TP JOUR 5 - Platform Engineering & SRE Avancé

Master 2 Full Stack - Docker & Kubernetes

Durée estimée : 3 heures

Contexte du TP

Vous êtes Platform Engineer dans TechMarket, une startup en pleine croissance qui développe une marketplace e-commerce (type Vinted/Leboncoin). L'entreprise compte maintenant 5 équipes de développement et doit industrialiser ses pratiques.

Problématiques actuelles

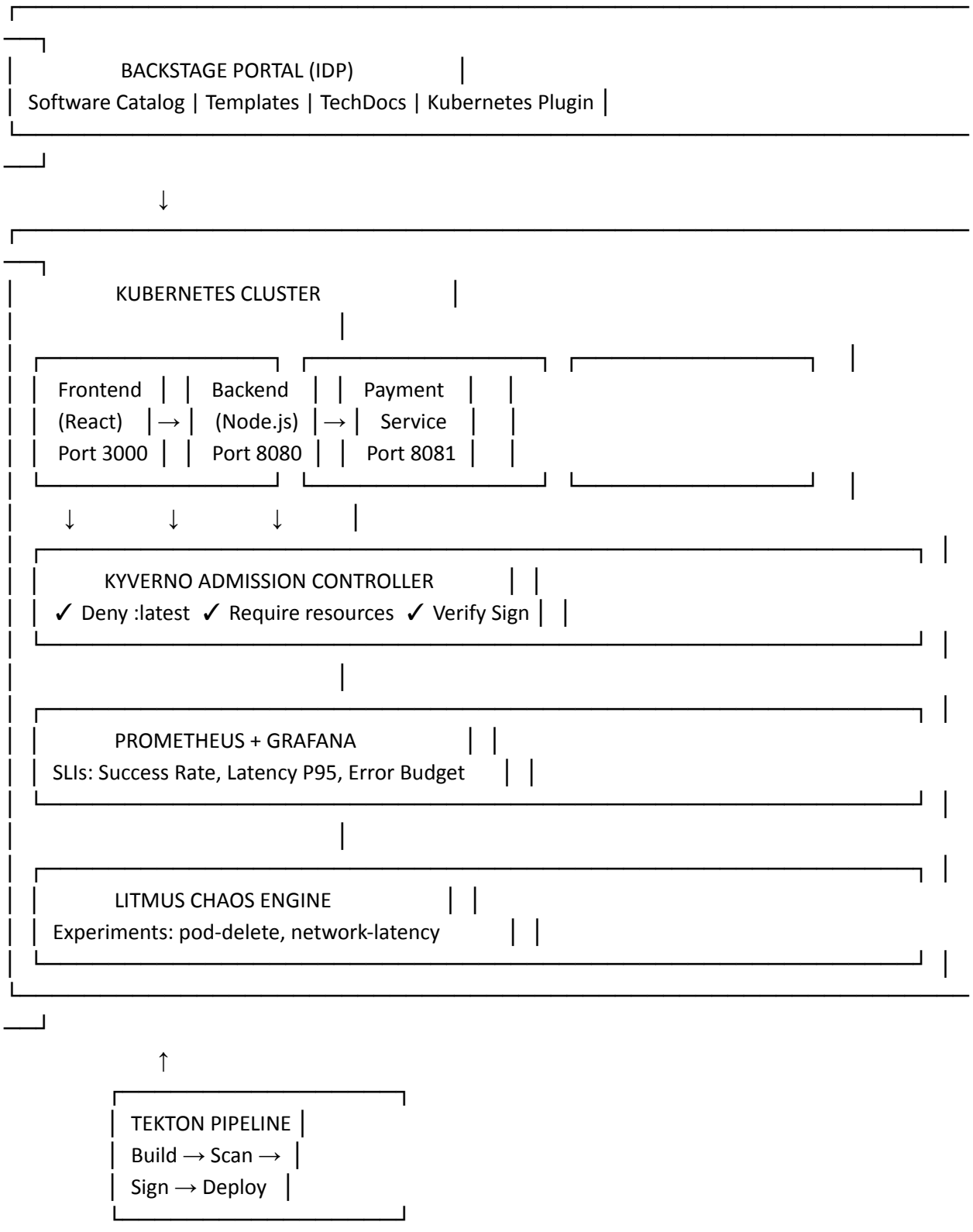
- Chaos organisationnel : Chaque équipe déploie différemment (scripts manuels, configurations divergentes)
- Incidents fréquents : Pannes en production dues à des images non testées, configurations non sécurisées
- Lenteur des déploiements : 2 semaines pour provisionner un nouvel environnement
- Manque de visibilité : Impossible de savoir qui possède quel service, quelle version est en prod
- Sécurité faible : Pas de validation des images, secrets en clair, containers en root

Votre mission

Mettre en place une Internal Developer Platform (IDP) complète intégrant :

1. Un portail self-service (Backstage)
2. Des politiques de sécurité automatisées (Kyverno)
3. Une culture SRE avec chaos engineering (Litmus)
4. Des pipelines CI/CD sécurisés (Tekton + Cosign)

🏗️ Architecture de la Plateforme TechMarket



📋 Prérequis

Outils nécessaires

- `kubectl` (v1.28+)
- `helm` (v3.12+)
- `kind` ou `minikube` (cluster Kubernetes local)
- `docker` (v24+)
- `cosign` (v2.0+)
- `tkn` (Tekton CLI)
- Compte GitHub (pour OAuth et registry)

Vérification de l'environnement

```bash
*Vérifier les versions*
kubectl version --client
helm version
docker version
cosign version
*Créer le cluster Kubernetes (Kind recommandé)*
kind create cluster --name techmarket --config kind-config.yaml
*Vérifier que le cluster est opérationnel*
kubectl cluster-info
kubectl get nodes
```

🎓 BLOC 1 - Platform Engineering & Backstage

Objectifs

- Déployer Backstage comme portail IDP
- Créer un Software Catalog pour les 3 microservices
- Créer un Software Template (Golden Path) pour scaffolder de nouveaux services
- Intégrer le plugin Kubernetes

Exercice 1.1 - Installation de Backstage

Contexte : L'équipe a besoin d'un point d'entrée unique pour visualiser tous les services.

```bash
*TODO: Installer Backstage via Helm*
*Namespace: backstage*
*Base de données: PostgreSQL incluse*
```

Ingress: Activé (NodePort pour local)
```
Questions :
1. Quelle est l'URL d'accès à Backstage après installation ?
2. Combien de pods sont créés dans le namespace `backstage` ?
Exercice 1.2 - Création du Software Catalog
Contexte : Les développeurs ne savent pas qui maintient quel service. Il faut centraliser cette information.

Créez 3 fichiers `catalog-info.yaml` pour :
- frontend-app (équipe: team-frontend, lifecycle: production)
- backend-api (équipe: team-backend, lifecycle: production)
- payment-service (équipe: team-payments, lifecycle: production)

Spécifications :
- Type: `service`
- Annotations Kubernetes: `backstage.io/kubernetes-id`
- Liens vers les repos GitHub (utilisez votre compte)
Fichier à créer : `td/backstage/catalog-info-frontend.yaml`


Exercice 1.3 - Software Template (Golden Path)
Contexte : Créer un nouveau service prend 2 semaines. Il faut un template pour générer automatiquement :
- Un repository Git
- Un Dockerfile optimisé
- Des manifests Kubernetes (Deployment, Service, HPA)
- Un fichier catalog-info.yaml
TODO : Créez un template `create-nodejs-service` qui :
1. Demande le nom du service
2. Demande l'équipe propriétaire
3. Génère un squelette Node.js Express
4. Crée le repo sur GitHub
5. Enregistre automatiquement le service dans le catalog

Fichier à créer : `td/backstage/template-nodejs-service.yaml`
Aide : Utilisez les actions Backstage :
- `fetch:template`
- `publish:github`
- `catalog:register`
Exercice 1.4 - Plugin Kubernetes
Contexte : Les développeurs veulent voir l'état de leurs pods directement dans Backstage.
```

TODO :
1. Configurez le plugin Kubernetes dans Backstage
2. Ajoutez l'annotation `backstage.io/kubernetes-id` dans les catalog-info.yaml
3. Vérifiez que les pods sont visibles dans l'interface
Livrable : Screenshot de l'interface Backstage montrant les pods du `payment-service`

🔒 BLOC 2 - Policy as Code & Supply Chain Security (45 min)

Objectifs

- Déployer Kyverno comme admission controller
- Créer des ClusterPolicies pour sécuriser le cluster
- Signer des images avec Cosign (keyless)
- Vérifier les signatures à l'admission
Exercice 2.1 - Installation de Kyverno
Contexte : Incident récent : un développeur a déployé un pod en `privileged: true` qui a compromis le cluster.
```bash
*TODO: Installer Kyverno en mode High Availability*
*Namespace: kyverno*
*Replicas: 3*
```
Question : Listez les CRDs créées par Kyverno (`kubectl get crd | grep kyverno`)

Exercice 2.2 - Politique "Deny Latest Tag"

Contexte : 30% des incidents sont dus à des images `:latest` non versionnées.

TODO : Créez une `ClusterPolicy` qui :
- Nom: `disallow-latest-tag`
- Mode: `Enforce` (bloque les déploiements)
- Cible: Tous les Pods
- Règle: Refuse les images avec le tag `:latest`
Fichier à créer : `td/kyverno/policy-deny-latest.yaml`
Test : Essayez de déployer `nginx:latest` → doit être refusé
Exercice 2.3 - Politique "Require Resource Limits"
Contexte : Des pods sans limites de ressources ont causé un OOMKill généralisé.
TODO : Créez une `ClusterPolicy` qui :

- Nom: `require-resource-limits`
- Exige `resources.limits.memory` et `resources.limits.cpu` sur tous les containers
- Mode: `Enforce`
Fichier à créer : `td/kyverno/policy-require-limits.yaml`
Test : Déployez un pod sans limits → doit être refusé

Exercice 2.4 - Mutation : Ajout automatique de labels

Contexte : Pour le FinOps, tous les pods doivent avoir un label `cost-center`.

TODO :
Créez une `ClusterPolicy` de type Mutation qui :
- Ajoute automatiquement le label `cost-center: techmarket`
- Ajoute le label `managed-by: platform-team`
Fichier à créer : `td/kyverno/policy-add-labels.yaml`
Test : Déployez un pod sans ces labels → ils doivent être ajoutés automatiquement

Exercice 2.5 - Signature d'images avec Cosign (Keyless)

Contexte : Supply chain attack : une image malveillante a été déployée. Il faut garantir la provenance.
TODO :

1. Construisez l'image du `payment-service`
2. Poussez-la sur GitHub Container Registry (ghcr.io)

3. Signez-la avec Cosign en mode keyless (OIDC GitHub)
```bash
*TODO: Compléter ces commandes*
docker build -t ghcr.io/VOTRE_USER/payment-service:v1.0.0 ./payment-service
docker push ghcr.io/VOTRE_USER/payment-service:v1.0.0
*Signature keyless*
cosign sign --yes ghcr.io/VOTRE_USER/payment-service:v1.0.0
*Vérification*
cosign verify \
  --certificate-identity=VOTRE_EMAIL \
  --certificate-oidc-issuer=https://github.com/login/oauth \
  ghcr.io/VOTRE_USER/payment-service:v1.0.0
```

Livrable : Output de la commande `cosign verify`

Exercice 2.6 - Vérification de signature avec Kyverno

Contexte : Seules les images signées doivent pouvoir être déployées en production.

TODO : Créez une `ClusterPolicy` qui :

- Nom: `verify-image-signature`
- Vérifie la signature Cosign pour toutes les images `ghcr.io/VOTRE_USER/`
- Utilise l'identité OIDC de votre email GitHub

Fichier à créer : `td/kyverno/policy-verify-signature.yaml`

Aide :

```yaml
spec:
  rules:
  - name: check-signature
    match:
      resources:
        kinds:
        - Pod
    verifyImages:
    - imageReferences:
      - "ghcr.io/VOTRE_USER/"
      attestors:
      - entries:
        - keyless:
            subject: "VOTRE_EMAIL"
            issuer: "https://github.com/login/oauth"
            rekor:
              url: https://rekor.sigstore.dev
```

Test : Déployez une image non signée → doit être refusée

—

📊 BLOC 3 - SRE Culture & Chaos Engineering (45 min)

Objectifs

- Définir des SLIs/SLOs pour le payment-service
- Calculer un Error Budget
- Installer Litmus Chaos

- Exécuter un chaos experiment (pod-delete)
- Rédiger un postmortem blameless

Exercice 3.1 - Définition des SLIs/SLOs

Contexte : Le payment-service est critique. Un downtime = perte de revenus directe.

SLO cible : 99.9% de disponibilité sur 30 jours (43 minutes 49 secondes de downtime max)
TODO : Créez des recording rules Prometheus pour :
1. SLI Success Rate : Ratio de requêtes HTTP réussies (status != 5xx)
2. SLI Latency P95 : 95e percentile de latence < 500ms

Fichier à créer : `td/prometheus/slo-rules.yaml`
```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-slo-rules
  namespace: monitoring
data:
  slo-rules.yml: |
    groups:
    - name: payment-service-slo
      interval: 30s
      rules:
       TODO: Créer les recording rules
      - record: payment_service:success_rate:ratio
        expr: |
          Votre expression PromQL ici

      - record: payment_service:latency:p95
        expr: |
          Votre expression PromQL ici
```

Exercice 3.2 - Calcul de l'Error Budget

Contexte : Vous devez calculer combien de temps d'indisponibilité il reste ce mois-ci.

Données :

- SLO: 99.9%

- Période: 30 jours
- Downtime déjà consommé: 15 minutes

Questions :

1. Quel est l'Error Budget total en minutes ?
2. Combien de minutes reste-t-il ?
3. En pourcentage, combien d'Error Budget reste-t-il ?
4. Si le service tombe pendant 30 minutes supplémentaires, que se passe-t-il ?

Livrable : Fichier `td/sre/error-budget-calculation.md` avec vos calculs

Exercice 3.3 - Installation de Litmus Chaos

Contexte : Avant de déployer en prod, il faut valider la résilience du système.

```bash
 TODO: Installer Litmus Chaos
 Namespace: litmus
 Mode: Cluster (pas Namespace)
 Portal: Activé (NodePort)
```

Question : Quelle est l'URL du Chaos Center ?

Exercice 3.4 - Chaos Experiment : Pod Delete

Contexte : Que se passe-t-il si un pod du payment-service crashe brutalement ?

Hypothèse : "Le payment-service doit rester disponible même si 1 pod sur 3 est supprimé"

TODO : Créez un `ChaosEngine` qui :
- Cible: `payment-service` deployment
- Experiment: `pod-delete`
- Durée: 30 secondes
- Nombre de pods à supprimer: 1

Fichier à créer : `td/litmus/chaos-pod-delete.yaml`

Aide :

```yaml
apiVersion: litmuschaos.io/v1alpha1
kind: ChaosEngine
metadata:
  name: payment-chaos
  namespace: default
```

```
spec:
  appinfo:
    appns: 'default'
    applabel: 'app=payment-service'
    appkind: 'deployment'
  engineState: 'active'
  chaosServiceAccount: litmus-admin
  experiments:
  - name: pod-delete
    spec:
      components:
        env:
        - name: TOTAL_CHAOS_DURATION
          value: '30'
        - name: CHAOS_INTERVAL
          value: '10'
        - name: FORCE
          value: 'false'
```

Test :

1. Appliquez le ChaosEngine
2. Observez les pods : `kubectl get pods -w`
3. Vérifiez les métriques Prometheus (Success Rate doit rester > 99%)

Exercice 3.5 - Postmortem Blameless

Contexte : Le chaos test a révélé un problème : le Success Rate est tombé à 95% pendant 20 secondes.

TODO : Rédigez un postmortem selon le template suivant :

Fichier à créer : `td/sre/postmortem-pod-delete.md`

```markdown
Postmortem - Chaos Test Pod Delete Payment Service
Date : [DATE]
Durée de l'incident : 20 secondes
Impact : Success Rate à 95% (SLO: 99.9%)
Détecté par : Chaos Experiment Litmus
Chronologie
- 14:00:00 - Démarrage du chaos experiment
- 14:00:05 - Suppression d'un pod payment-service
- 14:00:05 - Success Rate chute à 95%
- 14:00:25 - Nouveau pod démarré et healthy
- 14:00:25 - Success Rate revient à 100%
Cause Racine
```

[À COMPLÉTER : Pourquoi le service a-t-il été impacté ?]
Ce qui a bien fonctionné
[À COMPLÉTER]
Ce qui a mal fonctionné
[À COMPLÉTER]
Actions correctives
[À COMPLÉTER : Que faut-il améliorer ?]
- [ ] Action 1
- [ ] Action 2
Leçons apprises
[À COMPLÉTER]
```
Indices :
- Vérifiez le `minReplicas` du HPA
- Vérifiez le `PodDisruptionBudget`
- Vérifiez les `readinessProbe` et `livenessProbe`

🚀 BLOC 4 - CI/CD Avancé avec Tekton (45 min)
Objectifs

- Installer Tekton Pipelines
- Créer un pipeline Build → Test → Scan → Sign → Deploy
- Intégrer la signature Cosign dans le pipeline
- Générer un SBOM (Software Bill of Materials)

Exercice 4.1 - Installation de Tekton

Contexte : Les pipelines actuels (scripts bash) ne sont pas reproductibles ni auditables.
```bash
*TODO: Installer Tekton Pipelines et Dashboard*
kubectl apply -f https://storage.googleapis.com/tekton-releases/pipeline/latest/release.yaml
kubectl apply -f https://storage.googleapis.com/tekton-releases/dashboard/latest/release.yaml
*Vérifier l'installation*
kubectl get pods -n tekton-pipelines
```
Exercice 4.2 - Task : Build & Push avec Kaniko

Contexte : Construire des images Docker dans Kubernetes nécessite Kaniko (pas de Docker daemon).
```

TODO : Créez une `Task` Tekton qui :

- Nom: `kaniko-build-push`
- Paramètres: `IMAGE` (nom complet de l'image)
- Workspace: `source` (code source)
- Action: Build et push vers ghcr.io

Fichier à créer : `td/tekton/task-kaniko.yaml`
Aide :
```yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: kaniko-build-push
spec:
  params:
  - name: IMAGE
    type: string
  workspaces:
  - name: source
  steps:
  - name: build-and-push
    image: gcr.io/kaniko-project/executor:latest
    args:
    - "--dockerfile=$(workspaces.source.path)/Dockerfile"
    - "--context=$(workspaces.source.path)"
    - "--destination=$(params.IMAGE)"
    - "--cache=true"
```
Exercice 4.3 - Task : Scan de vulnérabilités avec Trivy

Contexte : 15% des images déployées contiennent des CVE critiques.

TODO : Créez une `Task` qui :
- Nom: `trivy-scan`
- Paramètre: `IMAGE`
- Action: Scan de l'image et échec si CVE CRITICAL trouvées
Fichier à créer : `td/tekton/task-trivy.yaml`
```yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
```

```
  name: trivy-scan
spec:
  params:
  - name: IMAGE
    type: string
  steps:
  - name: scan
    image: aquasec/trivy:latest
    command:
    - trivy
    args:
    - image
    - --exit-code
    - "1"    Fail si vulnérabilités trouvées
    - --severity
    - CRITICAL,HIGH
    - $(params.IMAGE)
```

Exercice 4.4 - Task : Signature avec Cosign

TODO : Créez une `Task` qui signe l'image en mode keyless.

Fichier à créer : `td/tekton/task-cosign.yaml`

```yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: cosign-sign
spec:
  params:
  - name: IMAGE
    type: string
  steps:
  - name: sign
    image: gcr.io/projectsigstore/cosign:latest
    command:
    - cosign
    args:
    - sign
    - --yes
    - $(params.IMAGE)
```

Exercice 4.5 - Task : Génération du SBOM

Contexte : Pour la compliance, il faut un inventaire de toutes les dépendances.

TODO : Créez une `Task` qui génère un SBOM avec Syft.

Fichier à créer : `td/tekton/task-sbom.yaml`
```yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: generate-sbom
spec:
  params:
  - name: IMAGE
    type: string
  steps:
  - name: syft
    image: anchore/syft:latest
    command:
    - syft
    args:
    - $(params.IMAGE)
    - -o
    - spdx-json
    - --file
    - /workspace/sbom.json
```
Exercice 4.6 - Pipeline complete

TODO : Assemblez toutes les Tasks dans un `Pipeline` :

Fichier à créer : `td/tekton/pipeline-secure-build.yaml`
```yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: secure-build-pipeline
spec:
  params:
  - name: IMAGE
    type: string
```

```
  - name: GIT_REPO
    type: string
  - name: GIT_REVISION
    type: string
    default: main
  workspaces:
  - name: shared-workspace
  tasks:
    TODO: Définir les tasks dans l'ordre
    1. git-clone
    2. kaniko-build-push
    3. trivy-scan (runAfter: kaniko)
    4. generate-sbom (runAfter: trivy)
    5. cosign-sign (runAfter: sbom)
```

Exercice 4.7 - Exécution du Pipeline

TODO : Créez un `PipelineRun` pour builder le payment-service.
Fichier à créer : `td/tekton/pipelinerun-payment.yaml`

```yaml
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: payment-service-build-1
spec:
  pipelineRef:
    name: secure-build-pipeline
  params:
  - name: IMAGE
    value: ghcr.io/VOTRE_USER/payment-service:v1.0.0
  - name: GIT_REPO
    value: https://github.com/VOTRE_USER/techmarket-payment
  - name: GIT_REVISION
    value: main
  workspaces:
  - name: shared-workspace
    volumeClaimTemplate:
      spec:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
```

storage: 1Gi
```
Commandes :
```bash
*Appliquer le PipelineRun*
kubectl apply -f td/tekton/pipelinerun-payment.yaml
*Suivre l'exécution*
tkn pipelinerun logs payment-service-build-1 -f
*Vérifier le statut*
tkn pipelinerun describe payment-service-build-1
```
Livrable : Screenshot du Dashboard Tekton montrant le pipeline réussi

🎁 BONUS (si temps restant)
Bonus 1 - Argo Rollouts : Canary Deployment
Déployez le payment-service avec une stratégie Canary :
- 10% du trafic vers la nouvelle version
- Pause de 2 minutes
- Si Success Rate > 99%, continuer
- Sinon, rollback automatique
Fichier : `td/bonus/rollout-canary.yaml`

Bonus 2 - Crossplane : Provisionner une base de données
Utilisez Crossplane pour créer une instance PostgreSQL pour le payment-service.
Fichier : `td/bonus/crossplane-postgres.yaml`

Bonus 3 - Intégration complète
Intégrez le pipeline Tekton dans Backstage :
- Ajoutez l'annotation `tekton.dev/pipeline` dans catalog-info.yaml
- Visualisez les builds directement dans le portail

📦 Livrables attendus
À la fin du TP, vous devez avoir dans le dossier `td/` :
```
td/
├── backstage/
│   ├── catalog-info-frontend.yaml
│   ├── catalog-info-backend.yaml
│   ├── catalog-info-payment.yaml
```
```

```
│   └── template-nodejs-service.yaml
├── kyverno/
│   ├── policy-deny-latest.yaml
│   ├── policy-require-limits.yaml
│   ├── policy-add-labels.yaml
│   └── policy-verify-signature.yaml
├── prometheus/
│   └── slo-rules.yaml
├── sre/
│   ├── error-budget-calculation.md
│   └── postmortem-pod-delete.md
├── litmus/
│   └── chaos-pod-delete.yaml
├── tekton/
│   ├── task-kaniko.yaml
│   ├── task-trivy.yaml
│   ├── task-cosign.yaml
│   ├── task-sbom.yaml
│   ├── pipeline-secure-build.yaml
│   └── pipelinerun-payment.yaml
└── bonus/ (optionnel)
```

📚 Ressources et Aide

Documentation officielle

- [Backstage](https://backstage.io/docs)
- [Kyverno](https://kyverno.io/docs)
- [Cosign](https://docs.sigstore.dev/cosign/overview)
- [Litmus Chaos](https://docs.litmuschaos.io)
- [Tekton](https://tekton.dev/docs)

Commandes utiles

```bash
# Débug Kubernetes
kubectl get events --sort-by='.lastTimestamp'
kubectl describe pod <pod-name>
kubectl logs <pod-name> -f

# Débug Kyverno
kubectl get clusterpolicy
kubectl describe clusterpolicy <policy-name>
kubectl get policyreport -A

# Débug Tekton
tkn pipelinerun list
```

tkn pipelinerun logs <run-name> -f
tkn task list
```

Troubleshooting
Problème : Backstage ne démarre pas
Solution : Vérifiez les logs PostgreSQL : `kubectl logs -n backstage <postgres-pod>`
Problème : Kyverno n'applique pas les policies
Solution : Vérifiez que le webhook est configuré : `kubectl get validatingwebhookconfigurations`
Problème : Cosign sign échoue
Solution : Authentifiez-vous à GitHub : `echo $GITHUB_TOKEN | docker login ghcr.io -u USERNAME --password-stdin`
Problème : Tekton pipeline bloqué
Solution : Vérifiez les PVC : `kubectl get pvc` et les ServiceAccounts

✅ Critères de réussite
- [ ] Backstage accessible et affiche les 3 services
- [ ] Au moins 3 ClusterPolicies Kyverno fonctionnelles
- [ ] Une image signée avec Cosign et vérifiable
- [ ] Un chaos experiment exécuté avec succès
- [ ] Un pipeline Tekton complet fonctionnel
- [ ] Un postmortem rédigé
- [ ] Tous les fichiers YAML sont valides (pas d'erreurs de syntaxe)

Conseils
1. Commencez simple : Testez chaque composant individuellement avant de les intégrer
2. Lisez les erreurs : Kubernetes est verbeux, les messages d'erreur sont précis
3. Utilisez les exemples : La documentation officielle contient des exemples fonctionnels
4. Travaillez en équipe : Répartissez les blocs entre membres du groupe
5. Documentez : Notez les commandes qui fonctionnent pour le rapport final
---
Bon courage ! 🚀
```