Engineer

Déploiement canari

publier d'abord pour une petite part des utilisateurs

Ce que c’est

Un déploiement canari expose d’abord une nouvelle version d’un service à une petite part de son trafic réel, tandis que la version remplacée continue à servir tous les autres utilisateurs. Si la nouvelle version se comporte correctement, sa part augmente par étapes jusqu’à recevoir tout le trafic, et l’ancienne est retirée. Sinon, le trafic retourne vers l’ancienne version, qui ne s’est jamais arrêtée, et seule cette petite part a rencontré le défaut. Danilo Sato a décrit la technique sur le bliki de Martin Fowler en juin 2014, en précisant qu’elle porte aussi les noms de déploiement par phases ou de déploiement incrémentiel. Les outils Kubernetes emploient généralement canary deployment : Argo Rollouts, contrôleur Kubernetes qui ajoute ce type de stratégie de publication, nomme le réglage `canary` .

Le nom vient des mines de charbon. Les mineurs descendaient un canari en cage sous terre : une fuite de gaz terrassait l’oiseau avant d’atteindre les personnes, et sa détresse les avertissait de sortir. La petite part de trafic joue le même rôle : elle rencontre le problème la première, pendant qu’il est encore temps d’agir.

La composition de cet échantillon est un choix. Sato cite un échantillon aléatoire d’utilisateurs, le personnel de l’entreprise avant le public et des utilisateurs choisis selon leur profil ou leurs caractéristiques démographiques. Les grands systèmes utilisent souvent des divisions déjà existantes : d’abord une région, un lieu ou une marque. Selon le récit de Sato, Facebook utilisait plusieurs canaris. Le premier n’était visible que par son personnel, avec tous les indicateurs de fonctionnalité activés, afin que les problèmes des nouvelles fonctions apparaissent tôt.

La différence avec le déploiement bleu-vert tient à la façon dont le trafic est transféré. Le bleu-vert construit une seconde copie complète et transfère tout le trafic en une bascule, en gardant l'ancienne copie comme voie de retour. Un canari déplace le trafic progressivement : pendant un temps, deux versions servent donc de vrais utilisateurs côte à côte. Une mise à jour progressive garde elle aussi deux versions actives à la fois, mais remplace les instances dès qu'elles sont prêtes, sans pause pour juger le comportement de la nouvelle version. Un canari se met en pause à chaque étape et attend une réponse.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'un déploiement canari ? » 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

Certains défauts ne se révèlent qu'en production : sous une charge réelle, avec de vraies données et face à la diversité réelle des clients. Un environnement de préproduction peut s'en approcher sans jamais être identique. Un canari l'accepte et limite plutôt les dégâts. Si une requête sur vingt atteint la nouvelle version, une version défaillante affecte une requête sur vingt jusqu'à sa détection. La correction consiste alors à renvoyer cette part vers l'ancienne version.

Sato signale une seconde utilisation : les tests de capacité. Augmenter lentement la part montre le comportement de la nouvelle version sous une charge de production croissante, avec un retour sûr, plutôt que de construire un environnement séparé pour les seuls tests de charge. Celui-ci ne peut jamais ressembler autant à la production que la production elle-même.

La difficulté est qu'un canari ne vaut que par le signal qui le juge. Les erreurs, la latence et les chiffres métier importants, comme les inscriptions, paiements ou commandes, doivent être visibles par version et assez tôt pour arrêter l'étape suivante. Sans cela, un canari n'est qu'un déploiement lent avec des attentes supplémentaires. Sato note qu'IMVU a été précurseur de ce qu'on appelle

un « système immunitaire de cluster » : surveiller les métriques métier et revenir automatiquement sur une publication en cas de régression statistiquement significative.

Il existe aussi des coûts d'exploitation. Deux versions ou plus sont simultanément en service, et tout ce qu'elles partagent — base de données, cache, file d'attente, API appelée par les clients — doit fonctionner avec les deux pendant toute cette période. Sato conseille de limiter au minimum le nombre de versions actives en même temps. La technique convient aussi mal aux logiciels installés sur les ordinateurs ou téléphones des utilisateurs : le fournisseur ne décide pas du moment de leur mise à jour. Dans ce cas, il suggère de garder une couche serveur capable de servir les deux versions, de surveiller les versions clientes encore utilisées et de ne retirer l'ancienne que lorsque sa part a suffisamment baissé.

À quoi cela ressemble en pratique

Dans sa forme la plus simple, un canari sur Kubernetes comprend deux Services, l'un sélectionnant les pods stables et l'autre les pods canaris, et une route qui pondère le trafic entre eux. Avec la Gateway API, les poids sont définis dans une HTTPRoute :

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: web
spec:
  parentRefs:
    - name: prod-web
  hostnames:
    - "www.example.com"
  rules:
    - backendRefs:
        - name: web-stable
          port: 8080
          weight: 95
        - name: web-canary
          port: 8080
          weight: 5
```

Les poids sont des proportions, pas des pourcentages : chaque service cible reçoit une part du trafic égale à son poids divisé par la somme des poids de la règle. Les valeurs 95 et 5 envoient donc une requête sur vingt vers le canari. Un

poids omis vaut 1, et un poids de 0 n'envoie rien. Le guide de la Gateway API sur la répartition du trafic montre aussi une règle correspondant à un en-tête de requête, `traffic: test` dans son exemple, qui n'envoie que ces requêtes vers la nouvelle version. Les personnes chargées des tests peuvent ainsi l'atteindre avant tout client. Faire avancer le canari consiste à modifier deux nombres ; le retirer revient à fixer son poids à 0.

Argo Rollouts automatise les étapes. Un `Rollout` remplace un `Deployment`, avec le même modèle de pod, et énumère les étapes que le contrôleur suit chaque fois que ce modèle change :

```
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: web
spec:
  replicas: 10
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: web
          image: registry.example.com/web:1.4.2
  strategy:
    canary:
      maxSurge: "25%"
      maxUnavailable: 0
      steps:
        - setWeight: 10
        - pause:
            duration: 1h
        - setWeight: 20
        - pause: {}
```

Sans routeur de trafic, Argo reproduit approximativement le poids avec le nombre de pods : dix répliques à 10 % signifient un pod canari et neuf pods stables. La première pause dure une heure ; la seconde dure jusqu'à ce qu'une décision soit prise. Un déploiement et ses deux issues possibles ressemblent à ceci :

```
kubectl argo rollouts set image web web=registry.example.com/web:1.5.0
# at the open-ended pause, either carry on ...
kubectl argo rollouts promote web
# ... or stop and put the stable version back in front of everyone
kubectl argo rollouts abort web
```

Toute cette séquence dépend de la possibilité de savoir, pendant la pause, si le canari fait moins bien que la version qu'il remplace. C'est le travail derrière la prestation de fiabilité des systèmes et de supervision, construite autour d'alertes qui se manifestent tant qu'un défaut est encore petit. La supervision des erreurs et le traçage internes, mis en place dans le cadre d'un poste occupé depuis 2025, montrent la forme d'un tel signal : les erreurs sont regroupées en problèmes récurrents plutôt qu'énumérées une par une, et pics et régressions sont détectés avec un délai de temporisation, afin qu'un mauvais déploiement n'appelle pas quarante fois une personne d'astreinte. Une étape canari a précisément besoin de cela : une régression qui se signale pendant la pause, pas une régression qu'un utilisateur découvre par hasard le lendemain matin.

Où les problèmes surviennent

Le canari est trop petit pour dire quoi que ce soit. Un pod sur dix à une heure calme peut traiter quelques centaines de requêtes pendant sa pause, sans jamais révéler un défaut touchant une requête sur mille. L'horloge est la mauvaise mesure ; le nombre de requêtes servies par le canari est la bonne. Dimensionnez ensemble la part et la pause pour qu'une erreur rare ait la possibilité d'apparaître plusieurs fois. Faites durer une étape plus longtemps la nuit qu'à midi.

Les tableaux de bord ne distinguent pas le canari de la version stable. Un canari qui fait échouer une requête sur cinquante de plus que la version stable, avec une part de 5 %, ajoute environ une requête en échec sur mille au total. Sur un tableau de bord montrant un seul taux d'erreur pour tout le service, c'est invisible. Donnez aux pods canaris et stables des étiquettes différentes, ajoutées par Argo Rollouts au moyen de `canaryMetadata` et `stableMetadata`. Transmettez l'étiquette aux métriques et comparez le canari à la version stable plutôt qu'à un seuil fixe. Une `analysis` en arrière-plan sur le Rollout provoque l'abandon de la mise à jour en cas d'échec de cette analyse.

Les deux versions écrivent dans les mêmes données. Pendant tout le déploiement, l'ancien et le nouveau code lisent et écrivent dans la même base. Avec un routage aléatoire, les requêtes d'un seul utilisateur peuvent atteindre

les deux en une minute. Une nouvelle colonne que l'ancienne version ne remplit jamais, ou un format qu'elle ne peut lire, provoque des erreurs du côté stable, celui qui devait être sûr. La réponse de Sato est le changement parallèle : la base prend en charge les deux versions de l'application pendant tout le déploiement, et l'ancienne structure n'est retirée qu'après la disparition de l'ancienne version.

Un abandon arrête le trafic, pas le changement. `kubectl argo rollouts abort` renvoie le trafic vers le ReplicaSet stable, mais la spécification du Rollout nomme encore la nouvelle image. Reprendre le Rollout remet cette image en cible. Le retour arrière n'est complet que lorsque le modèle est rétabli à la version précédente. Si la spécification est conservée dans un dépôt, effectuez-y la modification pour que la source et le cluster s'accordent sur ce qui doit fonctionner.

Le canari sert aussi de test A/B. Le routage est semblable dans les deux cas : il est donc tentant de mesurer l'effet d'une nouvelle fonction sur les conversions avec le trafic du canari. Sato conseille de séparer les deux : un canari cherche des problèmes et doit se terminer en quelques minutes ou heures, tandis qu'une expérience teste une hypothèse et peut demander plusieurs jours de données pour être statistiquement significative. Si le canari est jugé selon des métriques métier, l'effet recherché par l'expérience brouille aussi le verdict. Publiez avec le canari, puis lancez l'expérience avec sa propre répartition du trafic, par exemple au moyen d'un indicateur de fonctionnalité, une fois la version disponible.

Pour aller plus loin

- Déploiement canari — la description de Danilo Sato de 2014 sur le bliki de Martin Fowler : choix des utilisateurs, tests de capacité, tests A/B et base de données.
- Argo Rollouts : Canary — étapes et pauses, poids approximés par le nombre de pods et mise à l'échelle avec un routeur de trafic.
- Répartition du trafic HTTP — le guide de la Gateway API sur les services cibles pondérés et une route canari sélectionnée par en-tête.
- kubectl argo rollouts abort — ce qu'un abandon fait au trafic et ce qu'il laisse dans la spécification.

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 canari ? » 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/canary-release/