Lab 3 - Déploiement Stack Monitoring Grafana (Loki, Mimir, Tempo)

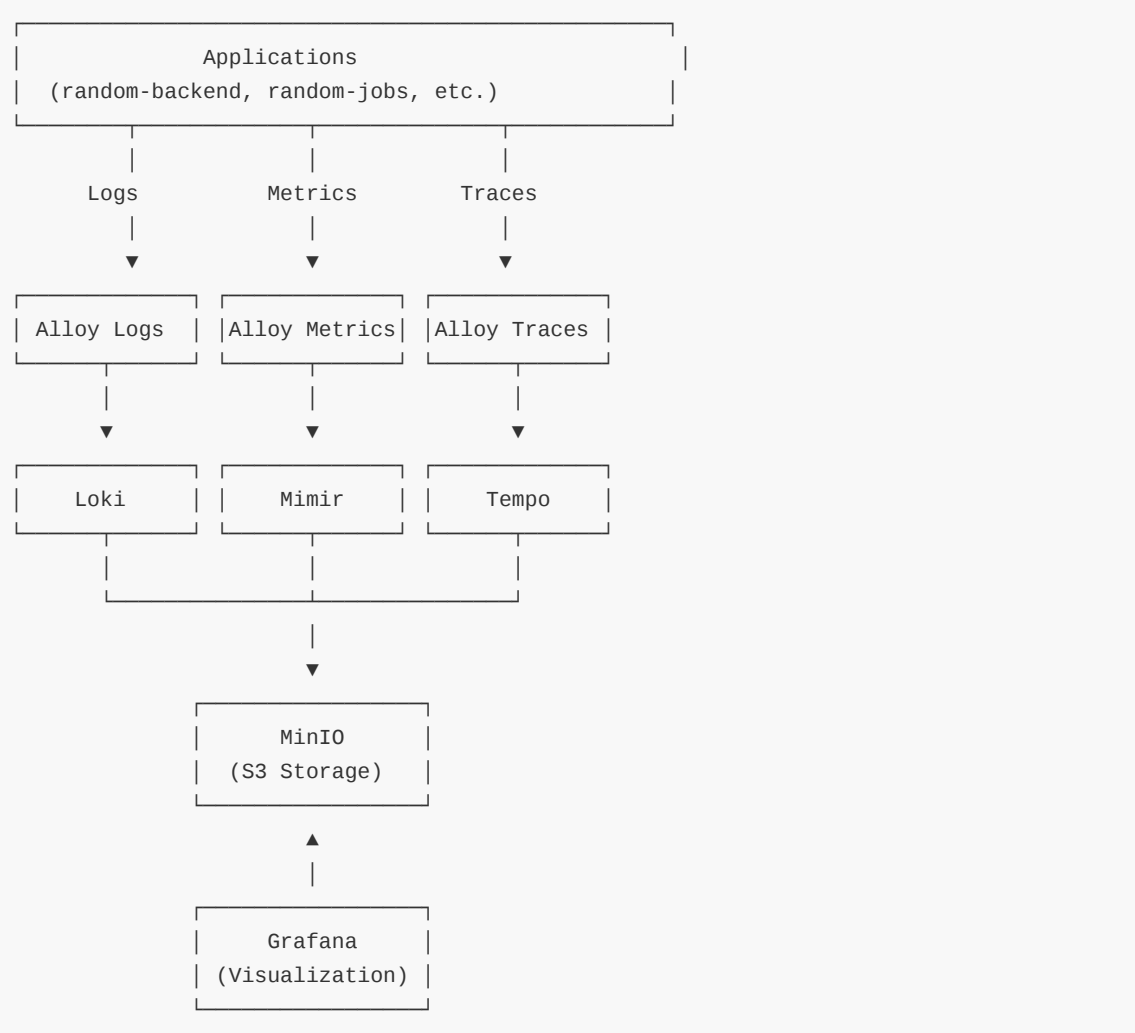
Objectif

Déployer une stack complète d'observabilité basée sur l'écosystème Grafana comprenant la collecte des logs (Loki), métriques (Mimir), et traces (Tempo), en suivant l'architecture du document de passation.

Contexte

Vous devez mettre en place l'infrastructure de monitoring décrite dans le document "Projet Monitoring des Cluster" pour assurer l'observabilité complète de la plateforme Random.

Architecture des Trois Piliers



Prérequis

- Cluster Kubernetes fonctionnel
- Helm 3 installé
- kubectl configuré
- 32Gi RAM minimum disponible sur le cluster
- 100Gi de stockage disponible
- Namespaces des Labs 1 et 2 déployés

Exercice 1 : Déploiement MinIO - Backend de Stockage (30 min)

MinIO servira de stockage objet (compatible S3) pour Loki, Mimir et Tempo.

Namespace : `minio`

Configuration requise :

- Mode : Standalone (ou Distributed selon ressources)
- Stockage : PVC de 50Gi (extensible)
- Exposition : Service ClusterIP sur port 9000 (API) et 9001 (Console)
- Credentials : À stocker dans un Secret
- Buckets à créer automatiquement :
  - `loki-data`
  - `mimir-blocks`
  - `mimir-ruler`
  - `mimir-alertmanager`
  - `tempo-data`

Sécurité :

- Activer le chiffrement des données au repos
- Configurer des policies IAM par bucket

Livrables :

- Fichier `minio-deployment.yaml` ou values Helm
- Script de création des buckets
- Vérification de l'accès à la console MinIO
- Documentation des credentials et endpoints

Exercice 2 : Déploiement Loki - Agrégation de Logs (40 min)

Namespace : `loki`

Architecture Loki :

- Distributor : 3 replicas (ingestion des logs)
- Ingester : 3 replicas (écriture dans le stockage)
- Querier : 2 replicas (lecture/requêtes)
- Query-frontend : 2 replicas (cache des requêtes)
- Compactor : 1 replica (compaction des données)

Configuration :

- Backend de stockage : MinIO (bucket loki-data)
- Schéma de rétention :

    - Logs applicatifs : 30 jours
    - Logs système : 15 jours
    - Logs debug : 7 jours
- Index : boltdb-shipper
- Compression : snappy
- Limites d'ingestion : 10MB/s par tenant

Livrables :

- Fichier `loki-values.yaml` pour Helm
- Commande de déploiement Helm
- Vérification des composants déployés
- Test d'ingestion d'un log simple
- Configuration de rétention par namespace

Exercice 3 : Déploiement Mimir - Métriques TSDB (45 min)

Namespace : `mimir`

Architecture Mimir :

- Distributor : 3 replicas
- Ingester : 3 replicas avec données persistées
- Querier : 2 replicas
- Query-frontend : 2 replicas
- Store-gateway : 2 replicas
- Compactor : 1 replica
- Ruler : 2 replicas
- Alertmanager : 3 replicas

Configuration :

- Backend de stockage : MinIO (buckets mimir-*)
- Multi-tenancy : Activé (OrgID: pods, nodes)
- Rétention : 365 jours
- High Availability : Activée
- Compaction : Toutes les 2 heures
- Limites par tenant :
    - Max series : 1 million
    - Max samples/s : 100k

Ressources :

- Ingesters : 4Gi RAM, 2 CPU par replica
- Queriers : 2Gi RAM, 1 CPU par replica

Livrables :

- Fichier `mimir-values.yaml` pour Helm
- Configuration multi-tenant documentée
- Test d'ingestion de métriques Prometheus
- Vérification du stockage dans MinIO

Exercice 4 : Déploiement Tempo - Traces Distribuées (40 min)

Namespace : `tempo`

Architecture Tempo :

- Distributor : 2 replicas (réception des traces)
- Ingester : 3 replicas (buffer et écriture)
- Querier : 2 replicas
- Query-frontend : 1 replica
- Compactor : 1 replica
- Metrics-generator : 1 replica

Configuration :

- Backend de stockage : MinIO (bucket tempo-data)
- Protocoles supportés :
  - OTLP (gRPC port 4317, HTTP port 4318)
  - Jaeger
  - Zipkin
- Rétention : 30 jours
- Génération de métriques depuis les traces : Activée
- Search : Activée

Livrables :

- Fichier `tempo-values.yaml` pour Helm
- Configuration des endpoints de réception
- Test d'envoi d'une trace OTLP
- Vérification de la recherche de traces

Exercice 5 : Déploiement Alloy Logs (35 min)

Namespace : `alloy-logs`

Configuration Alloy Logs :

- Mode : DaemonSet (un pod par node)
- Collecte : Logs de tous les pods Kubernetes
- Parser : docker/cri automatique
- Découverte : Kubernetes service discovery
- Labels ajoutés :
  - namespace
  - pod_name
  - container_name
  - node_name
  - job
- Destination : Loki
- Buffer : 1Gi en cas d'indisponibilité de Loki

Filtres à configurer :

- Exclure les logs de kube-system (sauf erreurs)
- Exclure les health checks
- Parser JSON automatique si détecté

Livrables :

- Fichier `alloy-logs-config.yaml`

- DaemonSet avec configuration montée en ConfigMap
- Vérification des logs dans Loki
- Test de recherche par label

Exercice 6 : Déploiement Alloy Metrics (40 min)

Namespace : `alloy-metrics`

Sources de métriques à scraper :

1. **kube-state-metrics** (métriques sur les objets K8s)
2. **node-exporter** (métriques système des nodes)
3. **cAdvisor** (métriques des conteneurs)
4. **ServiceMonitors** (applications custom)
5. **PodMonitors** (pods annotés)

Configuration Alloy Metrics :

- Découverte : operator.servicemonitors et operator.podmonitors
- Multi-tenancy : Envoyer avec OrgID approprié
  - OrgID "pods" pour métriques applicatives
  - OrgID "nodes" pour métriques infrastructure
- Destination : Mimir
- Scrape interval : 15s par défaut
- Remote write : Batch de 1000 samples

Déploiement kube-state-metrics et node-exporter :

- Via kube-prometheus-stack ou séparément
- Exposition en tant que ServiceMonitor

Livrables :

- Fichier `alloy-metrics-config.yaml`
- Déploiement kube-state-metrics
- Déploiement node-exporter en DaemonSet
- Vérification des métriques dans Mimir
- Test de requête PromQL

Exercice 7 : Déploiement Alloy Traces (35 min)

Namespace : `alloy-traces`

Configuration Alloy Traces :

- Réception : OTLP gRPC (4317) et HTTP (4318)
- Processors :
  - **k8sattributes** : Enrichir avec métadonnées K8s
  - **batch** : Grouper les traces pour efficacité
  - **servicegraph** : Générer des graphes de dépendances
- Destination :
  - Traces → Tempo
  - Métriques dérivées → Mimir
- Échantillonnage : Configurable par service

OpenTelemetry Operator :

- Installer l'operator
- Créer OpenTelemetryCollector CR
- Instrumentation automatique pour applications

Livrables :

- Installation OpenTelemetry Operator
- Fichier `alloy-traces-config.yaml`
- Service exposant ports 4317 et 4318
- Test avec application Python FastAPI instrumentée
- Visualisation du service graph dans Grafana

Exercice 8 : Déploiement Grafana (30 min)

Namespace : `grafana`

Configuration Grafana :

- Datasources pré-configurées :
  - Loki (logs)
  - Mimir (métriques)
  - Tempo (traces)
- Liens entre datasources (trace → logs → metrics)
- Authentification : OAuth2 ou LDAP
- Provisioning automatique :
  - Dashboards
  - Datasources
  - Alerting

Dashboards à provisionner :

1. Overview cluster (CPU, RAM, disque)
2. Monitoring de la stack elle-même
3. PostgreSQL (du Lab 2)
4. Namespaces Random
5. Service dependencies (depuis Tempo)

Plugins à installer :

- JSON API
- Pie Chart
- Status Panel

Livrables :

- Fichier `grafana-values.yaml` pour Helm
- ConfigMaps pour datasources
- ConfigMaps pour dashboards
- Test de connexion à toutes les datasources
- Capture d'écran d'un dashboard fonctionnel

Exercice 9 : Configuration OpenTelemetry pour Applications (40 min)

Instrumenter automatiquement les applications Python FastAPI du projet Random.

Composants :

- OpenTelemetry Operator installé
- Instrumentation automatique pour Python
- Configuration dans namespace de l'application

Configuration :

- Endpoint : Service Alloy Traces
- Propagation : W3C Trace Context et Baggage
- Sampling : 100% en dev, 10% en prod
- Ressource attributes :
  - service.name
  - service.namespace
  - deployment.environment

Livrables :

- Fichier `instrumentation.yaml` pour namespace random-backend
- Annotation des Deployments pour auto-instrumentation
- Test d'une requête tracée de bout en bout
- Corrélation trace → logs → métriques dans Grafana

Exercice 10 : cAdvisor pour Métriques Conteneurs (25 min)

Namespace : `monitoring`

Déploiement :

- Type : DaemonSet (un pod par node)
- Image : gcr.io/cadvisor/cadvisor:latest
- Volumes montés :
  - /rootfs
  - /var/run
  - /sys
  - /var/lib/docker
- Exposition : NodePort 30090 ou Service + ServiceMonitor

Métriques exposées :

- Utilisation CPU/RAM par conteneur
- I/O disque et réseau
- Caractéristiques de performance

Livrables :

- Fichier `cadvisor-daemonset.yaml`
- ServiceMonitor pour scraping par Alloy
- Vérification des métriques dans Mimir
- Dashboard Grafana utilisant les métriques cAdvisor

Exercice 11 : Kube-Prometheus-Stack (30 min)

Déployer le kube-prometheus-stack pour compléter la surveillance infrastructure.

Composants :

- Prometheus Operator
- Prometheus (pour compatibilité legacy)
- Alertmanager
- Grafana (peut réutiliser celui du Lab)
- kube-state-metrics
- node-exporter

Configuration :

- Prometheus stocke 7 jours localement
- Prometheus remote-write vers Mimir (stockage long terme)
- Alertmanager intégré à la stack Grafana
- Dashboards K8s pré-configurés

Livrables :

- Fichier `kube-prometheus-values.yaml`
- Configuration remote-write vers Mimir
- Vérification des alertes K8s par défaut
- Test d'une alerte (ex: pod crashlooping)

Exercice 12 : Surveillance de la Stack Monitoring (30 min)

Mettre en place l'auto-surveillance : surveiller les composants de monitoring eux-mêmes.

ServiceMonitors à créer :

- Loki (distributor, ingester, querier)
- Mimir (tous les composants)
- Tempo (distributor, ingester, querier)
- MinIO
- Alloy (chaque instance)
- Grafana

Dashboards de surveillance :

1. Loki Operations (ingestion rate, query latency)
2. Mimir Operations (ingester health, compactor)
3. Tempo Operations (trace ingestion, search performance)
4. MinIO Storage (capacité, IOPS)
5. Alloy Health (scrape success, buffer usage)

Alertes sur la stack :

- Loki ingester unhealthy
- Mimir compaction failing
- Tempo distributor overloaded
- MinIO storage > 80%
- Alloy remote write errors

Livrables :

- ServiceMonitors pour tous les composants
- Dashboard "Monitoring Stack Health"
- PrometheusRules pour alertes stack

- Test de détection d'un composant failing

Exercice 13 : Mise à l'Échelle (25 min)

Configurer l'auto-scaling pour les composants de la stack.

HorizontalPodAutoscaler :

- Loki querier : Scale 2-5 basé sur CPU > 70%
- Mimir querier : Scale 2-6 basé sur CPU > 70%
- Tempo querier : Scale 2-4 basé sur CPU > 70%
- Alloy metrics : Scale basé sur nombre de targets

VerticalPodAutoscaler (optionnel) :

- Ingesters Loki/Mimir : Ajuster RAM automatiquement

PodDisruptionBudget :

- Minimum 2 pods disponibles pour composants critiques
- Protéger contre les evictions massives

Livrables :

- Fichiers HPA pour composants queryables
- PodDisruptionBudgets pour haute disponibilité
- Test de montée en charge
- Documentation des seuils de scaling

Exercice 14 : Configuration des Rétentions et Compaction (30 min)

Optimiser le stockage et les performances à long terme.

Loki :

- Rétention par stream/label
- Compaction logs anciens
- Suppression automatique après rétention

Mimir :

- Compaction blocks toutes les 2h
- Downsampling (5m après 7j, 1h après 30j)
- Suppression automatique après 365j

Tempo :

- Rétention 30 jours
- Compaction traces anciennes

MinIO :

- Lifecycle policies par bucket
- Transition vers storage class froid après X jours

Livrables :

- Configuration rétention dans values.yaml
- MinIO lifecycle policies

- Script de vérification de l'espace libéré
- Documentation des politiques de rétention

Exercice 15 : Troubleshooting et Runbook (35 min)

Créer un guide de dépannage pour les problèmes courants.

Sections du runbook :

1. **Vérification de l'état des composants**
    - Commandes kubectl pour chaque namespace
    - Vérification des logs
    - Healthchecks
2. **Problèmes d'ingestion**
    - Logs non visibles dans Loki
    - Métriques manquantes dans Mimir
    - Traces perdues dans Tempo
    - Diagnostics et solutions
3. **Problèmes de performance**
    - Requêtes lentes dans Grafana
    - Queriers surchargés
    - Ingesters à saturation
    - Solutions d'optimisation
4. **Problèmes de stockage**
    - MinIO indisponible
    - Buckets pleins
    - Compaction échouée
    - Procédures de récupération
5. **Problèmes réseau**
    - Communication inter-composants
    - NetworkPolicies bloquantes
    - DNS resolution issues

Livrables :

- Document Markdown complet du runbook
- Scripts de diagnostic automatisés
- Checklist de vérification rapide
- Procédures de rollback par composant

Validation Finale

Checklist Infrastructure :

- [ ] MinIO déployé et buckets créés
- [ ] Loki opérationnel (tous les composants)
- [ ] Mimir opérationnel (tous les composants)
- [ ] Tempo opérationnel (tous les composants)

- [ ] Alloy Logs collecte les logs K8s
- [ ] Alloy Metrics scrape kube-state-metrics et node-exporter
- [ ] Alloy Traces reçoit les traces OTLP
- [ ] Grafana déployé avec datasources configurées
- [ ] OpenTelemetry Operator installé
- [ ] cAdvisor exposant les métriques
- [ ] Kube-prometheus-stack déployé
- [ ] Auto-surveillance configurée
- [ ] Dashboards provisionnés
- [ ] Alertes configurées

Tests End-to-End :

1. Déployer une application de test dans random-backend
2. Générer des logs, métriques et traces
3. Rechercher les logs dans Loki via Grafana
4. Requêter les métriques dans Mimir via Grafana
5. Visualiser les traces dans Tempo via Grafana
6. Naviguer d'une trace vers les logs corrélés
7. Visualiser le service graph
8. Déclencher une alerte et vérifier la réception
9. Tester la rétention (supprimer des données anciennes)
10. Simuler une panne d'un composant et vérifier l'auto-healing

QCM - Évaluation des Connaissances

Question 1

Pourquoi utiliser MinIO comme backend pour Loki, Mimir et Tempo ?

- A) C'est gratuit
- B) Il offre un stockage objet compatible S3 hautement performant
- C) C'est obligatoire pour Grafana
- D) Il remplace Prometheus

Question 2

Quelle est la différence entre Loki distributor et Loki ingester ?

- A) Aucune différence
- B) Distributor reçoit les logs et les route, Ingester les écrit dans le stockage
- C) Distributor est pour les logs, Ingester pour les métriques
- D) Ingester est obsolète

Question 3

Pourquoi Mimir utilise-t-il plusieurs OrgID (pods, nodes) ?

- A) Pour des raisons de sécurité uniquement
- B) Pour implémenter la multi-tenancy et séparer les métriques
- C) C'est une erreur de configuration
- D) Pour économiser du stockage

Question 4

Quel protocole OpenTelemetry utilise le port 4317 ?

- A) HTTP
- B) gRPC
- C) WebSocket
- D) TCP brut

Question 5

Pourquoi déployer Alloy en trois instances séparées (logs, metrics, traces) ?

- A) Pour la redondance
- B) Pour isoler les responsabilités et optimiser chaque collecteur
- C) C'est une obligation Grafana
- D) Pour répartir la charge CPU

Question 6

Dans Tempo, que fait le metrics-generator ?

- A) Il génère des fausses métriques
- B) Il dérive des métriques (RED metrics, service graph) depuis les traces
- C) Il convertit les métriques en traces
- D) Il optimise les requêtes

Question 7

Pourquoi utiliser boltdb-shipper comme index pour Loki ?

- A) C'est le plus rapide
- B) Il permet le stockage distribué de l'index dans le stockage objet
- C) Il est gratuit
- D) C'est obligatoire avec MinIO

Question 8

Quelle est la fonction du processor k8sattributes dans Alloy Traces ?

- A) Filtrer les traces Kubernetes
- B) Enrichir les traces avec des métadonnées Kubernetes (namespace, pod, etc.)
- C) Compresser les traces
- D) Authentifier les traces

Question 9

Pourquoi le Mimir compactor doit-il tourner en un seul replica ?

- A) Pour économiser des ressources
- B) Pour éviter les conflits lors de la compaction des blocks
- C) C'est plus rapide
- D) C'est une limitation technique

Question 10

Que signifie "remote write" dans le contexte Prometheus/Mimir ?

- A) Écrire sur un disque distant

- B) Envoyer les métriques vers un système de stockage externe (Mimir)
- C) Lire depuis un serveur distant
- D) Synchroniser deux Prometheus

Question 11

Pourquoi utiliser un DaemonSet pour Alloy Logs ?

- A) Pour avoir un pod sur chaque node et collecter tous les logs
- B) Pour économiser des ressources
- C) C'est obligatoire pour Loki
- D) Pour la redondance

Question 12

Dans Grafana, comment corréler une trace avec les logs correspondants ?

- A) Manuellement en cherchant le timestamp
- B) Via le traceID présent dans les logs et les traces
- C) Ce n'est pas possible
- D) Via l'adresse IP

Question 13

Pourquoi le Loki querier nécessite-t-il un cache ?

- A) Pour accélérer les requêtes répétées sur les mêmes logs
- B) Pour économiser du stockage
- C) Pour la sécurité
- D) C'est obligatoire

Question 14

Que fait le servicegraph connector dans Alloy Traces ?

- A) Il dessine des graphiques
- B) Il génère un graphe des dépendances entre services depuis les traces
- C) Il optimise les traces
- D) Il filtre les services

Question 15

Pourquoi surveiller la stack de monitoring elle-même ?

- A) Pour faire joli dans Grafana
- B) Pour détecter les problèmes de la stack avant qu'ils n'impactent l'observabilité
- C) C'est obligatoire
- D) Pour avoir plus de métriques

Question 16

Dans MinIO, à quoi servent les lifecycle policies ?

- A) À gérer le cycle de vie du service
- B) À automatiser la suppression ou transition de données anciennes
- C) À gérer les utilisateurs
- D) À monitorer les performances

Question 17

Pourquoi le kube-state-metrics est-il important ?

- A) Il remplace Prometheus
- B) Il génère des métriques sur l'état des objets Kubernetes (pods, deployments, etc.)
- C) Il surveille les nodes
- D) Il collecte les logs

Question 18

Que signifie "WaitForFirstConsumer" dans un StorageClass utilisé par MinIO ?

- A) Attendre le premier utilisateur
- B) Créer le volume dans la même zone que le pod qui l'utilisera
- C) Retarder la création du volume
- D) Partager le volume entre consumers

Question 19

Pourquoi utiliser Tempo plutôt que Jaeger seul ?

- A) Tempo est plus moderne
- B) Tempo utilise le stockage objet et scale mieux pour de gros volumes
- C) Jaeger est obsolète
- D) Tempo est gratuit

Question 20

Dans le document de passation, quels sont les 3 piliers de l'observabilité mentionnés ?

- A) CPU, RAM, Disque
- B) Logs, Métriques, Traces
- C) Loki, Mimir, Tempo
- D) Alloy, Grafana, MinIO

Réponses

1. B | 2. B | 3. B | 4. B | 5. B | 6. B | 7. B | 8. B | 9. B | 10. B | 11. A | 12. B | 13. A | 14. B | 15. B | 16. B | 17. B | 18. B | 19. B | 20. B