# *TP Master 2 - GitOps & Observabilité en Production*

Monitoring de Ferme Solaire Distribuée

**Niveau :** Master 2
**Technologies :** Kubernetes, GitOps (ArgoCD), Prometheus, Grafana, Node.js/Java

Contexte Entreprise

Vous êtes ingénieur DevOps chez **SolarGrid Energy**, une entreprise qui gère 3 fermes solaires photovoltaïques réparties en France :

- **Ferme de Provence** (Marseille) : 5 000 panneaux - 2 MW installés
- **Ferme d'Occitanie** (Montpellier) : 3 500 panneaux - 1.4 MW installés
- **Ferme de Nouvelle-Aquitaine** (Bordeaux) : 4 200 panneaux - 1.68 MW installés

Chaque ferme dispose de capteurs IoT qui remontent en temps réel :
- Production électrique (kW)
- Température des panneaux (°C)
- Irradiance solaire (W/m²)
- Tension et courant (V, A)
- État de santé des onduleurs
- Taux de disponibilité

Problématique Business

Le directeur technique vous confie une mission critique :

> *"Nous perdons 150 000€/an à cause de pannes non détectées et d'une maintenance réactive. Nous avons besoin d'une plateforme de monitoring temps réel avec alertes prédictives, déployée via GitOps pour garantir la reproductibilité sur nos 3 sites. Le système doit respecter nos SLO : 99.5% de disponibilité et détection d'anomalie < 2 minutes."*

Objectifs Pédagogiques

À l'issue de ce TP, vous serez capable de :
1. Implémenter une architecture GitOps complète avec ArgoCD
2. Déployer une stack d'observabilité (Prometheus + Grafana)
3. Créer des métriques métier personnalisées (SLI/SLO)
4. Configurer des alertes intelligentes basées sur des seuils réels
5. Optimiser les coûts (FinOps) avec requests/limits appropriés
6. Documenter une architecture production-ready

Données Techniques Réelles

Dataset Fourni

**Vous disposez d'un dataset complet de 30 jours de production réelle** dans le dossier `data/` :

- **provence_data.csv** : 720 lignes (30 jours × 24h) - Ferme de Marseille
- **occitanie_data.csv** : 720 lignes - Ferme de Montpellier
- **aquitaine_data.csv** : 720 lignes - Ferme de Bordeaux
- **anomalies_log.csv** : Log des 15+ anomalies injectées
- **README_DATASET.md** : Documentation complète du dataset

**Données incluses par ligne :**
- Timestamp, irradiance (W/m²), températures (°C)
- Production réelle vs théorique (kW)
- État des onduleurs, rendement (%)
- Revenus journaliers (€)
- Type et sévérité des anomalies

**Anomalies réelles injectées :**
- Canicule (Provence, jours 15-17) : Surchauffe > 70°C
- Panne onduleur (Occitanie, jour 8) : -33% production
- Dégradation (Aquitaine, jours 5-25) : -15% sur certains panneaux
- Ombrage matinal (Aquitaine, jours 12-14) : Brouillard 6h-10h
- Défaillance capteur (Provence, jour 20) : Données manquantes

**Note :** Vous pouvez utiliser ce dataset pour :
- Tester votre simulateur (comparer avec données réelles)
- Alimenter Prometheus avec des données historiques
- Valider vos alertes sur des anomalies connues
- Créer des visualisations Grafana réalistes

Spécifications Panneaux Solaires (Standard industriel)

**Modèle :** Panneaux monocristallins 400Wc

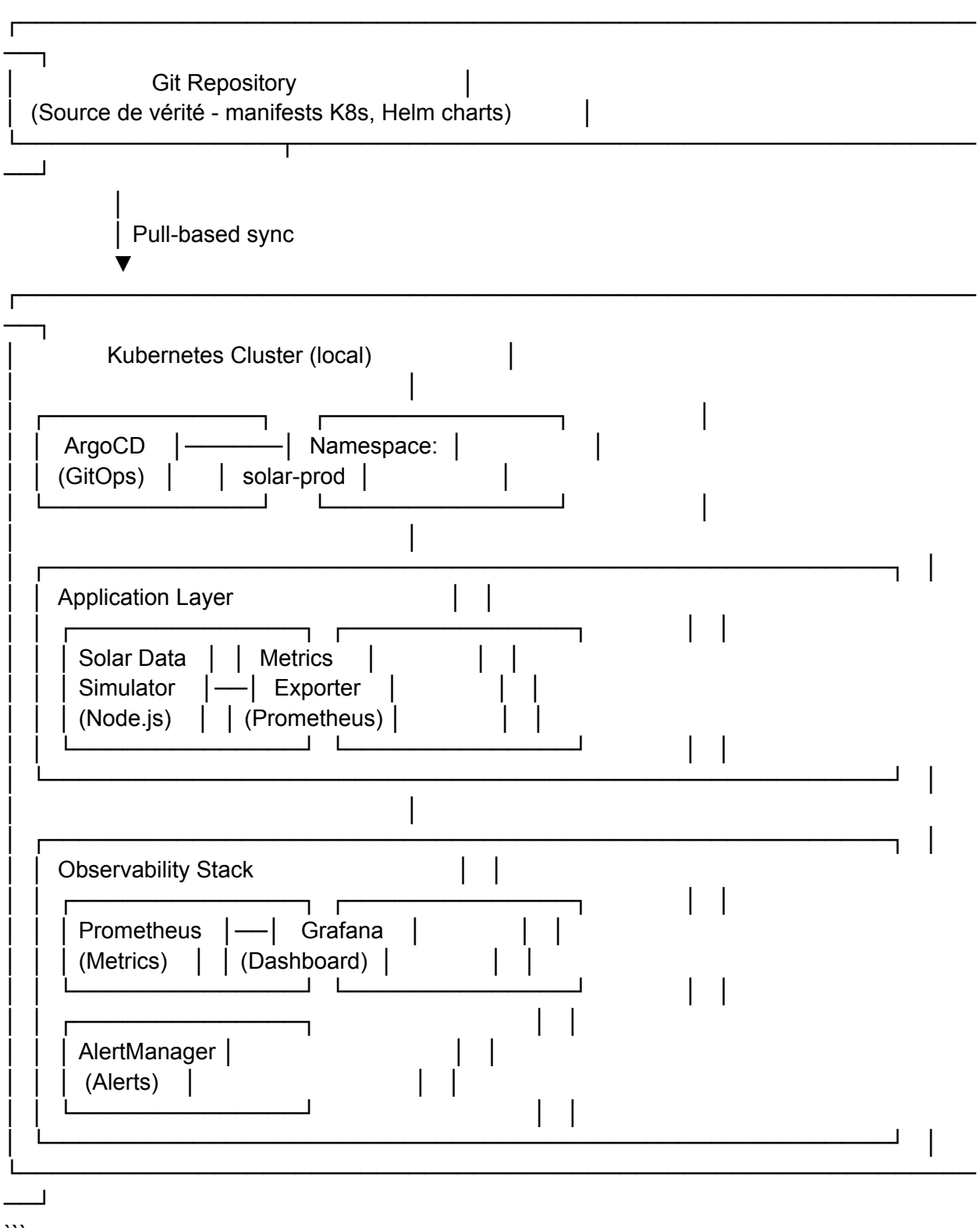
| Paramètre | Valeur Nominale | Plage Normale | Seuil Critique |
|--|-|||-|
| Puissance crête | 400 W | 360-400 W | < 320 W |
| Tension optimale (Vmp) | 41.5 V | 38-44 V | < 35 V ou > 48 V |
| Courant optimal (Imp) | 9.65 A | 8.5-10.2 A | < 7 A |
| Température fonctionnement | 25°C (STC) | 15-45°C | > 65°C |
| Rendement | 20.5% | 18-21% | < 15% |
| Coefficient température | -0.35%/°C | - | - |

Formules de Calcul
**Production théorique instantanée :**
```
P(t) = Nombre_panneaux × Puissance_crête × (Irradiance / 1000) × η_système ×
Facteur_température
```
Où :

- `Irradiance` : W/m² (0-1200 W/m² en France)
- `η_système` : 0.85 (pertes câblage, onduleur, poussière)
- `Facteur_température` : 1 + (T_panneau - 25) × (-0.0035)
**Irradiance selon l'heure (modèle simplifié) :**
```
Irradiance(h) = Irradiance_max × sin(π × (h - 6) / 12)  pour h ∈ [6h, 18h]
Irradiance(h) = 0  pour h ∈ [18h, 6h]
```
**Revenus journaliers :**
```
Revenus = Énergie_produite(kWh) × Tarif_rachat(€/kWh)
Tarif_rachat = 0.18 €/kWh (contrat EDF OA)
```
Patterns d'Anomalies à Détecter

1. **Dégradation progressive** : Baisse de rendement > 10% sur 7 jours
2. **Panne onduleur** : Production = 0 alors que Irradiance > 200 W/m²
3. **Surchauffe** : Température > 65°C pendant > 10 minutes
4. **Ombrage partiel** : Production < 50% du théorique avec irradiance normale
5. **Déconnexion capteur** : Absence de données > 5 minutes

Architecture à Implémenter



Travail à Réaliser
Phase 1 : Préparation de l'Infrastructure
1.1 Setup Kubernetes Local
```bash
*Créer un cluster local (kind ou minikube)*
kind create cluster --name solar-monitoring
```
1.2 Installation ArgoCD

```bash

*Installer ArgoCD dans le cluster*
kubectl create namespace argocd
kubectl apply -n argocd -f
https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
*Exposer l'UI ArgoCD*
kubectl port-forward svc/argocd-server -n argocd 8080:443
```

**Livrables :**

- Cluster Kubernetes opérationnel
- ArgoCD installé et accessible
- Screenshot de l'UI ArgoCD

Phase 2 : Développement de l'Application (1h15)

2.1 Simulateur de Données Solaires

Créer une application (Node.js **OU** Java) qui :

**Fonctionnalités requises :**
1. Simule les 3 fermes solaires avec données réalistes
2. Génère des métriques toutes les 30 secondes
3. Expose un endpoint `/metrics` au format Prometheus
4. Implémente les formules de calcul fournies
5. Injecte des anomalies aléatoires (10% du temps)

**Métriques à exposer :**
```prometheus
HELP solar_power_watts Production électrique instantanée
TYPE solar_power_watts gauge
solar_power_watts{farm="provence",panel_id="P001"} 385.2
HELP solar_irradiance_wm2 Irradiance solaire mesurée
```

```
 TYPE solar_irradiance_wm2 gauge
solar_irradiance_wm2{farm="provence"} 850.5
 HELP solar_panel_temperature_celsius Température du panneau
 TYPE solar_panel_temperature_celsius gauge
solar_panel_temperature_celsius{farm="provence",panel_id="P001"} 42.3
 HELP solar_inverter_status État de l'onduleur (1=OK, 0=KO)
 TYPE solar_inverter_status gauge
solar_inverter_status{farm="provence",inverter_id="INV01"} 1
 HELP solar_daily_revenue_euros Revenus journaliers estimés
 TYPE solar_daily_revenue_euros counter
solar_daily_revenue_euros{farm="provence"} 1250.80
```

**Contraintes techniques :**

- Dockeriser l'application
- Image < 200 MB
- CPU request: 100m, limit: 200m
- Memory request: 128Mi, limit: 256Mi

**Livrables :**

- Code source de l'application
- Dockerfile optimisé
- Tests unitaires (au moins 3)
- Documentation API

Phase 3 : Configuration GitOps

3.1 Structure du Repository Git

Créer la structure suivante :

```
solar-monitoring/
├── README.md
├── apps/
│   └── solar-simulator/
│       ├── deployment.yaml
│       ├── service.yaml
│       ├── configmap.yaml
│       └── kustomization.yaml
├── monitoring/
│   ├── prometheus/
│   │   ├── prometheus-config.yaml
```

```
│   │   ├── prometheus-deployment.yaml
│   │   ├── service.yaml
│   │   └── servicemonitor.yaml
│   ├── grafana/
│   │   ├── deployment.yaml
│   │   ├── service.yaml
│   │   └── dashboards/
│   │       └── solar-dashboard.json
│   └── alertmanager/
│       ├── config.yaml
│       └── rules.yaml
└── argocd/
    ├── application-solar.yaml
    └── application-monitoring.yaml
```

### 3.2 Règles d'Alerting Prometheus

Créer au minimum 5 alertes pertinentes :

**Exemple d'alerte :**

```yaml
groups:
  - name: solar_alerts
    interval: 30s
    rules:
      - alert: SolarPanelOverheating
        expr: solar_panel_temperature_celsius > 65
        for: 10m
        labels:
          severity: critical
          component: panel
        annotations:
          summary: "Panneau en surchauffe"
          description: "Le panneau {{ $labels.panel_id }} de la ferme {{ $labels.farm }} dépasse 65°C"
```

**Alertes à implémenter :**

1. Surchauffe panneau (> 65°C pendant 10 min)
2. Panne onduleur (status = 0)
3. Production anormalement basse (< 50% du théorique)
4. Perte de données capteur (absence > 5 min)
5. SLO breach : Disponibilité < 99.5%

**Livrables :**

- Repository Git structuré
- Manifests Kubernetes valides
- Configuration Prometheus complète
- 5 règles d'alerting fonctionnelles

Phase 4 : Observabilité & Dashboards

4.1 Dashboard Grafana

Créer un dashboard avec **minimum 6 panneaux** :
1. **Production Totale en Temps Réel** (Gauge)
   - Agrégation des 3 fermes
   - Seuils : Vert > 80%, Orange 50-80%, Rouge < 50%
2. **Historique Production par Ferme** (Time Series)
   - Courbes distinctes pour chaque ferme
   - Période : 24h
3. **Carte de Chaleur Température** (Heatmap)
   - Distribution des températures par panneau
4. **Taux de Disponibilité (SLO)** (Stat)
   - Calcul : (uptime / total_time) × 100
   - Objectif : 99.5%
5. **Revenus Journaliers** (Bar Chart)
   - Par ferme
   - Comparaison J vs J-1
6. **Alertes Actives** (Table)
   - Liste des alertes en cours
   - Severité + timestamp

**Requêtes PromQL attendues :**
```promql
 Production totale
sum(solar_power_watts) / 1000
 Disponibilité sur 24h
(1 - (sum(rate(solar_inverter_status{status="0"}[24h])) / count(solar_inverter_status))) * 100
 Revenus journaliers
sum(increase(solar_daily_revenue_euros[1d])) by (farm)
```

**Livrables :**
- Dashboard Grafana exporté (JSON)
- Screenshot du dashboard avec données
- Documentation des requêtes PromQL

Phase 5 : FinOps & Optimisation

5.1 Analyse des Coûts

Calculer le coût mensuel estimé de votre infrastructure :
**Hypothèses (cluster cloud équivalent) :**
- Node : 0.05 €/h par vCPU
- Stockage : 0.10 €/GB/mois
- Réseau : 0.09 €/GB sortant
**Tableau à compléter :**
| Composant | CPU (m) | Memory (Mi) | Stockage (GB) | Coût/mois |
|--||-||--|
| Solar Simulator | ? | ? | 0 | ? |
| Prometheus | ? | ? | 10 | ? |
| Grafana | ? | ? | 2 | ? |
| AlertManager | ? | ? | 1 | ? |
| **TOTAL** | | | | **?** |

5.2 Optimisations

Proposer 3 optimisations concrètes avec impact chiffré :
**Exemple :**
- Avant : `limits.memory: 512Mi` (surdimensionné)
- Après : `limits.memory: 256Mi` (ajusté via VPA)
- Économie : -50% sur ce composant = -15€/mois
**Livrables :**
- Tableau de coûts complété
- 3 optimisations documentées
- Manifests mis à jour avec optimisations

Phase 6 : Documentation & Démo

6.1 README Technique

Votre README doit contenir :

1. **Architecture** : Schéma + explications
2. **Installation** : Commandes pas-à-pas
3. **Utilisation** : Accès aux interfaces
4. **Métriques** : Liste exhaustive avec descriptions
5. **Alertes** : Conditions de déclenchement
6. **Troubleshooting** : 5 problèmes courants + solutions
7. **Améliorations futures** : 3 évolutions possibles

6.2 Scénario de Démonstration

Préparer une démo de 5 minutes montrant :
1. Déploiement via ArgoCD (sync automatique)
2. Production normale des 3 fermes
3. Injection d'une anomalie (surchauffe)
4. Déclenchement de l'alerte (< 2 min)
5. Visualisation dans Grafana

**Livrables :**
- README.md complet et professionnel
- Script de démo automatisé
- Vidéo ou screenshots de la démo

Bonus possibles

- Chiffrement secrets avec SOPS (+2)
- Dashboard mobile-responsive (+1)
- Tests d'intégration avec Helm test (+2)
- Multi-cluster simulation (+3)
- Prédiction ML des pannes (+5)

Format de Rendu

Repository GitHub

**Structure attendue :**

```
<votre-repo>/
├── README.md (documentation principale)
├── docs/
│   ├── ARCHITECTURE.md
│   ├── INSTALLATION.md
│   └── screenshots/
├── src/
│   └── solar-simulator/
│       ├── app/ (code source)
│       ├── Dockerfile
│       ├── package.json (ou pom.xml)
│       └── tests/
├── k8s/
│   ├── apps/
│   ├── monitoring/
│   └── argocd/
└── scripts/
    ├── setup.sh
```

└── demo.sh
```

Modalités

1. **Push sur GitHub** : Repository public ou privé (nous donner accès)
2. **Nom du repo** : `solar-monitoring-gitops-<nom-prenom>`
3. **Deadline** : À définir par le professeur
4. **Envoi** : Lien GitHub par email

**Checklist avant envoi :**

- Le README contient les instructions d'installation
- Tous les manifests K8s sont valides (`kubectl apply --dry-run`)
- L'application se build sans erreur
- Les screenshots sont présents dans `/docs/screenshots/`
- Le repository est propre (pas de `node_modules/`, `.env`, etc.)

Ressources Utiles

Documentation Officielle

- [ArgoCD Getting Started](https://argo-cd.readthedocs.io/en/stable/getting_started/)
- [Prometheus Querying](https://prometheus.io/docs/prometheus/latest/querying/basics/)
- [Grafana Dashboards](https://grafana.com/docs/grafana/latest/dashboards/)
- Kubernetes Best Practices

Bibliothèques Utiles

- **Node.js** : `prom-client` (métriques Prometheus)
- **Java** : `io.prometheus:simpleclient` (métriques Prometheus)
- **Helm** : Charts pour Prometheus/Grafana

Dataset Fourni (Dossier `data/`)

**Vous disposez de données réelles françaises :**
- **provence_data.csv** : 720 lignes (30 jours × 24h)
- **occitanie_data.csv** : 720 lignes
- **aquitaine_data.csv** : 720 lignes
- **anomalies_log.csv** : Log des anomalies injectées
- **README_DATASET.md** : Documentation complète

Note : Utilisez ces données pour valider votre simulateur et tester vos alertes.
```

FAQ

**Q : Peut-on utiliser Helm au lieu de manifests bruts ?**
R : Oui, c'est même encouragé. Helm + Kustomize = best practice.
**Q : Faut-il vraiment 3 fermes ou peut-on simplifier ?**
R : Minimum 2 fermes pour démontrer l'agrégation multi-sites.
**Q : Les données doivent-elles être persistées ?**
R : Non obligatoire pour ce TP, mais un PVC pour Prometheus est un plus.
**Q : Peut-on travailler en binôme ?**
R : À décider avec le professeur. Si oui, préciser les contributions de chacun.

Conseils de Réussite

**Tips importants :**
1. Commencez par lire TOUT l'énoncé
2. Gérez votre temps
3. Testez au fur et à mesure
4. Prenez des screenshots régulièrement
5. N'hésitez pas à poser des questions

Ce TP reflète des problématiques réelles d'industrialisation. Les compétences acquises sont directement applicables en entreprise sur des projets de monitoring IoT, FinTech, e-commerce, etc.