Présentation P7
Mise en place d'une chaîne CI/CD avec l'Infrastructure as Code (IaC)

Août 2025

CréaLogiciels

Soutenue par
Ayaka MAJAULT

Contexte

Automatiser le déploiement continu des applications (frontend/backend) pour une livraison plus rapide et une gestion d'infrastructure efficace.

1. Analyse et Planification

- Analyse des stacks technologiques existantes et des besoins CD.
- Élaboration d'un plan d'action et des solutions IaC adaptées.

2. Conception et Développement de l'IaC

- Conteneurisation : **Docker** pour isoler les applications.
- Orchestration : **Kubernetes** et **Helm** pour gérer les déploiements et la performance.
- Déploiement : **Terraform** et **Ansible** pour provisionner l'infrastructure **AWS (EKS)** et automatiser le déploiement.

3. Intégration et Validation

- Mise en place d'un pipeline **GitLab CI/CD** automatisé.
- Intégration du build, des tests, du scan de sécurité (**Trivy**) et du déploiement.
- Validation par des tests rigoureux de l'application et de l'infrastructure.

4. Documentation et Bonnes Pratiques

- Création d'un **Document préparatoire** et d'un **Plan de CD complet** avec captures d'écran.
- Rédaction d'un **Guide des bonnes pratiques CD** pour la maintenance future.

Problèmes Actuels - Synthèse

Catégorie	Problèmes détectés	Solutions proposées
CI/CD	Pipelines Jenkins peu développés Déclenchements manuels Faible couverture de tests	Pipelines GitLab CI/CD avancés (tests, build, scan, déploiement) Automatisation complète
Sécurité	Accès non standardisé aux scripts Pas de scans de vulnérabilités systématiques Journalisation limitée	Intégration de Trivy pour les images Gestion sécurisée des secrets (GitLab, Vault) Audit et logs renforcés
Qualité logicielle	Tests automatisés manquants Documentation obsolète ou incomplète	Ajout de tests unitaires/intégration Documentation centralisée et à jour
Git / GitLab	Utilisation limitée à Jenkins Pas de pipelines GitLab Secrets non intégrés	GitLab CI/CD comme orchestrateur unique Utilisation de GitLab Secrets
Registre Docker	Images construites manuellement via scripts Pas de cache ni scan intégré	CI/CD avec Kaniko Scan Trivy Cache Docker Registry
Portabilité	Déploiements manuels peu reproductibles Pas d'orchestration	Conteneurisation systématique Kubernetes pour orchestrer et standardiser
Maintenance	Forte dépendance aux scripts Bash Compétences DevOps inégales	Formation Kubernetes/Ansible/Terraform Standardisation des processus
Orchestration	Kubernetes absent Ansible sous-utilisé	Déploiement Kubernetes (k3s local, AWS avec Terraform) Automatisation via Ansible
Déploiement d'infrastructure	Provisionnement manuel Pas d'IaC (Terraform absent)	Terraform pour l'IaC Ansible pour la configuration

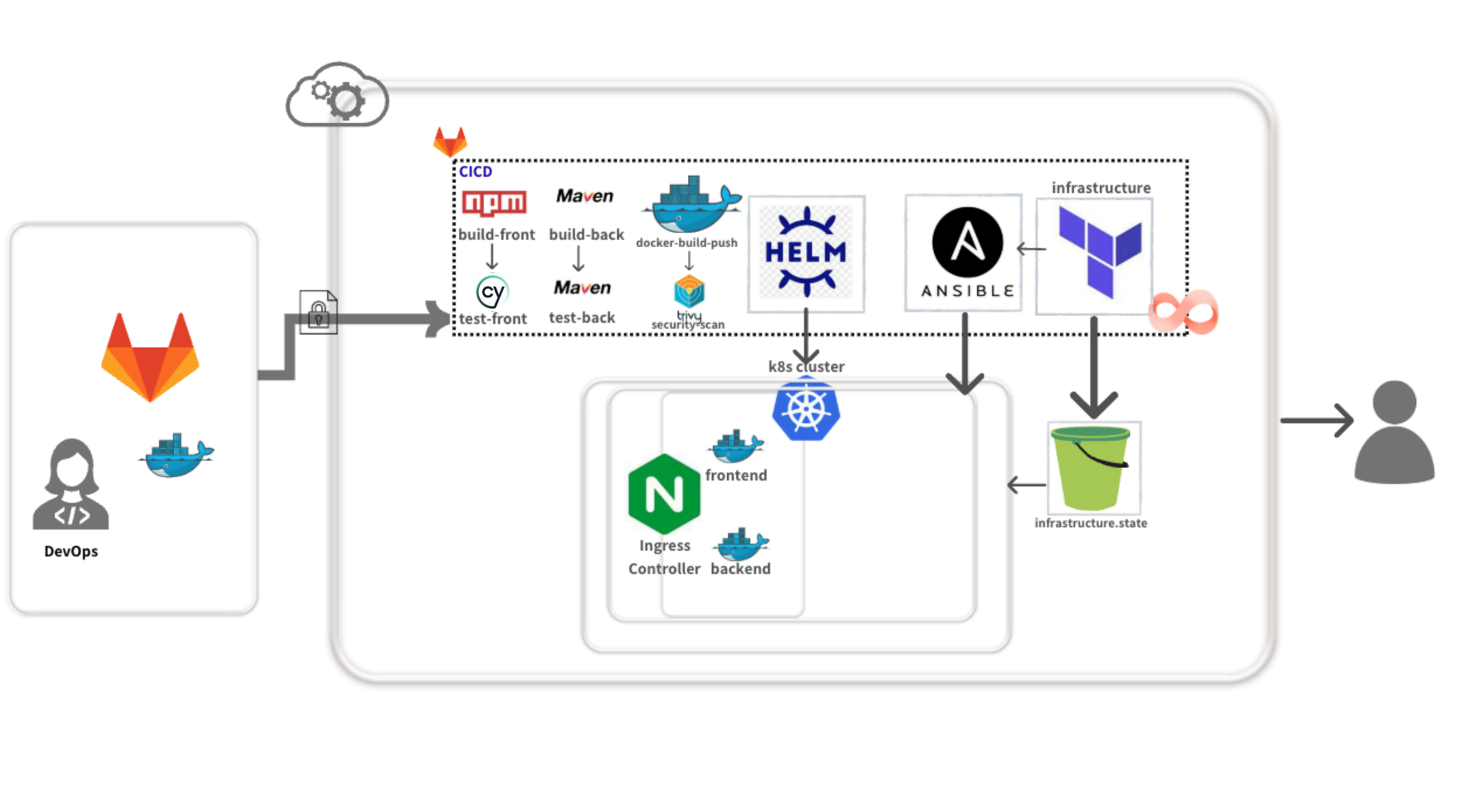
💡 Constat Clé

Automatisation et orchestration insuffisantes: Jenkins + scripts manuels, et tests et des déploiements sont peu automatisés, k8s/Terraform manquent → agilité limitée

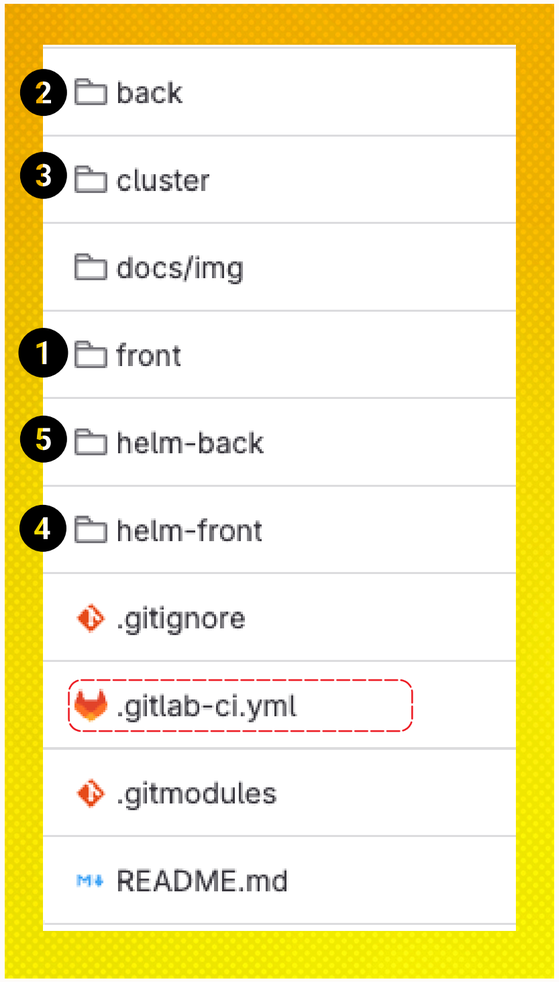
🎯 Objectif

Standardiser et automatiser avec des outils modernes (GitLab CI/CD, K8s, Terraform, Ansible)

Architecture globale



Structure du Repo



1

```
.
├ README.md
├ front
│  ├ Dockerfile
│  ├ nginx.conf
│  ├ tsconfig.app.json
│  ├ README.md
│  ├ package-lock.json
│  ├ tsconfig.json
│  ├ angular.json
│  ├ package.json
│  ├ tsconfig.spec.json
│  ├ karma.conf.js
│  └ src
│     ├ app
│     │  ├ app.component.html
│     │  ├ app.component.scss
│     │  ├ app.component.spec.ts
│     │  ├ app.component.ts
│     │  ├ app.module.ts
│     │  ├ model
│     │  │  └ joke.model.ts
│     │  └ services
│     │     ├ jokes.service.spec.ts
│     │     └ jokes.service.ts
│     ├ assets
│     ├ environments
│     │  ├ environment.prod.ts
│     │  └ environment.ts
│     ├ favicon.ico
│     ├ index.html
│     ├ main.ts
│     ├ polyfills.ts
│     ├ proxy.config.json
│     ├ styles.scss
│     └ test.ts
```

2

```
├ back
│  ├ Dockerfile
│  ├ mvnw
│  ├ mvnw.cmd
│  ├ pom.xml
│  └ src
│     ├ main
│     │  ├ java
│     │  │  └ com
│     │  │     └ openclassrooms
│     │  │        └ bobapp
│     │  │           ├ BobappApplication.java
│     │  │           ├ controller
│     │  │           │  └ JokeController.java
│     │  │           ├ data
│     │  │           │  └ JsonReader.java
│     │  │           ├ model
│     │  │           │  └ Joke.java
│     │  │           └ service
│     │  │              └ JokeService.java
│     │  └ resources
│     │     ├ application.properties
│     │     └ json
│     │        └ jokes.json
│     └ test
│        └ java
│           └ com
│              └ openclassrooms
│                 └ bobapp
│                    └ BobappApplicationTests.java
```

3

```
├ cluster
│  ├ ansible
│  │  ├ deploy-eks.yml
│  │  ├ deploy.yml
│  │  └ inventory
│  │     └ roles
│  │        └ app
│  │           └ tasks
│  │              └ main.yml
│  └ terraform
│     ├ main.tf
│     ├ output.tf
│     ├ provider.tf
│     ├ terraform.tfstate
│     ├ terraform.tfvars
│     └ variables.tf
├ docs
│  └ img
│     └ devops.jpg
```

4

```
├ helm-front
│  ├ Chart.yaml
│  ├ values-main.yaml
│  └ templates
│     ├ _helpers.tpl
│     ├ deployment.yaml
│     ├ ingress.yaml
│     └ service.yaml
```

5

```
└ helm-back
   ├ Chart.yaml
   ├ values-main.yaml
   └ templates
      ├ _helpers.tpl
      ├ deployment.yaml
      ├ ingress.yaml
      └ service.yaml
```

.gitlab-ci.yml

Build (Compilation)
Frontend : Compilation **Angular** avec **npm**.
Backend : Compilation **Java** avec **Maven**.
👉 Résultat : artefacts prêts pour les tests et la construction des images.

Test (Vérification)
Frontend : Tests end-to-end avec **Cypress**.
Backend : Tests unitaires avec **Maven (JUnit)**.
👉 Résultat : validation de la qualité du code avant packaging.

Package (Construction des images Docker)
Construction des images **frontend** et **backend** avec **Kaniko**.

Push automatique vers le **GitLab Container Registry**.
👉 Résultat : images prêtes à être déployées.

Security Scan (Vérification sécurité)

Analyse des **images Docker** avec **Trivy**.

Détection des vulnérabilités critiques ou hautes.
👉 Résultat : sécurité renforcée avant déploiement.

Helm Package (Gestion des chartes Helm)

Validation des **charts Helm (helm lint)**.

Création des **packages Helm** pour **frontend & backend**.

Simulation de déploiement (helm template --dry-run).
👉 Résultat : manifestes Kubernetes valides et versionnés.

.gitlab-ci.yml 8.52 KiB

```yaml
stages:
  - build
  - test
  - package
  - security-scan
  - helm-package
  - deploy-k3s
  - deploy

variables:
  DOCKER_DRIVER: overlay2
  CONTAINER_FRONTEND: registry.gitlab.com/majayaka/oc-p7/front
  CONTAINER_BACKEND: registry.gitlab.com/majayaka/oc-p7/back
  GIT_SUBMODULE_STRATEGY: recursive
  HELM_CHART_VERSION: ${CI_PIPELINE_ID}

# ------- Build & Test --------
build-front:
  image: node:latest
  stage: build
  before_script:
    - npm cache clean --force
  script:
    - cd front
    - npm ci
    - npm audit fix --force || true
    - npx @angular/cli build --optimization
  artifacts:
    paths:
      - front/dist
      - front/package*.json
      - front/node_modules/
  tags: [new-runner]
  cache:
    key: { files: [front/package.json] }
    paths: [front/node_modules/]

build-back:
  image: maven:3.9.9
  stage: build
  script:
    - cd back
    - mvn clean install -DskipTests
  artifacts:
    paths:
      - back/target/
      - back/.m2/repository/
  tags: [new-runner]
  cache:
    key: { files: [back/pom.xml] }
    paths: [back/.m2/repository/]

test-front:
  image: cypress/browsers:latest
  stage: test
  needs: [build-front]
  script:
    - cd front
    - CHROME_BIN=/opt/google/chrome/chrome npx @angular/cli test --no-watch --no-progress --browsers=Chrome
  artifacts:
    paths: [front/test-results.log]
  tags: [new-runner]
  cache:
    key: { files: [front/package.json] }
    paths: [front/node_modules/]
    policy: pull

test-back:
  image: maven:3.9.9
  stage: test
  needs: [build-back]
  script:
    - cd back
    - mvn test
  artifacts:
    paths: [back/target/surefire-reports/]
  tags: [new-runner]
  cache:
    key: { files: [back/pom.xml] }
    paths: [back/.m2/repository/]
    policy: pull
```

```yaml
# ------- Package (Docker) --------
docker-build-push:
  image:
    name: gcr.io/kaniko-project/executor:v1.23.2-debug
    entrypoint: [""]
  stage: package
  variables:
    FF_NETWORK_PER_BUILD: "true"
  before_script:
    - mkdir -p /kaniko/.docker
    - echo "{\"auths\":{\"$CI_REGISTRY\":{\"auth\":\"$(printf "%s:%s" "${CI_REGISTRY_USER}" "${CI_REGISTRY_PASSWORD}" | base64 | tr -d
  script:
    - |
      echo "Building frontend image..."
      /kaniko/executor \
        --context "${CI_PROJECT_DIR}/front" \
        --dockerfile "${CI_PROJECT_DIR}/front/Dockerfile" \
        --destination "${CONTAINER_FRONTEND}" \
        --cache=true \
        --cache-repo "registry.gitlab.com/majayaka/oc-p7/cache/frontend" \
        --cache-ttl=168h

    - |
      echo "Building backend image..."
      /kaniko/executor \
        --context "${CI_PROJECT_DIR}/back" \
        --dockerfile "${CI_PROJECT_DIR}/back/Dockerfile" \
        --destination "${CONTAINER_BACKEND}" \
        --cache=true \
        --cache-repo "registry.gitlab.com/majayaka/oc-p7/cache/backend" \
        --cache-ttl=168h
  needs: [build-front, build-back]
  tags: [new-runner]

# ------- Security Scan --------
security-scan:
  image:
    name: aquasec/trivy:latest
    entrypoint: [""]
  stage: security-scan
  script:
    - mkdir -p trivy-results
    - trivy image --exit-code 1 --severity HIGH,CRITICAL "$CONTAINER_FRONTEND" > trivy-results/front-scan.txt || true
    - trivy image --exit-code 1 --severity HIGH,CRITICAL "$CONTAINER_BACKEND" > trivy-results/back-scan.txt || true
  allow_failure: true
  needs: [docker-build-push]
  artifacts:
    paths: [trivy-results/]
    when: always
  tags: [new-runner]
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'

# ------- Helm Chart Management --------
helm-check-and-package:
  image:
    name: alpine/helm:latest
    entrypoint: [""]
  stage: helm-package
  before_script:
    - apk add --no-cache git
  script:
    - echo "Linting Helm charts with specific values..."
    - helm lint helm-front/ --values helm-front/values-main.yaml
    - helm lint helm-back/ --values helm-back/values-main.yaml || echo "helm-back not ready yet, skipping..."
    - echo "Packaging Helm charts..."
    - mkdir -p helm-packages
    - helm package helm-front/ --version ${HELM_CHART_VERSION} --destination helm-packages/
    - helm package helm-back/ --version ${HELM_CHART_VERSION} --destination helm-packages/ || echo "helm-back not ready yet, skipping..
    - echo "Testing Helm charts with dry-run..."
    - helm template frontend helm-front/ --values helm-front/values-main.yaml --dry-run
    - helm template backend helm-back/ --values helm-back/values-main.yaml --dry-run || echo "helm-back not ready yet, skipping..."
    - echo "Validating Kubernetes manifests..."
    - helm template frontend helm-front/ --values helm-front/values-main.yaml > frontend-manifests.yaml
    - helm template backend helm-back/ --values helm-back/values-main.yaml > backend-manifests.yaml || echo "helm-back not ready yet, s
    - echo "Generated Helm packages:"
    - ls -la helm-packages/
  needs: [docker-build-push]
  artifacts:
    paths:
      - helm-packages/
      - frontend-manifests.yaml
      - backend-manifests.yaml
    expire_in: 1 hour
  tags: [new-runner]
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
```

.gitlab-ci.yml

Deploy (Déploiement automatisé)

Stage deploy-k3s :

- Utilisation d'une image **Alpine** avec **Helm** et installation de **kubectl**.
- Décodage du kubeconfig encodé en Base64 et configuration de l'accès au cluster **K3s**.
- Déploiement des **charts Helm** pour **frontend** et **backend** sur le cluster cible.
- Vérification que les pods sont bien créés et opérationnels (**kubectl get pods -o wide**).

👉 Résultat : applications déployées sur **K3s**, prêtes à être utilisées. C'est une démonstration locale et légère : **K3s** permet de montrer rapidement que le pipeline sait construire les images, appliquer les **charts Helm** et lancer les pods.
But : prouver que la chaîne CI/CD fonctionne de bout en bout dans un environnement maîtrisé.

Terraform :

- Création et configuration de l'infrastructure AWS (EKS, IAM, VPC, etc.). Ici, on passe à l'échelle infrastructure : Terraform automatise la création de tous les composants AWS (EKS, VPC, IAM...).
- But : montrer l'Infrastructure as Code (IaC), reproductible et versionnée. Plus besoin de configurer AWS manuellement.

Ansible :

- Une fois l'infrastructure en place, **Ansible** prend le relais pour installer et configurer les applications (via **Helm** sur le **cluster Kubernetes**).

👉 But : séparer la logique infra (**Terraform**) de la logique applicative (**Ansible**).

🎯 **Emphase**

👉 Cette chaîne CI/CD automatise entièrement :

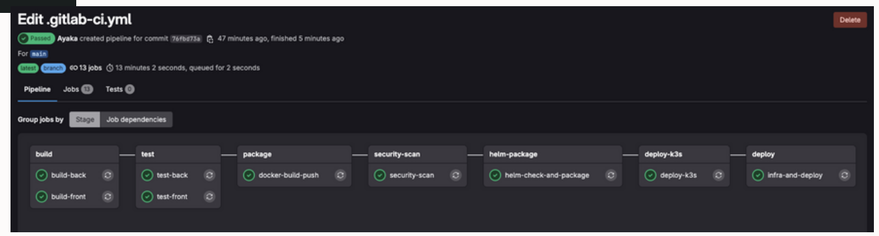
- la compilation
- les tests
- le contrôle de sécurité
- le packaging applicatif
- le déploiement de l'infrastructure et des applications

⚡ Cela élimine les étapes manuelles, réduit les erreurs humaines et accélère la mise en production.

```yaml
# ------- Deploy (k3s)--------
deploy-k3s:
  image:
    name: alpine/helm:latest
    entrypoint: [""]
  stage: deploy-k3s
  needs:
    - job: helm-check-and-package
      artifacts: true
  before_script:
    - apk add --no-cache curl bash git
    - curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
    - chmod +x kubectl && mv kubectl /usr/local/bin/
    - echo "$KUBE_CONFIG_CONTENT" | base64 -d > kubeconfig.yaml
    - export KUBECONFIG=$PWD/kubeconfig.yaml
    - echo "=== Checking kubeconfig ==="
    - head -n 5 kubeconfig.yaml
    - kubectl version --client
    - kubectl cluster-info || echo "❌ Cannot connect to the cluster. Verify KUBE_CONFIG_CONTENT."
  script:
    - FRONTEND_CHART=$(ls helm-packages/frontend-*.tgz)
    - helm upgrade --install frontend "$FRONTEND_CHART" --values helm-front/values-main.yaml
    - BACKEND_CHART=$(ls helm-packages/backend-*.tgz)
    - helm upgrade --install backend "$BACKEND_CHART" --values helm-back/values-main.yaml
    - kubectl get pods -o wide
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
  tags: [new-runner]

# ------- Deploy (Terraform and Ansible)--------
infra-and-deploy:
  image:
    name: alpine/helm:latest
    entrypoint: [""]
  stage: deploy
  needs:
    - job: helm-check-and-package
      artifacts: true
  before_script:
    - apk add --no-cache bash git curl gcc musl-dev libffi-dev python3-dev openssl-dev unzip jq aws-cli python3 py3-pip

    # Terraform setup
    - curl -fsSL https://releases.hashicorp.com/terraform/1.9.4/terraform_1.9.4_linux_amd64.zip -o terraform.zip
    - unzip terraform.zip && chmod +x terraform && mv terraform /usr/local/bin/

    # kubectl setup
    - curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
    - chmod +x kubectl
    - mv kubectl /usr/local/bin/kubectl

    # Python setup
    - python3 -m venv ansible-env
    - source ansible-env/bin/activate
    - pip install --upgrade pip
    - pip install ansible-core==2.15.5 boto3==1.34.0 botocore==1.34.0 kubernetes==29.0.0
    - ansible-galaxy collection install amazon.aws:6.5.0 community.aws:6.4.0 kubernetes.core:2.4.0

  script: |
    source ansible-env/bin/activate
    echo "=== Tool Verification ==="
    terraform --version
    helm version
    kubectl version --client
    aws --version

    echo "=== Python package check ==="
    ansible-env/bin/python -m pip show kubernetes || echo "kubernetes NOT installed!"

    echo "=== Available Helm Packages ==="
    ls -la helm-packages/

    # Terraform apply
    cd cluster/terraform
    terraform init
    terraform apply -auto-approve \
      -var="aws_access_key=$AWS_ACCESS_KEY_ID" \
      -var="aws_secret_key=$AWS_SECRET_ACCESS_KEY" \
      -var="region=$AWS_DEFAULT_REGION" \
      -var='subnet_ids=["subnet-0d97c69fd65001250","subnet-05e4ce3be43abf097","subnet-0c982df9bf03eac28"]' \
      -var="vpc_id=vpc-0a70039bdb6f247e3" \
      -var="eks_role_arn=$EKS_ROLE_ARN"

    CLUSTER_NAME=$(terraform output -raw cluster_name)
    # Ansible deploy
    cd ../ansible
    echo "[localhost]" > inventory.ini
    echo "127.0.0.1 ansible_connection=local" >> inventory.ini

    FRONTEND_CHART=$(ls ../../helm-packages/frontend-*.tgz)
    BACKEND_CHART=$(ls ../../helm-packages/backend-*.tgz)

    ansible-playbook -i inventory.ini deploy-eks.yml \
      --extra-vars "cluster_name=$CLUSTER_NAME \
                    aws_region=$AWS_DEFAULT_REGION \
                    frontend_chart=$FRONTEND_CHART \
                    backend_chart=$BACKEND_CHART"
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
  tags: [new-runner]
```

💡 **Élément visuel**

Capture d'écran d'un pipeline GitLab réussi (toutes les étapes vertes ✅).



La conteneurisation avec Docker

La conteneurisation avec Docker

Objectif (pourquoi Docker ?)

- Isoler l'application et ses dépendances pour garantir un environnement d'exécution cohérent et reproductible.
- Faciliter le déploiement de bout en bout dans la chaîne **CI/CD.**

Dockerfiles (où et pour quoi ?)

- **Frontend : front/Dockerfile**
  - Build de l'app Angular puis serving statique via Nginx (front/nginx.conf).
- **Backend : back/Dockerfile**
  - Packaging de l'app **Java (Spring Boot)** en image exécutable.

Chaîne de build & push (CI/CD GitLab)

- Construction des images avec **Kaniko** (job **docker-build-push**).
- Push vers **GitLab Container Registry** :
  - Frontend → registry.gitlab.com/majayaka/oc-p7/front
  - Backend → registry.gitlab.com/majayaka/oc-p7/back
- Cache de couches dédié pour accélérer les builds :

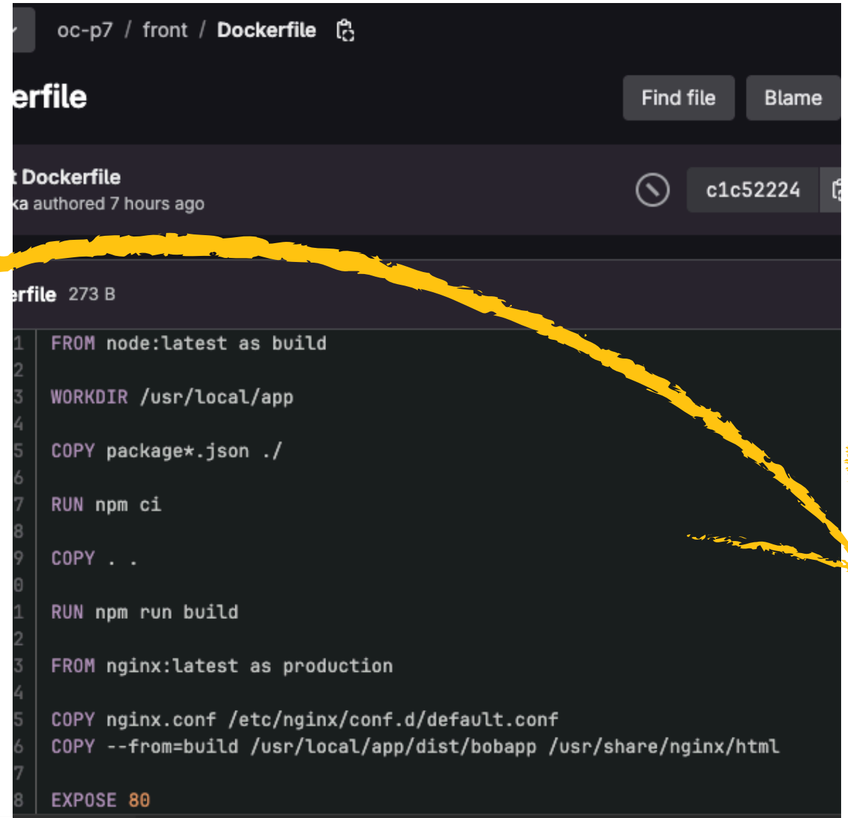
- --cache=true
- --cache-repo registry.gitlab.com/majayaka/oc-p7/cache/frontend
- --cache-repo registry.gitlab.com/majayaka/oc-p7/cache/backend
- --cache-ttl=168h

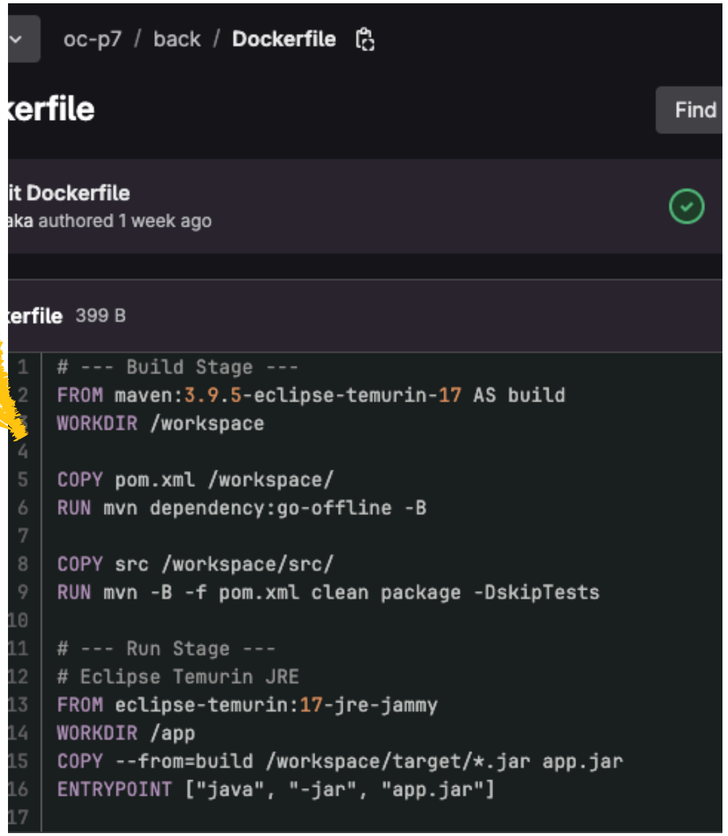
Pourquoi **Kaniko** ? (le choix technique)

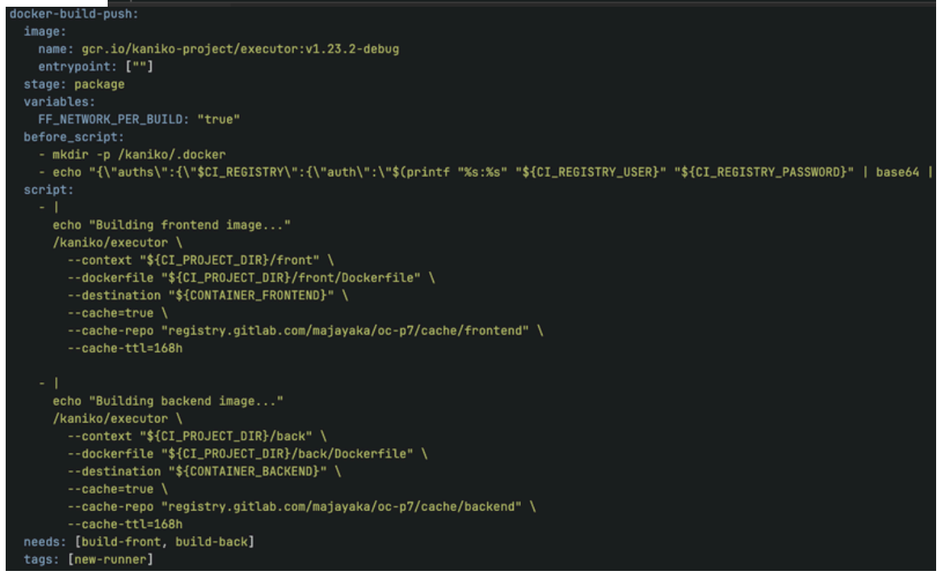
- Sans démon Docker : **pas besoin de Docker-in-Docker ni de privilèges élevés** → sécurité renforcée sur les runners.
- Intégration CI facile : s'exécute dans un container, pousse directement vers le registry.
- Performances : réutilisation des couches via un cache de registry séparé (front/back).

Point clé à retenir

Nous n'avons pas seulement "mis Docker partout" : nous avons choisi **Kaniko** pour sécuriser et fiabiliser la construction en CI, et séparé les caches pour accélérer les builds — un choix guidé par la sécurité, la vitesse et la reproductibilité.







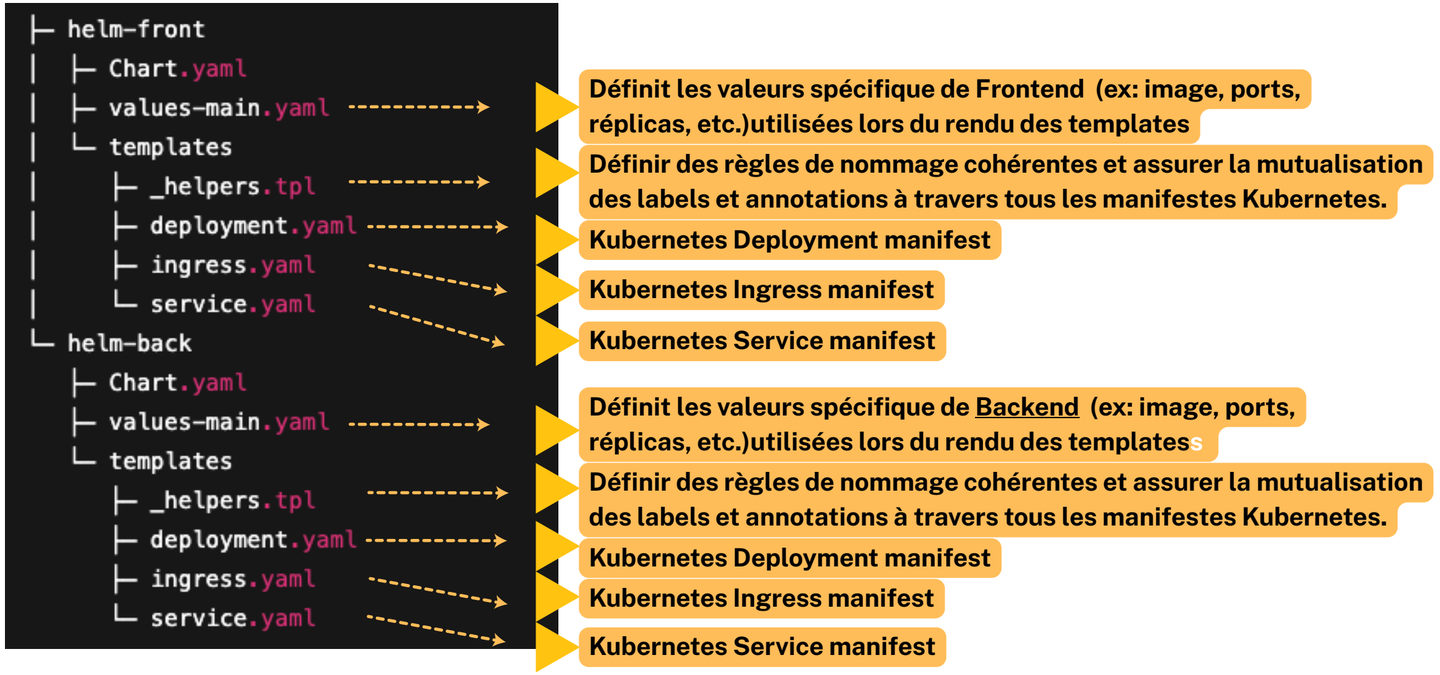
L'orchestration avec K3s et Helm

Chaîne de build & push (CI/CD GitLab)

- **Rôle de K3s** :
  - Automatisation du déploiement, de la mise à l'échelle et de la gestion des applications conteneurisées.
  - Assure la haute disponibilité et la résilience des services.

- Utilisation de **Helm** :

  - Templatisation **des manifestes Kubernetes** pour simplifier des configurations complexes.
  - **Charts Helm** distincts créés pour le **frontend** (**helm-front**) et le **backend** (**helm-back**).
  - Chaque chart gère ses propres objets Kubernetes (Deployment, Service, Ingress, etc.) via les fichiers suivants :
    - **templates/deployment.yaml** → génère les ressources Deployment pour exécuter les pods.
    - **templates/service.yaml** → définit les Services qui exposent les pods.
    - **templates/ingress.yaml** → configure les règles Ingress pour l'accès externe.
  - Fichiers de valeurs (**values-main.yaml**) pour adapter facilement les déploiements de main branch.



Lors de la pipeline GitLab CI/CD, ces templates sont utilisés à plusieurs étapes :

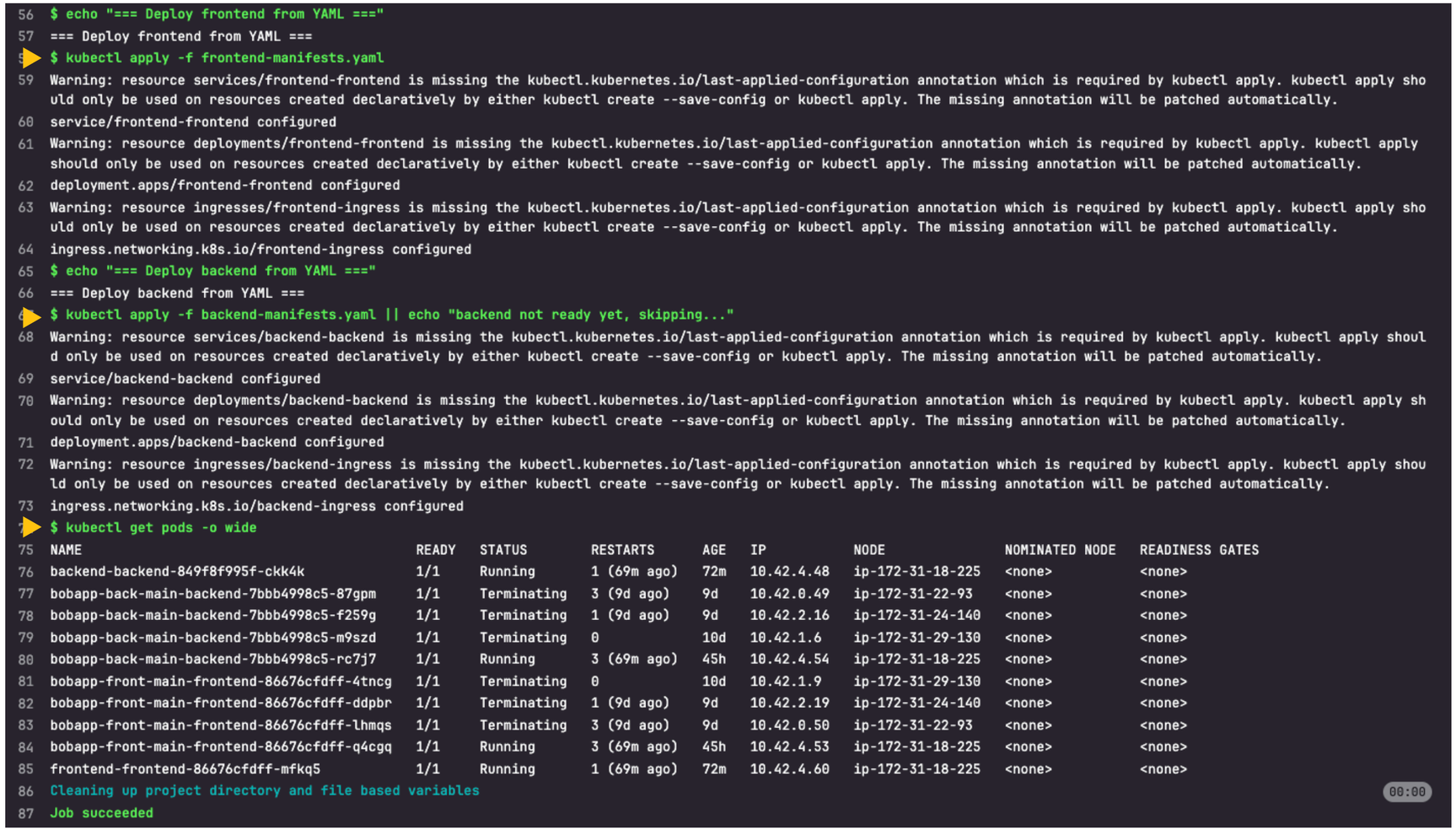
1. **Vérification (Lint)**
   - **helm lint helm-front/ --values helm-front/values-main.yaml**
   - Ici, le moteur de templating de Helm analyse **les fichiers templates/*.yaml** et valide leur cohérence.
2. **Packaging**
   - **helm package helm-front/ ...**
   - **Les fichiers templates/*.yaml** sont inclus dans l'archive **Helm**, prête à être déployée.
3. **Rendu des manifestes Kubernetes**
   - **helm template frontend helm-front/ --values helm-front/values-main.yaml --dry-run** >Le chart **helm-front** est rendu en utilisant **le fichier values-main.yaml** pour générer **les manifestes Kubernetes** et **les afficher,** mais sans les appliquer au cluster.
   - Les fichiers **deployment.yaml, service.yaml et ingress.yaml** sont rendus en manifestes Kubernetes complets (YAML).
4. **Export pour déploiement**
   - **helm template frontend helm-front/ --values helm-front/values-main.yaml > frontend-manifests.yaml**
   - Les manifestes générés à partir des templates sont exportés dans des fichiers (**frontend-manifests.yaml, backend-manifests.yaml**) utilisés ensuite pour le déploiement ou la validation.

```yaml
# ------- Helm Chart Management --------
helm-check-and-package:
  image:
    name: alpine/helm:latest
    entrypoint: [""]
  stage: helm-package
  before_script:
    - apk add --no-cache git
  script:
    - echo "Linting Helm charts with specific values..."
    - helm lint helm-front/ --values helm-front/values-main.yaml
    - helm lint helm-back/ --values helm-back/values-main.yaml || echo "helm-back not ready yet, skipping..."
    - echo "Packaging Helm charts..."
    - mkdir -p helm-packages
    - helm package helm-front/ --version ${HELM_CHART_VERSION} --destination helm-packages/
    - helm package helm-back/ --version ${HELM_CHART_VERSION} --destination helm-packages/ || echo "helm-back n
    - echo "Testing Helm charts with dry-run..."
    - helm template frontend helm-front/ --values helm-front/values-main.yaml --dry-run
    - helm template backend helm-back/ --values helm-back/values-main.yaml --dry-run || echo "helm-back not rea
    - echo "Validating Kubernetes manifests..."
    - helm template frontend helm-front/ --values helm-front/values-main.yaml > frontend-manifests.yaml
    - helm template backend helm-back/ --values helm-back/values-main.yaml > backend-manifests.yaml || echo "he
    - echo "Generated Helm packages:"
    - ls -la helm-packages/
  needs: [docker-build-push]
  artifacts:
    paths:
      - helm-packages/
      - frontend-manifests.yaml
      - backend-manifests.yaml
    expire_in: 1 hour
  tags: [new-runner]
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
```

```yaml
# ------- Deploy (k3s)--------
deploy-k3s:
  image:
    name: alpine/helm:latest
    entrypoint: [""]
  stage: deploy-k3s
  needs:
    - job: helm-check-and-package
      artifacts: true
  before_script:
    - apk add --no-cache curl bash git
    - curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/a
    - chmod +x kubectl && mv kubectl /usr/local/bin/
    - echo "$KUBE_CONFIG_CONTENT" | base64 -d > kubeconfig.yaml
    - export KUBECONFIG=$PWD/kubeconfig.yaml
    - echo "=== Checking kubeconfig ==="
    - head -n 5 kubeconfig.yaml
    - kubectl version --client
    - kubectl cluster-info || echo "❌ Cannot connect to the cluster. Verify KUBE_CONFIG_CONTENT."
  script:
    - echo "=== Deploy frontend from YAML ==="
    - kubectl apply -f frontend-manifests.yaml
    - echo "=== Deploy backend from YAML ==="
    - kubectl apply -f backend-manifests.yaml || echo "backend not ready yet, skipping..."
    - kubectl get pods -o wide
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
  tags: [new-runner]
```

💡 Élément visuel

Capture d’écran d’un k3s deploy réussi.



```
$ echo "=== Deploy frontend from YAML ==="
=== Deploy frontend from YAML ===
$ kubectl apply -f frontend-manifests.yaml
Warning: resource services/frontend-frontend is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
service/frontend-frontend configured
Warning: resource deployments/frontend-frontend is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
deployment.apps/frontend-frontend configured
Warning: resource ingresses/frontend-ingress is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
ingress.networking.k8s.io/frontend-ingress configured
$ echo "=== Deploy backend from YAML ==="
=== Deploy backend from YAML ===
$ kubectl apply -f backend-manifests.yaml || echo "backend not ready yet, skipping..."
Warning: resource services/backend-backend is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
service/backend-backend configured
Warning: resource deployments/backend-backend is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
deployment.apps/backend-backend configured
Warning: resource ingresses/backend-ingress is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
ingress.networking.k8s.io/backend-ingress configured
$ kubectl get pods -o wide
NAME                                    READY   STATUS        RESTARTS      AGE   IP           NODE               NOMINATED NODE   READINESS GATES
backend-backend-849f8f995f-ckk4k        1/1     Running       1 (69m ago)   72m   10.42.4.48   ip-172-31-18-225   <none>           <none>
bobapp-back-main-backend-7bbb4998c5-87gpm   1/1     Terminating   3 (9d ago)    9d    10.42.0.49   ip-172-31-22-93    <none>           <none>
bobapp-back-main-backend-7bbb4998c5-f259g   1/1     Terminating   1 (9d ago)    9d    10.42.2.16   ip-172-31-24-140   <none>           <none>
bobapp-back-main-backend-7bbb4998c5-m9szd   1/1     Terminating   0             10d   10.42.1.6    ip-172-31-29-130   <none>           <none>
bobapp-back-main-backend-7bbb4998c5-rc7j7   1/1     Running       3 (69m ago)   45h   10.42.4.54   ip-172-31-18-225   <none>           <none>
bobapp-front-main-frontend-86676cfdff-4tncg   1/1     Terminating   0             10d   10.42.1.9    ip-172-31-29-130   <none>           <none>
bobapp-front-main-frontend-86676cfdff-ddpbr   1/1     Terminating   1 (9d ago)    9d    10.42.2.19   ip-172-31-24-140   <none>           <none>
bobapp-front-main-frontend-86676cfdff-lhmqs   1/1     Terminating   3 (9d ago)    9d    10.42.0.50   ip-172-31-22-93    <none>           <none>
bobapp-front-main-frontend-86676cfdff-q4cgq   1/1     Running       3 (69m ago)   45h   10.42.4.53   ip-172-31-18-225   <none>           <none>
frontend-frontend-86676cfdff-mfkq5      1/1     Running       1 (69m ago)   72m   10.42.4.60   ip-172-31-18-225   <none>           <none>
Cleaning up project directory and file based variables
Job succeeded
```

Déploiement de l'infrastructure avec Terraform et Ansible

Terraform

- **Provisionnement codifié :** création automatique du cluster EKS AWS, VPC, sous-réseaux et rôles IAM.
- **Reproductibilité garantie :** chaque exécution applique les mêmes règles d'infrastructure.
- **Gestion du changement :** suivi versionné des évolutions infra via **GitLab**.

Ansible

- **Déploiement applicatif** : installation des **charts Helm (frontend et backend)** sur le cluster provisionné.
- **Intégration fluide** : récupération du nom du cluster depuis Terraform et passage comme variable aux playbooks.
- **Flexibilité :** gestion fine de la configuration via les collections amazon.aws, community.aws, kubernetes.core.

🎯 **Point clé**

👉 Synergie **Terraform + Ansible** : démonstration concrète d'un déploiement de bout en bout, de l'infrastructure jusqu'aux applications, dans une seule étape du pipeline CI/CD.

```
# ------- Deploy (Terraform and Ansible)--------
infra-and-deploy:
  image:
    name: alpine/helm:latest
    entrypoint: [""]
  stage: deploy
  needs:
    - job: helm-check-and-package
      artifacts: true
  before_script:
    - apk add --no-cache bash git curl gcc musl-dev libffi-dev python3-dev openssl-dev unzip jq aws-cli python3 py3-pip

    # Terraform setup
    - curl -fsSL https://releases.hashicorp.com/terraform/1.9.4/terraform_1.9.4_linux_amd64.zip -o terraform.zip
    - unzip terraform.zip && chmod +x terraform && mv terraform /usr/local/bin/

    # kubectl setup
    - curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
    - chmod +x kubectl
    - mv kubectl /usr/local/bin/kubectl

    # Python setup
    - python3 -m venv ansible-env
    - source ansible-env/bin/activate
    - pip install --upgrade pip
    - pip install ansible-core==2.15.5 boto3==1.34.0 botocore==1.34.0 kubernetes==29.0.0
    - ansible-galaxy collection install amazon.aws:6.5.0 community.aws:6.4.0 kubernetes.core:2.4.0

  script: |
    source ansible-env/bin/activate
    echo "=== Tool Verification ==="
    terraform --version
    helm version
    kubectl version --client
    aws --version

    echo "=== Python package check ==="
    ansible-env/bin/python -m pip show kubernetes || echo "kubernetes NOT installed!"

    echo "=== Available Helm Packages ==="
    ls -la helm-packages/

    # Terraform apply
    cd cluster/terraform
    terraform init
    terraform apply -auto-approve \
      -var="aws_access_key=$AWS_ACCESS_KEY_ID" \
      -var="aws_secret_key=$AWS_SECRET_ACCESS_KEY" \
      -var="region=$AWS_DEFAULT_REGION" \
      -var='subnet_ids=["subnet-0d97c69fd65001250","subnet-05e4ce3be43abf097","subnet-0c982df9bf03eac28"]' \
      -var="vpc_id=vpc-0a70039bdb6f247e3" \
      -var="eks_role_arn=$EKS_ROLE_ARN"

    CLUSTER_NAME=$(terraform output -raw cluster_name)
    # Ansible deploy
    cd ../ansible
    echo "[localhost]" > inventory.ini
    echo "127.0.0.1 ansible_connection=local" >> inventory.ini

    FRONTEND_CHART=$(ls ../../helm-packages/frontend-*.tgz)
    BACKEND_CHART=$(ls ../../helm-packages/backend-*.tgz)

    ansible-playbook -i inventory.ini deploy-eks.yml \
      --extra-vars "cluster_name=$CLUSTER_NAME \
                    aws_region=$AWS_DEFAULT_REGION \
                    frontend_chart=$FRONTEND_CHART \
                    backend_chart=$BACKEND_CHART"
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
  tags: [new-runner]
```

```
aws_eks_node_group.this: Still creating... [3m50s elapsed]
aws_eks_node_group.this: Still creating... [4m0s elapsed]
aws_eks_node_group.this: Still creating... [4m10s elapsed]
aws_eks_node_group.this: Still creating... [4m20s elapsed]
aws_eks_node_group.this: Still creating... [4m30s elapsed]
aws_eks_node_group.this: Still creating... [4m40s elapsed]
aws_eks_node_group.this: Creation complete after 4m49s [id=oc-p7-cluster:oc-p7-cluster-node-group]
Apply complete! Resources: 33 added, 0 changed, 0 destroyed.
Outputs:
cluster_ca = "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURCVENDQWUyZ0F3SUJBZ0lJYmY2VFBqeHMzbzh3RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJFR0ExVUUKQXhNS2
BNE1qQXhOVEkyTlRCYUZ3MHpOVEE0TVRneE5UTXhOVEJhTUJVeApFekFSQmdOVkJBTVRDbXQxWW1WeWJtVjBaWE13Z2dFaU1BMEdDU3FHU0liM0RRRUJBUVVBQTRJQkR3QXdnZ0VLCkFvSUJ
dFowNi9aaS9wQ2EvdGdaMlA3TEJnUWVYU3UxSXlQVmxWTVk1MFIKS08rY2VEeHcwckVFUnBZeTQybE8yZDhla0dRRjhwVGF6aThTUVV2TVd1VjVwbzdUd3ljUnY0UWZzN09Nc1lmUQpJVWhZ
VpncW1SSDdKOGI1SmNJTWdZdUdtNUVHVzFxeDBjeUVmcm9SSmt6ClQwdUhCYlJtb01Id0RXOUFmdlpqRmN0bEx0MFdwVXdoK2ZvbmtwaDdudUJ0bTRjQmlTUFFHTVFONUNkaW9oRXEKd3d4bU
NlQWpoVlNHRUFCblh1a2xoTWRFNm5jSGMxVk1CTzJndFpIendzbApuVjlSK0V6M285NWhZa0hHalZWa29CL3dXN2tQQWdNQkFBR2pXVEJYTUE0R0ExVWREd0VCL3dRRUF3SUNwREFQQkJnTlZ
xVWREZ1FXQkJTNExWYmI0YnNCY3lvSy94WS9xcVBoejBqNE1EQVYKQmd0VkhSRUVEakFNZ2dwcmRXSmxjbTVsZEdWek1BMEdDU3FHU0liM0RRRUJDd1VBQTRJQkFRQTd1c0UwZUZWbQpFbDhE
bDVkR25lQjFtUWE3WHFjYjE4T3E4amV0cVlZR3FsbFc1bDM0MG9BCktybXdad3pXeUVyR1BGWSt0SW9zR3FNb2t4a2Zha2MxL3hYSUZTWGlFZGZaQ3A5YUU5OUVndThEczBhMWhGcGMKUzUza
Tc2Rnk1Nk8vN3IyczUrWmNUaytPN1lldms1UlgxMmlrV3VYYW81ZgpFQkx3S1ZJUFpJV3lVOEZjd01VVjY5M1ZmdGRFQ1lJcDRYTGRIRHJyTFMvVES3U1FIQWwwNTBpTVhRanFHZXg5CldqZ
5KVTQ0dLJhY0U2M25RZ1hDNnN3N0w2Y0xCS0ZRSy9qUmhSbi9SS1IKbmNNT085ZzJlN1kvCi0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K"
cluster_endpoint = "https://33D79E30B13C0247202BF06CFC06FAD4.gr7.eu-west-3.eks.amazonaws.com"
cluster_name = "oc-p7-cluster"
eks_role_arn = ""
security_group_id = "sg-0cb7566470a8621e3"
subnet_ids = [
  "subnet-05bd2ad5f17c0b82c",
  "subnet-08d6d146cb97a8769",
  "subnet-099c144ed9e6f211e",
]
PLAY [Deploy app to EKS with Helm] *********************************************
TASK [Create .kube directory] **************************************************
changed: [127.0.0.1]
TASK [Configure kubectl for EKS] ***********************************************
changed: [127.0.0.1]
TASK [Wait for EKS cluster to be ready] ****************************************
ok: [127.0.0.1]
TASK [Deploy frontend via Helm] ************************************************
changed: [127.0.0.1]
TASK [Deploy backend via Helm] *************************************************
changed: [127.0.0.1]
PLAY RECAP *********************************************************************
127.0.0.1                  : ok=5    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
Cleaning up project directory and file based variables
Job succeeded
```

💡 **Élément visuel**

Capture d'écran d'un terraform apply réussi (cluster créé).

Documents à presenter

Document préparatoire

Plan de CD

Guide des bonnes pratiques

Août 2025

Merci !