Engineer

Déploiement

mise en service d'une modification là où elle est utilisée

Ce que c’est

Le déploiement est la suite d’étapes qui fait sortir une version testée d’une application du système de construction pour la mettre en service : placer les fichiers construits ou l’image de conteneur sur les machines ou la plateforme qui l’exécuteront, lui fournir la configuration de cet environnement, la démarrer et lui transférer les requêtes réelles. Le mot désigne à la fois l’action et son résultat. Une équipe parle donc d’un déploiement bloqué à l’étape de la base de données, mais aussi des déploiements actuellement en cours d’exécution dans chaque environnement. On peut déployer un site web, une API, une tâche planifiée, une modification de base de données ou le backend d’une application mobile.

Déployer et rendre disponible sont souvent deux opérations distinctes. Le déploiement place le nouveau code là où il peut s’exécuter ; la mise à disposition décide qui le voit. Une fonctionnalité cachée derrière un indicateur d’activation peut être déployée le lundi et ouverte aux clients le jeudi sans retoucher les serveurs. Cette séparation permet de livrer le code par petites étapes fréquentes, tout en laissant les décisions de calendrier produit aux personnes qui en ont la responsabilité.

Beaucoup de déploiements livrent aujourd’hui une image de conteneur. L’Open Container Initiative, créée sous l’égide de la Linux Foundation en juin 2015, publie trois spécifications : la structure d’une image, la façon dont un moteur d’exécution l’exécute une fois dépaquetée et la manière dont les registres stockent et distribuent les images. Une image est identifiée par son empreinte, un hachage de son contenu ; la même empreinte désigne donc toujours les mêmes octets. Une étiquette telle que `1.4.2` est un pointeur lisible vers une image. Publier à nouveau sous cette étiquette la fait pointer vers autre chose.

La méthode Twelve-Factor App, souvent citée pour construire des services web, divise le chemin du code au service en fonctionnement en trois étapes strictement séparées. La construction transforme un commit en paquet exécutable. La préparation d'une version associe ce paquet à la configuration d'un environnement. L'exécution démarre les processus à partir d'une version choisie. Chaque version reçoit un identifiant unique et n'est jamais modifiée après sa création. Tout changement crée une nouvelle version ; revenir en arrière consiste alors à en choisir une précédente.

La bascule dépend de ce que le service peut tolérer. Un déploiement par recréation arrête l'ancienne version puis démarre la nouvelle, en acceptant une interruption entre les deux. Un déploiement progressif remplace quelques instances à la fois : les deux versions servent donc le trafic pendant un temps. C'est la stratégie préconfigurée d'un Deployment Kubernetes. Un déploiement bleu-vert exécute la nouvelle version à côté de l'ancienne et déplace tout le trafic d'un coup, en gardant l'ancienne prête à le reprendre. Un déploiement canari envoie d'abord une petite part du trafic à la nouvelle version, puis l'augmente tant que les taux d'erreur et la latence restent satisfaisants.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'un déploiement ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des changements testés, publiés et réversibles, pour livrer aux clients un mardi après-midi plutôt qu'un vendredi soir tard.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Déploiement SaaS GIS automatisé : GitHub Actions — déploiement, données et reporting automatisés

Déploiement SaaS GIS automatisé : GitHub Actions Contactez-nous →

Pourquoi c'est important

Le déploiement est le moment où une modification commence à être utile, mais aussi celui où une erreur commence à atteindre les utilisateurs. Les équipes qui déploient rarement accumulent souvent de nombreuses modifications dans chaque version. Chaque déploiement comporte alors plus de risques et, s'il échoue, plus de causes possibles. Des déploiements fréquents et de faible ampleur rendent chaque étape facile à comprendre et à annuler.

La constance d'un déploiement manuel dépend entièrement de la personne qui l'exécute. Des étapes sont omises, changent d'ordre ou sont effectuées un peu autrement par un collègue qui assure le remplacement pendant les congés. Un

pipeline exécute les mêmes étapes chaque fois et consigne le commit, l'artefact et l'environnement, avec un horodatage. Cette trace est le point de départ d'une analyse d'incident et souvent le moyen le plus rapide de savoir ce qui a changé.

L'habitude la plus solide consiste à construire une fois puis à promouvoir le résultat. L'artefact testé en préproduction devrait être identique, octet pour octet, à celui qui atteint la production. Seule la configuration diffère, et l'environnement la fournit à l'arrivée de l'artefact. Enregistrer l'empreinte à la construction et déployer par empreinte rend cette promesse vérifiable :

```
podman build -t registry.example.com/web:1.4.2 .
podman push --digestfile web.digest registry.example.com/web:1.4.2
kubectl -n staging set image deployment/web \
  web=registry.example.com/web@$(cat web.digest)
# after staging checks pass, the same digest goes to production
kubectl -n production set image deployment/web \
  web=registry.example.com/web@$(cat web.digest)
```

Le retour arrière dépend de la même discipline. Avec des versions immuables et identifiées, revenir en arrière consiste à pointer vers la précédente, ce qui prend quelques secondes. Si chaque environnement reconstruit depuis les sources, il faut reconstruire un ancien commit et espérer que ses dépendances se résolvent encore de la même façon.

À quoi cela ressemble en pratique

Les déploiements d'une plateforme vers Azure étaient pris en charge de bout en bout. Chaque modification suivait le même itinéraire fixe du développement à la préproduction puis à la production, sans publication directe dans un environnement en service. La CI/CD construit et livre les images, tandis que les modèles Bicep maintiennent la même infrastructure cible dans chaque environnement : chaque promotion arrive donc sur une infrastructure correspondante. Les paramètres propres à chaque environnement sont conservés à part des secrets. Une construction unique peut ainsi traverser toutes les étapes et recevoir les paramètres de chaque environnement à son arrivée, sans être reconstruite en chemin. Les modifications atteignent chaque environnement par cet itinéraire défini, sans déploiement manuel improvisé ; la mise à disposition est devenue une étape contrôlée qui s'exécute de la même façon chaque fois.

Une mission antérieure montre ce que cette constance représente en heures. L'automatisation de la construction, des tests et du déploiement a économisé environ 4 000 heures en standardisant le chemin d'une modification jusqu'à la production et en supprimant les étapes manuelles qui variaient selon leur exécutant. Les mises à disposition sont devenues à la fois plus rapides et plus fiables.

Sur la plateforme de l'entreprise, le déploiement est séparé du provisionnement. Dans une couche de conteneurs fondée sur Podman et Quadlet, root prépare une fois le chemin de déploiement. Ensuite, un utilisateur non privilégié redéploie en exécutant un unique script fixe, par une règle étroitement limitée qui n'autorise que cette commande sans argument. Le script construit l'image et redémarre le service. Une mise à disposition courante n'exige donc ni shell root ni commandes arbitraires.

Où les problèmes surviennent

La plupart des difficultés de déploiement viennent d'un écart entre ce qui a été contrôlé et ce qui a réellement changé.

La production exécute ce que la préproduction n'a jamais vu. Quand chaque environnement construit sa copie, les plages de dépendances se résolvent vers des versions plus récentes, une image de base change entre les constructions ou un indicateur de construction diffère. L'artefact ayant passé les tests n'est alors pas celui qui sert les clients. Construisez une fois, enregistrez l'empreinte et promouvez-la. Gardez tout ce qui dépend de l'environnement dans la configuration plutôt que dans l'artefact.

Une étiquette change de cible et le retour arrière atteint la mauvaise version. Un déploiement qui nomme `latest`, ou réutilise une étiquette de version après reconstruction, dépend de sa cible le jour de l'exécution. La documentation Kubernetes déconseille `:latest` en production : il devient plus difficile de savoir quelle version s'exécute et de revenir correctement en arrière. Utilisez une étiquette de version jamais réutilisée et verrouillez l'image sur son empreinte lorsque la plateforme le permet.

Le pipeline annonce un succès avant que la nouvelle version fonctionne. Beaucoup de commandes de déploiement rendent la main dès que la plateforme accepte la demande, alors que les nouvelles instances démarrent encore et peuvent planter une minute plus tard. Une étape qui attend que la nouvelle

version se déclare prête, puis envoie quelques requêtes réelles et vérifie les réponses, fait échouer explicitement le processus de construction au lieu de laisser la défaillance passer inaperçue, alors que la version précédente reste à portée de main.

Le nouveau code rencontre l'ancien schéma au milieu du déploiement. Pendant un déploiement progressif ou canari, anciennes et nouvelles versions servent côte à côte le trafic en utilisant une seule base. Une migration renommant ou supprimant une colonne casse les anciennes instances encore actives et rend le retour arrière dangereux, puisque l'ancien code attend ce qui a été supprimé. Répartissez ces changements sur plusieurs déploiements : ajoutez d'abord la nouvelle structure, livrez un code compatible avec les deux et ne retirez l'ancienne que lorsqu'aucune version active n'en a plus besoin.

Pour aller plus loin

- Construction, préparation de version, exécution — l'argumentaire de Twelve-Factor App pour trois étapes séparées et des versions jamais modifiées après leur création.
- Open Container Initiative — le projet de la Linux Foundation à l'origine des spécifications d'image, d'exécution et de distribution dont dépendent la plupart des déploiements de conteneurs.
- Spécification du format d'image OCI — manifestes, couches, configuration et empreintes de contenu identifiant une image.
- Deployments Kubernetes — mises à jour progressives, stratégie de recréation, délai de progression et retours arrière.
- Images Kubernetes — étiquettes, empreintes, politiques de récupération et raisons pour lesquelles `:latest` complique les retours arrière.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'un déploiement ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/deployment/