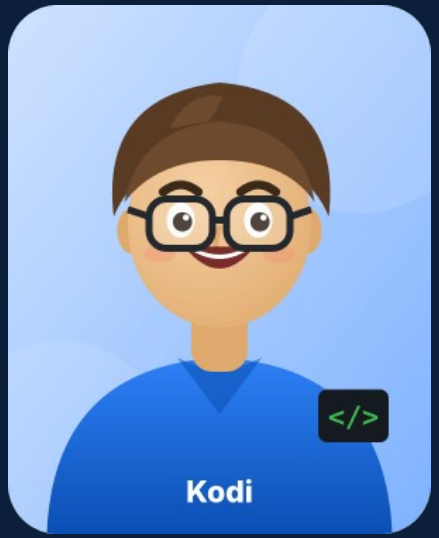
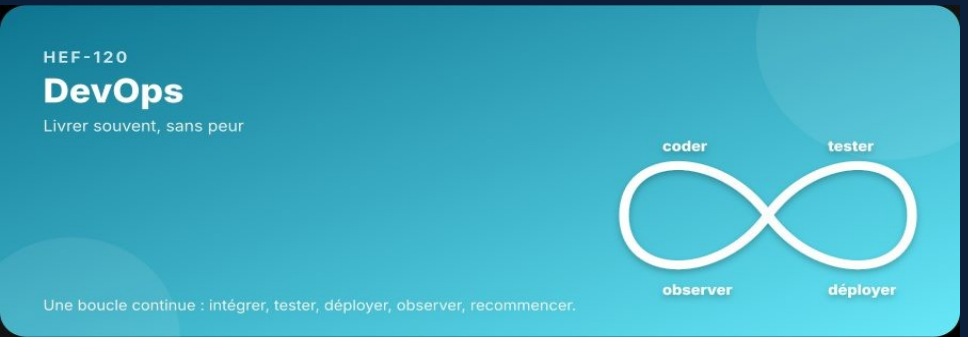
HEF-120 · MISSION 05 / 10

Configuration et secrets

Le même artefact doit tourner partout. Ce qui change d'un environnement à l'autre, c'est la configuration — et certains réglages sont des secrets.



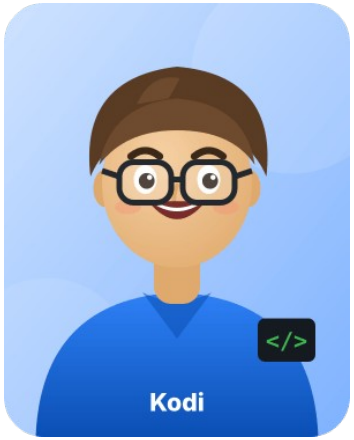


Badge : Gardien des réglages

95 min · intermédiaire

L'histoire

HashBox a trois environnements : développement, test, production. Dans le code, Kodi a écrit `if host == "prod-db": ...`. Un jour, quelqu'un copie la clé de l'API de paiement de production dans un fichier de configuration… et le pousse sur le dépôt. Un autre jour, l'application démarre en production avec l'adresse de la base de test : la configuration avait une faute de frappe, et personne ne l'a détectée avant les premiers clients.



> *« La configuration est du code qu'on ne compile pas, dit Sam. Elle mérite les mêmes soins : validation, versionnement, revue. Et les secrets, eux, ne vont jamais au même endroit. »*

Tu apprends les principes des applications « douze facteurs », à charger et valider une configuration par couches, à séparer les secrets, à détecter la dérive entre environnements et à faire tourner un secret sans interruption.

Ta mission

À la fin, je serai capable de…

1. séparer le code, la configuration et les secrets
2. charger la configuration par couches (défauts, fichier, environnement)
3. valider la configuration au démarrage avec des types et des plages
4. masquer les secrets dans les journaux et détecter la dérive entre environnements
5. faire la rotation d'un secret sans interruption de service

Défi d'échauffement

Une maison a un code d'alarme, un thermostat réglé à 19 °C et une clé de la porte d'entrée. Lesquels peut-on afficher dans le salon ? Lesquels doivent rester secrets ? Lesquels changent d'une saison à l'autre sans changer la maison ? Fais le parallèle avec : URL de la base, mot de passe de la base, niveau de journalisation.

Le concept

Code, configuration, secrets

	Définition	Exemples	Où
Code	logique de l'application, identique partout	fonctions, règles	dépôt, artefact
Configuration	réglages **qui varient selon l'environnement**	URL de service, taille de pool, niveau de journalisation	environnement, fichier versionné
Secret	configuration **sensible** : sa fuite cause un dommage	mot de passe, clé d'API, certificat	coffre-fort / gestionnaire de secrets

Test pratique : si tu pouvais publier ton dépôt demain en open source sans rien révéler de dangereux, tes secrets sont bien séparés.

Le concept

Les douze facteurs : configuration

Le principe III de l'application « douze facteurs » : **stocker la configuration dans l'environnement** (variables d'environnement), pas dans le code. Un seul artefact, plusieurs configurations → la mission 3 (« construire une fois »).

Charger par couches

Ordre de priorité habituel, du plus faible au plus fort :

```
1. valeurs par défaut (dans le code)
2. fichier de configuration (par environnement, versionné)
3. variables d'environnement
4. arguments de la ligne de commande
```

Le concept

Charger par couches (suite)

Chaque couche **remplace** la précédente. Les valeurs par défaut doivent être **sûres** (journalisation modérée, aucune fonction dangereuse activée).

Valider au démarrage : échouer vite

Une configuration fausse doit empêcher le **démarrage**, pas faire planter une requête dans la nuit :

- **types** (entier, booléen, URL) ;
- **plages** (port entre 1 et 65535) ;
- **obligatoire** vs optionnel ;
- **cohérences croisées** (`debug=True` interdit en production) ;
- message d'erreur **clair** qui nomme le réglage fautif (sans afficher un secret).

Le concept

Les secrets

Pratique	Détail
Jamais dans le dépôt	ni en clair, ni « temporairement », ni dans l'historique
Gestionnaire de secrets	coffre (Vault, gestionnaire du cloud) qui contrôle l'accès, journalise, fait tourner
Injection à l'exécution	variable d'environnement, fichier monté en mémoire, jamais dans l'image
Moindre privilège	un secret par service et par environnement, avec les droits strictement nécessaires
Masquage	jamais affichés dans les journaux ni les erreurs
Rotation	renouvelés régulièrement et après toute suspicion de fuite

Le concept

Les secrets (suite)

Pratique	Détail
Détection	analyse automatique des commits pour repérer les secrets (HEF-100, mission 5)

Dérive de configuration

La **dérive** (*drift*) : des environnements qui devraient être semblables se différencient peu à peu (modification manuelle, réglage oublié). Remède : tout décrire dans des fichiers **versionnés**, appliqués automatiquement, et **comparer** régulièrement les environnements : seules les différences **voulues** (adresses, tailles) doivent subsister.

Rotation sans interruption

Changer brutalement un secret coupe les clients qui utilisent l'ancien. La méthode **à chevauchement** :

Le concept

Rotation sans interruption (suite)

1. créer un **nouveau** secret, accepter **les deux** ;
2. mettre à jour les clients un par un ;
3. vérifier qu'**aucun** client n'utilise plus l'ancien ;
4. **révoquer** l'ancien.

Drapeaux de fonctionnalité

Les **drapeaux** (*feature flags*) activent ou non une fonctionnalité sans redéployer. Ils sont de la configuration **dynamique** : valeur modifiable à chaud, par environnement ou pourcentage d'utilisateurs (mission 7). Ils doivent être **retirés** une fois la fonctionnalité généralisée.

Le concept

Drapeaux de fonctionnalité (suite)

🧠 ***Astuce*** *: écris un test qui charge la configuration de* ***chaque*** *environnement et vérifie qu'elle est valide. Une erreur de configuration détectée en CI ne coûte rien ; détectée en production, elle coûte cher.*

Exemple pas à pas

Exemple pas à pas

Problème : écrire un chargeur de configuration par couches avec validation, masquage des secrets, comparaison d'environnements et rotation à chevauchement.

```
from dataclasses import dataclass, fields

# ---------------- 1. configuration typée et validée ----------------
@dataclass(frozen=True)
class Config:
    environnement: str = "dev"
    port: int = 8000
    debug: bool = False
    url_base: str = "sqlite:///local.db"
    niveau_journal: str = "INFO"
    mot_de_passe_base: str = ""                                  # SECRET : jamai
   s de valeur par défaut utile
    cle_api: str = ""                                            # SECRET

    SECRETS = ("mot_de_passe_base", "cle_api")
```

Exemple pas à pas

Exemple pas à pas (suite)

```
    def __repr__(self):                                          # JAMAIS de secr
   et dans les journaux
        morceaux = [f"{f.name}={'***' if f.name in self.SECRETS and getattr(self,
   f.name) else repr(getattr(self, f.name))}" for f in fields(self)]
        return f"Config({', '.join(morceaux)})"

class ConfigInvalide(Exception):
    pass

def convertir(valeur, type_):
    if type_ is bool:
        if str(valeur).lower() in ("1", "true", "oui", "vrai"): return True
        if str(valeur).lower() in ("0", "false", "non", "faux"): return False
        raise ValueError(f"booléen invalide : {valeur!r}")
    return type_(valeur)

def charger(fichier=None, env=None, cli=None):
    couches = {}
```

Exemple pas à pas

Exemple pas à pas (suite)

```
    for source in (fichier or {}, {k[8:].lower(): v for k, v in (env or {}).items(
   ) if k.startswith("HASHBOX_")}, cli or {}):
        couches.update(source)                                  # chaque couche
   remplace la précédente
    erreurs, valeurs = [], {}
    noms = {f.name: f for f in fields(Config)}
    for cle in couches:
        if cle not in noms:
            erreurs.append(f"réglage inconnu : {cle} (faute de frappe ?)")
    for nom, f in noms.items():
        if nom in couches:
            try:
                valeurs[nom] = convertir(couches[nom], f.type if isinstance(f.type
   , type) else eval(f.type))
            except (ValueError, TypeError) as e:
                erreurs.append(f"{nom} : {e}")
    if erreurs:
        raise ConfigInvalide("; ".join(erreurs))
    config = Config(**valeurs)
```

Exemple pas à pas (suite)

```
    # validations de plage et de cohérence
    if not 1 <= config.port <= 65535:
        erreurs.append(f"port hors plage : {config.port}")
    if config.niveau_journal not in ("DEBUG", "INFO", "WARNING", "ERROR"):
        erreurs.append(f"niveau de journal inconnu : {config.niveau_journal}")
    if config.environnement == "prod":
        if config.debug: erreurs.append("debug interdit en production")
        if not config.mot_de_passe_base: erreurs.append("mot de passe de base obli
   gatoire en production")
        if not config.cle_api: erreurs.append("clé d'API obligatoire en production
   ")
        if config.url_base.startswith("sqlite"): erreurs.append("base SQLite inter
   dite en production")
    if erreurs:
        raise ConfigInvalide("; ".join(erreurs))
    return config

FICHIER_PROD = {"environnement": "prod", "url_base": "postgres://db-prod:5432/hash
   box", "niveau_journal": "WARNING"}
```

Exemple pas à pas (suite)

```
ENV_PROD = {"HASHBOX_MOT_DE_PASSE_BASE": "s3cr3t-de-prod", "HASHBOX_CLE_API": "sk_
   live_abc", "HASHBOX_PORT": "8080", "PATH": "/usr/bin"}
c = charger(FICHIER_PROD, ENV_PROD)
print(c)
assert c.port == 8080 and c.mot_de_passe_base == "s3cr3t-de-prod"
assert "s3cr3t" not in repr(c) and "sk_live" not in repr(c) and "***" in repr(c)

for mauvaise, motif in (({"environnement": "prod", "debug": "true", "url_base": "s
   qlite:///x"}, "debug interdit"),
                        ({"port": "99999"}, "hors plage"), ({"prot": "8080"}, "rég
   lage inconnu"), ({"port": "abc"}, "port")):
    try:
        charger(mauvaise, {}); raise AssertionError(motif)
    except ConfigInvalide as e:
        assert motif in str(e), (motif, str(e))
print("5 configurations invalides refusées au démarrage, avec message clair")

# ---------------- 2. dérive entre environnements ----------------
def deriver(a, b, attendues=frozenset({"environnement", "url_base", "mot_de_passe_
```

Exemple pas à pas (suite)

```
   base", "cle_api"})):
    voulues, suspectes = {}, {}
    for f in fields(Config):
        va, vb = getattr(a, f.name), getattr(b, f.name)
        if va != vb:
            (voulues if f.name in attendues else suspectes)[f.name] = (va, vb)
    return voulues, suspectes

staging = charger({"environnement": "staging", "url_base": "postgres://db-stg:5432
   /hashbox", "niveau_journal": "WARNING"}, {"HASHBOX_PORT": "8081"})
voulues, suspectes = deriver(c, staging)
print("différences suspectes entre prod et staging :", suspectes)
assert "port" in suspectes and "niveau_journal" not in suspectes

# ---------------- 3. rotation à chevauchement ----------------
class VerificateurCle:
    def __init__(self, actuelle):
        self.cles = {actuelle}
    def accepter(self, cle): return cle in self.cles
```

Exemple pas à pas

Exemple pas à pas (suite)

```
    def ajouter_nouvelle(self, nouvelle): self.cles.add(nouvelle)
    def revoquer(self, ancienne): self.cles.discard(ancienne)

serveur = VerificateurCle("cle-v1")
clients = {"api": "cle-v1", "worker": "cle-v1"}
serveur.ajouter_nouvelle("cle-v2")                                  # étape 1 : les
   deux sont acceptées
clients["api"] = "cle-v2"                                           # étape 2 : migr
   ation progressive
assert all(serveur.accepter(k) for k in clients.values())           # personne n'est
    coupé pendant la migration
clients["worker"] = "cle-v2"
assert not any(k == "cle-v1" for k in clients.values())             # étape 3 : plus
    aucun client sur l'ancienne
serveur.revoquer("cle-v1")                                          # étape 4
assert all(serveur.accepter(k) for k in clients.values()) and not serveur.accepter
   ("cle-v1")
print("rotation sans interruption : OK")
```

Exemple pas à pas (suite)

Lecture :

1. **Couches** : défauts < fichier < environnement < CLI. Le fichier de prod ne contient **aucun secret** ; ils arrivent par l'environnement au démarrage.
2. **Validation** : types, plages, réglage inconnu (la faute de frappe `prot` est détectée), cohérences croisées (debug interdit en prod). Le message nomme le réglage fautif, jamais la valeur d'un secret.
3. **Masquage** : `repr(config)` affiche `***` à la place des secrets — un `print(config)` malencontreux ne les fuit pas.
4. **Dérive** : la comparaison sépare les différences **voulues** (adresse de base, secrets) des **suspectes** (le port diffère entre prod et staging).
5. **Rotation** : les deux clés sont valides pendant la migration, la révocation vient en dernier : zéro coupure.

À toi, avec aide

Écris `scanner_secrets(texte)` qui repère des secrets probables dans des lignes de configuration ou de code :

- affectations à des noms contenant `password`, `secret`, `token`, `key` avec une valeur de 8 caractères ou plus ;
- clés d'API au format connu : `sk_live_` suivi de 10 caractères alphanumériques ou plus, `AKIA` + 16 majuscules/chiffres ;
- ignore les valeurs qui sont des références (`${VAR}`, `os.environ[...]`, `getenv`).

Renvoie `(numéro de ligne, type)`. Teste avec 8 lignes.

À toi, avec aide · réponse

Indices progressifs

Indice 1

Utilise deux expressions régulières : une pour les affectations, une pour les formats connus.

Indice 2

Ignore la ligne si elle contient `${`, `environ` ou `getenv`.

Indice 3

Compte les lignes à partir de 1.

À toi, avec aide · réponse

Réponse

```
import re

AFFECTATION = re.compile(r"(?i)\b\w*(password|passwd|secret|token|key)\w*\s*[:=]\s
   *['\"]?([^'\"\s]{8,})")
FORMATS = [("clé API live", re.compile(r"sk_live_[A-Za-z0-9]{10,}")), ("clé AWS",
   re.compile(r"AKIA[0-9A-Z]{16}"))]

def scanner_secrets(texte):
    trouves = []
    for n, ligne in enumerate(texte.splitlines(), 1):
        if "${" in ligne or "environ" in ligne or "getenv" in ligne:
            continue
        for nom, motif in FORMATS:
            if motif.search(ligne):
                trouves.append((n, nom))
        if AFFECTATION.search(ligne) and not any(m.search(ligne) for _, m in FORMA
   TS):
            trouves.append((n, "secret en dur"))
    return trouves
```

À toi, avec aide · réponse

Réponse (suite)

```
TEXTE = '''port = 8080
password = "hunter2hunter2"
api_token: abcdef123456
cle = os.environ["CLE_API"]
password = ${DB_PASSWORD}
STRIPE = "sk_live_ABCDEF1234567890"
niveau = "debug"
aws = AKIAABCDEFGHIJKLMNOP'''
res = scanner_secrets(TEXTE)
print(res)
assert res == [(2, "secret en dur"), (3, "secret en dur"), (6, "clé API live"), (8
   , "clé AWS")]
```

Brancher un tel contrôle (avec un outil spécialisé) dans les crochets de pré-commit et la CI détecte les fuites **avant** qu'elles n'atteignent le dépôt partagé.

À toi, en solo

Gestionnaire de secrets minimal avec audit. Écris `Coffre` :

1. `ecrire(nom, valeur, par)`, `lire(nom, par)` avec une **politique d'accès** (qui peut lire quoi) ;
2. chaque accès (réussi ou refusé) est **journalisé** (qui, quoi, quand — horloge injectée) sans jamais enregistrer la valeur ;
3. **versions** : écrire un secret crée une nouvelle version ; la lecture renvoie la dernière ; `lire_version` permet la migration ;
4. **expiration** : un secret expiré (durée de vie fixée) est signalé ;
5. `a_renouveler(maintenant)` liste les secrets dont l'âge dépasse 80 % de leur durée de vie ;
6. teste : accès refusé, journal sans valeur, rotation avec chevauchement de versions.

À toi, en solo · réponse

Pistes de correction

```
from datetime import datetime, timedelta

class AccesRefuse(Exception):
    pass

class Coffre:
    def __init__(self, politique, horloge):
        self.politique, self.horloge = politique, horloge          # politique : {
   secret: {identités autorisées}}
        self.versions, self.journal = {}, []
    def _noter(self, qui, quoi, resultat):
        self.journal.append({"t": self.horloge(), "qui": qui, "quoi": quoi, "resul
   tat": resultat})     # PAS de valeur
    def ecrire(self, nom, valeur, par, duree=timedelta(days=90)):
        if par not in self.politique.get(nom, ()):
            self._noter(par, f"ecrire:{nom}", "refusé"); raise AccesRefuse(nom)
        self.versions.setdefault(nom, []).append({"valeur": valeur, "cree": self.h
   orloge(), "duree": duree})
        self._noter(par, f"ecrire:{nom}", f"version {len(self.versions[nom])}")
```

À toi, en solo · réponse

Pistes de correction (suite)

```
    def lire(self, nom, par, version=None):
        if par not in self.politique.get(nom, ()):
            self._noter(par, f"lire:{nom}", "refusé"); raise AccesRefuse(nom)
        v = self.versions[nom][-1 if version is None else version - 1]
        self._noter(par, f"lire:{nom}", "ok")
        return v["valeur"]
    def a_renouveler(self, maintenant):
        return sorted(n for n, vs in self.versions.items() if maintenant - vs[-1][
   "cree"] >= vs[-1]["duree"] * 0.8)

t = {"now": datetime(2025, 1, 1)}
coffre = Coffre({"cle_api": {"service-api", "admin"}}, lambda: t["now"])
coffre.ecrire("cle_api", "valeur-v1", "admin")
assert coffre.lire("cle_api", "service-api") == "valeur-v1"
try:
    coffre.lire("cle_api", "intrus"); raise AssertionError
except AccesRefuse:
    pass
coffre.ecrire("cle_api", "valeur-v2", "admin")                                  #
```

À toi, en solo · réponse

Pistes de correction (suite)

```
   rotation : nouvelle version
assert coffre.lire("cle_api", "service-api") == "valeur-v2" and coffre.lire("cle_a
   pi", "service-api", version=1) == "valeur-v1"
assert not any("valeur-v" in str(e) for e in coffre.journal)                    #
   aucun secret dans le journal
assert coffre.a_renouveler(datetime(2025, 2, 1)) == []
assert coffre.a_renouveler(datetime(2025, 3, 20)) == ["cle_api"]                #
   > 80 % de 90 jours
print("journal :", [(e['qui'], e['quoi'], e['resultat']) for e in coffre.journal][
   :4])
```

Un coffre-fort réel (Vault, services des fournisseurs de cloud) fait exactement cela : **contrôle d'accès**, **audit**, **versions**, **rotation**, et ne montre jamais les valeurs dans les journaux.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Secrets dans le dépôt	Fuite définitive (l'historique garde tout)	Coffre, injection à l'exécution, détection de secrets
`if environnement == 'prod'` dans le code	Chemins de code jamais testés avant la production	Même code partout, comportement piloté par la configuration
Configuration non validée	Erreur découverte en pleine charge	Validation complète au démarrage, échec immédiat
Rotation brutale	Coupure des clients	Chevauchement des versions

Mini-boss

Système de configuration de HashBox. Construis un module complet :

1. configuration typée avec couches (défauts, fichier JSON par environnement, variables d'environnement, CLI) ;
2. validations croisées (par environnement) et messages d'erreur sans secret ;
3. coffre simulé avec politique, audit, versions et rotation à chevauchement ;
4. scanner de secrets pour un répertoire de configuration, appelable en pré-commit ;
5. comparaison automatique de trois environnements avec séparation différences voulues / suspectes ;
6. test qui charge **chaque** fichier d'environnement du dépôt et vérifie qu'il est valide ;
7. simule un incident : un secret fuite dans un commit ; écris la procédure (révocation, rotation, recherche d'usage, nettoyage de l'historique, post-mortem).

Mini-boss · réponse

Pistes de correction

Un bon travail : classes `Config`, `Coffre`, fonctions `charger`, `deriver`, `scanner_secrets` ; un test paramétré (une exécution par fichier d'environnement) ; procédure d'incident : révoquer immédiatement (le secret est compromis même si le dépôt est privé), faire tourner, auditer les accès entre la fuite et la révocation, réécrire l'historique seulement en complément (jamais en remplacement de la révocation), ajouter la détection en pré-commit, écrire le post-mortem sans blâme.

Quiz éclair

Quiz éclair

1. Un secret est :

A. un nom de variable
B. un commentaire dans le code
C. une configuration sensible dont la fuite cause un dommage

2. La configuration d'une application douze facteurs est stockée :

A. dans l'environnement, hors du code
B. dans le code source
C. dans l'image du conteneur

3. Pour changer un secret sans interruption, on :

A. remplace brutalement le secret

Quiz éclair

Quiz éclair (suite)

B. accepte l'ancien et le nouveau pendant la migration, puis révoque l'ancien
C. attend la panne

4. 🔁 *Révision mission 04* : Un conteneur est :

A. une machine virtuelle complète
B. un fichier de configuration
C. un processus isolé par le noyau, instance d'une image

5. 🔁 *Révision mission 04* : Une bonne pratique de sécurité d'un Dockerfile est :

A. exécuter le processus avec un utilisateur non root
B. utiliser `latest`
C. copier les secrets dans l'image

Quiz éclair · réponse

Réponses

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

Je réfléchis

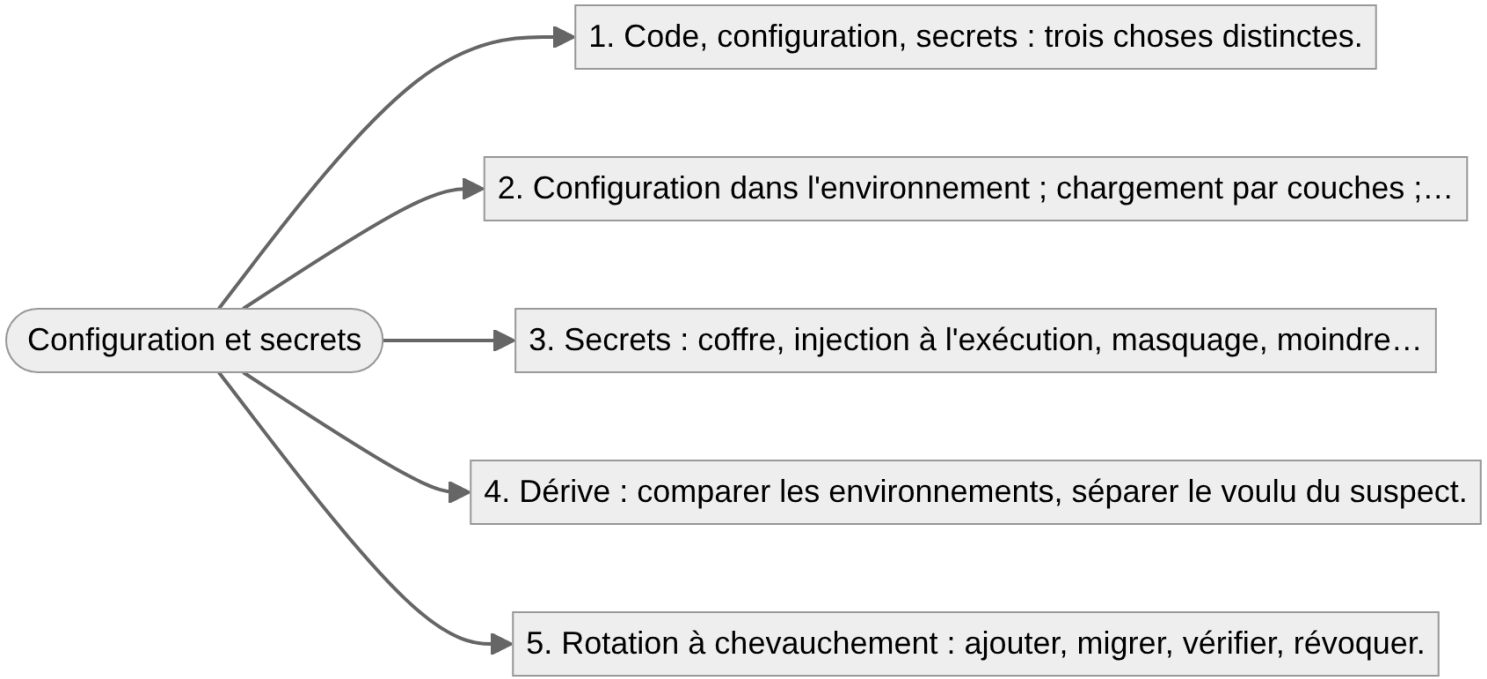
1. Quels secrets de ton projet pourraient être déjà dans ton historique Git ?
2. Combien de temps te faudrait-il pour faire tourner ta clé la plus critique sans interruption ?

À retenir

1. Code, configuration, secrets : trois choses distinctes.
2. Configuration dans l'environnement ; chargement par couches ; validation au démarrage.
3. Secrets : coffre, injection à l'exécution, masquage, moindre privilège.
4. Dérive : comparer les environnements, séparer le voulu du suspect.
5. Rotation à chevauchement : ajouter, migrer, vérifier, révoquer.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Gardien des réglages

Prochaine mission : Intégration continue (CI)

