

**RBER**

# Guide d'installation DevOps

## Guide d'Installation DevOps

Gitea • Harbor • ArgoCD • CI/CD

Infrastructure RBER - Kubernetes RKE2

PREPARE POUR  
**SBIN**

PREPARE PAR  
**IDADU TECH**



Connecting business need with IT



# Table des matières

## Table des matières

Table des matières .....	3
1. Vue d'ensemble .....	5
1.1 Architecture de la stack DevOps .....	5
1.2 Flux de travail.....	5
2. Prérequis .....	6
2.1 Cluster Kubernetes.....	6
2.2 Réseau.....	6
2.3 Outils locaux.....	6
3. Installation Gitea .....	7
3.1 Créer le namespace .....	7
3.2 Déployer PostgreSQL avec CloudNativePG .....	7
3.3 Installer Gitea via Helm .....	7
3.4 Configurer SSH (optionnel) .....	8
3.5 Vérification .....	8
4. Installation Harbor.....	9
4.1 Créer le namespace et les secrets .....	9
4.2 Déployer PostgreSQL .....	9
4.3 Installer Harbor via Helm .....	9
4.4 Configuration post-installation .....	10
4.5 Configurer Docker pour Harbor (HTTP).....	11
5. Installation ArgoCD .....	12
5.1 Créer le namespace et installer .....	12
5.2 Configurer l'Ingress .....	12
5.3 Récupérer le mot de passe admin .....	12
5.4 Configurer le repository Gitea.....	12
5.5 Créer une Application ArgoCD .....	13
6. Configuration CI/CD avec Drone.....	14
6.1 Créer l'application OAuth dans Gitea.....	14
6.2 Installer Drone Server .....	14
6.3 Installer Drone Runner .....	15
6.4 Exemple de pipeline .drone.yml.....	16
7. Administration .....	18
7.1 Gestion des utilisateurs .....	18
Gitea.....	18
Harbor .....	18
ArgoCD.....	18
7.2 Sauvegarde et restauration .....	18
Backup PostgreSQL (CloudNativePG).....	18

Backup Gitea (repositories) .....	18
Backup Harbor.....	19
7.3 Mise à jour des services .....	19
Gitea.....	19
Harbor .....	19
ArgoCD.....	19
7.4 Monitoring .....	19
7.5 Sécurité .....	20
Rotation des secrets .....	20
Audit des accès .....	20
8. Troubleshooting.....	21
8.1 Problèmes courants .....	21
8.2 Commandes de diagnostic .....	21
8.3 Réinitialisation d'urgence.....	21
Annexes .....	23
A. Récapitulatif des URLs .....	23
B. Secrets à sauvegarder .....	23
C. Ports internes Kubernetes .....	23

# 1. Vue d'ensemble

## 1.1 Architecture de la stack DevOps

Cette stack fournit une chaîne complète de développement et déploiement continu pour les universités du réseau RBER.

Service	Rôle	URL
Gitea	Gestion du code source (Git)	git.rber.bj
Harbor	Registre Docker privé	harbor.rber.bj
Drone CI	Pipeline CI/CD	drone.rber.bj
ArgoCD	Déploiement GitOps	argocd.rber.bj

## 1.2 Flux de travail

1. Développeur pousse le code sur Gitea
2. Drone détecte le changement via webhook
3. Drone build l'image Docker et la pousse sur Harbor
4. ArgoCD détecte le changement de manifeste et déploie sur Kubernetes

## 2. Prérequis

### 2.1 Cluster Kubernetes

Composant	Version	État requis
RKE2	v1.32+	Cluster opérationnel
Longhorn	1.6+	StorageClass par défaut
HAProxy Ingress	1.11+	Contrôleur configuré
Cert-Manager	1.14+	Optionnel (TLS)
CloudNativePG	1.23+	Pour PostgreSQL HA

### 2.2 Réseau

- VIP HAProxy : 10.29.112.100
- Domaines DNS configurés vers la VIP
- Ports 80/443 accessibles depuis le campus

### 2.3 Outils locaux

```
# Vérifier les versions
kubectl version --client
helm version
git --version
```

## 3. Installation Gitea

### 3.1 Créer le namespace

```
kubectl create namespace gitea
kubectl label namespace gitea app=gitea environment=production
```

### 3.2 Déployer PostgreSQL avec CloudNativePG

Créer le fichier gitea-pg-cluster.yaml :

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: gitea-pg-cluster
  namespace: gitea
spec:
  instances: 3
  imageName: ghcr.io/cloudnative-pg/postgresql:16.3
  storage:
    size: 20Gi
    storageClass: longhorn
  bootstrap:
    initdb:
      database: gitea
      owner: gitea
      encoding: UTF8
kubectl apply -f gitea-pg-cluster.yaml
kubectl wait --for=condition=Ready cluster/gitea-pg-cluster -n gitea --
timeout=300s
```

### 3.3 Installer Gitea via Helm

```
helm repo add gitea-charts https://dl.gitea.com/charts/
helm repo update
```

Créer values-gitea.yaml :

```
replicaCount: 2

gitea:
  admin:
    username: admin
    password: CHANGE_ME_SECURE_PASSWORD
    email: admin@rber.bj
  config:
    APP_NAME: RBER Git
  server:
    DOMAIN: git.rber.bj
    ROOT_URL: http://git.rber.bj
    SSH_DOMAIN: git.rber.bj
    SSH_PORT: 22
  database:
    DB_TYPE: postgres
    HOST: gitea-pg-cluster-rw:5432
    NAME: gitea
```

```

    USER: gitea
    service:
      DISABLE_REGISTRATION: false
      REQUIRE_SIGNIN_VIEW: false

postgresql:
  enabled: false # On utilise CloudNativePG

postgresql-ha:
  enabled: false

persistence:
  enabled: true
  size: 50Gi
  storageClass: longhorn

ingress:
  enabled: true
  className: haproxy
  hosts:
    - host: git.rber.bj
      paths:
        - path: /
          pathType: Prefix

```

Récupérer le mot de passe PostgreSQL et installer :

```

PG_PASS=$(kubectl get secret gitea-pg-cluster-app -n gitea \
  -o jsonpath='{.data.password}' | base64 -d)

helm install gitea gitea-charts/gitea \
  --namespace gitea \
  --values values-gitea.yaml \
  --set gitea.config.database.PASSWD=$PG_PASS \
  --wait --timeout 10m

```

### 3.4 Configurer SSH (optionnel)

Pour le clone SSH, exposer le port 22 via NodePort ou LoadBalancer :

```

kubectl patch svc gitea-ssh -n gitea \
  -p '{"spec":{"type":"NodePort","ports":[{"port":22,"nodePort":30022}]}}'

```

### 3.5 Vérification

```

kubectl get pods -n gitea
kubectl get ingress -n gitea
curl -I http://git.rber.bj

```



## 4. Installation Harbor

### 4.1 Créer le namespace et les secrets

```
kubectl create namespace harbor

# Générer les mots de passe
HARBOR_ADMIN_PASS=$(openssl rand -base64 16)
HARBOR_DB_PASS=$(openssl rand -base64 32)
HARBOR_SECRET_KEY=$(openssl rand -hex 16)

# Créer le secret
kubectl create secret generic harbor-credentials \
  --from-literal=HARBOR_ADMIN_PASSWORD=$HARBOR_ADMIN_PASS \
  --from-literal=POSTGRES_PASSWORD=$HARBOR_DB_PASS \
  --from-literal=secretKey=$HARBOR_SECRET_KEY \
  -n harbor
```

### 4.2 Déployer PostgreSQL

Créer harbor-pg-cluster.yaml :

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: harbor-pg-cluster
  namespace: harbor
spec:
  instances: 3
  imageName: ghcr.io/cloudnative-pg/postgresql:16.3
  storage:
    size: 30Gi
    storageClass: longhorn
  bootstrap:
    initdb:
      database: registry
      owner: harbor
      encoding: UTF8
      postInitSQL:
        - CREATE DATABASE notary_signer OWNER harbor;
        - CREATE DATABASE notary_server OWNER harbor;
kubectl apply -f harbor-pg-cluster.yaml
kubectl wait --for=condition=Ready cluster/harbor-pg-cluster -n harbor --
timeout=300s
```

### 4.3 Installer Harbor via Helm

```
helm repo add harbor https://helm.goharbor.io
helm repo update
```

Créer values-harbor.yaml :

```
expose:
  type: ingress
  ingress:
    hosts:
```

```

    core: harbor.rber.bj
    className: haproxy
  tls:
    enabled: false

externalURL: http://harbor.rber.bj

persistence:
  enabled: true
  persistentVolumeClaim:
    registry:
      storageClass: longhorn
      size: 200Gi
    jobservice:
      storageClass: longhorn
      size: 5Gi
    trivy:
      storageClass: longhorn
      size: 10Gi

database:
  type: external
  external:
    host: harbor-pg-cluster-rw
    port: 5432
    username: harbor
    coreDatabase: registry

redis:
  type: internal

trivy:
  enabled: true

metrics:
  enabled: true

```

#### Installer Harbor :

```

PG_PASS=$(kubectl get secret harbor-pg-cluster-app -n harbor \
-o jsonpath='{.data.password}' | base64 -d)

helm install harbor harbor/harbor \
  --namespace harbor \
  --values values-harbor.yaml \
  --set harborAdminPassword=$HARBOR_ADMIN_PASS \
  --set database.external.password=$PG_PASS \
  --set secretKey=$HARBOR_SECRET_KEY \
  --wait --timeout 15m

```

## 4.4 Configuration post-installation

### Créer un projet pour les images :

```
# Via l'interface : http://harbor.rber.bj
# Login: admin / $HARBOR_ADMIN_PASS
# Projects > New Project > Name: rber
```

### Créer un compte robot pour CI/CD :

```
# Administration > Robot Accounts > New Robot Account
# Name: drone-ci
# Permissions: Push, Pull sur projet rber
```

## 4.5 Configurer Docker pour Harbor (HTTP)

Sur chaque nœud et poste développeur :

```
# /etc/docker/daemon.json
{
  "insecure-registries": ["harbor.rber.bj"]
}

sudo systemctl restart docker
```

Test de connexion :

```
docker login harbor.rber.bj
docker pull hello-world
docker tag hello-world harbor.rber.bj/rber/hello-world:test
docker push harbor.rber.bj/rber/hello-world:test
```

## 5. Installation ArgoCD

### 5.1 Créer le namespace et installer

```
kubectl create namespace argocd

# Installation standard
kubectl apply -n argocd \
  -f https://raw.githubusercontent.com/argoproj/argo-
cd/stable/manifests/install.yaml

# Attendre que les pods soient prêts
kubectl wait --for=condition=Ready pods --all -n argocd --timeout=300s
```

### 5.2 Configurer l'Ingress

Créer argocd-ingress.yaml :

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: argocd-server
  namespace: argocd
  annotations:
    haproxy.org/ssl-passthrough: "true"
    haproxy.org/backend-config-snippet: |
      option httpchk GET /healthz
spec:
  ingressClassName: haproxy
  rules:
  - host: argocd.rber.bj
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: argocd-server
            port:
              number: 80
kubectl apply -f argocd-ingress.yaml
```

### 5.3 Récupérer le mot de passe admin

```
# Mot de passe initial
kubectl -n argocd get secret argocd-initial-admin-secret \
  -o jsonpath="{.data.password}" | base64 -d; echo

# Connexion CLI
argocd login argocd.rber.bj --username admin --password <PASSWORD> --insecure
```

### 5.4 Configurer le repository Gitea

```
# Ajouter le repo Gitea
argocd repo add http://git.rber.bj/infrastructure/k8s-manifests.git \
```

```
--username admin \
--password <GITEA_TOKEN>

# Ou via l'interface web :
# Settings > Repositories > Connect Repo
```

## 5.5 Créer une Application ArgoCD

Exemple d'application GitOps :

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: moodle-unstim
  namespace: argocd
spec:
  project: default
  source:
    repoURL: http://git.rber.bj/infrastructure/k8s-manifests.git
    targetRevision: main
    path: moodle/unstim
  destination:
    server: https://kubernetes.default.svc
    namespace: moodle-unstim
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
```

## 6. Configuration CI/CD avec Drone

### 6.1 Créer l'application OAuth dans Gitea

1. Se connecter à Gitea en admin
2. Aller dans Site Administration > Applications
3. Create OAuth2 Application :
  - Name: Drone CI
  - Redirect URI: http://drone.rber.bj/login
4. Noter le Client ID et Client Secret

### 6.2 Installer Drone Server

```
kubectl create namespace drone
```

Créer drone-server.yaml :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: drone-server
  namespace: drone
spec:
  replicas: 1
  selector:
    matchLabels:
      app: drone-server
  template:
    metadata:
      labels:
        app: drone-server
    spec:
      containers:
        - name: drone
          image: drone/drone:2
          env:
            - name: DRONE_GITEA_SERVER
              value: http://git.rber.bj
            - name: DRONE_GITEA_CLIENT_ID
              valueFrom:
                secretKeyRef:
                  name: drone-secrets
                  key: gitea-client-id
            - name: DRONE_GITEA_CLIENT_SECRET
              valueFrom:
                secretKeyRef:
                  name: drone-secrets
                  key: gitea-client-secret
            - name: DRONE_RPC_SECRET
              valueFrom:
                secretKeyRef:
                  name: drone-secrets
                  key: rpc-secret
            - name: DRONE_SERVER_HOST
```

```

    value: drone.rber.bj
  - name: DRONE_SERVER_PROTO
    value: http
  - name: DRONE_DATABASE_DRIVER
    value: sqlite3
  - name: DRONE_DATABASE_DATASOURCE
    value: /data/database.sqlite
  ports:
  - containerPort: 80
  volumeMounts:
  - name: data
    mountPath: /data
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: drone-data

```

### Créer les secrets :

```

RPC_SECRET=$(openssl rand -hex 16)

kubectl create secret generic drone-secrets \
  --from-literal=gitea-client-id=YOUR_CLIENT_ID \
  --from-literal=gitea-client-secret=YOUR_CLIENT_SECRET \
  --from-literal=rpc-secret=$RPC_SECRET \
  -n drone

```

## 6.3 Installer Drone Runner

### Créer drone-runner.yaml :

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: drone-runner
  namespace: drone
spec:
  replicas: 2
  selector:
    matchLabels:
      app: drone-runner
  template:
    metadata:
      labels:
        app: drone-runner
    spec:
      containers:
      - name: runner
        image: drone/drone-runner-docker:1
        env:
        - name: DRONE_RPC_HOST
          value: drone-server.drone.svc.cluster.local
        - name: DRONE_RPC_PROTO
          value: http

```

```

- name: DRONE_RPC_SECRET
  valueFrom:
    secretKeyRef:
      name: drone-secrets
      key: rpc-secret
- name: DRONE_RUNNER_CAPACITY
  value: "4"
- name: DRONE_RUNNER_NAME
  valueFrom:
    fieldRef:
      fieldPath: metadata.name
volumeMounts:
- name: docker-socket
  mountPath: /var/run/docker.sock
volumes:
- name: docker-socket
  hostPath:
    path: /var/run/docker.sock

```

## 6.4 Exemple de pipeline .drone.yml

Placer ce fichier à la racine de chaque repository :

```

kind: pipeline
type: docker
name: default

steps:
- name: test
  image: node:20-alpine
  commands:
    - npm ci
    - npm test

- name: build
  image: docker:dind
  volumes:
    - name: docker-socket
      path: /var/run/docker.sock
  commands:
    - docker build -t
      harbor.rber.bj/rber/${DRONE_REPO_NAME}:${DRONE_COMMIT_SHA:0:8} .
    - docker push
      harbor.rber.bj/rber/${DRONE_REPO_NAME}:${DRONE_COMMIT_SHA:0:8}
  when:
    branch:
      - main

- name: deploy
  image: alpine/git
  commands:
    - git clone http://git.rber.bj/infrastructure/k8s-manifests.git
    - cd k8s-manifests

```



```
- sed -i "s|image:.*|image:
harbor.rber.bj/rber/${DRONE_REPO_NAME}:${DRONE_COMMIT_SHA:0:8}|"
app/deployment.yaml
- git add .
- git commit -m "Update image to ${DRONE_COMMIT_SHA:0:8}"
- git push
when:
  branch:
    - main

volumes:
- name: docker-socket
  host:
    path: /var/run/docker.sock
```

## 7. Administration

### 7.1 Gestion des utilisateurs

#### Gitea

Action	Commande / Procédure
Créer un utilisateur	Interface > Site Administration > User Accounts > Create
Désactiver un compte	User Accounts > Edit > Prohibit Login
Reset mot de passe	<code>gitea admin user change-password --username USER --password NEWPASS</code>
Créer une organisation	Interface > + > New Organization

#### Harbor

Action	Procédure
Créer un projet	Projects > New Project
Ajouter un membre	Project > Members > + User
Créer un robot account	Administration > Robot Accounts > New
Quotas de stockage	Projects > Configuration > Storage Quota

#### ArgoCD

```
# Créer un utilisateur local
argocd account update-password --account USER --new-password PASS

# Ajouter un utilisateur (dans configmap)
kubectl edit configmap argocd-cm -n argocd
# data:
#   accounts.USER: apiKey, login

# Définir les permissions RBAC
kubectl edit configmap argocd-rbac-cm -n argocd
```

### 7.2 Sauvegarde et restauration

#### Backup PostgreSQL (CloudNativePG)

```
# Backup à la demande
kubectl apply -f - <<EOF
apiVersion: postgresql.cnpg.io/v1
kind: Backup
metadata:
  name: gitea-backup-$(date +%Y%m%d)
  namespace: gitea
spec:
  cluster:
    name: gitea-pg-cluster
EOF

# Vérifier le backup
kubectl get backup -n gitea
```

#### Backup Gitea (repositories)

```
# Dump via CLI Gitea
```

```
kubectl exec -it -n gitea deploy/gitea -- gitea dump -c
/data/gitea/conf/app.ini

# Copier l'archive
kubectl cp gitea/gitea-POD_NAME:/app/gitea/gitea-dump-*.zip ./backup/
```

## Backup Harbor

```
# Exporter la configuration Harbor
kubectl get all,pvc,secrets,configmaps -n harbor -o yaml > harbor-backup.yaml

# Backup des images : utiliser la réplication vers un Harbor distant
# Interface > Administration > Replications > New Replication Rule
```

## 7.3 Mise à jour des services

### Gitea

```
# Vérifier la version actuelle
helm list -n gitea

# Mettre à jour
helm repo update
helm upgrade gitea gitea-charts/gitea \
  --namespace gitea \
  --values values-gitea.yaml \
  --wait
```

### Harbor

```
# Backup préalable obligatoire !
helm upgrade harbor harbor/harbor \
  --namespace harbor \
  --values values-harbor.yaml \
  --wait --timeout 20m
```

### ArgoCD

```
# Appliquer la nouvelle version
kubectl apply -n argocd \
  -f https://raw.githubusercontent.com/argoproj/argo-
cd/v2.10.0/manifests/install.yaml
```

## 7.4 Monitoring

Service	Endpoint Prometheus	Métriques clés
Gitea	/metrics	gitea_users, gitea_repositories
Harbor	/metrics	harbor_project_count, harbor_artifact_count
ArgoCD	argocd-metrics:8082	argocd_app_sync_status
Drone	/metrics	drone_build_total, drone_pending_builds

Exemple de requête Prometheus :

```
# Nombre de builds échoués dans la dernière heure
sum(increase(drone_build_total{status="failure"}[1h]))

# Apps ArgoCD hors sync
sum(argocd_app_sync_status{sync_status!="Synced"})
```

## 7.5 Sécurité

### Rotation des secrets

```
# Régénérer le secret Drone RPC
NEW_SECRET=$(openssl rand -hex 16)
kubectl patch secret drone-secrets -n drone \
  -p '{"stringData":{"rpc-secret":"'NEW_SECRET'"} }'

# Redémarrer les pods
kubectl rollout restart deployment -n drone
```

### Audit des accès

```
# Logs Gitea
kubectl logs -n gitea -l app=gitea --tail=100 | grep -i auth

# Logs Harbor
kubectl logs -n harbor -l component=core --tail=100

# Logs ArgoCD
kubectl logs -n argocd -l app.kubernetes.io/name=argocd-server
```

## 8. Troubleshooting

### 8.1 Problèmes courants

Symptôme	Cause probable	Solution
Gitea : 502 Bad Gateway	Pod non prêt ou DB inaccessible	Vérifier les logs pod et connexion PostgreSQL
Harbor : push échoue	Registry insecure non configuré	Ajouter harbor.rber.bj à insecure-registries
Drone : builds en attente	Runners non connectés	Vérifier logs runners et RPC_SECRET
ArgoCD : OutOfSync	Drift manuel ou erreur manifeste	Vérifier diff et sync manuellement
PostgreSQL : cluster dégradé	Réplica en retard ou storage plein	kubectl cnpg status CLUSTER -n NS

### 8.2 Commandes de diagnostic

```
# État général d'un namespace
kubectl get all,pvc,secrets,ingress -n gitea

# Logs d'un pod avec erreur
kubectl logs -n NAMESPACE POD_NAME --previous

# Events récents (erreurs)
kubectl get events -n NAMESPACE --sort-by='.lastTimestamp' | tail -20

# État du cluster PostgreSQL
kubectl cnpg status gitea-pg-cluster -n gitea

# Test de connectivité interne
kubectl run test --rm -it --image=busybox -- wget -O-
http://SERVICE.NS.svc:PORT

# Vérifier les ressources
kubectl top pods -n gitea
```

### 8.3 Réinitialisation d'urgence

#### Reset mot de passe admin Gitea :

```
kubectl exec -it -n gitea deploy/gitea -- \
  gitea admin user change-password --username admin --password NEW_PASSWORD
```

#### Reset mot de passe admin Harbor :

```
# Supprimer le job de reset s'il existe
kubectl delete job harbor-core-reset-password -n harbor --ignore-not-found

# Recréer via Helm
helm upgrade harbor harbor/harbor -n harbor \
  --values values-harbor.yaml \
  --set harborAdminPassword=NEW_PASSWORD
```

#### Reset mot de passe admin ArgoCD :

```
# Générer le hash bcrypt
NEW_HASH=$(argocd account bcrypt --password NEW_PASSWORD)
```

```
# Mettre à jour le secret
kubectl patch secret argocd-secret -n argocd \
  -p '{"stringData":{"admin.password":"'$NEW_HASH'"} }'

# Redémarrer le serveur
kubectl rollout restart deployment argocd-server -n argocd
```

## Annexes

### A. Récapitulatif des URLs

Service	URL	Port
Gitea	http://git.rber.bj	80
Harbor	http://harbor.rber.bj	80
Drone	http://drone.rber.bj	80
ArgoCD	http://argocd.rber.bj	80

### B. Secrets à sauvegarder

Secret	Namespace	Contenu
gitea-pg-cluster-app	gitea	Credentials PostgreSQL
harbor-pg-cluster-app	harbor	Credentials PostgreSQL
harbor-credentials	harbor	Admin password, secret key
drone-secrets	drone	OAuth credentials, RPC secret
argocd-initial-admin-secret	argocd	Admin password initial

### C. Ports internes Kubernetes

Service	Port interne	Usage
gitea-http	3000	Interface web
gitea-ssh	22	Git SSH
harbor-core	8080	API Harbor
harbor-registry	5000	Docker registry
drone-server	80	Interface + API
argocd-server	8080	Interface web
argocd-repo-server	8081	Git sync

— Fin du document —

Équipe Infrastructure RBER - devops@rber.bj