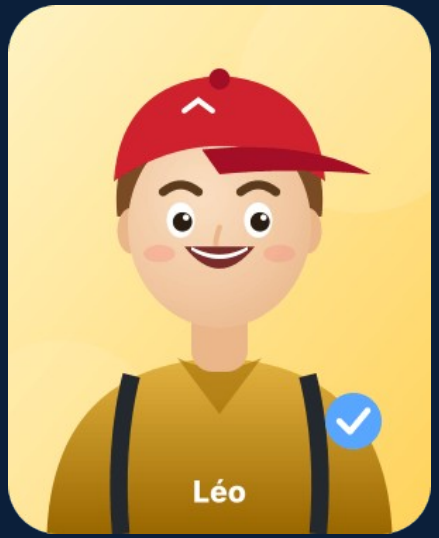
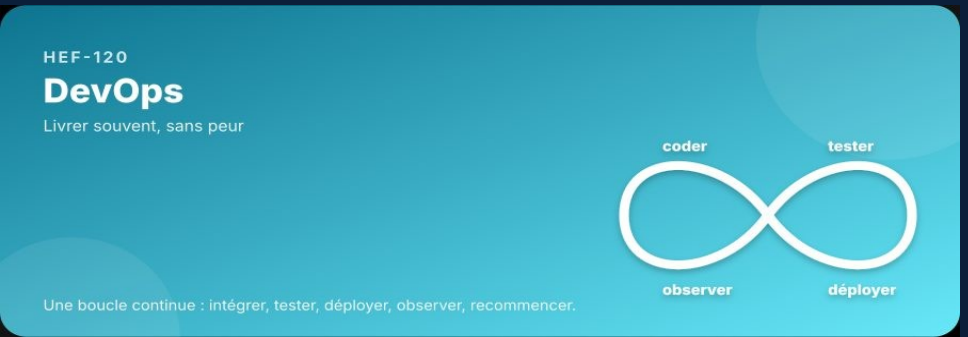
HEF-120 · MISSION 08 / 10

Observabilité



Un système en production est une boîte noire. Journaux, métriques et traces sont les fenêtres qui permettent de voir ce qui se passe à l'intérieur.

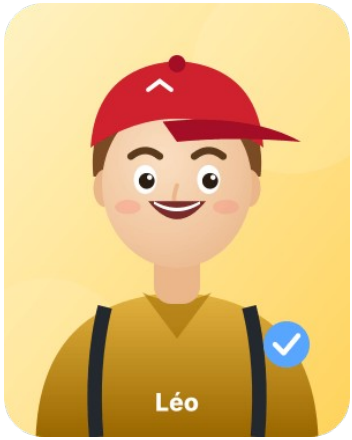


Badge : Sentinelle des signaux

105 min · intermédiaire

L'histoire

Mardi, les utilisateurs de HashBox se plaignent de lenteurs. L'équipe regarde les serveurs : le processeur est calme, la mémoire aussi. Quelques journaux contiennent « erreur » mais sans heure précise ni contexte. Après deux heures, un développeur remarque qu'un appel à un service tiers prend 8 secondes — depuis le déploiement de lundi soir.



> *« On a perdu deux heures parce qu'on ne voyait pas, dit Sam. Observer un système, c'est pouvoir poser des questions nouvelles à partir de ce qu'il émet, sans avoir à le modifier. »*

Tu apprends les trois piliers de l'observabilité (journaux, métriques, traces), les indicateurs d'objectif de service (SLI/SLO), le budget d'erreurs, et comment alerter sans noyer les gens.

Ta mission

À la fin, je serai capable de…

1. distinguer journaux, métriques et traces et leur usage
2. écrire des journaux structurés avec identifiant de corrélation
3. mesurer des latences par percentiles et des taux d'erreur
4. définir des SLI, SLO et un budget d'erreurs
5. concevoir des alertes qui signalent de vrais problèmes (taux de consommation du budget)

Défi d'échauffement

Défi d'échauffement

Tu conduis une voiture. Le tableau de bord affiche la vitesse, le niveau de carburant et les voyants. Si la voiture n'avait aucun affichage et qu'elle se mettait à vibrer, que ferais-tu ? Cite trois informations que tu voudrais voir. Lesquelles sont des mesures continues, lesquelles des événements isolés ?

Le concept

Surveillance et observabilité

- **Surveillance** (*monitoring*) : vérifier des choses **connues** (« le service répond-il ? »).
- **Observabilité** : pouvoir comprendre l'état interne d'un système à partir de ses **sorties**, y compris pour des problèmes **inattendus**.

Les trois piliers

Pilier	Quoi	Question typique	Coût
Journaux (*logs*)	événements discrets horodatés	que s'est-il passé pour *cette* requête ?	volumineux
Métriques	valeurs numériques agrégées dans le temps	quelle est la tendance ? quel taux d'erreurs ?	compactes

Le concept

Les trois piliers (suite)

Pilier	Quoi	Question typique	Coût
Traces	parcours d'une requête à travers les services (suite de *spans*)	où le temps est-il passé ?	échantillonnées

Des journaux utiles

- **Structurés** (JSON) : analysables par machine ;
- **niveaux** cohérents (DEBUG, INFO, WARNING, ERROR) ;
- **horodatage** UTC précis, **service**, **version** ;
- **identifiant de corrélation** (`request_id`) transmis entre services ;
- **pas de secrets** ni de données personnelles inutiles (HEF-100) ;
- un événement = un message clair avec ses champs.

Le concept

Les métriques : types

Type	Exemple
Compteur (*counter*)	nombre de requêtes, d'erreurs : ne fait que croître
Jauge (*gauge*)	mémoire utilisée, taille de la file : monte et descend
Histogramme	distribution des durées par tranches : permet les percentiles

Percentiles, pas moyennes : la moyenne cache la « queue » des requêtes lentes. p50 (médiane), p95, p99 disent ce que vivent la majorité et les moins chanceux.

Quelles métriques regarder ?

- **RED** (services) : *Rate* (débit), *Errors* (erreurs), *Duration* (durée).

Le concept

Quelles métriques regarder ? (suite)

- **USE** (ressources) : *Utilization* (utilisation), *Saturation* (file d'attente), *Errors*.
- Les « quatre signaux d'or » : latence, trafic, erreurs, saturation.

Traces

Une **trace** suit une requête : une suite de **spans** (opération, début, durée, span parent). Elle révèle quel service ou quelle requête de base de données consomme le temps.

```
GET /cours (320 ms)
├─ authentification (12 ms)
├─ lecture base (45 ms)
└─ appel service tiers (260 ms)   ← le coupable
```

Le concept

SLI, SLO, budget d'erreurs

Sigle	Sens	Exemple
SLI	indicateur mesuré du niveau de service	proportion de requêtes réussies en moins de 500 ms
SLO	objectif visé sur une période	99,9 % sur 30 jours
SLA	engagement contractuel (avec pénalités)	99,5 %
Budget d'erreurs	part d'échec acceptable : 100 % − SLO	0,1 % ≈ 43,2 min d'indisponibilité par mois

Le budget d'erreurs **arbitre** entre vitesse et fiabilité : tant qu'il reste du budget, on peut livrer des changements risqués ; quand il est épuisé, on se concentre sur la fiabilité.

Le concept

Alerter : le taux de consommation (*burn rate*)

Alerter sur chaque erreur épuise les gens. On alerte quand le budget est consommé **trop vite** :

```
burn rate = taux d'erreurs observé / taux d'erreurs toléré (1 − SLO)
```

Un burn rate de 1 épuise le budget exactement à la fin de la période ; 14,4 sur 1 heure consomme **2 % du budget mensuel en 1 heure** : alerte urgente. On utilise **deux fenêtres** (longue et courte) pour réagir vite et éviter les fausses alertes.

Principes d'une bonne alerte

- **Actionnable** : quelqu'un sait quoi faire ;
- **liée à l'impact utilisateur**, pas à un détail interne (processeur à 85 %) ;

Le concept

Principes d'une bonne alerte (suite)

- avec un **lien vers un guide d'intervention** (*runbook*) ;
- peu nombreuse ; chaque fausse alerte est un défaut à corriger.

🧠 ***Astuce*** *: avant de livrer une fonctionnalité, demande-toi « comment saurai-je en production qu'elle fonctionne ? ». Si la réponse est vague, ajoute la mesure.*

Exemple pas à pas

Problème : instrumenter un service simulé (journaux structurés avec corrélation, compteur, histogramme, trace), détecter un déploiement fautif par les percentiles, et calculer SLI, SLO et budget d'erreurs.

```
import contextvars, io, json, logging, math, random, time
from contextlib import contextmanager

# ---------------- journaux structurés avec identifiant de corrélation -----------
   -----
id_requete = contextvars.ContextVar("id_requete", default="-")

class FormatJSON(logging.Formatter):
    def format(self, enreg):
        return json.dumps({"niveau": enreg.levelname, "message": enreg.getMessage(
   ), "id_requete": id_requete.get(),
                           **getattr(enreg, "champs", {})}, ensure_ascii=False, so
   rt_keys=True)

flux = io.StringIO()
```

Exemple pas à pas (suite)

```
journal = logging.getLogger("hashbox"); journal.handlers.clear(); journal.setLevel
   (logging.INFO)
gestionnaire = logging.StreamHandler(flux); gestionnaire.setFormatter(FormatJSON()
   ); journal.addHandler(gestionnaire); journal.propagate = False

id_requete.set("req-42")
journal.info("inscription créée", extra={"champs": {"cours": "HEF-010", "duree_ms"
   : 37}})
ligne = json.loads(flux.getvalue().splitlines()[0])
print(ligne)
assert ligne["id_requete"] == "req-42" and ligne["cours"] == "HEF-010"

# ---------------- métriques : compteur et histogramme ----------------
class Histogramme:
    def __init__(self): self.valeurs = []
    def observer(self, v): self.valeurs.append(v)
    def percentile(self, p):
        v = sorted(self.valeurs)
        return v[min(len(v) - 1, math.ceil(p / 100 * len(v)) - 1)]
```

Exemple pas à pas

Exemple pas à pas (suite)

```
# ---------------- traces : spans imbriqués ----------------
class Trace:
    def __init__(self): self.spans, self._pile, self._horloge = [], [], 0.0
    @contextmanager
    def span(self, nom, duree_ms):
        parent = self._pile[-1] if self._pile else None
        self._pile.append(nom)
        debut = self._horloge
        try:
            yield
        finally:
            self._horloge = max(self._horloge, debut + duree_ms)
            self._pile.pop()
            self.spans.append({"nom": nom, "parent": parent, "duree_ms": duree_ms}
   )

# ---------------- simulation d'un service sur 10 000 requêtes, avec un déploiemen
   t fautif à la requête 6 000 ----------------
```

Exemple pas à pas (suite)

```
def simuler(n=10000, deploiement_a=6000, graine=8):
    rng = random.Random(graine)
    requetes = []
    for i in range(n):
        apres = i >= deploiement_a
        latence = rng.lognormvariate(math.log(80), 0.35)                    # ~
   80 ms typique
        if apres and rng.random() < 0.15:                                   # a
   près le déploiement : 15 % des requêtes appellent le service lent
            latence += 800
        erreur = rng.random() < (0.02 if apres else 0.0005)
        requetes.append({"i": i, "latence": latence, "erreur": erreur})
    return requetes

reqs = simuler()
avant, apres = [r for r in reqs if r["i"] < 6000], [r for r in reqs if r["i"] >= 6
   000]
for nom, groupe in (("avant", avant), ("après", apres)):
    h = Histogramme()
```

Exemple pas à pas (suite)

```
    for r in groupe: h.observer(r["latence"])
    moyenne = sum(h.valeurs) / len(h.valeurs)
    taux = sum(r["erreur"] for r in groupe) / len(groupe)
    print(f"{nom:6s} moyenne {moyenne:6.0f} ms | p50 {h.percentile(50):5.0f} | p95
    {h.percentile(95):6.0f} | p99 {h.percentile(99):6.0f} | erreurs {taux:6.2%}")
h_av, h_ap = Histogramme(), Histogramme()
for r in avant: h_av.observer(r["latence"])
for r in apres: h_ap.observer(r["latence"])
assert h_ap.percentile(99) > 5 * h_av.percentile(99)              # le p99 révèle
    le problème
assert h_ap.percentile(50) < 1.2 * h_av.percentile(50)            # la médiane n'
   a presque pas bougé : la moyenne/médiane aurait caché le problème

# ---------------- la trace localise le coupable ----------------
t = Trace()
with t.span("GET /cours", 360):
    with t.span("authentification", 12): pass
    with t.span("lecture base", 45): pass
    with t.span("appel service tiers", 300): pass
```

Exemple pas à pas (suite)

```
coupable = max((s for s in t.spans if s["parent"]), key=lambda s: s["duree_ms"])
print("span le plus coûteux :", coupable["nom"], coupable["duree_ms"], "ms")
assert coupable["nom"] == "appel service tiers"

# ---------------- SLI, SLO, budget d'erreurs ----------------
SLO = 0.999
def sli(groupe):                                                # réussite = pa
   s d'erreur ET moins de 500 ms
    return sum(1 for r in groupe if not r["erreur"] and r["latence"] < 500) / len(
   groupe)

total = sli(reqs)
budget_total = (1 - SLO) * len(reqs)                            # nombre d'éche
   cs tolérés
echecs = sum(1 for r in reqs if r["erreur"] or r["latence"] >= 500)
print(f"SLI = {total:.3%} (objectif {SLO:.1%}) | budget : {echecs} échecs sur {bud
   get_total:.0f} tolérés → budget dépassé {echecs / budget_total:.0f} fois")
assert total < SLO and echecs > budget_total
```

Exemple pas à pas (suite)

```
minutes_mois = 30 * 24 * 60
print(f"budget d'indisponibilité à 99,9 % sur 30 jours : {minutes_mois * (1 - SLO)
   :.1f} minutes")
assert round(minutes_mois * (1 - SLO), 1) == 43.2

# ---------------- burn rate ----------------
def burn_rate(groupe):
    echecs = sum(1 for r in groupe if r["erreur"] or r["latence"] >= 500)
    return (echecs / len(groupe)) / (1 - SLO)

print(f"burn rate avant déploiement : {burn_rate(avant):.1f} | après : {burn_rate(
   apres):.1f}")
assert burn_rate(apres) > 14.4 > burn_rate(avant)
```

Lecture :

1. Le **journal structuré** porte l'`id_requete` : toutes les lignes d'une même requête se retrouvent d'un coup d'œil, même entre services.

Exemple pas à pas (suite)

2. Après le déploiement, la **médiane** (p50) ne bouge presque pas, mais le **p99** est multiplié par plus de cinq : une moyenne ou une médiane aurait caché le problème.
3. La **trace** pointe immédiatement `appel service tiers` comme le span le plus long.
4. Le **SLI** (réussite en moins de 500 ms sans erreur) tombe sous le **SLO** de 99,9 % ; le **budget** est largement dépassé. Le **burn rate** après déploiement dépasse 14,4 : une alerte urgente aurait sonné en quelques minutes.

À toi, avec aide

À toi, avec aide

Un service a un SLO de **99,95 %** de disponibilité sur **30 jours**.

1. Calcule le budget d'indisponibilité en minutes.
2. Un incident de 18 minutes a eu lieu cette semaine, un autre de 9 minutes il y a 10 jours : quelle part du budget est consommée ? Combien reste-t-il ?
3. À ce rythme, sur 30 jours, le SLO sera-t-il tenu ? (Suppose que d'autres incidents suivent le même rythme : 27 minutes en 10 jours.)
4. Que recommandes-tu à l'équipe concernant les déploiements risqués ?

À toi, avec aide · réponse

Indices progressifs

Indice 1

Budget = 30 × 24 × 60 × (1 − 0,9995).

Indice 2

Part consommée = minutes d'incident / budget.

Indice 3

Projection : 27 minutes pour 10 jours → 81 minutes pour 30 jours.

Réponse

```
minutes = 30 * 24 * 60
budget = minutes * (1 - 0.9995)
consomme = 18 + 9
print(f"budget {budget:.1f} min, consommé {consomme} min ({consomme / budget:.0%})
   , reste {budget - consomme:.1f} min")
projection = consomme * 3
print(f"projection à 30 jours : {projection} min ({projection / budget:.0%} du bud
   get)")
assert round(budget, 1) == 21.6 and consomme > budget
```

Le budget est de **21,6 minutes**. Avec 27 minutes d'incident, il est **déjà dépassé** (125 %), et la projection à 30 jours (81 minutes) serait de 375 %. **Le SLO ne sera pas tenu.** Recommandation : **geler les déploiements risqués**, concentrer l'équipe sur la fiabilité (cause des incidents, tests, retour arrière automatique), et ne reprendre les nouvelles fonctionnalités qu'une fois la tendance rétablie. C'est le rôle du budget d'erreurs : transformer la fiabilité en décision objective.

À toi, en solo

Alerte à double fenêtre sur le taux de consommation. Implémente une alerte de type « burn rate » :

1. SLO 99,9 % ; flux de requêtes simulé minute par minute (échecs par minute et total par minute) ;
2. règle d'**alerte urgente** : burn rate > 14,4 sur **1 heure** ET > 14,4 sur **5 minutes** ;
3. règle d'**alerte lente** : burn rate > 1 sur **3 jours** (ticket, pas d'appel) ;
4. teste trois scénarios : (a) panne franche (taux d'erreur de 5 % pendant 20 min) → alerte urgente rapide ; (b) micro-pic d'une minute (30 % d'erreurs) → aucune alerte ; (c) dégradation lente (0,3 % d'erreurs en continu) → alerte lente seulement ;
5. mesure le délai de détection de (a).

À toi, en solo · réponse

Pistes de correction

```
SLO = 0.999
BUDGET = 1 - SLO

def burn(minutes, fin, fenetre):
    debut = max(0, fin - fenetre + 1)
    echecs = sum(m[0] for m in minutes[debut:fin + 1]); total = sum(m[1] for m in
   minutes[debut:fin + 1])
    return (echecs / total) / BUDGET if total else 0.0

def evaluer(minutes):
    alertes = []
    for t in range(len(minutes)):
        if burn(minutes, t, 60) > 14.4 and burn(minutes, t, 5) > 14.4 and not any(
   a[0] == "urgente" for a in alertes):
            alertes.append(("urgente", t))
        if t >= 3 * 24 * 60 - 1 and burn(minutes, t, 3 * 24 * 60) > 1 and not any(
   a[0] == "lente" for a in alertes):
            alertes.append(("lente", t))
    return alertes
```

À toi, en solo · réponse

Pistes de correction (suite)

```
TRAFIC = 1000                                                          # requêt
   es par minute
def flux(duree, taux, debut_panne=None, fin_panne=None, taux_panne=0.0, pics=()):
    minutes = []
    for t in range(duree):
        p = taux_panne if (debut_panne is not None and debut_panne <= t < fin_pann
   e) else taux
        if t in pics: p = 0.30
        minutes.append((round(TRAFIC * p), TRAFIC))
    return minutes

a = evaluer(flux(24 * 60, 0.0002, debut_panne=600, fin_panne=620, taux_panne=0.05)
   )
b = evaluer(flux(24 * 60, 0.0002, pics=(600,)))
c = evaluer(flux(4 * 24 * 60, 0.003))
print("a (panne franche) :", a, "| b (micro-pic) :", b, "| c (dégradation lente) :
   ", c)
assert [x[0] for x in a] == ["urgente"] and 600 <= a[0][1] <= 620           # dét
```

Pistes de correction (suite)

```
   ection en moins de 20 minutes
assert b == []
assert [x[0] for x in c] == ["lente"]
print(f"délai de détection de la panne franche : {a[0][1] - 600} minute(s)")
```

La **double fenêtre** évite la fausse alerte du micro-pic (la fenêtre d'une heure ne dépasse pas le seuil) tout en réagissant en moins de vingt minutes à une vraie panne (et d'autant plus vite que la panne est sévère). L'alerte lente repère les dégradations discrètes qui épuiseraient le budget sans jamais déclencher l'urgence.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Se fier aux moyennes	Elles cachent la queue de latence	Percentiles p95 / p99, histogrammes
Journaux non structurés, sans corrélation	Enquête interminable	JSON + identifiant de requête
Alerter sur tout	Fatigue d'alerte, vraies alertes ignorées	Alertes liées à l'impact utilisateur et au burn rate
Journaliser des secrets ou des données personnelles	Fuite via les journaux (HEF-100)	Masquer, minimiser

Mini-boss

Mini-boss

Observer un incident de bout en bout. Simule un service HashBox à trois composants (api, base, service de paiement) sur 24 heures, avec un déploiement fautif à 14 h (le service de paiement ralentit pour 10 % des requêtes). Construis :

1. les journaux structurés, les compteurs, les histogrammes et les traces de chaque requête ;
2. les SLI (disponibilité et latence), le SLO, le budget d'erreurs consommé ;
3. les deux alertes à double fenêtre et leur délai de détection ;
4. une analyse automatique : compare les percentiles avant/après chaque déploiement, identifie par les traces le composant responsable ;
5. un rapport d'incident d'une page : chronologie, impact (requêtes affectées), cause, détection, correction, actions ;
6. 10 tests, dont l'absence de fausse alerte sans incident.

Mini-boss · réponse

Pistes de correction

Un bon travail : générateur reproductible avec composants et durées par span; histogrammes par minute; SLI/SLO/budget; alerte double fenêtre; comparaison des p99 avant/après le déploiement; agrégation des spans par composant pour désigner `paiement`; rapport de type post-mortem sans blâme (chronologie avec heures, impact chiffré, cause racine, ce qui a bien marché — l'alerte, ce qui a manqué — un test de latence en pré-production, actions avec responsables).

Quiz éclair

Quiz éclair

1. Pour mesurer la latence ressentie par les moins chanceux, on regarde :

A. la moyenne uniquement
B. le nombre de serveurs
C. les percentiles élevés (p95, p99)

2. Un budget d'erreurs de 99,9 % sur 30 jours correspond à environ :

A. 43 minutes d'indisponibilité
B. 4 heures
C. 4 secondes

3. Un identifiant de corrélation sert à :

A. chiffrer les journaux

Quiz éclair

Quiz éclair (suite)

B. relier toutes les lignes d'une même requête entre services
C. accélérer la base

4. 🔁 *Révision mission 07* : Un déploiement canary consiste à :

A. remplacer tout d'un coup
B. déployer seulement la nuit
C. envoyer une petite part du trafic vers la nouvelle version, mesurer, puis étendre

5. 🔁 *Révision mission 07* : Un artefact promu de staging à la production doit être :

A. le même, construit une seule fois
B. reconstruit pour la production
C. modifié à la main

Quiz éclair · réponse

Réponses

1. **C**.
2. **A**.
3. **B**.
4. **C**.
5. **A**.

Je réfléchis

1. Si ton service ralentissait cette nuit, quelle première mesure regarderais-tu, et existe-t-elle ?
2. Quelle alerte de ton équipe a sonné ces derniers mois sans qu'on ait rien à faire ?

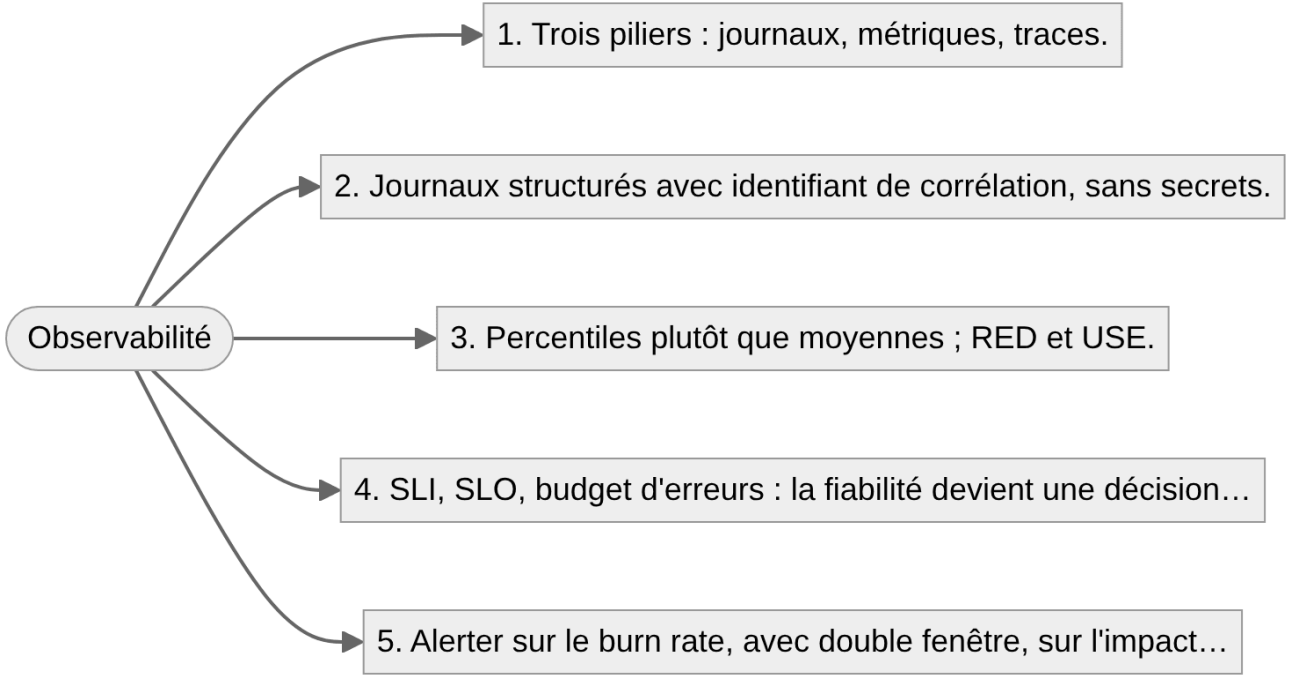
À retenir

À retenir

1. Trois piliers : journaux, métriques, traces.
2. Journaux structurés avec identifiant de corrélation, sans secrets.
3. Percentiles plutôt que moyennes ; RED et USE.
4. SLI, SLO, budget d'erreurs : la fiabilité devient une décision chiffrée.
5. Alerter sur le burn rate, avec double fenêtre, sur l'impact utilisateur.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Sentinelle des signaux

Prochaine mission : Rollback et résilience opérationnelle

