LISV Laboratoire d'ingénierie des systèmes de Versailles
UVSQ université PARIS-SACLAY

R5.04 – Cycle de vie d'un projet informatique
TP : installer et utiliser sa forge logicielle

Halim Djerroud

révision 1.0

Objectifs et organisation

Dans ce TP, votre équipe installe **sa propre forge logicielle** (Forgejo) avec Docker, puis l'utilise pour organiser et développer un premier sprint. Cette forge sera celle de la **SAÉ 5.01/5.02 « La Carotte Électronique »** : tout le suivi du projet (tickets, sprints, code, revues, tests) s'y fera jusqu'à la soutenance.

Ce TP met en pratique les deux cours de la ressource :

— **Cours 1** : user stories, planning poker, sprint, tableau Kanban, Definition of Done ;

— **Cours 2** : Git, branches, pull requests, protection de `main`, tags, intégration continue.

Organisation

— Vous travaillez avec **votre équipe de SAÉ** (2 ou 3 étudiants).

— Partie 1 : **chaque étudiant** installe Forgejo sur son poste, pour comprendre l'installation.

— À partir de la partie 2, l'équipe choisit **une seule forge**, installée sur une **machine qui reste disponible** pendant toute la SAÉ (VM de l'équipe) : c'est elle que tout le monde utilise.

— Les **points de validation** sont à faire vérifier par l'enseignant avant de continuer.

Prérequis

— Une machine Linux avec **Docker 25 ou plus récent** et **Docker Compose** (`docker compose version`).

— Les ports **3000** (web) et **2222** (SSH) libres sur cette machine.

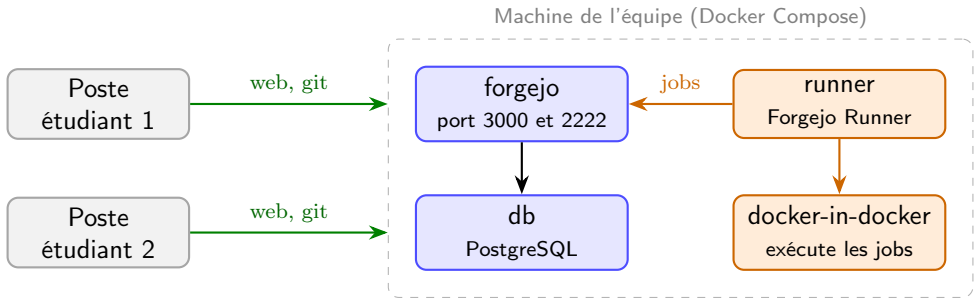
— `git` installé sur chaque poste.

— Les fichiers fournis par l'enseignant : dossier `ressources/` et dossier `depot-initial/`.

Déroulé indicatif

Partie	Contenu	Durée
1	Installer Forgejo avec Docker Compose	1h30
2	Administrer la forge : comptes, organisation, équipe, clés SSH	45 min
3	Créer le projet et organiser le sprint 1	1h
4	Travailler en équipe : branches, pull requests, conflit, version	1h15
5	Mettre en place l'intégration continue	1h30

Architecture cible



Les parties 1 à 4 installent et utilisent **forgejo** et **db** ; la partie 5 ajoute **runner** et **docker-in-docker**.

1 Installer Forgejo avec Docker Compose (1h30)

1.1 Préparer le dossier

Tout ce qui concerne la forge est rangé dans un seul dossier : fichiers de configuration et données.

```
$ mkdir ~/forge && cd ~/forge
$ cp /chemin/vers/ressources/docker-compose.yml .
$ cp /chemin/vers/ressources/env.exemple .env
```

Relevez l'**adresse IP** de la machine :

```
$ hostname -I
192.168.1.50 172.17.0.1
```

La première adresse est celle de la machine sur le réseau (ici `192.168.1.50`).

Attention. N'utilisez pas `localhost` dans la configuration de la forge. Les autres membres de l'équipe, et plus tard les jobs d'intégration continue (qui tournent dans des conteneurs), doivent pouvoir joindre la forge : pour eux, `localhost` désigne **leur propre machine**, pas la vôtre.

1.2 Le fichier `.env`

Les valeurs propres à votre installation, dont les **mots de passe**, ne sont pas écrites dans `docker-compose.yml` mais dans un fichier `.env`, lu automatiquement par Docker Compose. Complétez-le :

```
# .env
ADRESSE_IP=192.168.1.50
POSTGRES_PASSWORD=choisissez-un-mot-de-passe-solide
```

Question 1. Pourquoi sépare-t-on les mots de passe dans un fichier `.env` ? Si ce dossier était versionné avec Git, que faudrait-il ajouter au fichier `.gitignore` ? (voir cours 2)

1.3 Le fichier `docker-compose.yml`

Voici le fichier fourni. Lisez-le attentivement avant de lancer quoi que ce soit.

```
networks:
  forge:

services:
  forgejo:
```

1.3 *Le fichier* `docker-compose.yml`

```
    image: codeberg.org/forgejo/forgejo:15
    container_name: forgejo
    environment:
      - USER_UID=1000
      - USER_GID=1000
      - FORGEJO__database__DB_TYPE=postgres
      - FORGEJO__database__HOST=db:5432
      - FORGEJO__database__NAME=forgejo
      - FORGEJO__database__USER=forgejo
      - FORGEJO__database__PASSWD=${POSTGRES_PASSWORD}
      - FORGEJO__server__DOMAIN=${ADRESSE_IP}
      - FORGEJO__server__ROOT_URL=http://${ADRESSE_IP}:3000/
      - FORGEJO__server__SSH_DOMAIN=${ADRESSE_IP}
      - FORGEJO__server__SSH_PORT=2222
      - FORGEJO__security__INSTALL_LOCK=true
    restart: unless-stopped
    networks:
      - forge
    volumes:
      - ./forgejo:/data
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "3000:3000"
      - "2222:22"
    depends_on:
      - db

  db:
    image: postgres:16
    container_name: forgejo-db
    environment:
      - POSTGRES_USER=forgejo
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
      - POSTGRES_DB=forgejo
    restart: unless-stopped
    networks:
      - forge
    volumes:
      - ./postgres:/var/lib/postgresql/data
```

Quelques repères :

— `codeberg.org/forgejo/forgejo:15` : Forgejo **15**, version à support long (LTS), maintenue jusqu'en juillet 2027. Toute la SAÉ se fera sans changement de version.

— `FORGEJO__section__CLE` : chaque variable de ce type remplace une option du fichier de configuration de Forgejo (`app.ini`).

— `INSTALL_LOCK=true` : la configuration est entièrement donnée ici, la page d'installation web est désactivée.

— `volumes` : les données sont stockées dans les dossiers `./forgejo` et `./postgres` de la machine.

Question 2. Quel est le rôle de chacun des deux services ? Comment le service `forgejo` trouve-t-il la base de données (regardez `DB_TYPE`, `HOST` et `networks`) ?

Question 3. Le port `22` du conteneur est publié sur le port `2222` de la machine. Pourquoi ne pas utiliser directement le port 22 de la machine ?

Question 4. Que deviendraient les dépôts et les tickets de l'équipe si on supprimait les lignes `volumes` puis qu'on recréait les conteneurs ?

1.4 Démarrer la forge

```
$ docker compose config          # vérifie le fichier et affiche les valeurs de .env
$ docker compose up -d           # télécharge les images et démarre en arrière-plan
$ docker compose ps              # état des conteneurs
$ docker compose logs -f forgejo # journal (Ctrl+C pour quitter)
```

La forge est prête quand elle répond au test de santé :

```
$ curl http://192.168.1.50:3000/api/healthz
$ curl http://192.168.1.50:3000/api/v1/version
{"version":"15.0.8+gitea-1.22.0"}
```

1.5 Créer le compte administrateur

Le premier compte est créé **en ligne de commande**, dans le conteneur, avec l'utilisateur système `git` :

```
$ docker exec -u git forgejo forgejo admin user create --admin \
    --username admin-equipe --email votre.email@uvsq.fr \
    --password 'Un-Mot-De-Passe-Solide' --must-change-password=false
New user 'admin-equipe' has been successfully created!
```

Ouvrez `http://192.168.1.50:3000` dans un navigateur et connectez-vous avec ce compte.

Question 5. Que fait `docker exec -u git forgejo ...` ? Pourquoi est-il préférable de créer le compte administrateur ainsi plutôt que via une page d'inscription ouverte à tous ?

Question 6. Arrêtez la forge avec `docker compose down`, puis relancez-la avec `docker compose up -d`. Le compte administrateur existe-t-il toujours ? Pourquoi ?

Point de validation. Chaque étudiant montre sa forge qui répond sur `http://ADRESSE_IP:3000` avec son compte administrateur. L'équipe indique à l'enseignant **quelle machine** hébergera la forge commune pour la suite du TP et pour la SAÉ.

2 Administrer la forge (45 min)

À partir d'ici, toute l'équipe travaille sur la forge commune.

2.1 Créer les comptes de l'équipe

L'administrateur crée un compte pour chaque membre de l'équipe **et pour l'enseignant**. Sans l'option `--must-change-password=false`, l'utilisateur devra choisir son propre mot de passe à la première connexion :

```
$ docker exec -u git forgejo forgejo admin user create \
    --username prenom.nom --email prenom.nom@uvsq.fr \
    --password 'Mot-De-Passe-Provisoire'
```

2.2 Organisation et équipe

Pour que personne d'autre ne puisse créer de compte, désactivez l'inscription libre : ajoutez la variable suivante au service `forgejo`, puis relancez avec `docker compose up -d`.

```
      - FORGEJO__service__DISABLE_REGISTRATION=true
```

Question 7. Vérifiez que le lien « S'inscrire » a disparu de la page d'accueil. Pourquoi `docker compose up -d` suffit-il à prendre en compte la modification ?

2.2 Organisation et équipe

Les dépôts de la SAÉ n'appartiennent pas à une personne mais à l'**équipe**. Dans l'interface web :

1. Créez une **organisation** nommée `carotte-equipeN` (N = numéro de votre équipe) avec le bouton « + » en haut à droite.
2. Dans l'organisation, créez une **équipe** `developpeurs` avec le droit d'**écriture** sur les dépôts, et ajoutez-y tous les membres.
3. Ajoutez le compte de l'**enseignant** aux propriétaires de l'organisation (*Owners*).

Question 8. Quelle différence faites-vous entre le rôle « propriétaire » et le droit « écriture » ? Pourquoi ne met-on pas tous les étudiants propriétaires ?

2.3 Clés SSH

Chaque membre crée une clé SSH sur **son poste** (si ce n'est pas déjà fait) et déclare sa **clé publique** dans ses paramètres Forgejo (*Paramètres* → *Clés SSH / GPG*) :

```
$ ssh-keygen -t ed25519 -C "prenom.nom@uvsq.fr"
$ cat ~/.ssh/id_ed25519.pub      # à copier dans Forgejo
```

Testez la connexion (port 2222, utilisateur `git`) :

```
$ ssh -T -p 2222 git@192.168.1.50
Hi there, prenom.nom! You've successfully authenticated with the key named ...,
but Forgejo does not provide shell access.
```

Attention. On ne copie **jamais** la clé privée (`id_ed25519`, sans `.pub`) : elle ne quitte pas votre poste.

Question 9. Bonus sécurité : activez l'authentification à deux facteurs (TOTP) sur le compte administrateur. Contre quel type d'attaque protège-t-elle ?

Point de validation. Tous les membres se connectent avec leur compte, appartiennent à l'équipe `developpeurs` de l'organisation, et `ssh -T` s'authentifie. L'enseignant a accès à l'organisation.

3 Créer le projet et organiser le sprint 1 (1h)

3.1 Le dépôt `carotte`

Dans l'organisation, créez un dépôt `carotte`, **vide** (ne cochez pas l'initialisation avec un README). Puis, sur le poste d'**un seul** membre, publiez le dépôt de départ fourni :

```
$ cp -r /chemin/vers/depot-initial ~/carotte && cd ~/carotte
$ git init -b main
$ git add .
$ git commit -m "Structure du projet et tests des APDU"
$ git remote add origin ssh://git@192.168.1.50:2222/carotte-equipeN/carotte.git
$ git push -u origin main
```

Le dépôt de départ contient :

— `carotte/apdu.py` : construction des APDU de la carte Rubrovitamina (lecture du solde, crédit, débit) et conversion des montants en 2 octets *big endian* ;

— `tests/test_apdu.py` : les tests unitaires de ce module ;

— `README.md` et `.gitignore`.

Les autres membres **clonent** le dépôt et vérifient que les tests passent :

```
$ git clone ssh://git@192.168.1.50:2222/carotte-equipeN/carotte.git
$ cd carotte
$ python3 -m unittest discover -s tests -v
```

3.2 Le backlog : des user stories dans les tickets

Dans le dépôt, créez des **étiquettes** (*Labels*) :

— `story`, `tâche`, `bug` ;

— `points: 1`, `points: 2`, `points: 3`, `points: 5`, `points: 8`.

Créez un **jalon** (*Milestone*) `Sprint 1` avec une date d'échéance fixée avec l'enseignant.

À partir du sujet de la SAÉ, rédigez **au moins 6 tickets** « story » au format du cours 1. Par exemple :

— *En tant qu'**agent administratif**, je veux **afficher la version de la carte** afin de vérifier qu'elle est compatible avant de l'attribuer.* (Lubiana)

— *En tant qu'**agent administratif**, je veux **attribuer un bonus** à un étudiant afin de récompenser son travail.* (Rodelika)

— *En tant qu'**étudiant**, je veux **acheter une boisson** avec ma carte afin de profiter de mes bonus.* (Lunar White)

Chaque ticket contient des **critères d'acceptation**.

3.3 Sprint planning

1. **Planning poker** : estimez chaque story en équipe (cartes du cours 1, ou application de planning poker), puis posez l'étiquette `points` correspondante.
2. Choisissez les stories du **sprint 1** et rattachez-les au jalon `Sprint 1`.
3. **Découpez** chaque story choisie en tâches, sous forme de liste à cocher dans le ticket :

```
## Tâches
- [ ] Construire l'APDU de lecture de version
- [ ] Afficher la version dans le menu de Lubiana
- [ ] Écrire les tests
```

4. Créez un **projet** (onglet *Projets*) de type Kanban avec les colonnes `À faire`, `En cours`, `En revue`, `Terminé`, et placez-y les tickets du sprint.
5. **Assignez** chaque ticket à un membre.

Question 10. Rédigez la **Definition of Done** de votre équipe dans le fichier `README.md` (vous la publierez dans la partie 4, par une pull request).

Point de validation. Le dépôt contient le code de départ ; le backlog compte au moins 6 stories estimées ; le jalon `Sprint 1` et le tableau Kanban sont remplis ; chaque ticket du sprint est assigné.

4 Travailler en équipe (1h15)

4.1 Protéger la branche `main`

Dans les paramètres du dépôt, onglet *Branches*, ajoutez une **règle de protection** pour `main` :

— **désactiver le push** direct ;

— exiger **1 approbation** avant la fusion.

Vérifiez que la protection fonctionne : un push direct sur `main` doit être refusé.

```
$ git push origin main
 ! [remote rejected] main -> main (pre-receive hook declined)
```

4.2 Le workflow : une branche par ticket

Chaque membre applique le workflow du cours 2 sur **un ticket qui lui est assigné**. Pour commencer, un ticket simple : ajouter la Definition of Done au `README.md`, ou compléter une fonction et ses tests.

```
$ git switch main && git pull              # partir d'un main à jour
$ git switch -c 3-definition-of-done       # numéro du ticket + description
# ... modifications ...
$ git add README.md
$ git commit -m "Ajout de la Definition of Done"
$ git push -u origin 3-definition-of-done
```

1. Sur la forge, ouvrez une **pull request** vers `main`. Dans sa description, écrivez `Closes #3`.
2. Déplacez la carte du ticket dans la colonne `En revue`.
3. Un **autre membre** relit la PR, commente si besoin, puis l'**approuve**.
4. L'auteur fusionne la PR et supprime la branche.
5. Vérifiez que le ticket #3 a été **fermé automatiquement**, puis déplacez sa carte dans `Terminé`.

Question 11. Essayez de fusionner une PR sans approbation. Que se passe-t-il ? Quel critère de la Definition of Done est ainsi garanti ?

4.3 Provoquer et résoudre un conflit

Deux membres, A et B, partent tous les deux du même `main` :

1. A crée la branche `8-message-solde` et modifie la fonction `formater_euros` pour afficher `"1.20 euros"` ;
2. B crée la branche `9-symbole-euro` et modifie **la même ligne** pour afficher `"1.20 EUR"` ;
3. les deux poussent et ouvrent une PR ; la PR de A est approuvée et fusionnée en premier.

La PR de B est maintenant en conflit. B le résout **sur son poste** :

```
$ git switch 9-symbole-euro
$ git pull origin main          # fusionne main dans la branche : CONFLICT
$ git status                    # fichiers en conflit
# éditer carotte/apdu.py : choisir la version, supprimer <<<<<<< ======= >>>>>>>
$ python3 -m unittest discover -s tests -v
$ git add carotte/apdu.py
$ git commit                    # termine la fusion
$ git push
```

Question 12. Les tests passent-ils encore après ces modifications ? Corrigez les tests si le format d'affichage a changé : qui, de A ou de B, aurait dû le faire ?

4.4 Publier une version

En fin de sprint, l'incrément est marqué par un **tag** :

```
$ git switch main && git pull
$ git tag -a v0.1.0 -m "Fin du sprint 1"
$ git push origin v0.1.0
```

Sur la forge, créez une **version** (*Release*) à partir de ce tag, avec une courte description de ce qu'elle apporte.

Question 13. Affichez l'historique avec `git log --oneline --graph --all`. Repérez les commits de fusion et le tag.

Point de validation. L'historique montre au moins une PR fusionnée **par membre**, avec approbation ; un conflit résolu ; le ticket correspondant fermé par `Closes #N` ; le tag `v0.1.0` et sa version publiée.

5 Mettre en place l'intégration continue (1h30)

On ajoute à la forge un **runner** : un programme qui récupère les jobs d'intégration continue et les exécute dans des conteneurs. Pour que ces conteneurs soient **isolés** de la machine, le runner utilise son propre démon Docker : le service **docker-in-docker**.

5.1 Ajouter les services au `docker-compose.yml`

Le fichier fourni contient déjà ces deux services, en commentaire, à la fin de la section `services` : **décommentez-les** et lisez-les.

```
  docker-in-docker:
    image: docker:dind
    container_name: forgejo-dind
    privileged: true
    command: ["dockerd", "-H", "tcp://0.0.0.0:2375", "--tls=false"]
    restart: unless-stopped
    networks:
      - forge

  runner:
    image: data.forgejo.org/forgejo/runner:13
    container_name: forgejo-runner
    user: "1001:1001"
```

```
    environment:
      - DOCKER_HOST=tcp://docker-in-docker:2375
    volumes:
      - ./runner:/data
    command: forgejo-runner daemon --config /data/config.yml
    restart: unless-stopped
    networks:
      - forge
    depends_on:
      - docker-in-docker
      - forgejo
```

Question 14. Le service `docker-in-docker` est lancé en mode `privileged`. Que signifie ce mode ? Pourquoi ne faut-il **pas** exposer son port 2375 sur la machine (pas de section `ports`) ?

5.2 Déclarer le runner dans Forgejo

Dans les paramètres de l'**organisation**, rubrique *Actions* → *Runners* (http://192.168.1.50:3000/org/carotte-equipeN/settings/actions/runners), cliquez sur **Créer un runner**. Forgejo affiche un **UUID** et un **jeton** (*token*) : notez-les, le jeton ne sera plus affiché.

Attention. Le jeton permet à n'importe qui de se faire passer pour votre runner et de recevoir votre code : c'est un **secret**, à ne jamais versionner ni partager.

5.3 Configurer et démarrer le runner

Créez le dossier du runner et son fichier de configuration (modèle fourni dans `ressources/runner-config.yml`) :

```
$ mkdir -p ~/forge/runner/.cache
$ cp /chemin/vers/ressources/runner-config.yml ~/forge/runner/config.yml
```

```
# runner/config.yml
log:
  level: info

runner:
  capacity: 1
  labels:
    - "docker:docker://node:20-bookworm"

container:
  network: ""
  docker_host: "-"

server:
  connections:
    forgejo:
      url: http://192.168.1.50:3000/
      uuid: UUID-AFFICHE-PAR-FORGEJO
      token: JETON-AFFICHE-PAR-FORGEJO
```

Le runner s'exécute avec l'utilisateur `1001` : il doit être propriétaire de son dossier. Puis on démarre les nouveaux services :

```
$ sudo chown -R 1001:1001 ~/forge/runner
```

```
$ docker compose up -d
$ docker logs -f forgejo-runner
```

Au démarrage, quelques erreurs `cannot ping the docker daemon` sont **normales** : le runner redémarre tant que docker-in-docker n'est pas prêt. Il doit ensuite afficher :

```
... runner: ..., with version: v13.1.0, with labels: [docker], ephemeral: false, declared successfully
... [poller] launched
```

Dans Forgejo, le runner apparaît alors **en ligne**.

Question 15. Expliquez le label `docker:docker://node:20-bookworm` : à quoi sert la partie avant le premier « : » ? et la partie après `docker://` ?

5.4 Premier workflow

En suivant le workflow de la partie 4 (ticket, branche, PR), ajoutez au dépôt le fichier `.forgejo/workflows/tests.yml`

```
name: Tests
on: [push, pull_request]

jobs:
  tests:
    runs-on: docker
    steps:
      - uses: actions/checkout@v4
      - name: Tests unitaires
        run: python3 -m unittest discover -s tests -v
```

Dès le push de la branche, ouvrez l'onglet *Actions* du dépôt : suivez l'exécution du job et lisez son journal. Sur la PR, le résultat des tests s'affiche à côté des commits.

Question 16. Dans quel ordre le job exécute-t-il ses étapes ? Où la ligne `runs-on: docker` fait-elle le lien avec la configuration du runner ?

5.5 Exiger une CI verte pour fusionner

Dans la règle de protection de `main`, activez la **vérification des statuts** et ajoutez le contrôle :

```
Tests / tests (pull_request)
```

Ce nom n'est connu de la forge qu'après une première exécution du workflow.

5.6 Casser exprès

1. Créez une branche `10-prix-boisson` et passez `PRIX_BOISSON` de `20` à `25` dans `carotte/apdu.py`.
2. Poussez et ouvrez une PR. Le job échoue : lisez le journal et trouvez le test en échec.
3. Approuvez la PR et essayez de la fusionner : la forge doit **refuser**, car les contrôles de statut requis ne sont pas au vert.
4. Corrigez (le prix d'une boisson est bien 20 centimes dans le sujet de SAÉ), poussez : la CI repasse au vert et la fusion est possible.

Question 17. Mettez à jour la Definition of Done dans le `README.md` : quels critères sont désormais **vérifiés automatiquement** par la forge ?

5.7 Écrire un test

Le sujet de SAÉ indique qu'un crédit qui ferait dépasser la capacité de la carte est refusé (65535 centimes au maximum). Dans un nouveau ticket :

1. écrivez d'abord un test : `apdu_crediter(70000)` doit lever une `ValueError` ;
2. vérifiez que la CI passe (ou corrigez le code si le test échoue) ;
3. fusionnez par une PR.

Point de validation. Le runner est en ligne ; le workflow s'exécute à chaque push et PR ; une PR en échec a été bloquée ; `main` exige une CI verte ; au moins un test a été ajouté par l'équipe.

Pour la SAÉ

— Votre forge est l'outil de pilotage de la SAÉ : **tout** le code et le suivi y sont hébergés. Le lien du dépôt est à fournir à l'enseignant.

— Un sprint = un jalon, un tableau à jour, un tag de version et une version publiée.

— Le **responsable d'équipe** (rôle tournant, au plus 2 semaines) anime le sprint planning, tient le tableau Kanban à jour et veille au respect du workflow : il joue le rôle de **Scrum Master**.

— Sauvegardez régulièrement la forge : dossier `~/forge` complet, conteneurs arrêtés (`docker compose down`).

A Dépannage

Symptôme	Cause probable et solution
`docker compose up` : *port is already allocated*	Le port 3000 ou 2222 est déjà utilisé sur la machine (`ss -ltn`). Arrêter le service en conflit ou changer le port publié.
Les liens de clonage affichent `localhost`	`ROOT_URL` ou `SSH_DOMAIN` mal renseignés : vérifier `ADRESSE_IP` dans `.env`, puis `docker compose up -d`.
`Permission denied (publickey)`	Clé publique non déclarée dans Forgejo, mauvaise clé, ou oubli du port : `ssh://git@IP:2222/...`
`remote rejected ... pre-receive hook declined`	Push direct sur `main` protégée : passer par une branche et une PR.
Runner : `cannot ping the docker daemon` en boucle	Normal pendant quelques secondes. Si cela dure : `docker logs forgejo-dind`.
Runner : erreur d'accès à `config.yml` ou `.cache`	Dossier non possédé par l'utilisateur 1001 : `sudo chown -R 1001:1001 ~/forge/runner`.
Job bloqué « en attente »	Aucun runner en ligne avec le label demandé : comparer `runs-on` et les `labels` de `config.yml`.
Le job échoue à l'étape `checkout`	Le conteneur du job ne joint pas la forge : `ROOT_URL` doit contenir l'**adresse IP** de la machine, pas `localhost`.
Le contrôle `Tests / tests (pull_request)` n'est pas proposé	Le workflow ne s'est encore jamais exécuté sur une PR : ouvrir une PR, puis revenir à la règle de protection.

B Mémo Docker Compose

`docker compose up -d`	créer et démarrer les services en arrière-plan
`docker compose ps`	état des services
`docker compose logs -f <service>`	suivre le journal d'un service
`docker compose down`	arrêter et supprimer les conteneurs (les données restent)
`docker compose pull`	télécharger les nouvelles images
`docker exec -u git forgejo forgejo <cmd>`	commande d'administration Forgejo

C Références

— Documentation Forgejo, installation avec Docker : https://forgejo.org/docs/latest/admin/installation/docker/

— Forgejo Actions, installation du runner : https://forgejo.org/docs/latest/admin/actions/installation/docker/

— Forgejo Actions, enregistrement du runner : https://forgejo.org/docs/latest/admin/actions/registration/

— Sujet de SAÉ 5.01/5.02 « La Carotte Électronique ».