  stage: build # On déclare que ce `job` fait partie de l'étape build
  script: make build

job:test:unit:
  stage: test # On déclare que ce `job` fait partie de l'étape test
  script: make test-unit

job:test:functional:
  stage: test # On déclare que ce `job` fait partie de l'étape test
  script: make test-functional

job:deploy:
  stage: deploy # On déclare que ce `job` fait partie de l'étape deploy
  script: make deploy
```

DIRECTIVES

Mettre en place des contraintes sur l'exécution d'une tâche.

ONLY ET EXCEPT

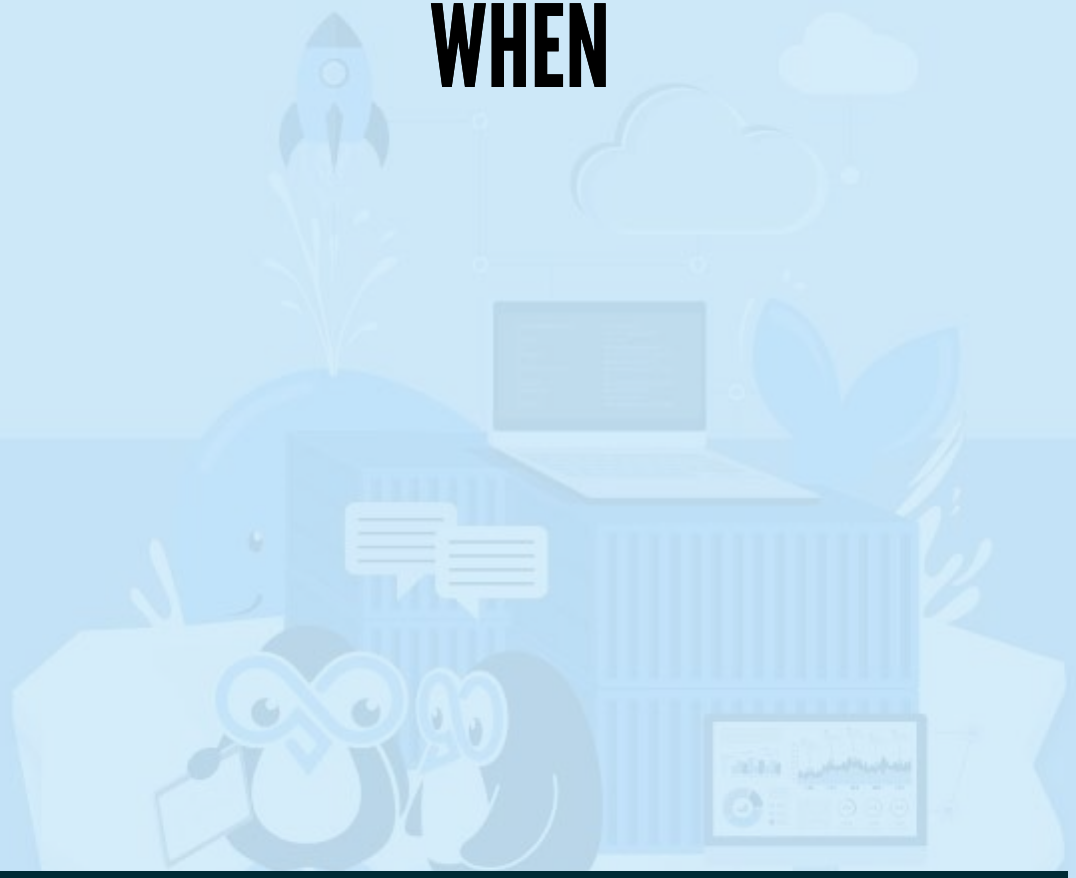
```yaml
job:only:master:
  script: make deploy
  only:
    - master # Le job sera effectué uniquement lors d'un événement sur la branche master

job:except:master:
  script: make test
  except:
    - master # Le job sera effectué sur toutes les branches lors d'un événement sauf sur la br
```

AUKFOOD
Infogérance et conseil en logiciel libre

WHEN

AUKFOOD
Infogérance et conseil en logiciel libre

WHEN



```yaml
job:clean:
  stage: clean
  script:
    - make clean # s'exécutera quoi qu'il se passe
  when: always
```

AUKFOOD
Infogérance et conseil en logiciel libre

WHEN

Il y a quatre modes possibles :

- **always** : le job s'exécutera quoi qu'il se passe (même en cas d'échec)

```yaml
job:clean:
  stage: clean
  script:
    - make clean # s'exécutera quoi qu'il se passe
  when: always
```

AUKFOOD
Infogérance et conseil en logiciel libre

WHEN

Il y a quatre modes possibles :

- **on_success** : le job sera exécuté uniquement si tous les jobs du stage précédent sont passés

- **always** : le job s'exécutera quoi qu'il se passe (même en cas d'échec)

```yaml
job:clean:
  stage: clean
  script:
    - make clean # s'exécutera quoi qu'il se passe
  when: always
```

AUKFOOD
Infogérance et conseil en logiciel libre

WHEN

Il y a quatre modes possibles :

- **on_success** : le job sera exécuté uniquement si tous les jobs du stage précédent sont passés
- **on_failure** : le job sera exécuté uniquement si un job est en échec
- **always** : le job s'exécutera quoi qu'il se passe (même en cas d'échec)

```
1 job:clean:
2   stage: clean
3   script:
4     - make clean # s'exécutera quoi qu'il se passe
5   when: always
```

AUKFOOD
Infogérance et conseil en logiciel libre

WHEN

Il y a quatre modes possibles :

- **on_success** : le job sera exécuté uniquement si tous les jobs du stage précédent sont passés
- **on_failure** : le job sera exécuté uniquement si un job est en échec
- **always** : le job s'exécutera quoi qu'il se passe (même en cas d'échec)
- **manual** : le job s'exécutera uniquement par une action manuelle

```
1 job:clean:
2   stage: clean
3   script:
4     - make clean # s'exécutera quoi qu'il se passe
5   when: always
```

AUKFOOD
Infogérance et conseil en logiciel libre

ALLOW_FAILURE

Cette directive permet d'accepter qu'un job échoue sans faire échouer la pipeline.

```yaml
job:clean:
  stage: clean
  script:
    - make clean
  when: always
  allow_failure: true # Ne fera pas échouer la pipeline
```

AUKFOOD
Infogérance et conseil en logiciel libre

ENVIRONMENT

Cette déclaration permet de définir un environnement spécifique au déploiement.

Il est possible de spécifier :

- un **name**,
- une **url**,
- une condition **on_stop**,
- une **action** en réponse de la condition précédente.

```
1 deploy:production:
2   environment: # Déclaration étendue de l'environnement
3     name: production
4     url: 'https://blog.eleven-labs/fr/gitlab-ci/' # Url de l'application
5   script:
6     - make deploy
```

AUKFOOD
Infogérance et conseil en logiciel libre

on_stop et **action** seront utilisés pour ajouter une action à la fin du déploiement, si vous souhaitez arrêter votre application sur commande.

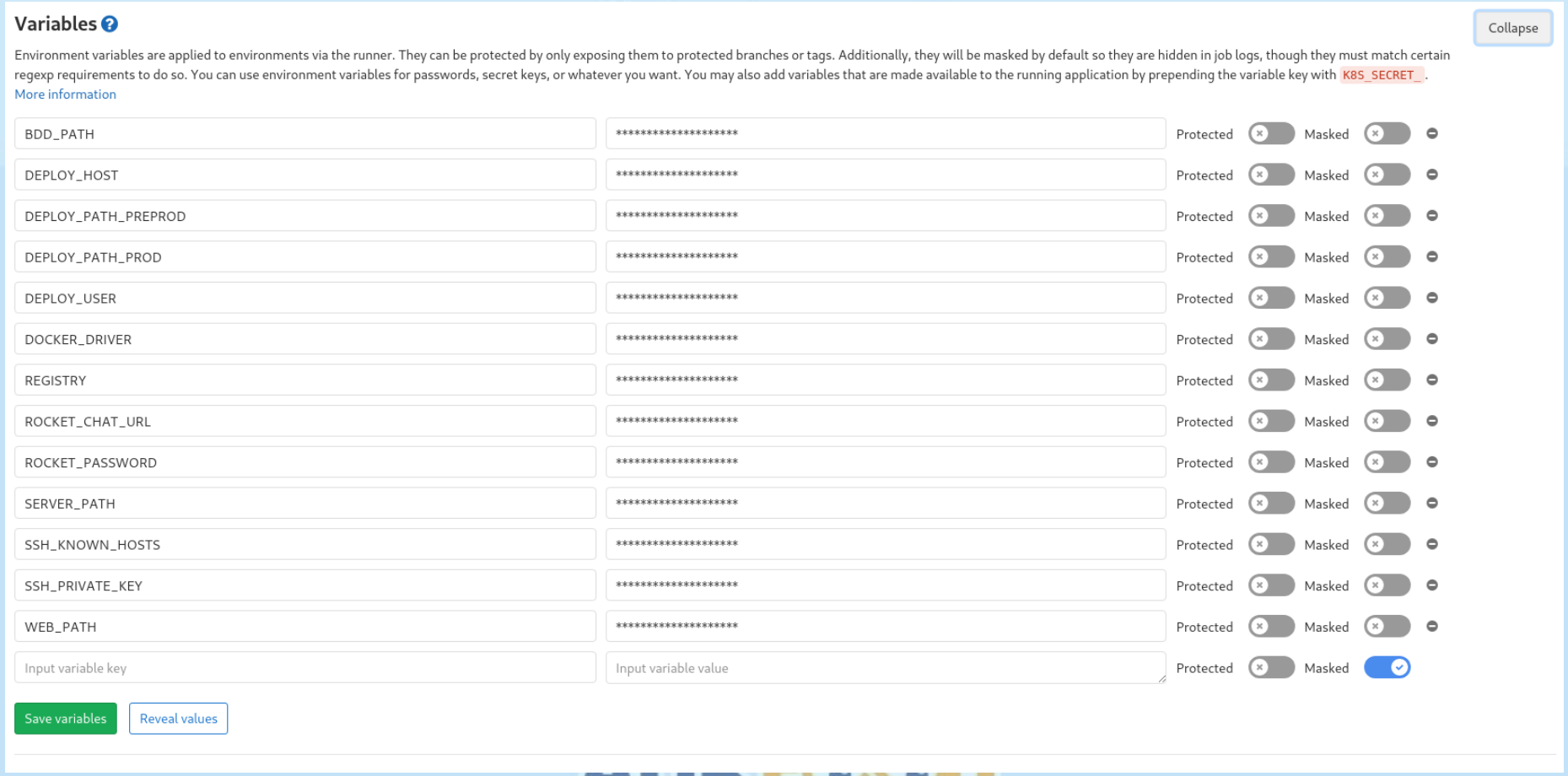
```
1  deploy:demo:
2    script: make deploy
3    environment:
4      name: demo
5      on_stop: stop:demo
6
7  stop:demo: # Ce job pourra être visible et exécuté uniquement après le job `deploy:demo`
8    script: make stop
9    environment:
10     name: demo
11     action: stop
```

VARIABLES

Cette déclaration permet de définir des variables pour tous les jobs ou pour un job précis.

```yaml
variables: # Déclaration de variables pour tous les `job`
  SYMFONY_ENV: prod

build:
  script: echo ${SYMFONY_ENV} # Affichera "prod"
```

Il est aussi possible de déclarer des variables depuis l'interface web de GitLab Settings > CI/CD > Variables et de leur spécifier un environnement.



Infogérance et conseil en logiciel libre

CACHE

Le cache est intéressant pour spécifier une liste de fichiers et de répertoires à mettre en cache tout le long de votre pipeline. Une fois la pipeline terminée le cache sera détruit.

```yaml
stages:
  - build
  - deploy

job:build:
  stage: build
  image: node:8-alpine
  script: yarn install && yarn build
  cache:
    paths:
      - build # répertoire mis en cache
    policy: push # le cache sera juste sauvegardé, pas de récupération d'un cache existant

job:deploy:
  stage: deploy
  script: make deploy
  cache:
    paths:
      - build
    policy: pull # récupération du cache
```

ARTIFACTS

Les artefacts sont un peu comme du cache mais ils peuvent être récupérés depuis une autre pipeline.

```yaml
job:
  script: make build
  artifacts:
    paths:
      - dist
    name: artifact:build
    when: on_success
    expire_in: 1 weeks
```

AUKFOOD
Infogérance et conseil en logiciel libre

ARTIFACTS

Les artefacts sont un peu comme du cache mais ils peuvent être récupérés depuis une autre pipeline.

- **paths** : obligatoire, elle permet de spécifier la liste des fichiers et/ou dossiers à mettre en artifact

```yaml
job:
  script: make build
  artifacts:
    paths:
      - dist
    name: artifact:build
    when: on_success
    expire_in: 1 weeks
```

AUKFOOD
Infogérance et conseil en logiciel libre

ARTIFACTS

Les artefacts sont un peu comme du cache mais ils peuvent être récupérés depuis une autre pipeline.

- **paths** : obligatoire, elle permet de spécifier la liste des fichiers et/ou dossiers à mettre en artifact
- **name** : facultative, elle permet de donner un nom à l'artifact. Par défaut elle sera nommée artifacts.zip

```yaml
job:
  script: make build
  artifacts:
    paths:
      - dist
    name: artifact:build
    when: on_success
    expire_in: 1 weeks
```

AUKFOOD
Infogérance et conseil en logiciel libre

ARTIFACTS

Les artefacts sont un peu comme du cache mais ils peuvent être récupérés depuis une autre pipeline.

- **paths** : obligatoire, elle permet de spécifier la liste des fichiers et/ou dossiers à mettre en artifact
- **name** : facultative, elle permet de donner un nom à l'artifact. Par défaut elle sera nommée artifacts.zip
- **untracked** : facultative, elle permet d'ignorer les fichiers définis dans le fichier .gitignore

```yaml
job:
  script: make build
  artifacts:
    paths:
      - dist
    name: artifact:build
    when: on_success
    expire_in: 1 weeks
```

AUKFOOD
Infogérance et conseil en logiciel libre

ARTIFACTS

Les artefacts sont un peu comme du cache mais ils peuvent être récupérés depuis une autre pipeline.

- **paths** : obligatoire, elle permet de spécifier la liste des fichiers et/ou dossiers à mettre en artifact
- **name** : facultative, elle permet de donner un nom à l'artifact. Par défaut elle sera nommée artifacts.zip
- **untracked** : facultative, elle permet d'ignorer les fichiers définis dans le fichier .gitignore
- **when** : facultative, elle permet de définir quand l'artifact doit être créé. Trois choix possibles :
  - **on_success**, La valeur on_success est la valeur par défaut.
  - **on_failure**
  - **always**

```yaml
job:
  script: make build
  artifacts:
    paths:
      - dist
    name: artifact:build
    when: on_success
    expire_in: 1 weeks
```

AUKFOOD
Infogérance et conseil en logiciel libre

ARTIFACTS

Les artefacts sont un peu comme du cache mais ils peuvent être récupérés depuis une autre pipeline.

- **paths** : obligatoire, elle permet de spécifier la liste des fichiers et/ou dossiers à mettre en artifact
- **name** : facultative, elle permet de donner un nom à l'artifact. Par défaut elle sera nommée artifacts.zip
- **untracked** : facultative, elle permet d'ignorer les fichiers définis dans le fichier .gitignore
- **when** : facultative, elle permet de définir quand l'artifact doit être créé. Trois choix possibles :
  - **on_success**, La valeur on_success est la valeur par défaut.
  - **on_failure**
  - **always**
- **expire_in** : facultative, elle permet de définir un temps d'expiration

```yaml
job:
  script: make build
  artifacts:
    paths:
      - dist
    name: artifact:build
    when: on_success
    expire_in: 1 weeks
```

AUKFOOD
Infogérance et conseil en logiciel libre

DEPENDENCIES

Cette déclaration fonctionne avec les artifacts, il rend un job dépendant d'un artifact. Si l'artifact a expiré ou a été supprimé / n'existe pas, alors la pipeline échouera.

```yaml
build:artifact:
  stage: build
  script: echo hello > artifact.txt
  artifacts: # On ajoute un `artifact`
    paths:
      - artifact.txt

deploy:ko:
  stage: deploy
  script: cat artifact.txt
  dependencies: # On lie le job avec 'build:artifact:fail' qui n'existe pas donc la pipeline
    - build:artifact:fail

deploy:ok:
  stage: deploy
  script: cat artifact.txt
  dependencies: # On lie le job avec 'build:artifact' qui existe donc la pipeline n'échouera
    - build:artifact
```

CONTACT !

Twitter : @aukfood

Facebook : @aukfood

Site Internet : https://www.aukfood.fr

Linkedin : company/aukfood

Mail : contact@aukfood.fr

AUKFOOD
Infogérance et conseil en logiciel libre