DevFest Toulouse

Votre architecture cloud est en réalité un organigramme

Horacio Gonzalez

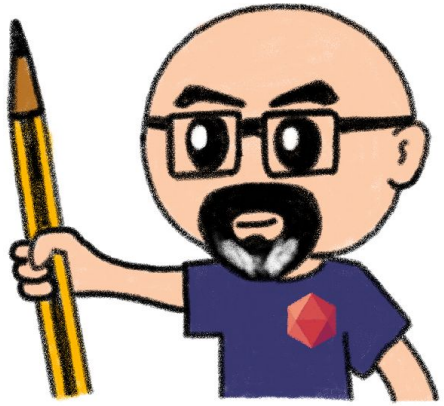
2026-11-19



clever cloud

Qui sommes-nous ?

Je me présente, et je présente Clever Cloud



clever cloud

DevFest Toulouse

@LostInBrittany



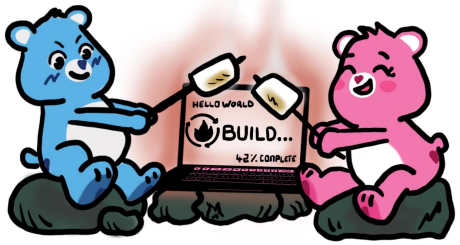
Horacio Gonzalez

@LostInBrittany

Un Espagnol perdu en Bretagne



Finist Devs



Head of DevRel

clever cloud

Clever Cloud

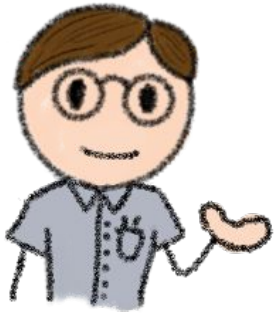
From Code to Product

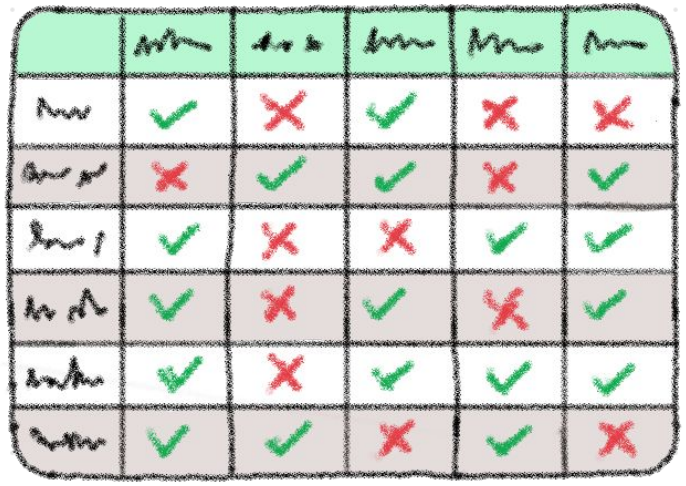




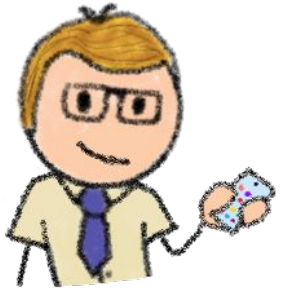
Le choix semblait rationnel

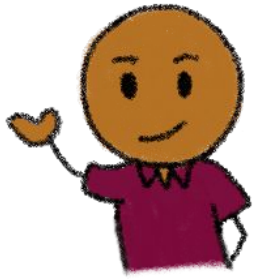
La meilleure architecture technique...









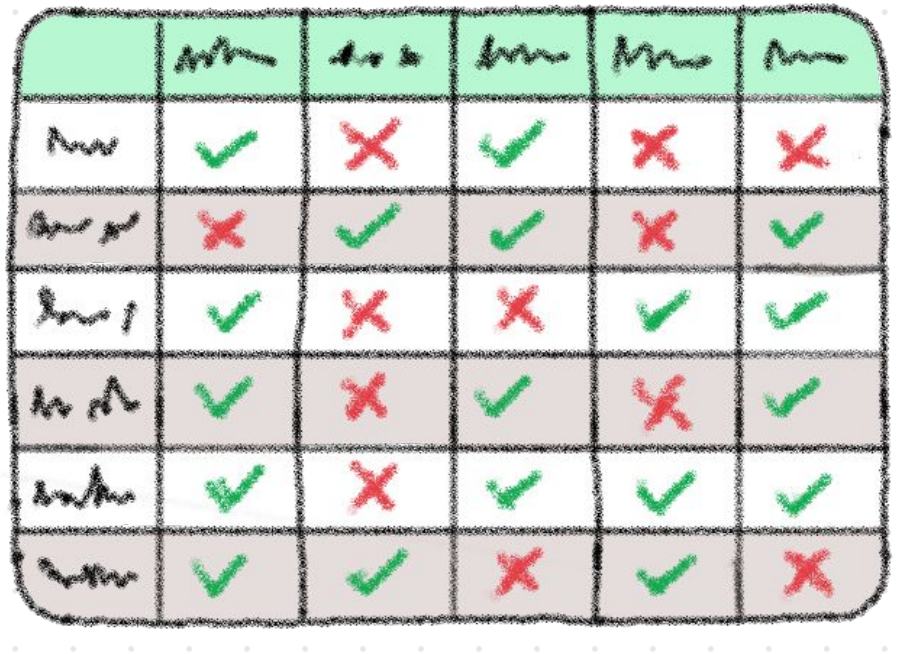




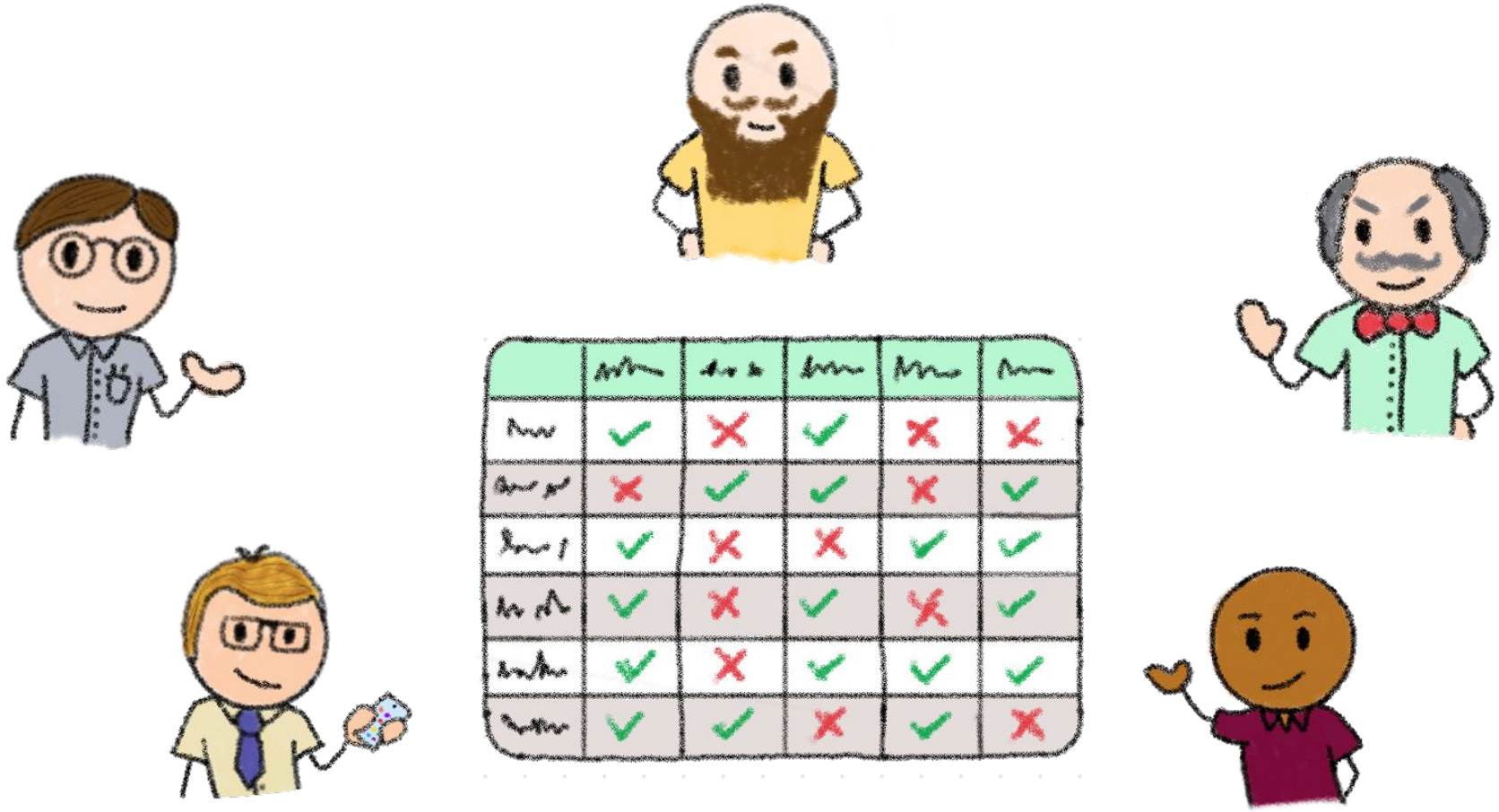
Comment choisit-on l'architecture technique ?

On compare les fonctionnalités et les contraintes

- Autoscaling, services managés, flexibilité de déploiement…
- Portabilité, coût, sécurité, performance…



Une comparaison technique sérieuse



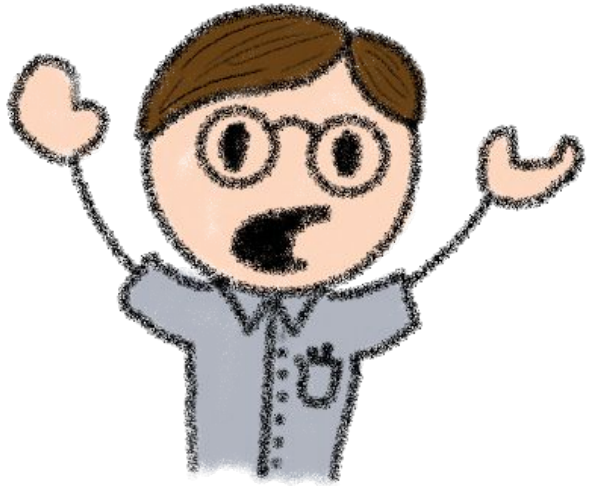
Le choix est rationnel, tout le monde est d'accord

Six mois plus tard, l'app est en production



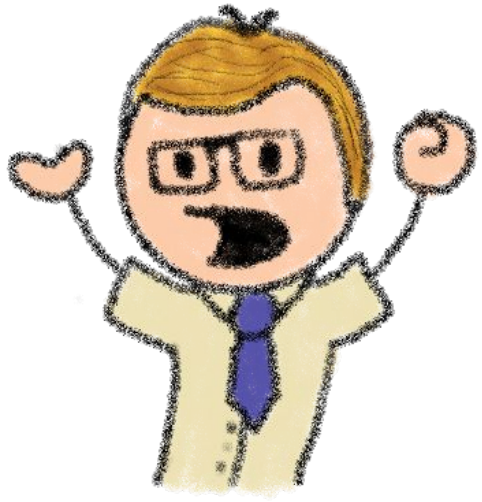
Tout va bien

Six mois plus tard, tout va bien ?



Les recrues ont besoin de semaines avant
de déployer en confiance

Six mois plus tard, tout va bien ?



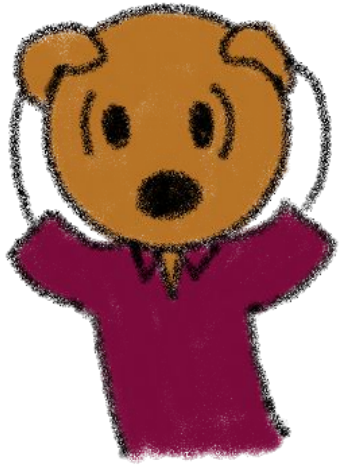
Incident : appli, plateforme, réseau ou base de données ?

Six mois plus tard, tout va bien ?



Une personne est devenue la documentation humaine

Six mois plus tard, tout va bien ?



Astreinte plus lourde que ce à quoi qui que ce soit s'était engagé

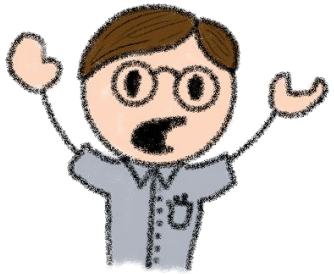
Six mois plus tard, tout va bien ?

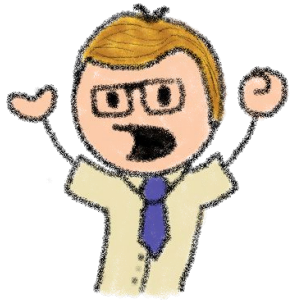


Plus de temps à comprendre les déploiements qu'à livrer

Ils viennent de découvrir la facture invisible

La facture cloud n'était pas la seule





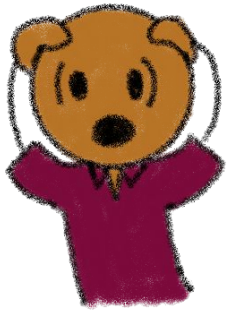
Il y en avait une **invisible** :

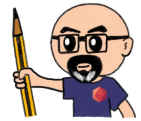
charge cognitive • responsabilité • recrutement

onboarding • astreinte









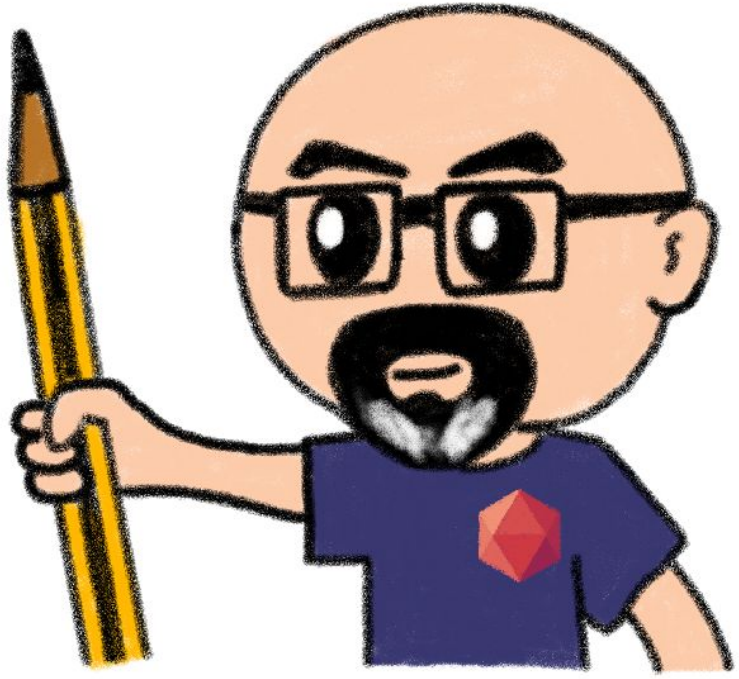
Votre architecture cloud est un organigramme



Chaque choix d'infrastructure est aussi un choix d'organisation

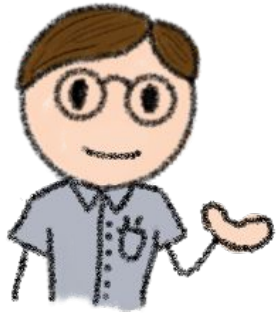
C'est quoi, ce talk ?

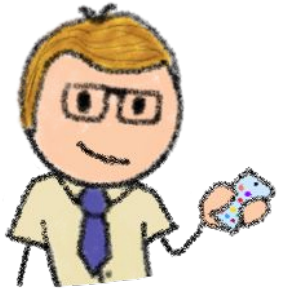
- Une appli, quatre modèles d'infrastructure
- Quatre organigrammes très différents
- Une matrice utilisable dès lundi
- Pas du Kubernetes-bashing,
  pas un pitch PaaS

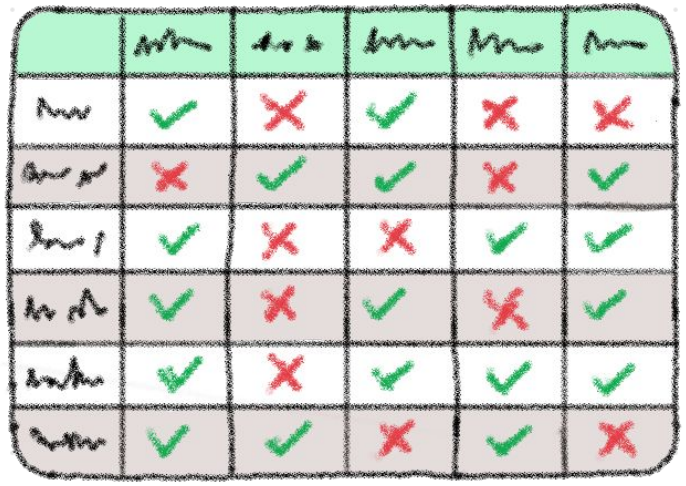


La matrice qu'on fait d'habitude

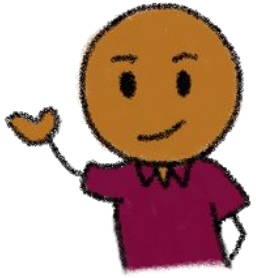
...et ce qui manque

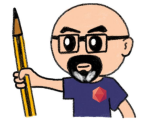






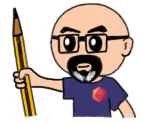






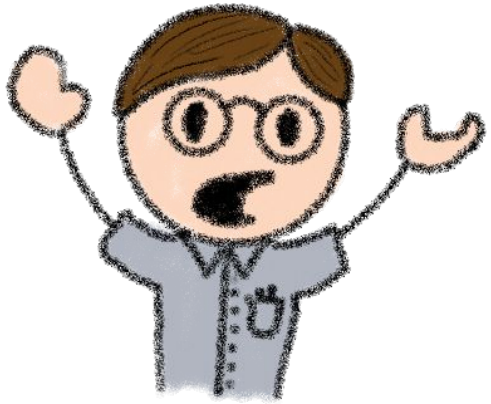
Ce qu'on compare d'habitude

Criterion	VMs	K8s	PaaS	Serverless
Autoscaling	🤨	✅	✅	✅
Pas d'hôte / OS à gérer	❌	🤨	✅	✅
Base de données managée	❌	❌	✅	✅
Flexibilité de déploiement	✅	✅	🤨	🤨
Portabilité	🤨	✅	🤨	❌
Observabilité intégrée	❌	🤨	✅	🤨
Écosystème	❌	✅	✅	✅
Scale à zéro / paiement à l'usage	❌	❌	🤨	❌
...	...	...	...	...



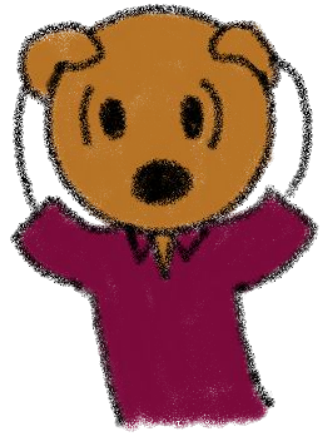
Incomplète

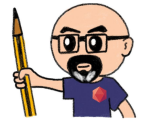
Cette matrice n'est pas fausse, juste incomplète



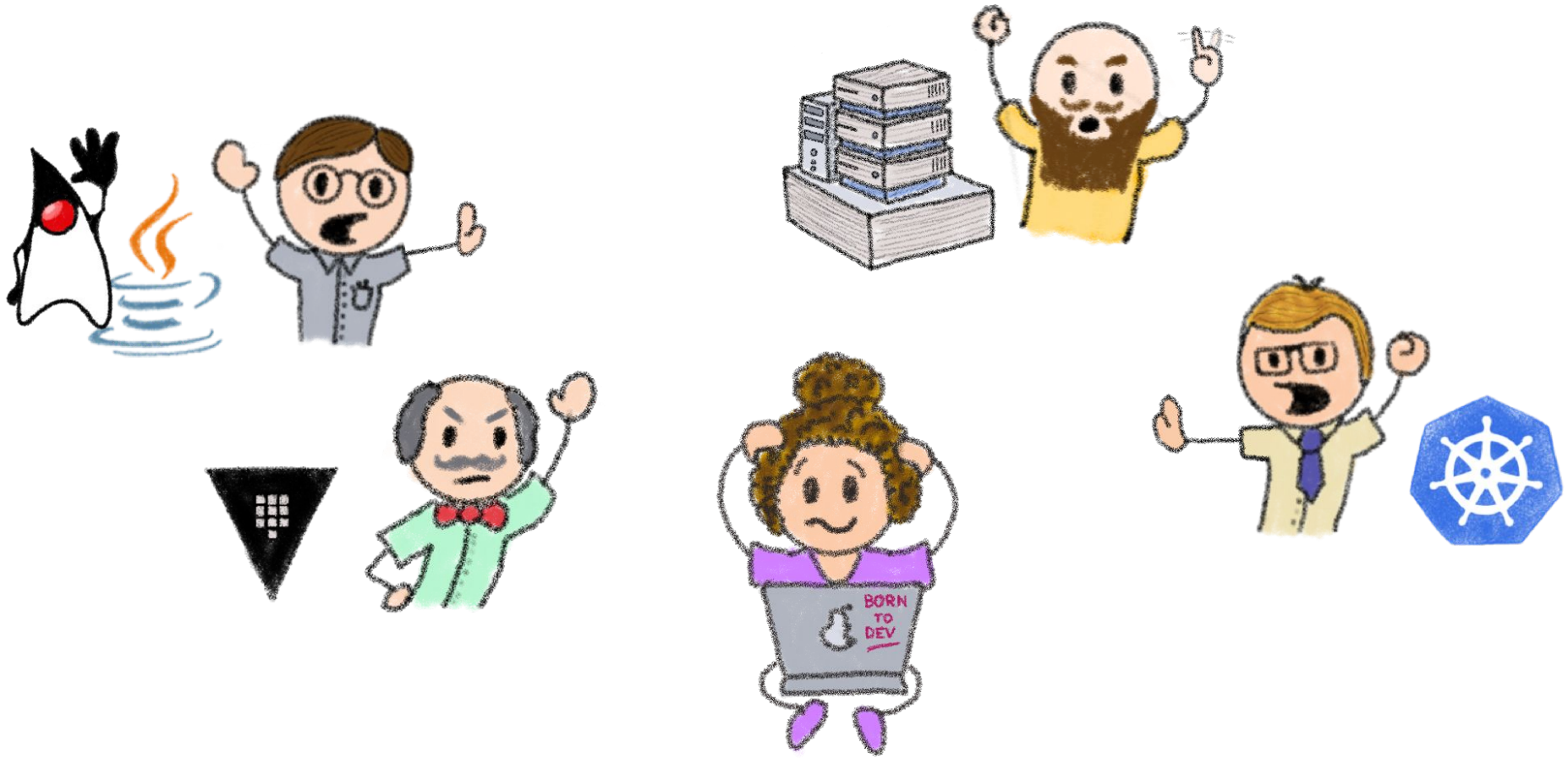
Elle vous dit ce que **la plateforme sait faire**...

pas qui est **réveillé à 3h du matin**





Ce qu'elle ne vous dit pas



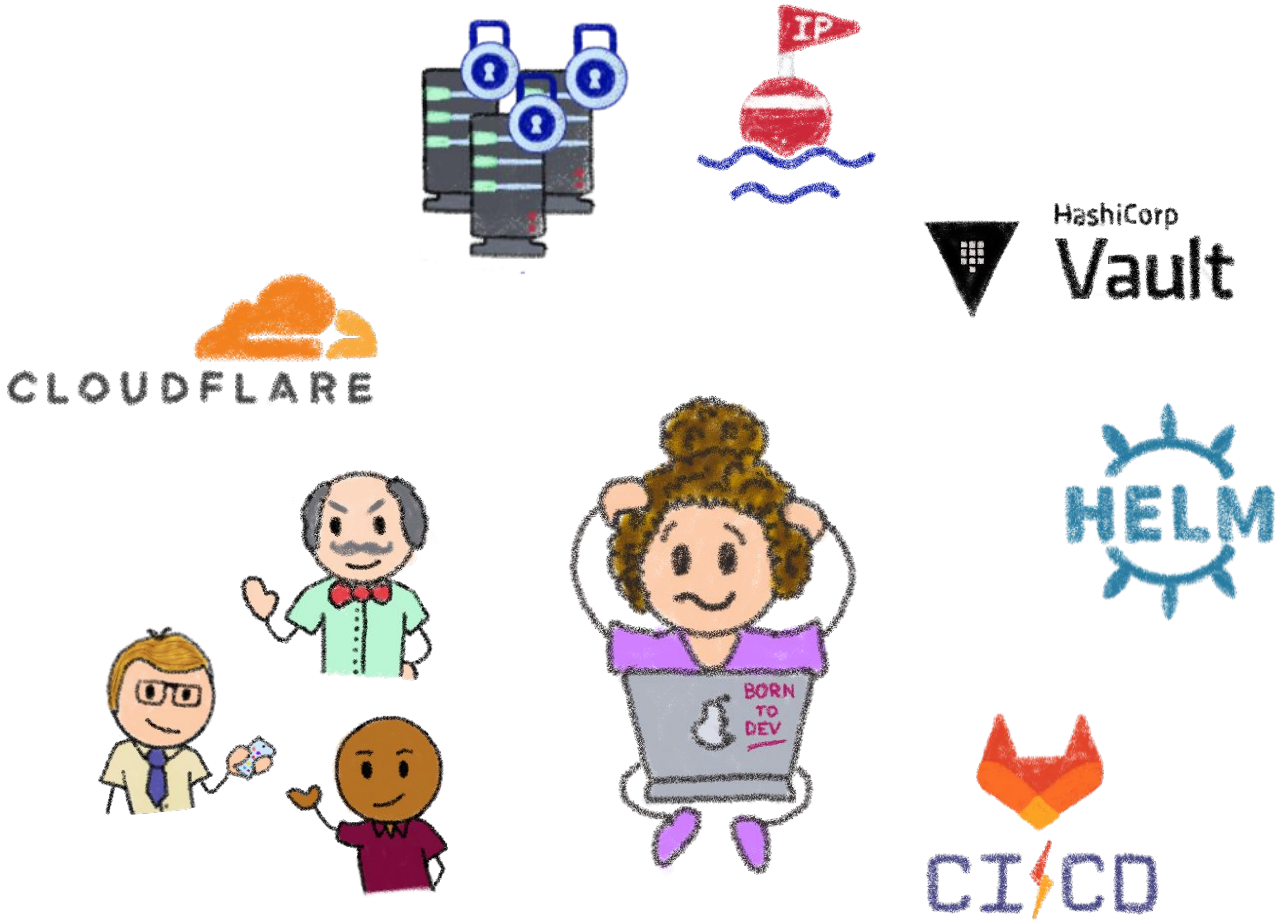
Qui possède la production

Ce qu'elle ne vous dit pas



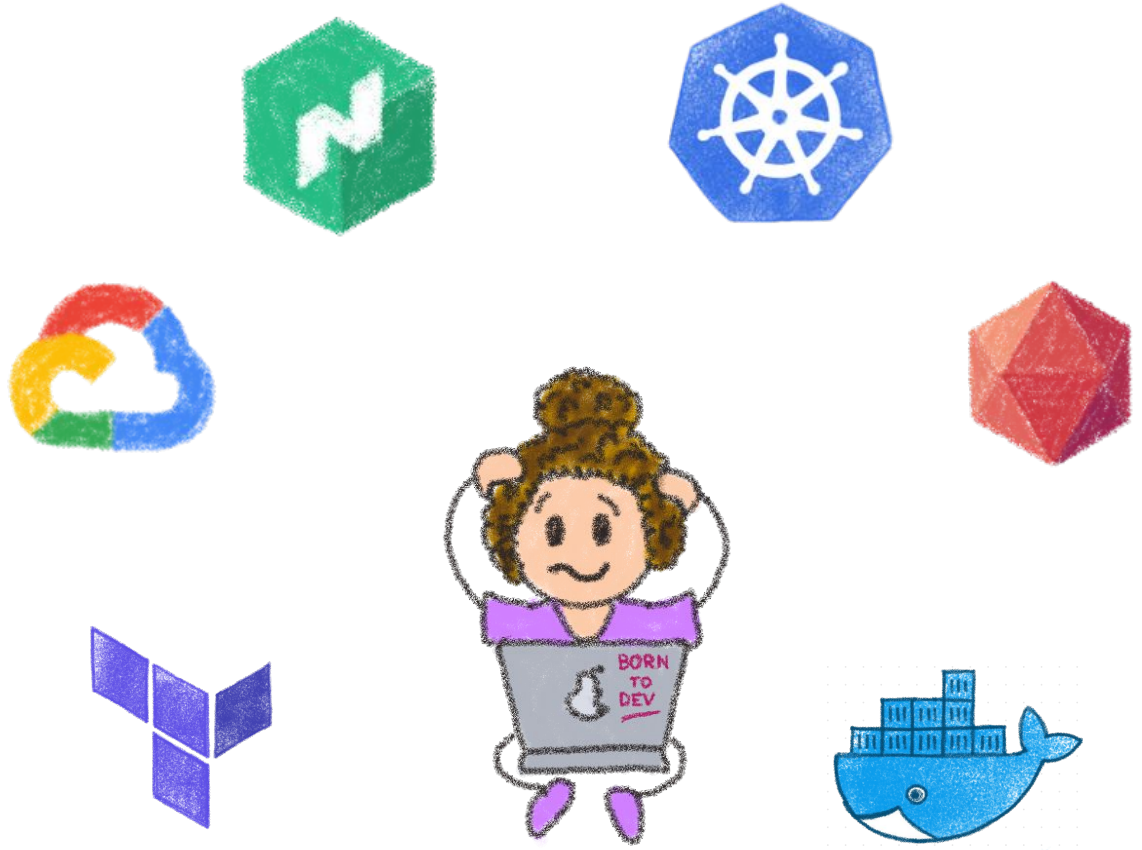
Qui est réveillé à 3h du matin

Ce qu'elle ne vous dit pas



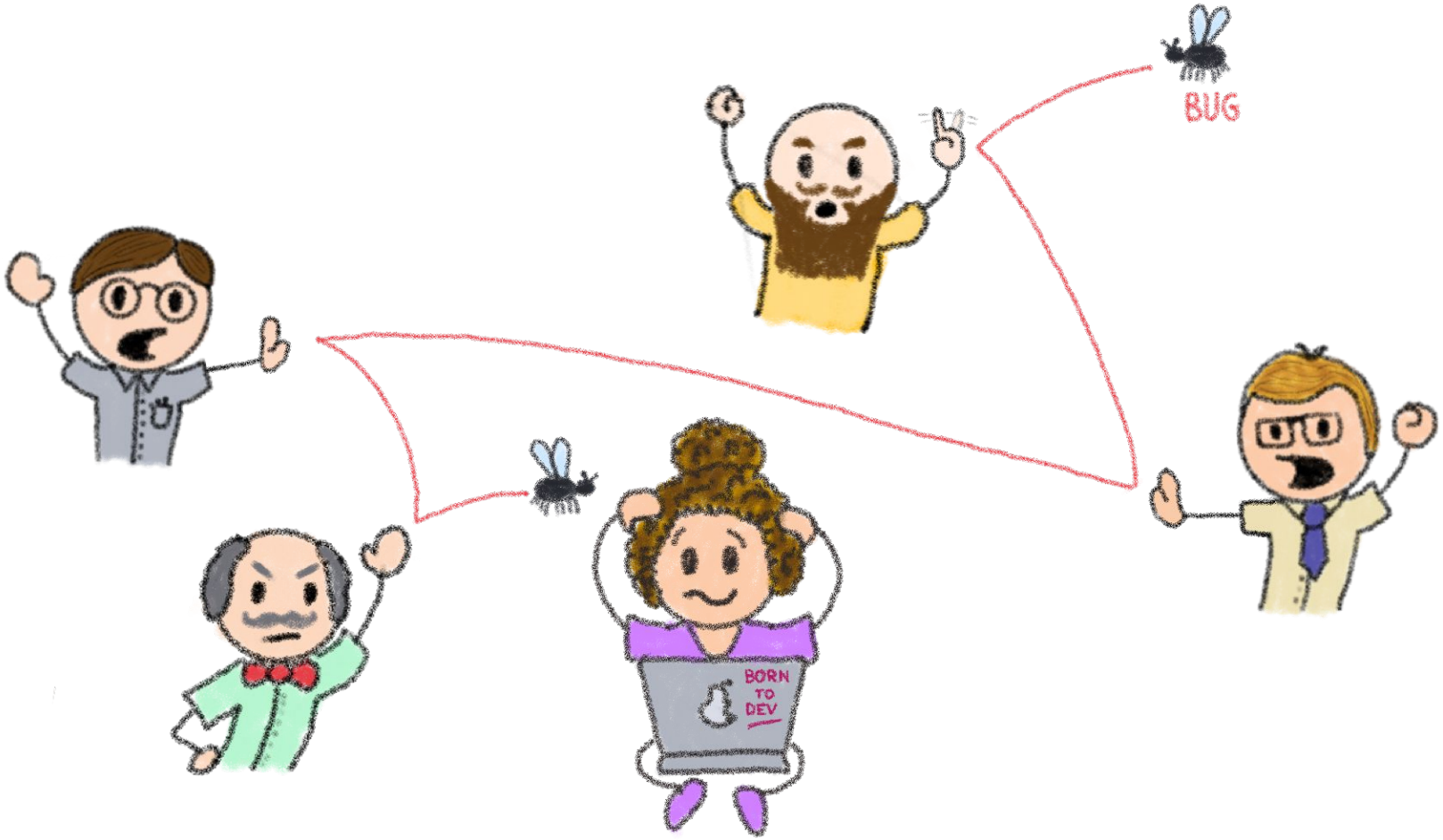
Ce qu'une nouvelle recrue doit savoir pour livrer

Ce qu'elle ne vous dit pas



Quelles compétences vous devez désormais recruter

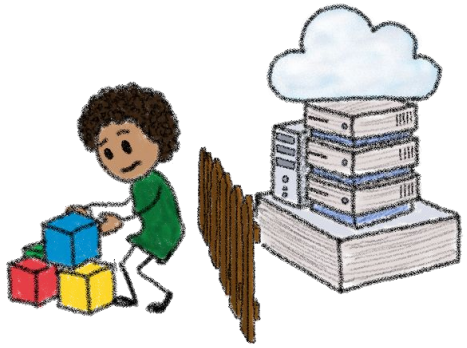
Ce qu'elle ne vous dit pas

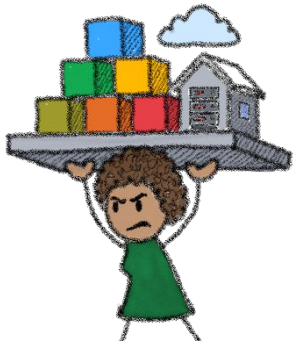


Où les incidents rebondissent entre les équipes

Le prisme du modèle opérationnel

Quatre dimensions que la matrice de fonctionnalités ignore



Les quatre dimensions



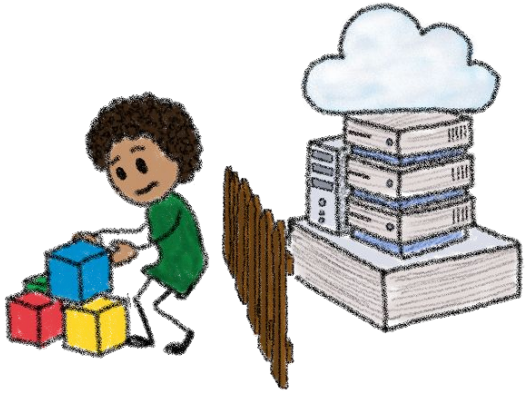
Charge cognitive

Ce qu'un dev doit savoir pour livrer en sécurité



Modèle d'astreinte

Qui est réveillé, et pourquoi



Frontières de responsabilité

Où l'appli s'arrête, où la plateforme commence



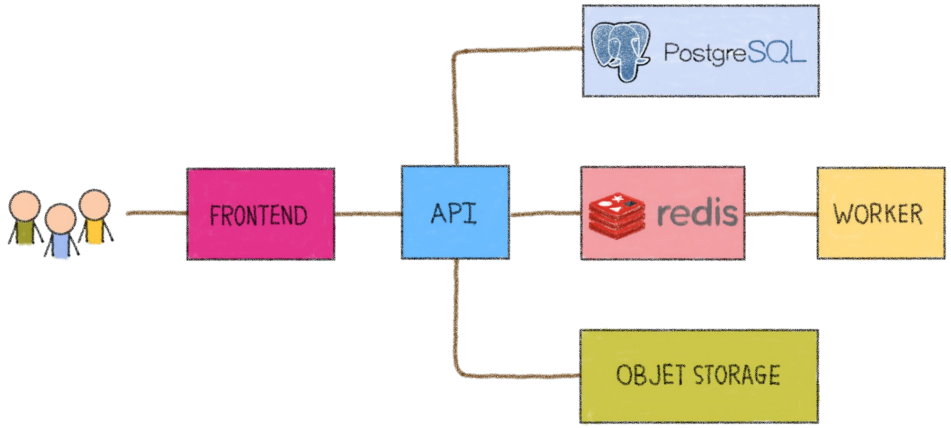
Personnes et résilience

Recruiter, onboarder, et bus factor

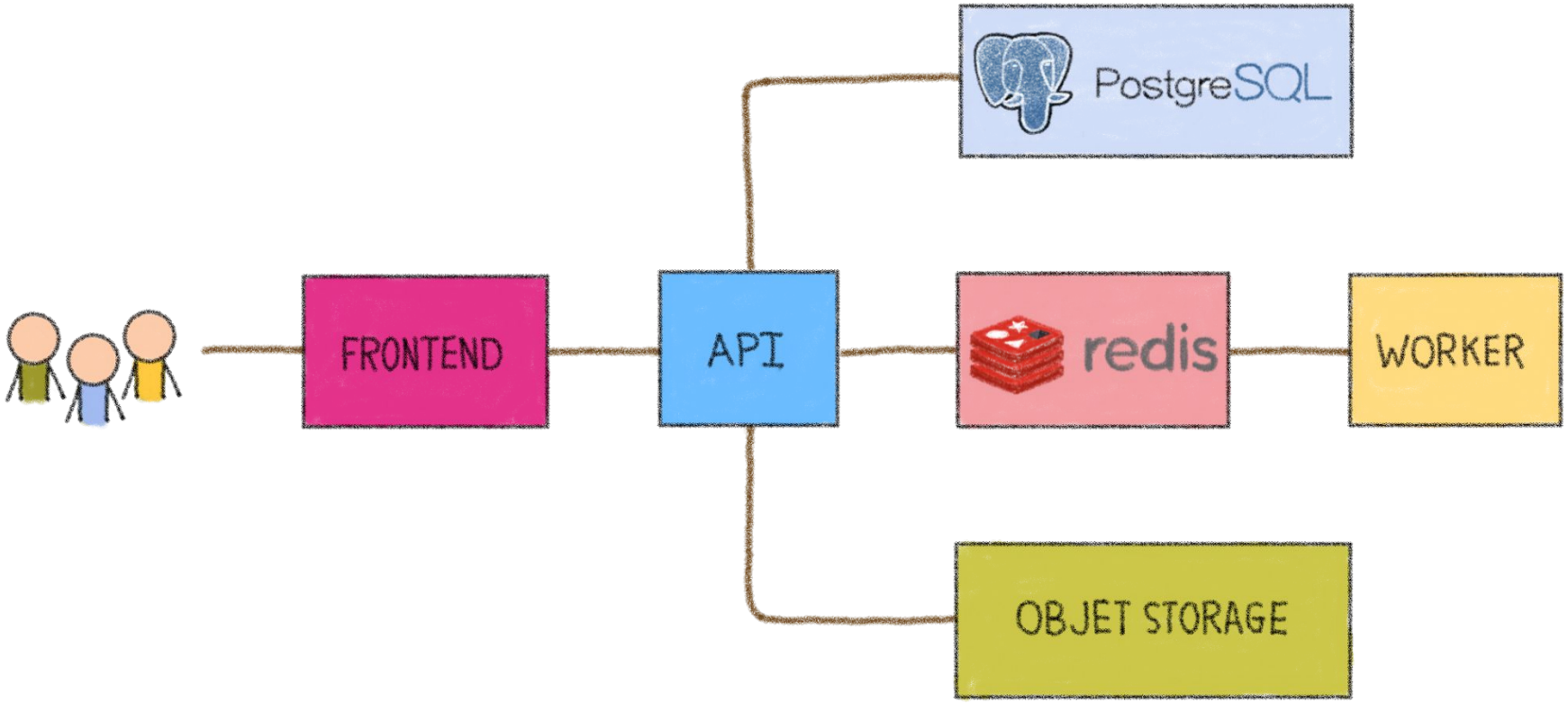


Une seule application

Délibérément banale, la seule chose qui ne changera pas



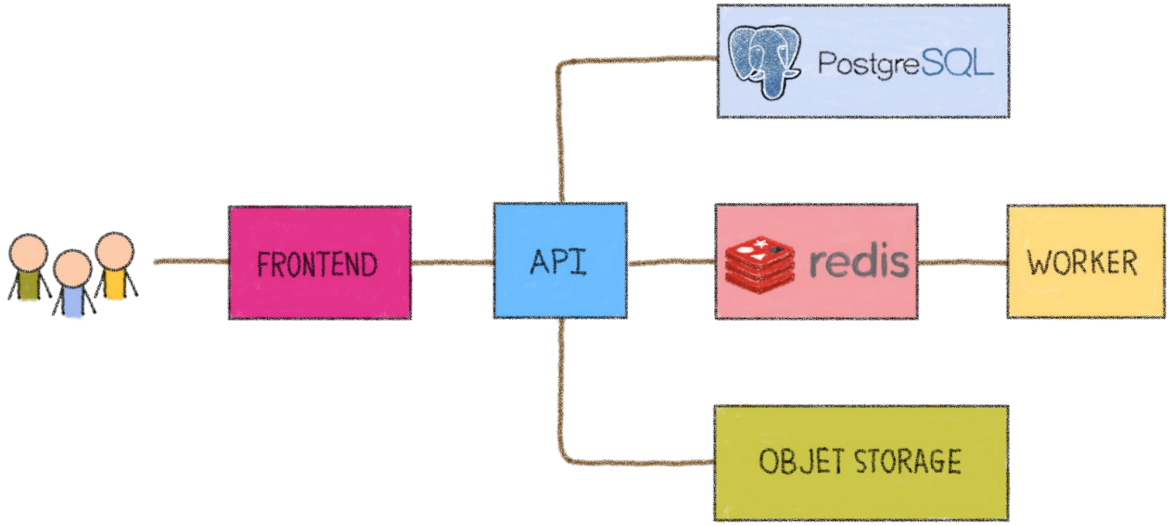
L'application ennuyeuse



6 développeurs · 1 tech lead · pas de SRE dédié · livre régulièrement

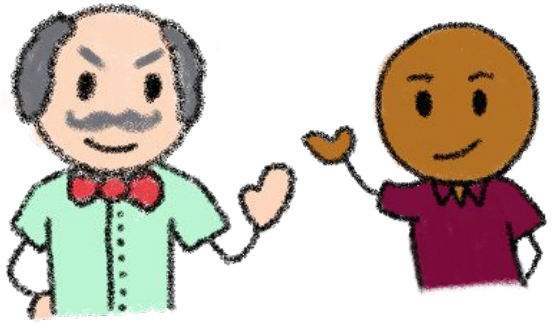
La constante

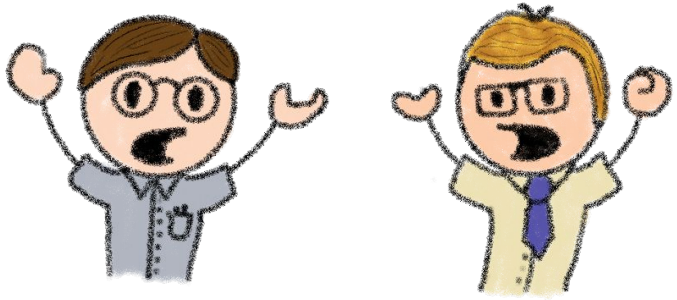
L'application ne change pas, l'organigramme, si



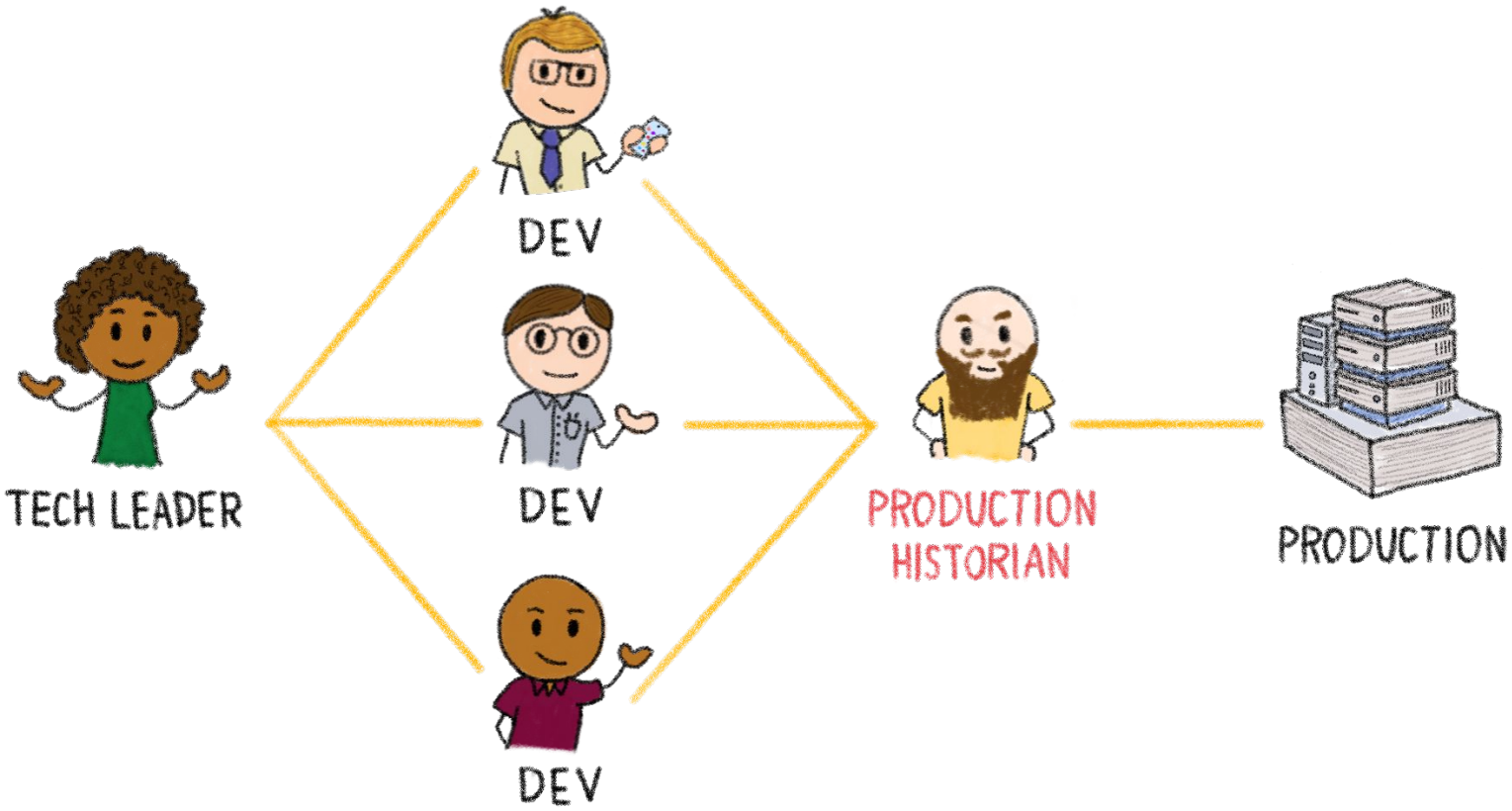
Une appli, quatre organigrammes

Même appli, même besoin métier, quatre équipes





VMs auto-gérées

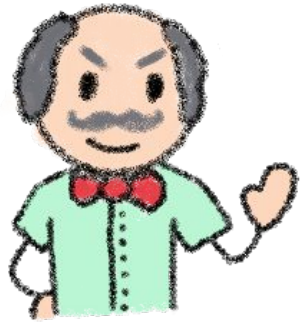


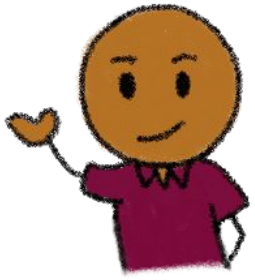
L'Historien·ne de la prod : le savoir concentré dans une seule personne

VMs auto-gérées

Ce qui devient plus facile

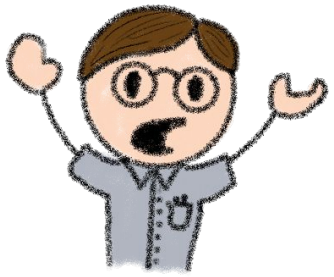
- Contrôle direct
- Des primitives compréhensibles
- Facile à démarrer, petits systèmes
- Convient au legacy ou aux contraintes spécifiques

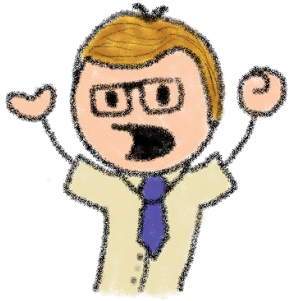




Ce qui devient un boulot à faire

- Patching, provisioning, pare-feu
- Sauvegardes et monitoring
- Scripts de déploiement, réponse aux incidents
- Docs en retard sur la réalité



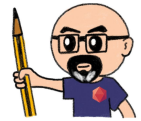




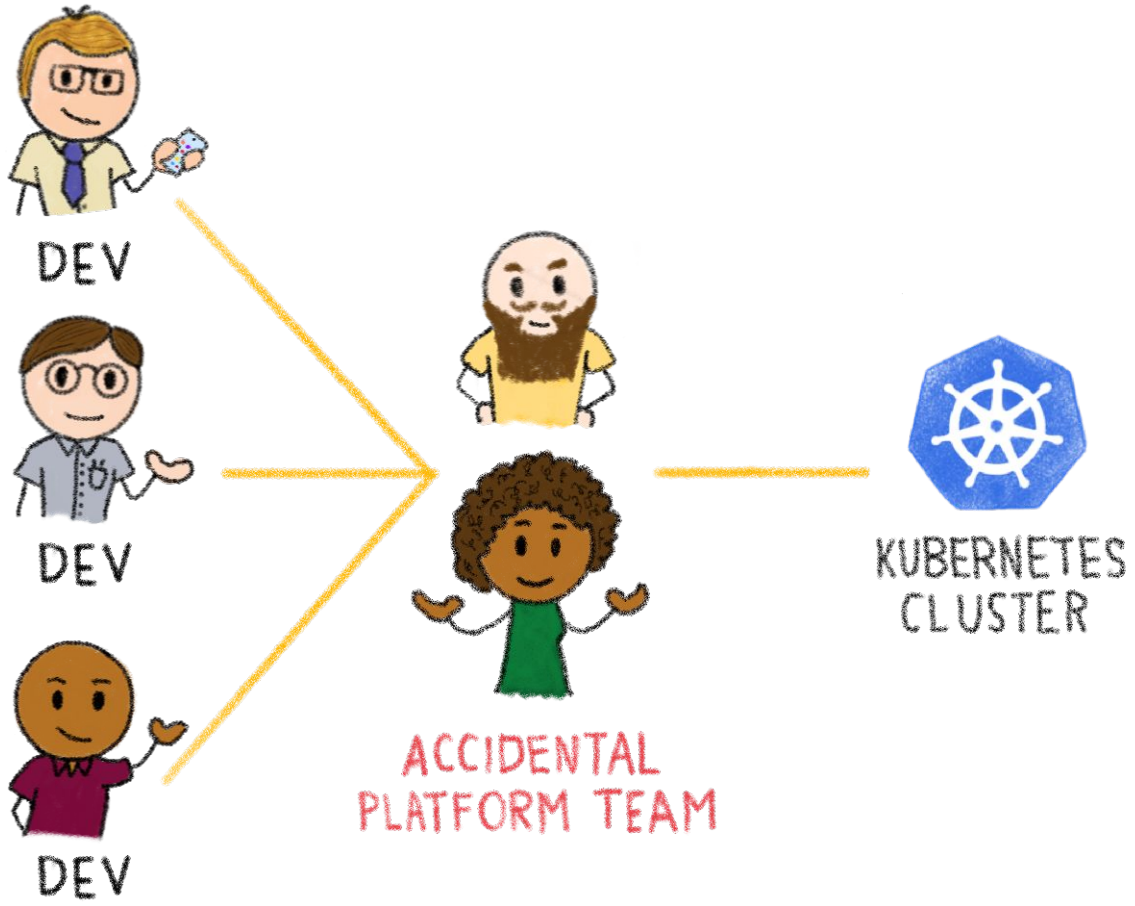
VMs auto-gérées

Vous n'avez peut-être pas d'équipe plateforme…

mais vous avez une **personne plateforme**



Kubernetes managé



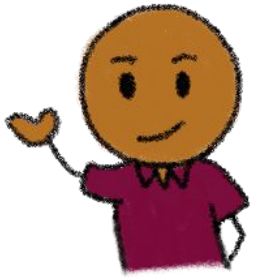
L'équipe plateforme accidentelle : née sans mandat

Managed Kubernetes

Ce qui devient plus facile

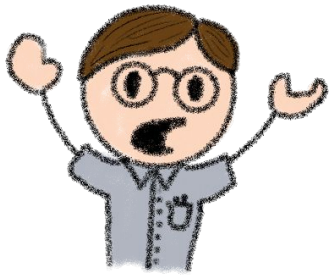
- Orchestration standard
- Écosystème riche, portabilité
- Des primitives puissantes
- Une base pour construire une plateforme

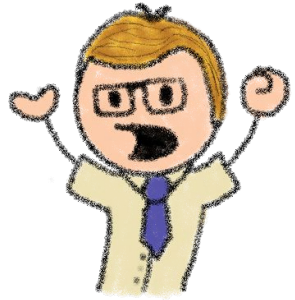


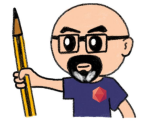


Ce qui devient un boulot à faire

- Upgrades du cluster, RBAC, réseau
- Capacity planning, observabilité
- Golden paths, expérience développeur
- Documentation de la plateforme



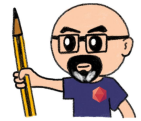




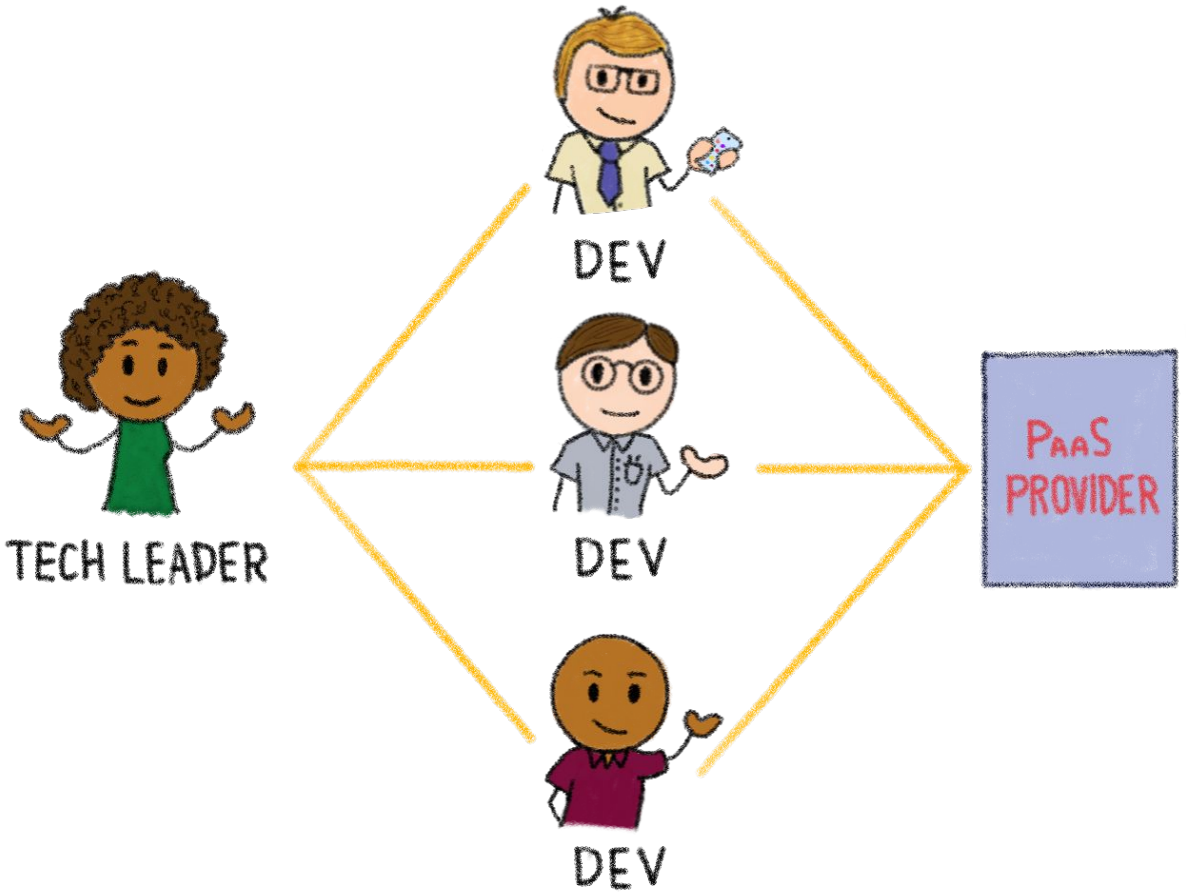
Managed Kubernetes

Vous avez acheté un **kit de construction**, pas un produit...

quelqu'un doit **assumer la construction**



PaaS

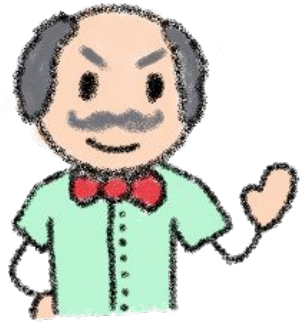


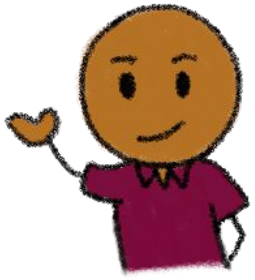
Pas de héros caché : le fournisseur est la personne plateforme que vous n'avez pas eu à recruter

PaaS

Ce qui devient plus facile

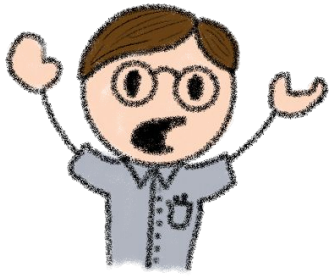
- Onboarding et déploiements rapides
- Autonomie des développeurs
- Responsabilité de la prod par une petite équipe
- Se concentrer sur le produit

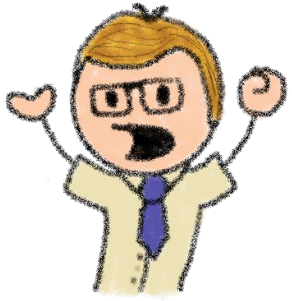




Ce qui devient un boulot à faire

- Comprendre les contraintes de la plateforme
- Le modèle de scaling et de sauvegarde
- Là où l'abstraction fuit
- Mais bien moins de gens ont à le faire







PaaS

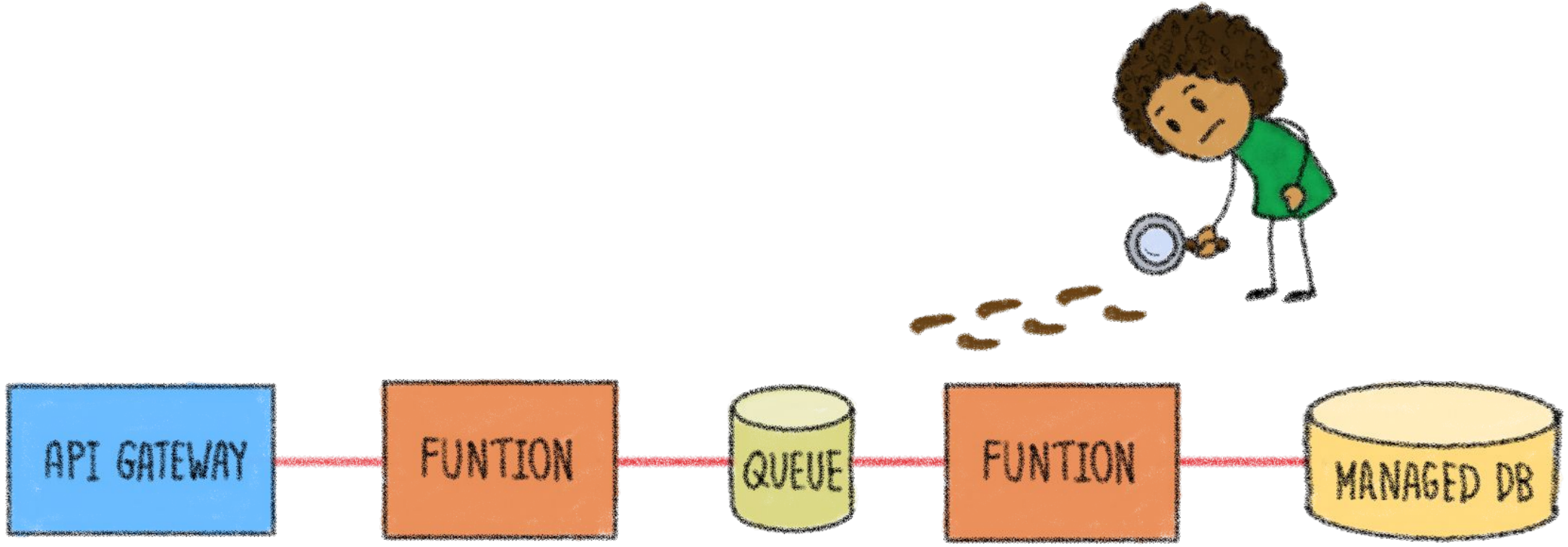
Un bon PaaS permet

à une **équipe produit d'opérer la production**

sans devoir d'abord devenir une équipe plateforme



Serverless

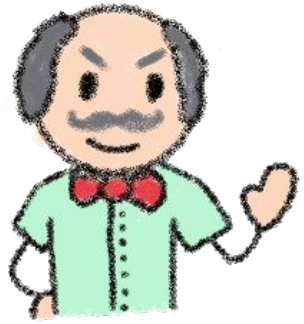


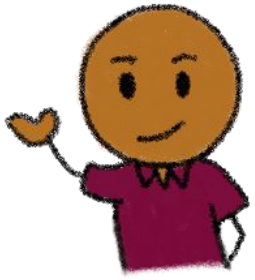
Le Détective des systèmes distribués :
la responsabilité se déplace vers l'architecture, l'IAM, le tracing

Serverless

Ce qui devient plus facile

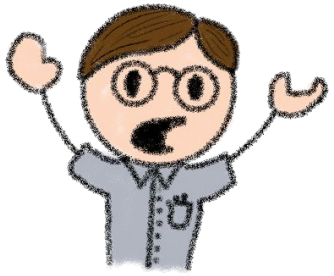
- Scaling en pic, pas de serveurs
- Charges event-driven
- Moins de capacité au repos
- Expérimentation rapide

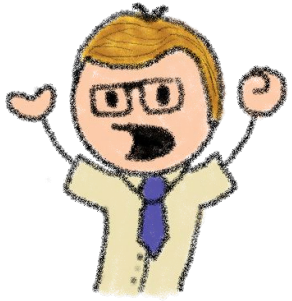




Ce qui devient un boulot à faire

- Limites du fournisseur et IAM
- Retries et idempotence
- Tracing à travers les services
- Prédictibilité des coûts







Serverless

Le serverless retire les serveurs de l'organigramme, pas les opérations :

Il transforme les **problèmes opérationnels** en **problèmes d'architecture**



Où va la complexité ?

Le motif commun aux quatre



Chaque plateforme déplace la complexité

Modèle	Où va la complexité ?
VMs auto-gérées	dans les gens et les scripts
Kubernetes	dans la couche plateforme
PaaS	dans le contrat du fournisseur
Serverless	dans l'architecture distribuée



Où elle vit

La question n'a jamais été quel modèle a de la complexité, ils en ont tous…

C'est où vous voulez que cette complexité vive





Vous avez ri

Parce que c'est toujours l'architecture d'une autre équipe.

Alors : qui a choisi la vôtre ?



Mardi, 15h05

Prépare le déploiement du service de notifications. **Production-ready, scalable.**

6 devs · 1 tech lead · pas de SRE · pas de réunion · pas d'ADR



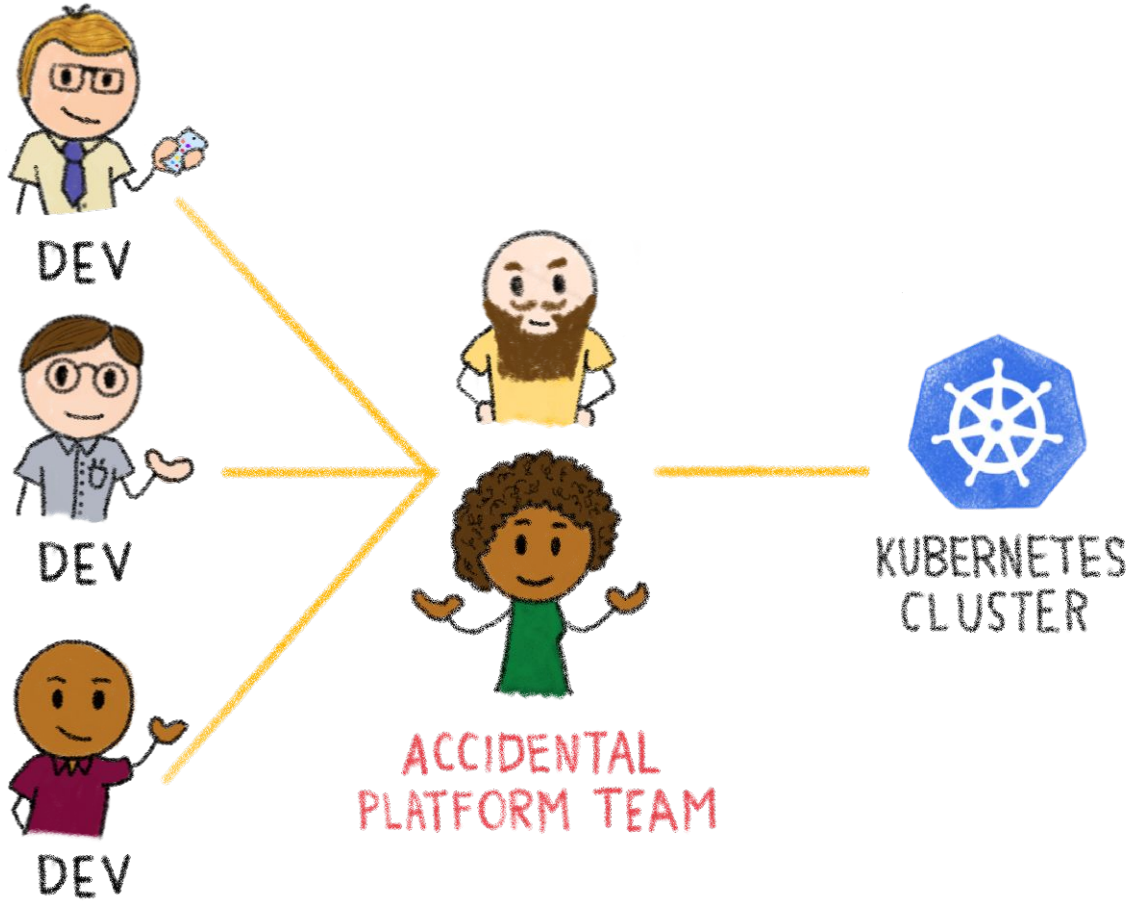
Mardi, 15h45

```
notification-service/
├── Dockerfile
├── helm/              cert-manager, external-secrets, HPA
├── terraform/         cluster Kubernetes managé
├── argocd/
├── monitoring/        Prometheus, Grafana
└── README.md          (magnifique)
```

+2 347 −0 · CI au vert · review : **" LGTM, propre."**



Mercredi : mergée



Personne n'a décidé de devenir une équipe plateforme

Samedi, 3h12

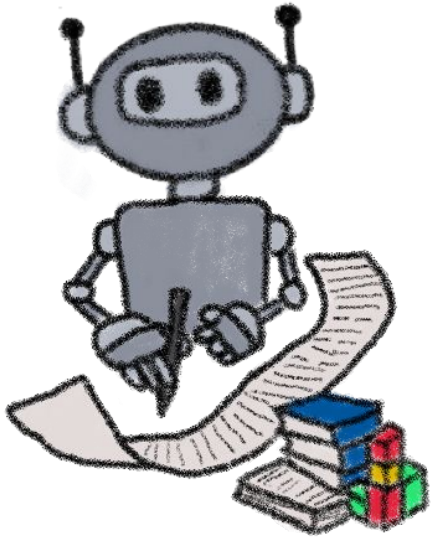
- Un chart utilisait une version d'API **qui n'existe plus**
- Le certificat **ne s'est pas renouvelé**. TLS expiré, site par terre
- D'astreinte : une dev backend **qui n'a jamais ouvert le cluster**



Trois correctifs sûrs d'eux proposés par l'agent.

Le deuxième marche à moitié. Retour en ligne à 5h30

Rétro du lundi : qui est responsable du cluster



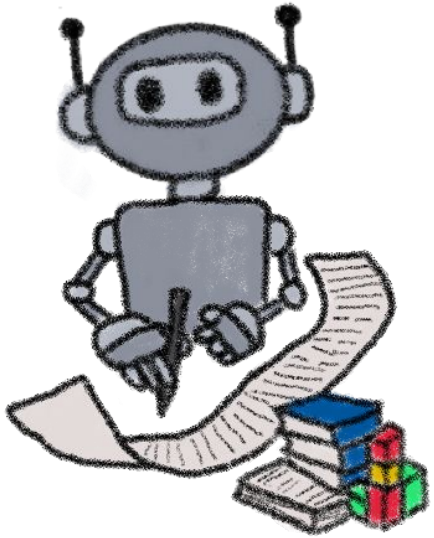


L'architecture avait un auteur

L'organigramme n'en avait aucun



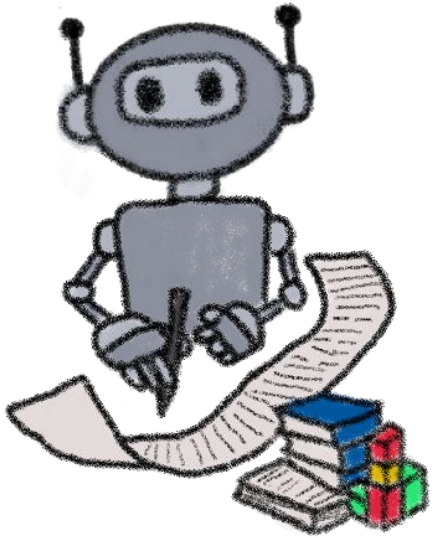
À vous





1. Levez la main si le dernier Dockerfile, chart Helm ou module Terraform de votre repo a été écrit en grande partie par un agent.

À vous

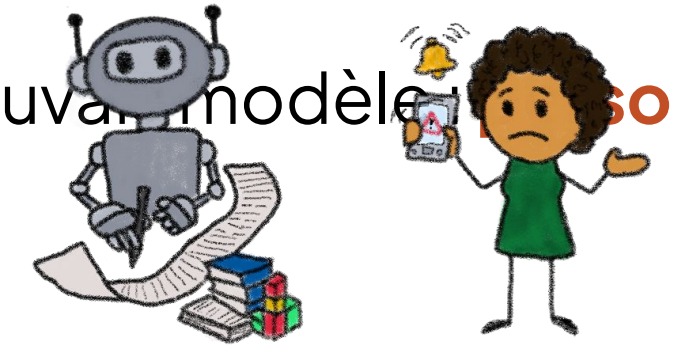




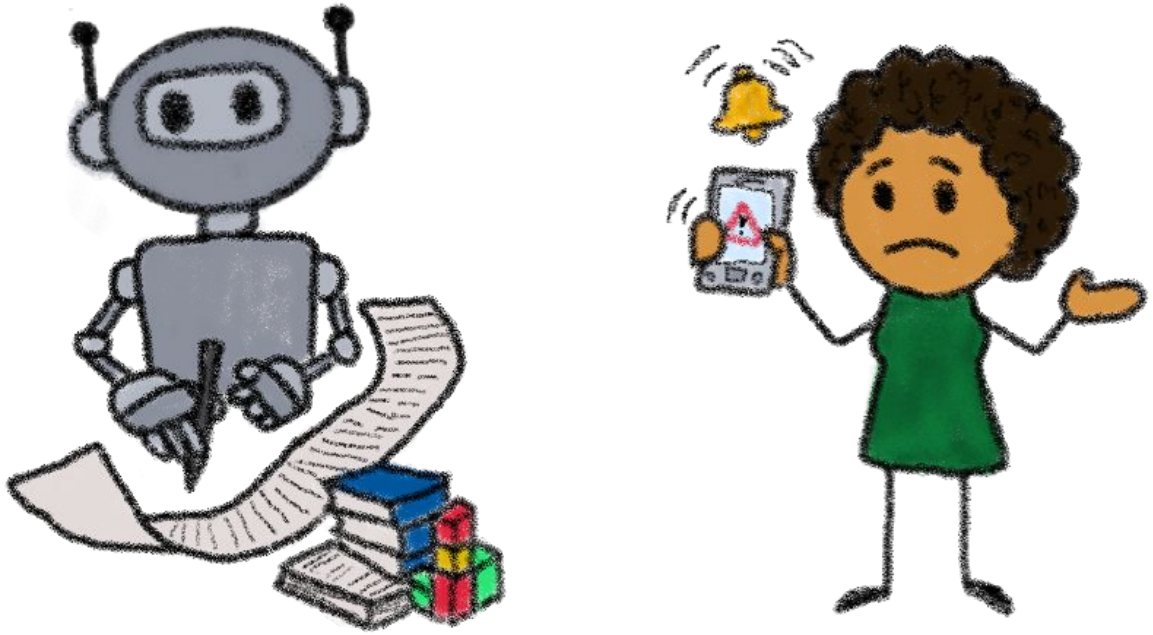
2. Gardez-la levée si quelqu'un a relu l'organigramme qui venait avec

Pas juste "relisez vos PR"

- **L'agent ne voit pas votre équipe** : six devs, pas de SRE
- Il a appris le *production-ready* d'orgas **avec des équipes plateforme**
- ***Scalable*** était une décision d'organigramme
- Il n'a pas choisi le mauvais modèle, **personne n'en a choisi**



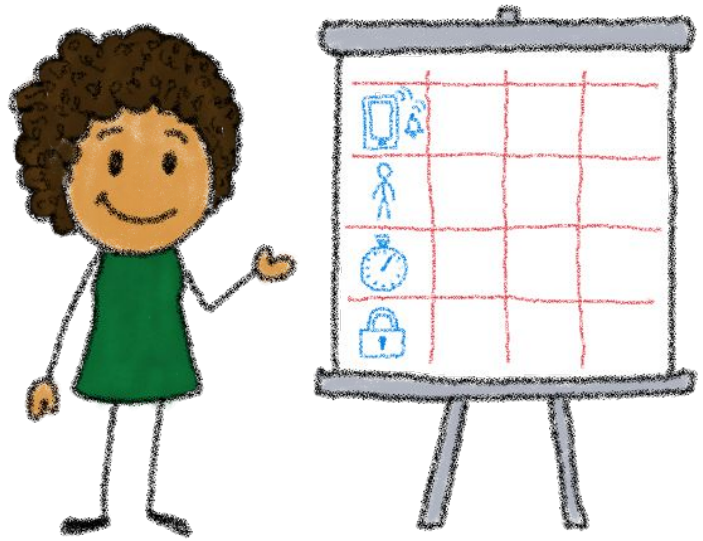
Produire n'est pas posséder



L'IA change qui fait le travail, pas qui en est responsable

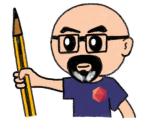
La matrice que vous devriez vraiment faire

Évaluez le modèle opérationnel, pas les fonctionnalités



La matrice du modèle opérationnel

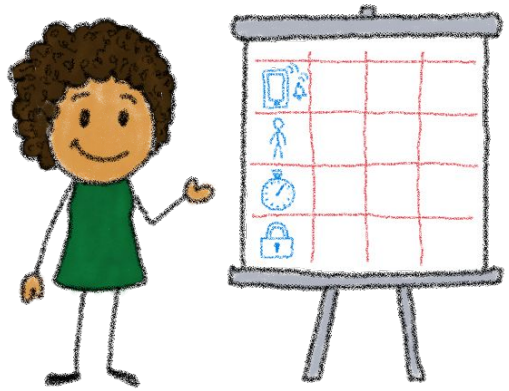
Question	Measures
Qui est réveillé à 3h du matin ?	Astreinte / responsabilité
Que doit savoir chaque dev pour déployer ?	Charge cognitive
Combien de temps avant qu'une recrue livre ?	Personnes et résilience
Quelle expertise devons-nous recruter ?	Personnes et résilience
Et si l'expert plateforme part ?	Personnes et résilience
Produit ou plateforme ?	Orientation stratégique
Posséder la complexité ou la louer ?	Le vrai arbitrage
Qui décide, et où est-ce écrit ?	La propriété de la décision

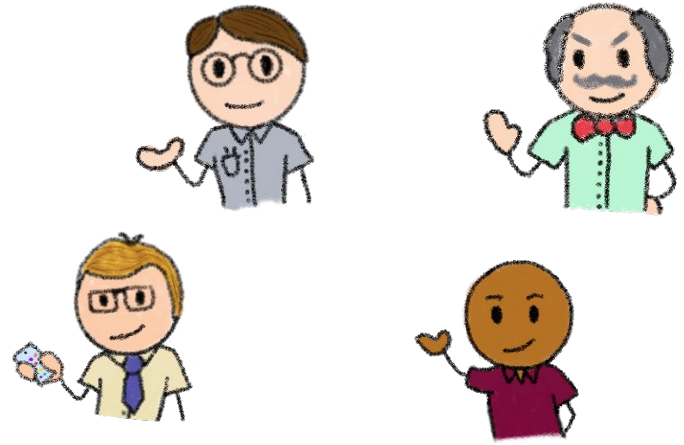


La vraie question

Ne demandez pas si la plateforme peut faire tourner votre application,

demandez quelle équipe vous devriez devenir pour bien l'opérer





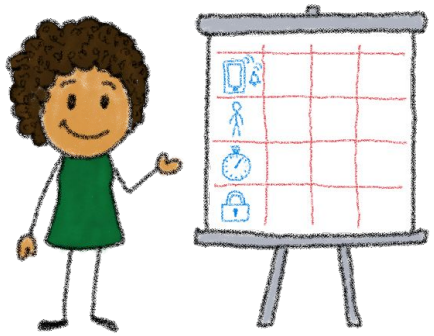
À choisir quand

VMs auto-gérées

- Contrôle, système petit ou stable
- Ops solides
- Pas de goulot unique

Kubernetes

- Vous avez besoin d'une plateforme
- Plusieurs équipes
- Capacité à investir dans le platform engineering



PaaS

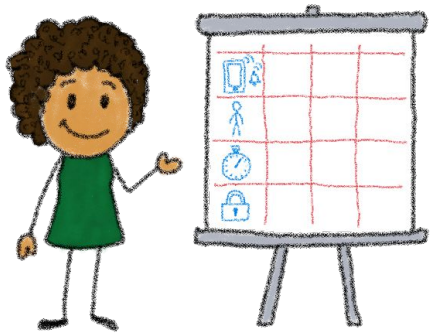
- Le produit avant le contrôle de la plateforme
- Petite équipe
- Onboarding rapide
- Les contraintes conviennent

Serverless

- Event-driven ou en pics
- L'équipe maîtrise les systèmes distribués
- Investir dans l'IAM et le tracing

Trois choses à faire la semaine prochaine

- Mesurez votre vrai organigramme : réveils à 3h, premier déploiement
- Trouvez votre PR du mardi : qui a décidé ?
- Écrivez l'organigramme là où **les gens et les agents** le lisent

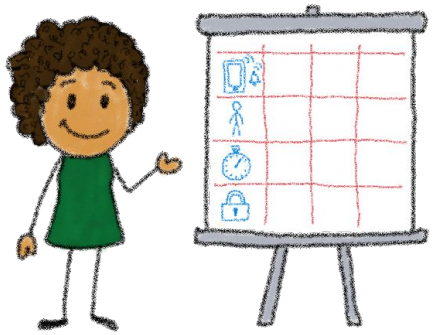




Rendez-le visible

Chaque diagramme d'architecture cache **un organigramme**

Rendez-le visible avant que la production ne s'en charge



C'est tout, les amis !

Merci à toutes et à tous !

Vous avez les slides disponibles
sur lostinbrittany.dev

