Technique

POC OpenHands avec Orchestration Multi-LLM - v1 Simplifiée

Version: 1.0 - Approche DevOps Classique
Date: Janvier 2025
Classification: POC / Proof of Concept
Auteur: Architecture DevOps Team

---

1. Résumé Exécutif

1.1 Objectif

Mise en place d'un POC permettant l'automatisation du cycle de développement logiciel via IA, depuis l'expression du besoin jusqu'au déploiement, avec une approche **DevOps classique (push)** pour valider rapidement le concept.

1.2 Principes directeurs

- **Simplicité avant tout** : Approche push traditionnelle via GitHub Actions
- **Time to market** : POC fonctionnel en 4 semaines au lieu de 8
- **Validation rapide** : Focus sur la valeur de l'IA, pas sur l'infrastructure
- **Évolution progressive** : Architecture permettant une migration GitOps future si pertinent

1.3 Périmètre v1

- **Orchestration IA** : OpenHands pour la coordination d'agents
- **Multi-LLM** : Support modèles locaux (Ollama) et cloud (OpenRouter)
- **CI/CD simplifié** : GitHub Actions avec kubectl direct
- **Infrastructure** : K3s mono-cluster
- **Monitoring** : Basique (logs + métriques essentielles)

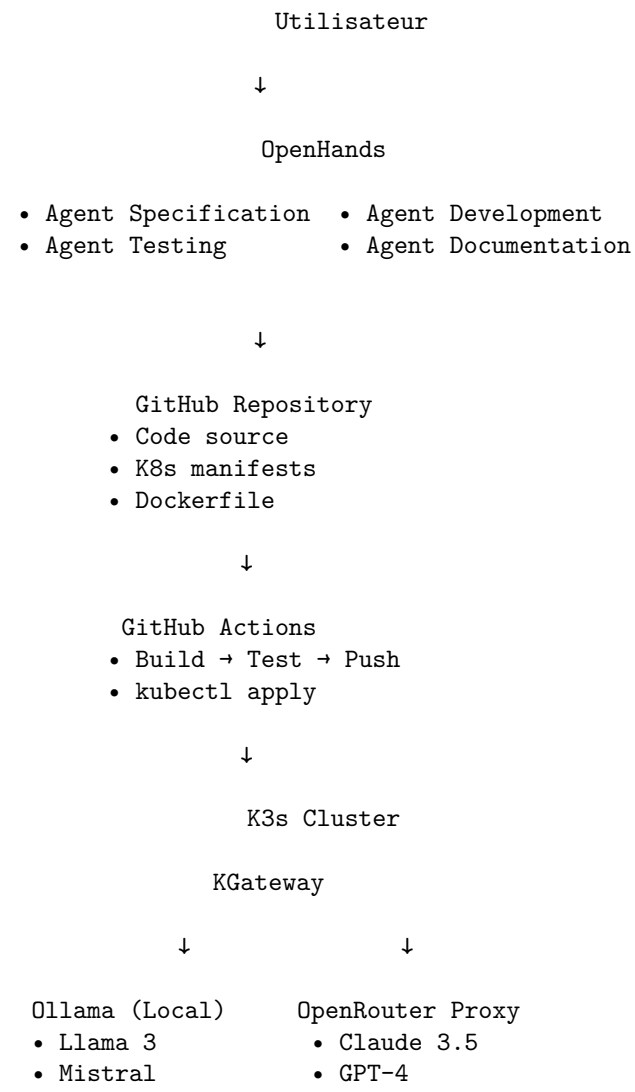
1.4 Hors périmètre v1 (simplifications)

- GitOps/Flux (complexité inutile pour POC)
- SOPS (secrets GitHub suffisants)
- Multi-environnements (dev/staging/prod)
- Haute disponibilité
- Observabilité avancée

---

2. Architecture Simplifiée

2.1 Vue d'ensemble



2.2 Flux de travail simplifié

1. Développeur → Prompt dans OpenHands
2. OpenHands → Génération code + tests + K8s manifests
3. OpenHands → Création PR sur GitHub

```
4. GitHub Actions → Build & Test automatique
5. Merge PR → Déploiement automatique via kubectl
6. Application → Running sur K3s
```

---

3. Stack Technique Allégée

3.1 Composants essentiels uniquement

Composant	Version	Rôle	Complexité
OpenHands	Latest	Orchestration IA	
GitHub	-	Source + CI/CD	
GitHub Actions	-	Pipeline push	
GHCR	-	Registry Docker	
K3s	1.29.x	Kubernetes léger	
KGateway	1.0	Routing LLM	
Ollama	0.3.x	LLM local	
OpenRouter	API v1	LLM cloud	

3.2 Composants différés à v2

Composant	Raison du report
Flux/ArgoCD	Complexité GitOps non nécessaire
SOPS	GitHub Secrets suffisants
Prometheus/Grafana	Monitoring basique suffisant
Testkube	Tests dans GitHub Actions OK
Jaeger	Tracing non prioritaire

---

4. Structure du Projet

4.1 Organisation des dossiers

```
poc-openhands/
  .github/
     workflows/
         deploy.yml          # Pipeline simple push
  src/
     api/                    # Code généré par OpenHands
     frontend/               # UI si nécessaire
  k8s/
     deployment.yaml         # Manifests simples
```

```
    service.yaml
    configmap.yaml
    kgateway-routes.yaml    # Routing LLM
  tests/
    unit/                   # Tests unitaires générés
    integration/            # Tests d'intégration
  docker/
    Dockerfile              # Image application
  README.md                  # Documentation
```

4.2 Pipeline GitHub Actions

```yaml
name: Build and Deploy POC

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  build-test-deploy:
    runs-on: ubuntu-latest

    steps:
    # 1. Checkout code
    - uses: actions/checkout@v3

    # 2. Login to GHCR
    - name: Log in to Container Registry
      uses: docker/login-action@v2
      with:
        registry: ${{ env.REGISTRY }}
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}

    # 3. Build and push Docker image
    - name: Build and push Docker image
      run: |
        docker build -t ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{ github.sha }} .
        docker build -t ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:latest .
        docker push ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{ github.sha }}
```

```
      docker push ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:latest

  # 4. Run tests
  - name: Run tests
    run: |
      docker run --rm ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{ github.sha }} npm test

  # 5. Deploy to K3s (only on main)
  - name: Deploy to K3s
    if: github.ref == 'refs/heads/main' && github.event_name == 'push'
    run: |
      # Setup kubectl
      mkdir -p $HOME/.kube
      echo "${{ secrets.KUBECONFIG }}" | base64 -d > $HOME/.kube/config

      # Update image tag in deployment
      sed -i "s|IMAGE_TAG|${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{ github.sha }}|g" k

      # Apply all manifests
      kubectl apply -f k8s/

      # Wait for rollout to complete
      kubectl rollout status deployment/app -n default --timeout=5m

      # Show deployment status
      kubectl get pods -n default
```

---

5. Configuration Multi-LLM Simplifiée

5.1 Stratégie de routage

Use Case	LLM	Mode	Justification
Données sensibles	Llama 3	Local (Ollama)	Confidentialité
Code complexe	GPT-4	Cloud (OpenRouter)	Performance
Tests	Mistral	Local (Ollama)	Économique
Documentation	Claude 3.5	Cloud (OpenRouter)	Qualité
Par défaut	Mistral	Local (Ollama)	Coût minimal

5.2 Configuration KGateway minimale

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
```

```
  name: llm-router
  namespace: default
spec:
  parentRefs:
  - name: kgateway
    namespace: default
  rules:
  # Route vers Ollama (local)
  - matches:
    - headers:
      - name: x-llm-mode
        value: local
    backendRefs:
    - name: ollama
      port: 11434

  # Route vers OpenRouter (cloud)
  - matches:
    - headers:
      - name: x-llm-mode
        value: cloud
    backendRefs:
    - name: openrouter-proxy
      port: 8080

  # Default = local
  - backendRefs:
    - name: ollama
      port: 11434
```

5.3 Déploiement Ollama simple

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ollama
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ollama
  template:
    metadata:
      labels:
        app: ollama
    spec:
```

```
      containers:
      - name: ollama
        image: ollama/ollama:latest
        ports:
        - containerPort: 11434
        resources:
          requests:
            memory: "4Gi"
            cpu: "2"
          limits:
            memory: "8Gi"
            cpu: "4"
        volumeMounts:
        - name: models
          mountPath: /root/.ollama
      volumes:
      - name: models
        emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: ollama
spec:
  selector:
    app: ollama
  ports:
  - port: 11434
    targetPort: 11434
```

---

6. Gestion des Secrets (Approche Simple)

6.1 GitHub Secrets nécessaires

```
Secrets à configurer dans GitHub (Settings → Secrets):
  KUBECONFIG:        # Config K3s encodée en base64
  OPENROUTER_KEY:    # Clé API OpenRouter
  GITHUB_TOKEN:      # Auto-généré pour GHCR
```

6.2 ConfigMap pour configuration non-sensible

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: llm-config
```

```yaml
data:
  default_model: "mistral"
  ollama_models: "llama3,mistral,codellama"
  max_retries: "3"
  timeout_seconds: "30"
```

---

7. Installation et Démarrage Rapide

7.1 Prérequis

- VM ou serveur avec 8GB RAM minimum
- Ubuntu 22.04 ou similaire
- Accès GitHub avec repo créé
- Compte OpenRouter (optionnel pour v1)

7.2 Setup en 30 minutes

```bash
# 1. Installer K3s (5 min)
curl -sfL https://get.k3s.io | sh -
sudo chmod 644 /etc/rancher/k3s/k3s.yaml

# 2. Récupérer kubeconfig pour GitHub (1 min)
cat /etc/rancher/k3s/k3s.yaml | base64 -w 0
# → Copier dans GitHub Secrets comme KUBECONFIG

# 3. Installer Ollama dans K3s (5 min)
kubectl apply -f k8s/ollama-deployment.yaml

# 4. Charger les modèles (10 min)
kubectl exec -it deployment/ollama -- ollama pull mistral
kubectl exec -it deployment/ollama -- ollama pull llama3

# 5. Déployer KGateway (5 min)
kubectl apply -f https://github.com/solo-io/gloo/releases/download/v1.15.0/gloo-gateway.yaml
kubectl apply -f k8s/kgateway-routes.yaml

# 6. Configurer OpenHands (5 min)
docker run -d \
  -e OPENROUTER_API_KEY=$OPENROUTER_KEY \
  -e LLM_BASE_URL=http://your-k3s-ip:31234 \
  -p 3000:3000 \
  ghcr.io/all-hands-ai/openhands:latest

# 7. Premier test
curl http://localhost:3000
```

---

8. Roadmap Simplifiée

Phase 1 : MVP (Semaines 1-2)

- Setup K3s basique
- GitHub + GitHub Actions
- Ollama avec 1-2 modèles
- Pipeline push simple
- Premier déploiement manuel

Phase 2 : Integration OpenHands (Semaine 3)

- Configuration OpenHands
- Connection avec GitHub
- Premier code généré
- PR automatique

Phase 3 : Multi-LLM (Semaine 4)

- KGateway routing
- OpenRouter integration
- Choix local/cloud
- Tests end-to-end

Phase 4 : Stabilisation (Semaine 5)

- Documentation
- Métriques basiques
- Optimisation coûts
- Demo finale

Évolutions futures (Post-POC)

- GitOps avec Flux (si volume justifie)
- Monitoring avancé (si production)
- Multi-environnements (si équipe grandit)
- HA et scaling (si charge augmente)

---

9. Métriques de Succès Simplifiées

9.1 KPIs essentiels uniquement

Métrique	Cible v1	Mesure
Temps setup complet	< 1 jour	Chronomètre
Code généré qui compile	$> 80\%$	GitHub Actions
Coût LLM par jour	$< €10$	OpenRouter dashboard
Temps dev économisé	$> 50\%$	Estimation manuelle
Déploiements réussis	$> 90\%$	GitHub Actions

9.2 Ce qu'on ne mesure PAS en v1

- Latence P95 (pas critique pour POC)
- Disponibilité 99.9% (pas de SLA)
- Métriques de scaling (mono-instance)
- Security scanning (POC isolé)

---

10. Risques et Mitigations

10.1 Risques principaux POC v1

Risque	Impact	Mitigation Simple
LLM génère code bugué	Haut	Review manuelle systématique
Coûts OpenRouter	Moyen	Utiliser Ollama par défaut
K3s crash	Faible	Backup configs, redéploiement rapide
Pipeline trop lent	Moyen	Cache Docker, parallélisation

10.2 Risques acceptés pour v1

- Pas de haute disponibilité
- Secrets en GitHub (pas en production)
- Pas de rollback automatique
- Monitoring minimal

---

11. Budget et Ressources

11.1 Coûts POC v1 (1 mois)

Poste	Coût estimé	Notes
VM/Serveur	€50	$1 \times$ 8GB RAM, 4 vCPU
OpenRouter	€30	Usage modéré
GitHub	€0	Free tier suffisant
Domaine	€0	IP directe pour POC

Poste	Coût estimé	Notes
Total	**€80/mois**	3× moins que v1 GitOps

11.2 Temps humain

Phase	Jours/homme	Compétences
Setup infra	1	DevOps junior OK
Config OpenHands	2	Dev + IA
Tests & debug	2	Dev
Documentation	1	Tech writer
Total	**6 jours**	vs 20 jours GitOps

---

12. Critères Go/No-Go pour v2

Passage en v2 SI :

- POC génère >60% code valide
- Économie >50% temps dev
- Coûts LLM <€200/mois
- Équipe adhère au concept
- Management valide business case

Alors v2 ajoutera :

- GitOps avec Flux
- Monitoring complet
- Multi-environnements
- Tests automatisés avancés
- Sécurité renforcée

---

13. Conclusion

Cette approche **DevOps classique simplifiée** permet de :

1. **Valider rapidement** la valeur ajoutée d'OpenHands (4 semaines vs 8)
2. **Minimiser la complexité** technique (push vs pull)
3. **Réduire les coûts** initiaux (€80/mois vs €250)
4. **Faciliter l'adoption** par l'équipe (courbe apprentissage douce)
5. **Garder la flexibilité** d'évoluer vers GitOps si succès

Le focus est mis sur l'innovation IA, pas sur l'infrastructure parfaite.

Annexes

A. Checklist de démarrage rapide

```
## Jour 1 - Infrastructure
- [ ] Provisionner VM Ubuntu 22.04
- [ ] Installer K3s
- [ ] Configurer kubectl local
- [ ] Créer repo GitHub
- [ ] Configurer GitHub Secrets

## Jour 2 - LLM Setup
- [ ] Déployer Ollama
- [ ] Charger modèles (mistral, llama3)
- [ ] Tester inference locale
- [ ] Configurer OpenRouter (optionnel)

## Jour 3 - Pipeline
- [ ] Créer workflow GitHub Actions
- [ ] Configurer GHCR
- [ ] Premier build/deploy manuel
- [ ] Vérifier pipeline complet

## Jour 4 - OpenHands
- [ ] Installer OpenHands
- [ ] Configurer agents
- [ ] Connecter à GitHub
- [ ] Premier prompt test

## Jour 5 - Integration
- [ ] Test génération code
- [ ] Test création PR
- [ ] Test déploiement auto
- [ ] Documentation
```

B. Commandes utiles

```
# Voir les pods
kubectl get pods -A

# Logs d'un pod
kubectl logs -f deployment/ollama

# Tester Ollama
curl http://localhost:11434/api/generate -d '{
  "model": "mistral",
  "prompt": "Hello world"
```

```
}'

# Redéployer manuellement
kubectl rollout restart deployment/app

# Voir les events K3s
kubectl get events --sort-by='.lastTimestamp'
```

C. Troubleshooting rapide

Problème	Solution
Ollama OOM	Augmenter memory limits
GitHub Actions fail	Vérifier KUBECONFIG secret
LLM timeout	Augmenter timeout dans config
Deploy fail	`kubectl describe pod <pod>`

Document simplifié pour POC v1
Approche: DevOps Classique (Push)
Complexité: /5
Time to Market: 4 semaines