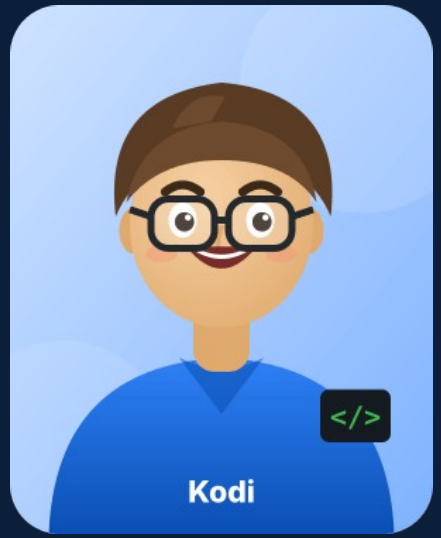
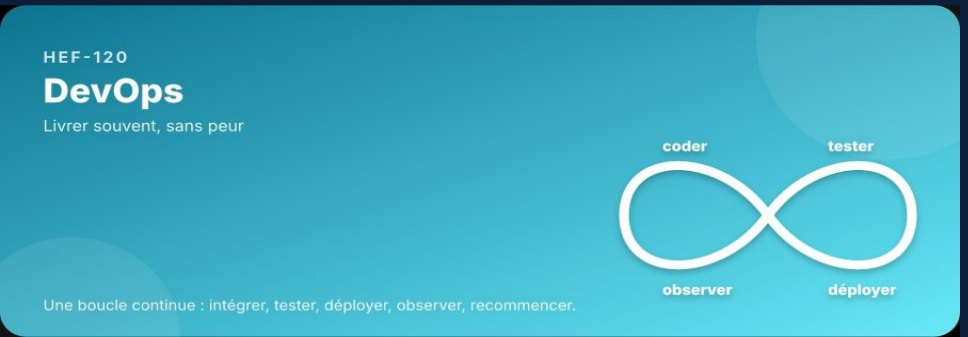
HEF-120 · MISSION 01 / 10

Culture et flux DevOps

DevOps n'est pas un outil ni un poste : c'est faire circuler le changement du code vers les utilisateurs, vite, sûrement, ensemble.



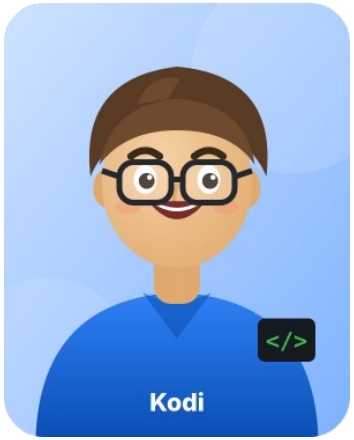


Badge : Passeur de pont

85 min · intermédiaire

L'histoire

À HashBox, l'équipe de développement termine une fonctionnalité en deux jours. Puis elle attend. Le ticket de déploiement passe chez les « ops », qui le planifient pour le jeudi suivant. Le déploiement échoue (« chez nous ça marchait ! »), on recommence la semaine d'après. Résultat : une fonctionnalité de deux jours met **trois semaines** à atteindre les utilisateurs.



> *« Les devs veulent livrer vite, les ops veulent que rien ne casse, dit Maya. Mis dos à dos, ils se bloquent. Mis dans la même équipe, avec les mêmes outils et les mêmes mesures, ils font les deux. »*

Tu apprends ce qu'est DevOps, comment mesurer la performance d'une chaîne de livraison avec les indicateurs DORA, et pourquoi les petits lots changent tout.

Ta mission

À la fin, je serai capable de…

1. expliquer la culture DevOps (CALMS) et les « trois voies »
2. calculer les quatre indicateurs DORA à partir d'un historique de déploiements
3. cartographier un flux de valeur (temps de travail contre temps d'attente)
4. comprendre pourquoi les petits lots réduisent délai et risque
5. identifier le goulot d'étranglement d'une chaîne de livraison

Défi d'échauffement

Un restaurant a deux équipes : la cuisine et la salle, qui ne se parlent jamais. La cuisine prépare des plats « parfaits » à son rythme ; la salle les sert froids et critique. Qui est responsable de la satisfaction du client ? Que changerais-tu pour que les deux équipes partagent le même objectif ?

Le concept

Le problème que DevOps résout

Historiquement, deux équipes aux **objectifs opposés** :

Développement	Exploitation (Ops)
mesuré sur le **changement** (nouvelles fonctionnalités)	mesuré sur la **stabilité** (pas de panne)
veut livrer souvent	veut limiter les changements

Le mur entre les deux produit des livraisons rares, grosses, risquées et stressantes.

DevOps = culture, pratiques et outils qui rassemblent développement et exploitation pour **livrer rapidement et fiablement**, avec une responsabilité partagée : *« you build it, you run it »*.

Le concept

CALMS : les cinq piliers

Lettre	Pilier	Idée
C	*Culture*	collaboration, confiance, responsabilité partagée, absence de blâme
A	*Automation*	automatiser build, tests, déploiement, infrastructure
L	*Lean*	petits lots, flux continu, éliminer le gaspillage
M	*Measurement*	décider avec des mesures, pas des opinions
S	*Sharing*	partager connaissances, outils, incidents

Les trois voies (The Phoenix Project)

1. **Le flux** : accélérer le passage du code vers la production (petits lots, automatisation).

Le concept

Les trois voies (The Phoenix Project) (suite)

2. **Le retour** : obtenir vite des retours (tests, monitoring, utilisateurs) pour corriger tôt.
3. **L'apprentissage continu** : expérimenter, tirer des leçons des incidents, partager.

Les quatre indicateurs DORA

Les recherches DORA (*DevOps Research and Assessment*) ont identifié quatre mesures qui distinguent les équipes performantes :

Indicateur	Question	Sens
Fréquence de déploiement	À quelle fréquence livre-t-on en production ?	vitesse
Délai de livraison des changements (*lead time*)	Combien de temps entre un commit et sa mise en production ?	vitesse
Taux d'échec des changements	Quelle part des déploiements provoque un incident ou un retour arrière ?	stabilité

Le concept

Les quatre indicateurs DORA (suite)

Indicateur	Question	Sens
Temps de rétablissement (*MTTR*)	En combien de temps rétablit-on le service après un échec ?	stabilité

Constat majeur : vitesse et stabilité **ne s'opposent pas** — les meilleures équipes sont les plus rapides **et** les plus fiables.

Le flux de valeur et le temps d'attente

Dans un flux de livraison, l'essentiel du délai est de l'**attente** (approbations, files d'attente, environnements indisponibles), pas du travail.

Le concept

Le flux de valeur et le temps d'attente (suite)

```
efficacité du flux = temps de travail actif / délai total
```

Une efficacité de 10 à 15 % est courante dans les organisations cloisonnées. Optimiser le travail actif (coder plus vite) apporte peu ; réduire l'**attente** apporte beaucoup.

Petits lots, grande différence

Loi de Little : pour un système stable, **délai moyen = travail en cours ÷ débit**. Pour le même débit, **moins de travail en cours** (lots plus petits) donne un délai plus court. De plus, un petit changement :

- est plus **facile à comprendre** et à relire ;
- cause des bugs plus **faciles à localiser** ;
- se **retire** (rollback) plus facilement ;
- expose moins de personnes et de temps à un risque.

Le concept

La théorie des contraintes

Une chaîne va à la vitesse de son **goulot d'étranglement**. Améliorer une étape qui n'est pas le goulot n'accélère pas le tout. On identifie le goulot (la plus longue attente, la file la plus longue), on le traite, puis on cherche le suivant.

Culture sans blâme

Après un incident, on demande « **comment** le système l'a permis ? » et non « **qui** a fait l'erreur ? ». Une équipe qui a peur d'être blâmée cache ses erreurs et déploie moins souvent, donc plus gros et plus risqué.

> 🧠 ***Astuce** : si déployer te fait peur, déploie **plus souvent**, pas moins. La peur vient de la taille et de la rareté, pas de l'acte.*

Exemple pas à pas

Problème : à partir de l'historique de déploiements de deux équipes, calculer les indicateurs DORA, comparer, puis simuler l'effet de la taille des lots sur le délai.

```
import statistics
from datetime import datetime, timedelta

def d(s):
    return datetime.fromisoformat(s)

# chaque déploiement : (date de mise en production, [dates des commits inclus], éc
   hec ?, date de rétablissement si échec)
EQUIPE_A = [  # grosses livraisons mensuelles
    (d("2025-01-31 18:00"), [d("2025-01-06 10:00"), d("2025-01-14 15:00"), d("2025
   -01-22 09:00")], True, d("2025-02-01 14:00")),
    (d("2025-02-28 18:00"), [d("2025-02-03 11:00"), d("2025-02-12 16:00"), d("2025
   -02-20 10:00")], True, d("2025-03-01 20:00")),
    (d("2025-03-31 18:00"), [d("2025-03-05 09:00"), d("2025-03-18 14:00"), d("2025
   -03-27 11:00")], False, None),
```

Exemple pas à pas (suite)

```
]
EQUIPE_B = [  # petites livraisons quotidiennes
    (d(f"2025-03-{jour:02d} 15:00"), [d(f"2025-03-{jour:02d} 09:30")], jour == 12,
    d("2025-03-12 15:40") if jour == 12 else None)
    for jour in range(3, 8 + 7)
]

def dora(deploiements, periode_jours):
    delais = [(dep - c).total_seconds() / 3600 for dep, commits, _, _ in deploieme
   nts for c in commits]
    echecs = [(dep, retab) for dep, _, echec, retab in deploiements if echec]
    return {
        "frequence_par_semaine": len(deploiements) / (periode_jours / 7),
        "delai_median_h": statistics.median(delais),
        "taux_echec": len(echecs) / len(deploiements),
        "mttr_h": statistics.mean((r - dep).total_seconds() / 3600 for dep, r in e
   checs) if echecs else 0.0,
    }
```

Exemple pas à pas (suite)

```
for nom, equipe, jours in (("A (mensuelle)", EQUIPE_A, 90), ("B (quotidienne)", EQ
   UIPE_B, 14)):
    m = dora(equipe, jours)
    print(f"équipe {nom:16s} fréquence {m['frequence_par_semaine']:5.2f}/sem | dél
   ai médian {m['delai_median_h']:6.1f} h | échecs {m['taux_echec']:4.0%} | MTTR {
   m['mttr_h']:5.1f} h")

a, b = dora(EQUIPE_A, 90), dora(EQUIPE_B, 14)
assert b["frequence_par_semaine"] > 10 * a["frequence_par_semaine"]
assert b["delai_median_h"] < a["delai_median_h"] / 50
assert b["mttr_h"] < a["mttr_h"] / 10

# ---- effet de la taille des lots (loi de Little) : débit identique, taille de lo
   t différente ----
def delai_moyen(taille_lot, changements_par_jour=4, duree_deploiement_jours=1):
    # on attend d'avoir `taille_lot` changements, puis on déploie en `duree_deploi
   ement_jours`
    attente = (taille_lot - 1) / 2 / changements_par_jour          # attente moyen
   ne dans le lot (en jours)
```

Exemple pas à pas (suite)

```
    return attente + duree_deploiement_jours

for lot in (1, 4, 20, 80):
    print(f"lot de {lot:2d} changements : délai moyen {delai_moyen(lot):5.2f} jour
   s")
assert delai_moyen(1) < delai_moyen(4) < delai_moyen(20) < delai_moyen(80)
```

Lecture :

- L'équipe B déploie **plus de vingt fois plus souvent** (6 fois par semaine contre 0,23), avec un délai médian de **5,5 heures** contre **16 jours**, un taux d'échec de 8 % contre 67 %, et un rétablissement en moins d'une heure contre près d'une journée. La rapidité n'a donc pas été payée par de l'instabilité : les petits changements sont plus faciles à comprendre, à tester et à annuler.
- La simulation de lot illustre la loi de Little : à débit constant, plus le lot est gros, plus **chaque changement attend**.

Exemple pas à pas (suite)

Note : sur un petit échantillon, les pourcentages bougent beaucoup ; en vrai on mesure sur des mois.

À toi, avec aide

Voici un flux de livraison, avec pour chaque étape le **temps de travail actif** et le **temps d'attente** (en heures) :

Étape	Travail	Attente avant
Revue de code	1	20
Tests manuels	6	48
Approbation du comité	0,5	72
Demande d'environnement	0,5	40
Déploiement	2	8

Calcule : le délai total, le temps de travail total, l'**efficacité du flux**, l'étape qui contribue le plus au délai, et propose ce que tu automatiserais en premier.

À toi, avec aide · réponse

Indices progressifs

Indice 1

Délai total = somme (travail + attente).

Indice 2

Efficacité = travail / délai.

Indice 3

Le plus gros contributeur est souvent une *attente*, pas un travail.

À toi, avec aide · réponse

Réponse

```
etapes = [("Revue de code", 1, 20), ("Tests manuels", 6, 48), ("Approbation du com
   ité", 0.5, 72), ("Demande d'environnement", 0.5, 40), ("Déploiement", 2, 8)]
travail = sum(t for _, t, _ in etapes)
total = sum(t + a for _, t, a in etapes)
print(f"travail {travail} h, délai {total} h, efficacité {travail / total:.1%}")
pire = max(etapes, key=lambda e: e[1] + e[2])
print("étape la plus longue :", pire[0], pire[1] + pire[2], "h")
assert total == 198 and travail == 10
assert pire[0] == "Approbation du comité"
```

Délai total : **198 h** (≈ 8 jours) pour **10 h** de travail actif : **5,1 %** d'efficacité. L'étape qui pèse le plus est l'**approbation** (72,5 h, presque toute en attente). À automatiser d'abord : les **environnements à la demande** (40 h d'attente) et les **tests** (48 h), puis remplacer le comité par des portes de qualité automatiques (mission 6).

À toi, en solo

Simulation de goulot d'étranglement. Modélise une chaîne à 4 étapes (développement, revue, tests, déploiement) où chaque étape a une capacité (changements par jour) : 6, 4, 2, 5. Simule 30 jours avec 5 nouveaux changements par jour :

1. suis la file d'attente devant chaque étape ;
2. mesure le débit réel de bout en bout et le délai moyen de chaque changement ;
3. identifie le goulot ;
4. double la capacité d'une étape **qui n'est pas** le goulot, puis celle du goulot : compare le délai moyen dans les deux cas ;
5. explique en 5 lignes pourquoi la première amélioration ne sert à rien.

À toi, en solo · réponse

Pistes de correction

```
def simuler(capacites, arrivees_par_jour=5, jours=30):
    files = [[] for _ in capacites]              # chaque élément = jour d'arrivé
   e du changement
    termines = []
    for jour in range(jours):
        files[0].extend([jour] * arrivees_par_jour)
        for i, cap in enumerate(capacites):
            traites, files[i] = files[i][:cap], files[i][cap:]
            if i + 1 < len(capacites):
                files[i + 1].extend(traites)
            else:
                termines.extend((jour - arrivee) for arrivee in traites)
    return len(termines) / jours, (sum(termines) / len(termines)) if termines else
    float("inf"), [len(f) for f in files]

base = (6, 4, 2, 5)
debit, delai, restes = simuler(base)
print(f"base       : débit {debit:.1f}/j, délai moyen {delai:.1f} j, en attente {r
   estes}")
```

Pistes de correction (suite)

```
hors_goulot = simuler((6, 8, 2, 5))
goulot = simuler((6, 4, 4, 5))
print(f"revue x2   : débit {hors_goulot[0]:.1f}/j, délai moyen {hors_goulot[1]:.1f
   } j")
print(f"tests x2   : débit {goulot[0]:.1f}/j, délai moyen {goulot[1]:.1f} j")
assert debit < 3 and hors_goulot[0] <= debit + 0.1
assert goulot[0] > debit and goulot[1] < delai
```

Doubler la capacité de la **revue** ne change presque rien : le débit reste plafonné à 2/jour par l'étape « tests ». Seule l'amélioration du **goulot** augmente le débit de bout en bout. Optimiser ailleurs ne fait que déplacer la file d'attente.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Réduire DevOps à des outils ou à un poste	Les silos restent, seul l'outil change	Responsabilité partagée, objectifs communs, mesures partagées
Mesurer la vitesse sans la stabilité	On livre vite des choses cassées	Les quatre indicateurs ensemble
Livrer rarement pour être plus sûr	Lots gros et risqués	Petits lots fréquents
Chercher un coupable après un incident	Les erreurs sont cachées	Analyse sans blâme centrée sur le système

Mini-boss

Diagnostic de livraison de HashBox. À partir d'un historique fictif de 40 déploiements sur 12 semaines (que tu génères avec une graine fixée : dates, commits inclus, échecs, rétablissements) et d'une description du flux (7 étapes avec travail et attente) :

1. calcule les quatre indicateurs DORA par mois et trace leur évolution en texte (tableau) ;
2. classe l'équipe (élite / haute / moyenne / basse) selon des seuils que tu justifies ;
3. fais la cartographie du flux de valeur : efficacité, goulot, 3 gaspillages ;
4. propose un plan de 4 changements, estime l'effet de chacun sur le délai avec ta simulation, et ordonne-les par rapport gain/effort ;
5. rédige 10 lignes pour convaincre un responsable prudent que livrer plus souvent *réduit* le risque.

Mini-boss · réponse

Pistes de correction

Un bon travail : un générateur de déploiements reproductible, des fonctions `dora()` par fenêtre mensuelle, une cartographie type (revue 20 h d'attente, comité 72 h, etc.), un plan priorisé (environnements à la demande, tests automatisés, remplacement des approbations manuelles par portes de qualité, déploiements quotidiens) avec l'effet simulé de chacun. L'argument final attendu : un petit changement est facile à comprendre, à localiser, à annuler ; la fréquence entraîne l'automatisation et la maîtrise ; les données DORA montrent la corrélation vitesse/stabilité.

Quiz éclair

Quiz éclair

1. DevOps cherche surtout à :

A. livrer vite et de façon fiable grâce à une responsabilité partagée
B. remplacer les développeurs par des administrateurs
C. supprimer les tests

2. Le délai de livraison (*lead time*) mesure :

A. le temps d'exécution du code
B. le temps entre un commit et sa mise en production
C. la durée d'une réunion

3. Pour réduire le délai moyen à débit constant, on :

A. augmente la taille des lots

Quiz éclair

Quiz éclair (suite)

B. retarde les déploiements
C. réduit le travail en cours et la taille des lots

4. 🔁 *Révision HEF-110, mission 10* : Un pipeline place en premier :

A. les étapes rapides et décisives
B. les étapes les plus longues
C. les tests E2E

5. 🔁 *Révision HEF-110, mission 09* : Un test instable est un test qui :

A. est très lent
B. passe ou échoue sur le même code sans modification
C. n'a pas d'assertion

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Combien de temps s'écoule entre « c'est prêt » et « c'est en production » dans ton équipe ? Où attend-on le plus ?
2. Quelle peur freine tes déploiements, et quelle pratique la réduirait ?

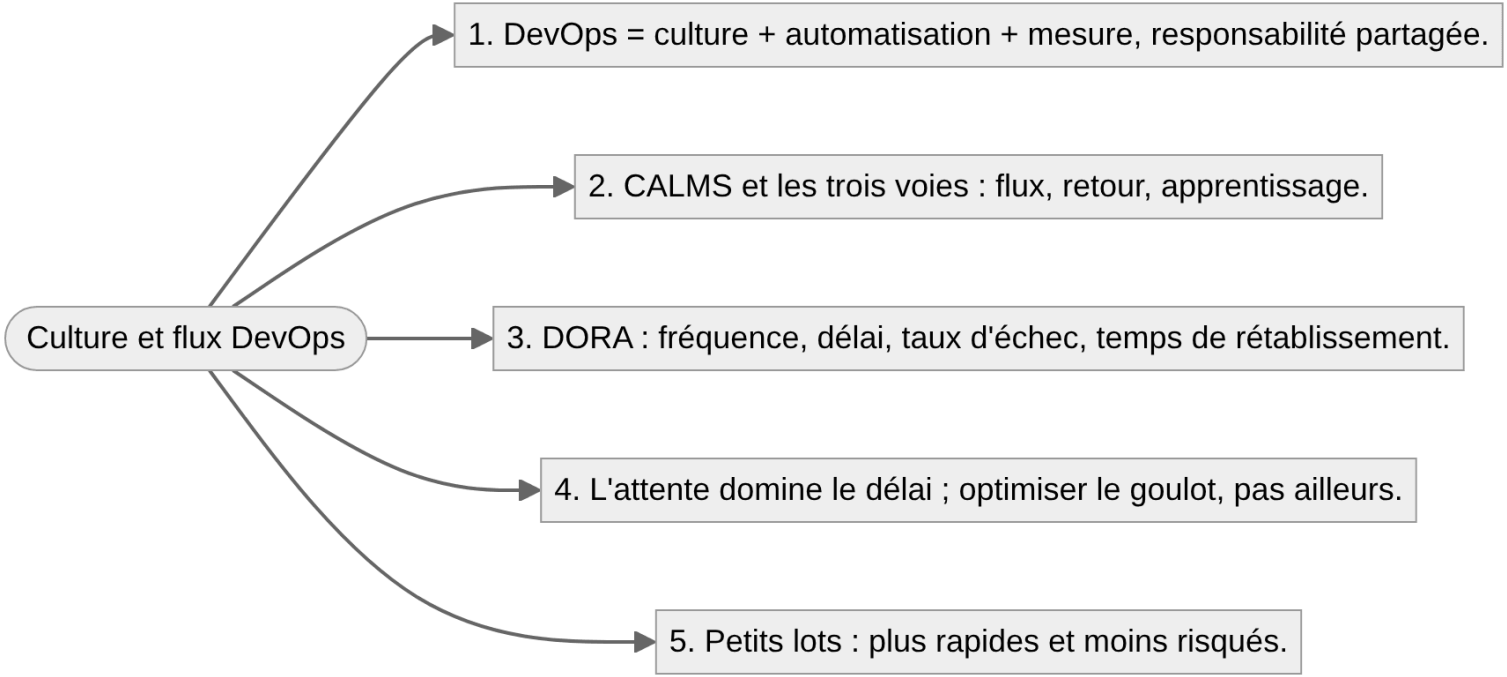
À retenir

À retenir

1. DevOps = culture + automatisation + mesure, responsabilité partagée.
2. CALMS et les trois voies : flux, retour, apprentissage.
3. DORA : fréquence, délai, taux d'échec, temps de rétablissement.
4. L'attente domine le délai ; optimiser le goulot, pas ailleurs.
5. Petits lots : plus rapides et moins risqués.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Passeur de pont

Prochaine mission : Git et stratégie de collaboration

