Du commit au déploiement

Pipeline CI/CD complet avec GitHub Actions, Docker et Orchestration sous N8N

Spécialement conçu pour vous les MICSI 27 !! • Formation DevOps • 1 journée (ou plus)

Programme de la journée

Bloc 1 — Culture DevOps & CI/CD (1h)

Bloc 2 — N8N et automatisation SI (45min)

Bloc 3 — TP1 : Pipeline CI + Docker (2h)

Bloc 4 — TP2 : Workflow n8n (1h)

Bloc 5 — Atelier final : Tout connecter (1h30)

Bloc 6 — Débrief et ouverture (15min)

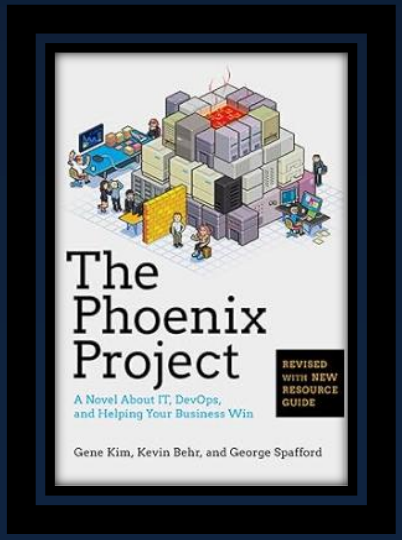
BLOC 1

Qui a déjà vécu une situation où l’équipe dev livre un code qui marche en local, mais qui plante en production ?

Culture DevOps & CI/CD — 1h de théorie

Culture DevOps : origines & enjeux

- Né en 2009 (DevOpsDays, Patrick Debois)
- Réponse au mur entre Dev et Ops
- Accélérer les cycles de livraison
- Réduire le Time-to-Market
- Améliorer la fiabilité des déploiements
- Culture de collaboration et d'amélioration continue



Les 3 piliers DevOps

Culture

Collaboration

Responsabilité partagée

Amélioration continue

Automatisation

CI/CD Pipelines

Infra as Code

Tests automatisés

ANSIBLE

Pulumi

HashiCorp Terraform

Mesure

Monitoring

Feedback loops

Métriques DORA

Prometheus

+

Grafana

https://xavki.blog/prometheus-grafana-tutoriaux-francais/

CI vs CD : définitions & différences

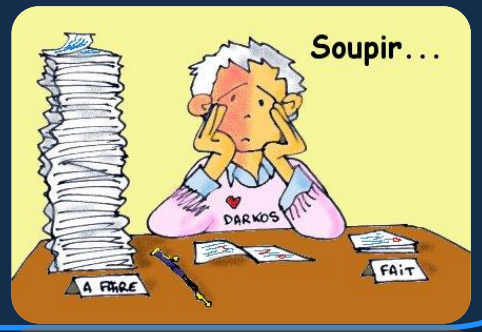
Continuous Integration

Intégration fréquente du code

Tests automatisés à chaque commit

Détection rapide des régressions

Build automatique de l'artefact



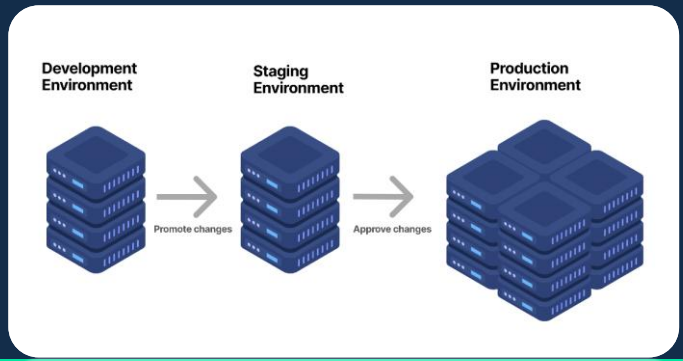
Continuous Delivery / Deployment

Delivery : prêt à déployer en 1 clic

Deployment : déploiement automatique

Environnements staging → production

Rollback automatique si échec



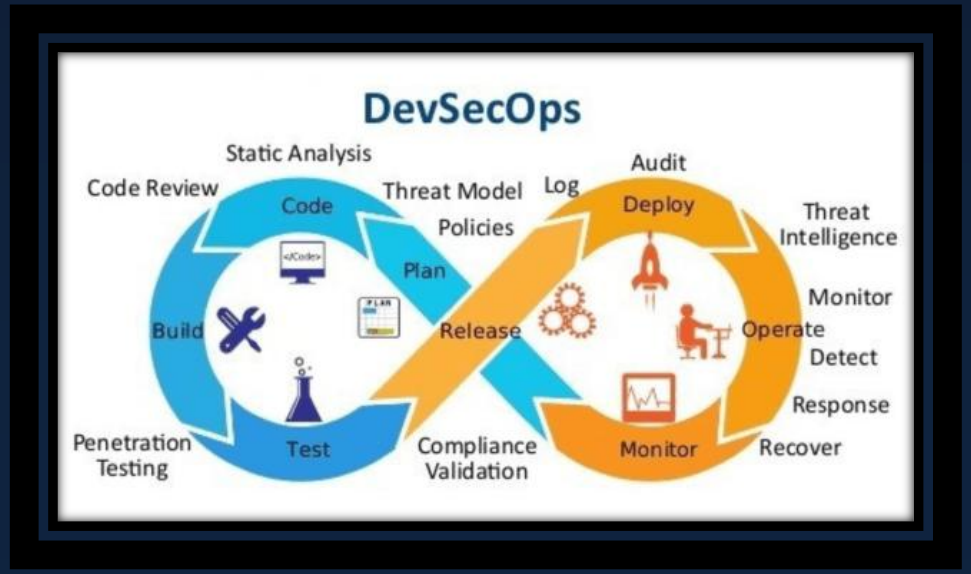
Bénéfices du CI/CD

- Réduction du time-to-market de 50 à 80%
- Détection des bugs en minutes, pas en jours
- Déploiements plus fréquents et plus sûrs
- Meilleure collaboration Dev ↔ Ops
- Traçabilité complète du code à la production
- Capacité de rollback instantané

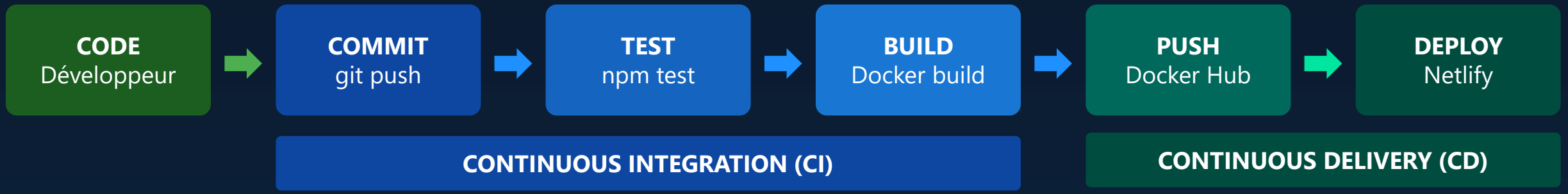
+ 100 Push / jour !

Spotify®

https://engineering.atspotify.com/2020/08/how-we-improved-developer-productivity-for-our-devops-teams



Fonctionnement CI/CD : le flux complet



Intégration Continue

→ Checkout + tests auto
→ Build image Docker
→ Push sur Docker Hub

Livraison Continue

→ Deploy auto sur Netlify
→ Staging puis Production
→ Rollback en 1 clic

Feedback Loop

→ Alertes Discord / Email
→ Métriques DORA
→ Amélioration continue

GitHub Actions orchestre tout le pipeline • Chaque étape est automatisée • Feedback en temps réel

Tour d'horizon des outils CI/CD

GitHub Actions

CI/CD natif GitHub

YAML workflows

GitLab CI

Pipeline intégré

.gitlab-ci.yml

Jenkins

Open source

Jenkinsfile

Docker

Conteneurisation

Dockerfile → Image

Netlify

Deploy frontend

Auto depuis Git

n8n

Automatisation

Workflow low-code

GitHub Actions : concepts clés

Workflow

Fichier YAML dans .github/workflows/

Github Actions

Event

push, pull_request schedule, manual

Job

Unité d'exécution Parallèle ou séquentiel

Step

Action individuelle run: ou uses:

Runner

Machine d'exécution ubuntu-latest

Secrets & Variables

secrets.DOCKER_USERNAME • secrets.DOCKER_PASSWORD • secrets.NETLIFY_TOKEN
Stockés chiffrés dans Settings → Secrets → Actions

Anatomie d'un workflow YAML

```yaml
name: CI Pipeline
on:
  push:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: npm test
      - run: docker build -t app .
      - run: docker push app
```

← Nom du workflow

← Déclencheur (event)

← Job "build"

← Runner GitHub

← Steps (actions)

DÉMO LIVE

Pipeline GitHub Actions en action — du push au Docker Hub

BLOC 2

N8N et automatisation SI — 45min de théorie

Positionnement : No-code → Low-code → Code

No-Code

Zapier, Make

Interface visuelle

Aucun code requis

Low-Code

n8n, Retool

Nodes + expressions

Code optionnel

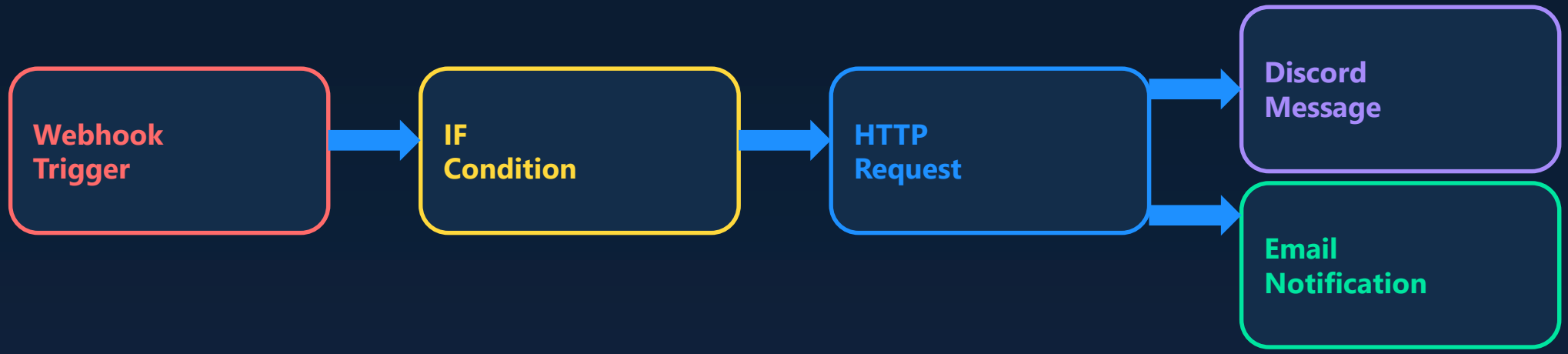
Full Code

Scripts Python

Terraform, Ansible

Contrôle total

n8n : concepts clés



Workflow = enchaînement de nodes • Node = action unitaire • Trigger = déclencheur

Webhook = point d'entrée HTTP • Credentials = authentification sécurisée • Expressions = données dynamiques

Cas d'usage métier en entreprise SI

CI/CD Notifications

Pipeline → Slack/Discord

alerte échec build

Monitoring

Alertes infra → ticket

Jira/ServiceNow auto

Onboarding

Nouvel employé → comptes

AD, email, Slack, VPN

Sécurité

CVE détectée → scan

rapport → équipe sécu

Data Sync

CRM → ERP sync

transformation + routing

Reporting

KPI quotidiens → PDF

email auto aux managers

DÉMO LIVE

Webhook n8n → notification Discord en temps réel

TP

Bloc 3 — TP1

Pipeline CI + Docker — 2h de pratique

Objectif : Écrire un pipeline GitHub Actions complet
test → build → push Docker Hub

TP1 — Vue d'ensemble du pipeline

- Repo template avec app React/Vue/HTML
- Dockerfile fonctionnel dans le repo
- Compte Docker Hub par groupe
- Accès GitHub avec droits admin sur le fork
- Durée estimée : 2h (30min par étape)

TP1

TP1 — Étape 1 : Fork & structure du repo

- Fork le repo template sur GitHub
- Cloner en local : git clone <votre-fork>
- Structure attendue :
- 📁 src/ — code source de l'app
- 📁 public/ — assets statiques
- 📄 Dockerfile
- 📄 package.json
- 📄 .github/workflows/ci.yml (à créer)
- Vérifier : npm install && npm start

TP1 — Étape 2 : Écriture du ci.yml

```yaml
name: CI Pipeline
on:
  push:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: npm ci
      - run: npm test

  build-push:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: docker build -t $IMAGE .
      - run: docker push $IMAGE
```

① Créer .github/workflows/ci.yml

② Job test : checkout + npm test

③ Job build-push : needs: test

④ Docker build + tag avec le nom de votre image

⑤ Docker push vers Docker Hub

TP1 — Étape 3 : Secrets & vérification

- Configurer les secrets GitHub :
- Settings → Secrets → Actions → New secret
- DOCKER_USERNAME = votre username Docker Hub
- DOCKER_PASSWORD = votre access token
- Push sur main pour déclencher le pipeline
- Vérifier dans l'onglet Actions de GitHub
- Vérifier l'image sur Docker Hub
- Test local : docker run -p 3000:80 <image>

TP

Bloc 4 — TP2

Workflow n8n — 1h de pratique

Objectif : Créer un workflow n8n
Webhook → IF condition → Discord / Email

TP2

TP2 — Création du workflow n8n

- Se connecter à l'instance n8n fournie
- Créer un nouveau workflow
- Ajouter un node Webhook (méthode POST)
- Ajouter un node IF (condition sur le status)
- Si success → branche Discord
- Si failure → branche Email
- Ajouter le node Discord (Bot message)
- Tester avec un curl sur l'URL du webhook

TP2 — Enrichir le message Discord

Payload du webhook

```
{
  "status": "success",
  "image": "user/app:latest",
  "branch": "main",
  "author": "alice",
  "commit": "feat: add login",
  "repo": "team/frontend-app",
  "timestamp": "2025-01-15T10:30"
}
```

Message Discord enrichi

✅ Deploy réussi !

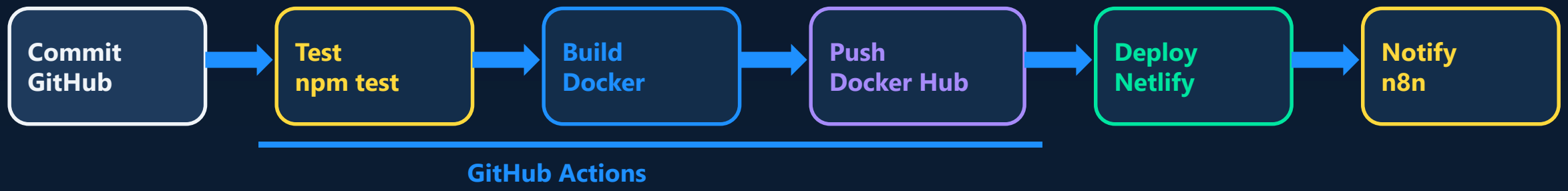
🎁 Image : user/app:latest
🌿 Branche : main
👤 Auteur : alice
💬 Commit : feat: add login
🕐 15 Jan 2025, 10:30

💡 **Utiliser les expressions n8n : {{ $json.image }}, {{ $json.author }}, {{ $json.branch }}**

BLOC 5

Atelier final : Tout connecter — 1h30

Pipeline complet : du commit au Discord



n8n Workflow

Flux complet automatisé : 1 commit → tests → image Docker → déploiement → notification
Temps moyen : ~3 minutes du commit au message Discord

Atelier — Ajouter l'étape notify

```
notify:
    needs: build-push
    runs-on: ubuntu-latest
    steps:
      - name: Notify n8n
        run: |
          curl -X POST $WEBHOOK \
            -H 'Content-Type: json' \
            -d '{
              "status": "success",
              "image": "$IMAGE",
              "branch": "$BRANCH",
              "author": "$AUTHOR"
            }'
```

① Ajouter le job notify après build-push

② needs: build-push (exécution séquentielle)

③ curl POST vers l'URL du webhook n8n

④ Payload JSON avec les infos du build

⑤ Variables d'env pour les secrets sensibles

Challenge : conditionner le push

- Objectif : le push Docker Hub ne se fait que sur main
- Utiliser la condition if dans GitHub Actions
- if: github.ref == 'refs/heads/main'
- Les tests tournent sur toutes les branches
- Le build+push uniquement sur main
- Bonus : ajouter un tag avec le SHA du commit
- docker build -t app:${{ github.sha }}

DÉMO LIVE PAR GROUPE

5 minutes par groupe — Montrez votre pipeline de bout en bout !

BLOC 6

Débrief et ouverture — 15min

GitHub Actions vs Netlify CI

GitHub Actions (Custom)

- ✅ Contrôle total du pipeline
- ✅ Multi-step : test, build, deploy
- ✅ Docker, notifications, etc.
- ✅ Réutilisable (marketplace)
- ⚠️ Configuration YAML manuelle
- ⚠️ Debugging plus complexe

Netlify CI (Managed)

- ✅ Zero configuration
- ✅ Deploy automatique sur push
- ✅ Preview par branche / PR
- ✅ CDN global intégré
- ⚠️ Limité au frontend statique
- ⚠️ Pas de pipeline custom

Bonnes pratiques DevOps

Secrets

- Jamais en clair dans le code
- Utiliser les secrets GitHub
- Rotation régulière
- Access tokens > passwords

Environments

- Séparer dev / staging / prod
- Protection rules sur prod
- Approvals obligatoires
- Variables par environnement

Branches

- Protection de main
- Pull requests obligatoires
- Code review avant merge
- CI doit passer (status check)

Ouverture : n8n + Intelligence Artificielle

- Analyse automatique de logs avec LLM
- Résumé automatique des PRs/commits
- Détection d'anomalies dans les métriques
- Génération de rapports d'incident
- Chatbot DevOps interne (Slack/Discord)
- n8n + OpenAI : workflow intelligent en quelques nodes

Ce que vous avez appris aujourd'hui

Culture DevOps

CI/CD Pipelines

GitHub Actions

Docker Build

n8n Workflows

Notifications

Du commit au déploiement — vous maîtrisez le pipeline complet !

Questions ?

Merci pour votre participation !

FICHE TP 1

Pipeline CI + Docker — 2h

2 heures

OBJECTIF

Configurer un pipeline GitHub Actions complet : test → build → push Docker Hub → run local
Chaque groupe produit : pipeline vert + image Docker publique + app qui tourne en local

Étapes de la fiche

1	Fork du repo template	15 min
2	Configurer les secrets GitHub	15 min
3	Compléter ci.yml	30 min
4	Push et observer le pipeline	20 min
5	Vérifier sur Docker Hub	10 min
6	Déployer en local	15 min

ÉTAPE 1/6 — Fork du repo template

 15 min

1. Aller sur le repo template : https://github.com/JulienDifferangles/devops-tp
2. Cliquer sur « Fork » en haut à droite
3. Nommer le fork : devops-tp-<nom-du-groupe>
4. Cloner le fork en local
5. Vérifier que les tests passent avant de continuer

✓ VALIDATION

□ Fork visible dans votre compte GitHub
□ npm test passe sans erreur
□ App visible sur localhost:3000

⚙ COMMANDES

```
git clone https://github.com/<votre-fork>.git
cd devops-tp-<groupe>
npm install
npm test
```

ASTUCE

Décochez « Copy the main branch only » au fork si le repo a d'autres branches utiles.

Si npm install échoue, vérifiez votre version de Node (v18+).

EN CAS DE BLOCAGE

- Permission denied au clone → Vérifiez que vous êtes connecté à GitHub (gh auth login ou SSH key)
- Tests échouent → Vérifiez Node v18+, supprimez node_modules et relancez npm install

ÉTAPE 2/6 — Configurer les secrets GitHub

 15 min

1. Sur Docker Hub : Account Settings → Security → New Access Token
   Nom du token : github-actions | Permission : Read & Write
   ⚠ Copier le token maintenant — il ne sera plus affiché !
2. Sur GitHub, dans votre fork : Settings → Secrets and variables → Actions
3. Créer deux secrets :

⚙ SECRETS À CRÉER

```
DOCKER_USERNAME  →  votre username Docker Hub
DOCKER_TOKEN     →  le token copié à l'étape 1
```

✓ VALIDATION

- □ 2 secrets visibles dans Settings → Secrets
- □ Noms exacts respectés

ASTUCE

Utilisez un Access Token Docker Hub, jamais votre mot de passe !

Le nom du secret doit correspondre exactement à ce qui est dans le ci.yml.

EN CAS DE BLOCAGE

- « Settings » non visible → Vous n'avez pas les droits admin sur le fork. Re-fork depuis votre propre compte.
- Token Docker Hub expiré → Regénérez-en un nouveau dans Account Settings → Security

ÉTAPE 3/6 — Compléter le fichier ci.yml

 30 min

⚙ .github/workflows/ci.yml

```
name: CI Pipeline
on:
  push:
    branches: [main]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: npm ci
      - run: npm test
  build-push:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: echo $DOCKER_TOKEN | docker login
            -u $DOCKER_USERNAME --password-stdin
      - run: docker build -t $USER/app:latest .
      - run: docker push $USER/app:latest
```

📝 À FAIRE

1. Créer le dossier : .github/workflows/

2. Créer le fichier : ci.yml

3. Copier le code ci-contre

✓ VALIDATION

- □ Fichier ci.yml commité
- □ Syntaxe YAML valide
- □ Secrets référencés

💡 ASTUCE

YAML est sensible à l'indentation (espaces, pas de tabulations !). Utilisez un validateur en ligne : yamllint.com

ÉTAPE 4/6 — Push et observer le pipeline

 20 min

1. Commit et push le fichier ci.yml
2. Sur GitHub → onglet Actions → voir le workflow en cours
3. Cliquer sur le run pour voir le détail de chaque étape
4. Provoquer un échec : casser un test volontairement
5. Observer l'échec dans Actions (icône rouge)
6. Corriger le test et repousser → pipeline redevient vert

⚙ COMMANDES

```
git add .github/workflows/ci.yml
git commit -m "feat: add CI pipeline"
git push origin main
```

✓ VALIDATION

□ Pipeline vert dans Actions
□ Job test ✓ puis build-push ✓
□ Échec provoqué puis corrigé

ASTUCE

Pour casser un test facilement :
ouvrez un fichier .test.js
et changez un expect().

Le pipeline est déclenché uniquement sur push vers main.

EN CAS DE BLOCAGE

- Workflow pas déclenché → Vérifiez que vous avez pushé sur main (pas une autre branche)
- Erreur « invalid workflow file » → Vérifiez l'indentation YAML et les noms de dossier (.github/workflows/)

ÉTAPE 5/6 — Vérifier sur Docker Hub

 10 min

1. Aller sur hub.docker.com → se connecter
2. Vérifier que le repository <username>/app existe
3. Cliquer sur le repo → onglet Tags
4. L'image « latest » doit apparaître avec la date du dernier push
5. Noter l'URL complète de l'image : docker.io/<username>/app:latest

✓ VALIDATION

- □ Image visible sur Docker Hub
- □ Tag « latest » présent
- □ Date = date du dernier push

🚀 CHALLENGE (groupes rapides)

Ajoutez un tag dynamique avec le SHA du commit : docker build -t <user>/app:${{ github.sha }} .
Vérifiez que le tag SHA apparaît dans Docker Hub en plus du tag « latest ».

ÉTAPE 6/6 — Déployer en local

 15 min

1. Ouvrir un terminal
2. Pull l'image depuis Docker Hub
3. Lancer le conteneur
4. Ouvrir le navigateur sur http://localhost:3000
5. Vérifier que l'app fonctionne correctement

✓ VALIDATION

- □ App accessible localhost:3000
- □ L'image vient de Docker Hub (pas un build local)

⚙ COMMANDES

```
docker pull <username>/app:latest
docker run -d -p 3000:3000 <username>/app:latest
# Ouvrir http://localhost:3000
```

ASTUCE

Port déjà utilisé ?
docker stop $(docker ps -q)
ou changez le port :
docker run -p 3001:3000 ...

LIVRABLE FINAL TP1

- □ Pipeline vert visible dans GitHub Actions (montrer l'onglet Actions)
- □ Image publique sur Docker Hub (montrer le repository + tag)
- □ App qui tourne en local depuis l'image Docker (ouvrir localhost:3000 devant le formateur)

FICHE TP 2

Workflow n8n — 1h

1 heure

OBJECTIF

Créer un workflow n8n : réception webhook CI → routage conditionnel → notification Discord/Email
Chaque groupe produit : workflow n8n actif + URL webhook + 2 notifications distinctes (succès/échec)

Étapes de la fiche

1	Se connecter à l'instance n8n	5 min
2	Ajouter le nœud Webhook	10 min
3	Tester le webhook manuellement	10 min
4	Ajouter le nœud IF (condition)	10 min
5	Ajouter les nœuds de notification	15 min
6	Tester les deux scénarios	10 min

ÉTAPE 1/6 — Se connecter à l'instance n8n

 5 min

1. Ouvrir l'URL fournie par le formateur
2. Se connecter avec les identifiants fournis
3. Créer un nouveau workflow vide
4. Le nommer : CI-Notify-<nom-du-groupe>

✓ VALIDATION

- □ Connecté à l'interface n8n
- □ Workflow vide créé
- □ Workflow nommé correctement

ASTUCE

Si Docker local, lancez :
docker run -p 5678:5678
n8nio/n8n
Puis ouvrez localhost:5678

EN CAS DE BLOCAGE

- Page blanche / erreur 502 → L'instance n8n n'est pas démarrée, prévenez le formateur
- « Invalid credentials » → Vérifiez les identifiants ou créez un nouveau compte

ÉTAPE 2/6 — Ajouter le nœud Webhook

 10 min

1. Cliquer sur + pour ajouter un nœud
2. Chercher « Webhook » → ajouter
3. Configurer : HTTP Method = POST
4. Copier l'URL du webhook (bouton Copy)
5. Activer le workflow (toggle en haut)
6. ⚠ Garder l'URL précieusement pour le TP3 !

⚙ URL DU WEBHOOK (noter ici)

```
https://votre-instance.n8n.cloud/
  webhook/xxxxxxxx-xxxx-xxxx

→ Collez votre URL : _________________
```

✓ VALIDATION

- □ Nœud Webhook ajouté
- □ Method = POST
- □ URL copiée et notée
- □ Workflow activé (toggle ON)

ASTUCE

Il existe deux URLs :
- Test URL (pour debug)
- Production URL

Utilisez la Production URL pour le TP3.

EN CAS DE BLOCAGE

- « Webhook not found » → Le workflow n'est pas activé. Activez le toggle !
- URL différente à chaque test → Utilisez la Production URL, pas la Test URL

ÉTAPE 3/6 — Tester le webhook manuellement

10 min

1. Ouvrir un terminal (ou Postman / Thunder Client)
2. Envoyer un payload de test avec curl
3. Vérifier dans n8n que les données arrivent
4. Cliquer sur le nœud Webhook → voir l'output

⚙ COMMANDE CURL

```
curl -X POST <VOTRE_URL_WEBHOOK> \
  -H "Content-Type: application/json" \
  -d '{"status":"success","branch":"main",
       "author":"votre-pseudo",
       "image":"user/app:latest"}'
```

✓ VALIDATION

- □ Code 200 en réponse
- □ Données visibles dans n8n
- □ Les 4 champs reçus

ASTUCE

Windows ? Utilisez PowerShell :
Invoke-WebRequest -Uri
<URL> -Method POST
-Body '{"status":"success"}'

EN CAS DE BLOCAGE

- « Connection refused » → L'URL n'est pas bonne ou le workflow n'est pas activé
- Pas de données dans n8n → Vérifiez le Content-Type: application/json dans le header

ÉTAPE 4/6 — Ajouter le nœud IF (condition)

 10 min

1. Ajouter un nœud IF après le Webhook
2. Configurer la condition :
   Value 1 : {{ $json.status }}
   Opération : Equal
   Value 2 : success
3. Connecter la sortie « true » vers la branche succès
4. Connecter la sortie « false » vers la branche échec

⚙ EXPRESSION N8N

```
{{ $json.status }} == "success"
→ true = notification succès | false = notification échec
```

✓ VALIDATION

□ Nœud IF ajouté
□ 2 branches connectées
□ Expression correcte

ASTUCE

Le nœud IF a deux sorties :
- output 0 = true (condition respectée)
- output 1 = false
Vérifiez les connexions !

ÉTAPE 5/6 — Nœuds de notification

 15 min

1. Branche succès → ajouter nœud Discord (ou Email)
2. Configurer le message de succès :
3. Branche échec → ajouter un second nœud Discord/Email
4. Message d'échec avec mention urgente
5. Utiliser les expressions n8n pour enrichir le message

Message SUCCÈS

```
✓ Build réussi !
Branche : {{ $json.branch }}
Auteur : {{ $json.author }}
Image : {{ $json.image }}
```

Message ÉCHEC

```
✗ ÉCHEC Build @here
Branche : {{ $json.branch }}
Auteur : {{ $json.author }}
Voir le run : {{ $json.run_url }}
```

ASTUCE — Discord Webhook

Pour Discord : Server Settings → Integrations → Webhooks → New Webhook → Copy URL. Collez cette URL dans les Credentials n8n.

ÉTAPE 6/6 — Tester les deux scénarios

 10 min

1. Envoyer un payload avec status = "success" → notification verte
2. Envoyer un payload avec status = "failure" → notification rouge
3. Vérifier les messages sur Discord ou dans la boîte email
4. Vérifier que les variables dynamiques sont bien remplacées

✓ VALIDATION

□ Notification succès reçue
□ Notification échec reçue
□ Variables dynamiques OK

⚙ TEST ÉCHEC

```
curl -X POST <VOTRE_URL> \
  -H "Content-Type: application/json" \
  -d '{"status":"failure","branch":"dev","author":"test"}'
```

LIVRABLE FINAL TP2

□ Workflow n8n actif avec URL webhook notée (nécessaire pour le TP3)
□ Deux notifications distinctes reçues : succès (message vert) et échec (message rouge)
□ Messages enrichis avec les variables dynamiques (branche, auteur, image)

CHALLENGE (groupes rapides)

Ajoutez un 3ème nœud qui enregistre chaque notification dans un Google Sheet (nœud Google Sheets).
Colonnes : date, statut, branche, auteur → historique de tous les builds !

FICHE TP 3

Atelier final : Tout connecter — 1h30

1h30

OBJECTIF FINAL

Connecter le pipeline CI au workflow n8n pour un flux bout-en-bout automatique :
commit → test → build → push Docker Hub → deploy Netlify → notify n8n → Discord

Étapes de la fiche

1	Ajouter le secret webhook dans GitHub	5 min
2	Ajouter l'étape notify dans ci.yml	20 min
3	Tester le flux complet	15 min
4	Provoquer un échec intentionnel	10 min
5	**CHALLENGE — Conditionner le push Docker**	15 min
6	**CHALLENGE BONUS — Ajouter Netlify**	25 min

ÉTAPE 1/6 — Ajouter le secret webhook dans GitHub

 5 min

1. Sur GitHub → Settings → Secrets and variables → Actions
2. Créer un nouveau secret :
   Nom : N8N_WEBHOOK_URL
   Valeur : l'URL webhook copiée dans le TP2
3. Vérifier que vous avez maintenant 3 secrets

⚙ SECRETS GITHUB (vérification)

```
DOCKER_USERNAME   → votre username Docker Hub
DOCKER_TOKEN      → votre token Docker Hub
N8N_WEBHOOK_URL   → URL du webhook n8n
```

✓ VALIDATION

- □ 3 secrets dans GitHub Actions
- □ N8N_WEBHOOK_URL ajouté
- □ URL du TP2 correcte

ASTUCE

Copiez l'URL exacte du TP2.
Ne pas confondre Test URL
et Production URL !
L'URL ne doit pas avoir
de / final.

🚧 EN CAS DE BLOCAGE

• Secret non visible après création → C'est normal ! GitHub masque les valeurs. Re-créez-le si besoin.

ÉTAPE 2/6 — Ajouter l'étape notify dans ci.yml

 20 min

1. Ouvrir le fichier ci.yml
2. Ajouter un job « notify » après build-push
3. Utiliser curl pour appeler le webhook n8n
4. Construire le payload JSON avec les variables GitHub
5. Le job notify doit toujours s'exécuter (même si build échoue)

⚙ JOB NOTIFY À AJOUTER

```
notify:
  needs: build-push
  if: always()
  runs-on: ubuntu-latest
  steps:
    - run: |
        curl -X POST "$N8N_WEBHOOK_URL" \
        -H "Content-Type: application/json" \
        -d '{"status":"${{ job.status }}",
             "branch":"${{ github.ref_name }}",
             "author":"${{ github.actor }}"}'
```

✓ VALIDATION

- □ Job notify ajouté
- □ if: always() présent
- □ Payload avec 3 variables

ASTUCE

if: always() est crucial !
Sans ça, le notify ne
s'exécute que si build-push
réussit → pas de notif échec.

EN CAS DE BLOCAGE

- « Unexpected value » → Vérifiez les guillemets dans le payload JSON (simples dehors, doubles dedans)

ÉTAPE 3/6 — Tester le flux complet

 15 min

1. Faire un petit changement dans le code (ex: modifier un texte)
2. Commit et push sur main
3. Observer le pipeline dans GitHub Actions : test → build-push → notify
4. Vérifier sur Docker Hub : image mise à jour
5. Vérifier sur Discord/Email : notification reçue automatiquement

✓ VALIDATION

- □ Pipeline complet vert
- □ Image Docker Hub à jour
- □ Notification reçue !

⚙ COMMANDES

```
git add .
git commit -m "feat: test full pipeline"
git push origin main
# Puis ouvrir l'onglet Actions sur GitHub
```

ASTUCE

Suivez le pipeline en temps réel dans l'onglet Actions. Chaque job prend ~1-2 min. Le notify arrive en dernier.

EN CAS DE BLOCAGE

• Notify échoue → Vérifiez l'URL webhook dans le secret. Testez-la manuellement avec curl en local d'abord.

ÉTAPE 4/6 — Provoquer un échec intentionnel

 10 min

1. Ouvrir un fichier de test (.test.js ou .spec.js)
2. Modifier un expect() pour qu'il échoue
3. Commit et push sur main
4. Observer dans Actions : le job test échoue (rouge)
5. Le job build-push est skippé (needs: test)
6. Le job notify s'exécute quand même (if: always()) → notification d'échec !

⚙ EXEMPLE DE TEST CASSÉ

```
// Dans le fichier de test, changez :
expect(result).toBe(true);
// en :
expect(result).toBe(false); // CASSER
```

✓ VALIDATION

- □ Job test = rouge
- □ Build-push = skippé
- □ Notify = notification échec

ASTUCE

N'oubliez pas de corriger le test après ! Recommittez pour repasser en vert.

C'est le scénario réel d'un bug détecté par la CI.

ÉTAPE 5/6 — 🚀 CHALLENGE : Condition branche

⏱ 15 min

OBJECTIF : Le push Docker Hub ne doit se faire que depuis la branche main.
Les pushes depuis d'autres branches (dev, feature/*) ne doivent que tester.

1. Ajouter une condition if sur le job build-push
2. Créer une branche dev et pusher dessus
3. Vérifier que seul le job test s'exécute
4. Merger dev dans main → le push Docker se déclenche

⚙️ MODIFICATION CI.YML

```
# Modifier le trigger pour toutes les branches
on:
  push:
    branches: ['**']  # ou [main, dev]

# Ajouter la condition sur build-push
  build-push:
    if: github.ref == 'refs/heads/main'
```

✅ VALIDATION

- ☐ Push dev → test only
- ☐ Push main → test + build
- ☐ Notify fonctionne toujours

💡 ASTUCE

Commandes Git utiles :
git checkout -b dev
git push origin dev

Puis sur GitHub :
Pull Request dev → main
Merge → déclenche le build

ÉTAPE 6/6 — 🚀 CHALLENGE BONUS : Netlify

25 min

1. Aller sur netlify.com → Sign up with GitHub
2. « Add new site » → « Import an existing project »
3. Sélectionner votre fork → laisser les paramètres auto
4. Cliquer « Deploy » → observer le déploiement automatique
5. Pousser un commit → Netlify redéploie automatiquement !
6. Discussion : GitHub Actions vs Netlify CI — quand utiliser quoi ?

💼 LIVRABLE FINAL TP3 — DÉMO LIVE

□ Démo 5 min devant les autres groupes
□ Flux complet : commit → test → build → push → notify → Discord
□ Chaque membre explique son rôle dans le pipeline

🏆 BARÈME INDICATIF

Pipeline vert + Docker Hub + notification : 15/20
Challenge branche : +3 pts | Challenge Netlify : +2 pts
Qualité de la démo et compréhension du flux : appréciation

✅ VALIDATION

□ Site déployé sur Netlify
□ Auto-deploy fonctionne
□ URL publique accessible

💡 COMPARAISON

GitHub Actions = contrôle total
(tests, Docker, notifications)

Netlify CI = magie auto
(zero config, CDN, previews)

→ Complémentaires, pas rivaux !