Engineer

Canary release

å slippe en endring til en liten andel av brukerne først

Hva det er

En canary release sender en ny versjon av en tjeneste ut til en liten del av den ekte trafikken først, mens versjonen den skal erstatte, fortsetter å betjene alle andre. Holder den nye versjonen, økes andelen trinn for trinn til den bærer alt, og den gamle versjonen kan fases ut. Holder den ikke, går trafikken tilbake til den gamle versjonen, som aldri sluttet å kjøre, og bare den lille andelen rakk å merke feilen. Danilo Sato beskrev teknikken på Martin Fowlers bliki i juni 2014 og nevner at den også kalles phased rollout eller incremental rollout. I Kubernetes-verdenen sier man oftere canary deployment, og på norsk ser man også canary-utrulling. Argo Rollouts, en Kubernetes-kontroller som legger til utrullingsstrategier av dette slaget, kaller innstillingen `canary`.

Navnet kommer fra kullgruvene. Gruvearbeiderne tok med seg en kanarifugl i bur ned i gruva, fordi gass som lekket inn, ville påvirke fuglen før den påvirket menneskene, og fuglens tegn på ubehag var varselet om å komme seg ut. Den lille andelen trafikk spiller samme rolle: den møter problemet først, mens det fortsatt er tid til å handle.

Hvem som havner i andelen, er et valg. Sato nevner et tilfeldig utvalg av brukerne, selskapets egne ansatte før resten av verden og brukere valgt ut etter profil eller demografi. Store systemer deler ofte langs skillelinjer de allerede har: én region, ett bestemt sted eller én merkevare først. Ifølge Sato brukte Facebook flere canaries, der den første bare var synlig for selskapets egne ansatte og hadde alle funksjonsflagg slått på, slik at problemer med nye funksjoner viste seg tidlig.

Forskjellen fra blue-green deployment ligger i hvordan trafikken flyttes. Blue-green bygger en komplett ekstra kopi og flytter all trafikk i ett skifte, mens den gamle kopien står klar som veien tilbake. En canary flytter trafikken gradvis, så to

versjoner i en periode betjener ekte brukere side om side. En rullerende utrulling har også to versjoner i drift samtidig, men den bytter ut instanser så fort de blir klare, uten en pause for å vurdere hvordan den nye versjonen klarer seg. En canary stopper ved hvert trinn og venter på et svar.

Bruk begrepet i en beslutning

Bruk «Hva er en canary release?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Testet, utgitt og mulig å reversere, slik at endringer når kundene en tirsdag ettermiddag i stedet for sent en fredagskveld.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Automatiserte GIS-SaaS-utrulling i GitHub Actions — produksjonssetting, data og rapportering, alt uten manuelle grep

Automatiserte GIS-SaaS-utrulling i GitHub Actions Ta kontakt →

Hvorfor det er viktig

Noen feil viser seg bare i produksjon: under ekte belastning, med ekte data og med den faktiske blandingen av klienter. Et stagingmiljø kan komme nær, men blir aldri identisk. En canary godtar det og begrenser heller skaden. Når hver tjuende forespørsel treffer den nye versjonen, går en dårlig utgivelse ut over hver tjuende forespørsel så lenge det tar å oppdage den, og løsningen er å sende den andelen tilbake.

Sato peker på en annen gevinst: kapasitetstesting. Når andelen skrus sakte opp, ser man hvordan den nye versjonen oppfører seg under økende produksjonslast, med en trygg vei tilbake, i stedet for å bygge et eget miljø bare for lasttester. Ikke noe testmiljø kommer nærmere produksjon enn produksjonen selv.

Haken er at en canary aldri er bedre enn signalet som vurderer den. Feil, svartider og forretningstallene som betyr noe, som registreringer, betalinger eller bestillinger, må være synlige per versjon og tidlig nok til å stoppe neste trinn. Uten det er en canary bare en langsom utrulling med ekstra venting. Sato forteller at IMVU var først ute med det som kalles et «cluster immune system»: man følger med på forretningstallene og ruller automatisk tilbake en utgivelse ved en statistisk signifikant regresjon.

Det er også løpende kostnader. To eller flere versjoner er i drift samtidig, og alt de deler (databasen, en cache, en kø, API-et klientene kaller), må fungere med begge så lenge det pågår. Sato råder til å holde antallet samtidige versjoner på et minimum. Teknikken passer også dårlig for programvare som er installert på

brukernes egne datamaskiner eller telefoner, der leverandøren ikke bestemmer når oppgraderingen skjer. Der foreslår han at backenden støtter begge versjonene, at man følger med på hvilke klientversjoner som fortsatt er i bruk, og at den gamle først fjernes når andelen dens har falt tilstrekkelig.

Slik ser det ut i praksis

I sin enkleste form er en canary på Kubernetes to Services, én som velger de stabile podene og én som velger canary-podene, pluss en rute som vekter trafikken mellom dem. Med Gateway API ligger vektene i en 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
```

Vekter er forholdstall, ikke prosenter: hver backend får vekten sin delt på summen av vektene i regelen, så 95 og 5 sender hver tjuende forespørsel til canary-versjonen. En utelatt vekt teller som 1, og en vekt på 0 sender ingenting. Gateway API-veiledningen om trafikkdeling viser også en regel som matcher en header i forespørselen, `traffic: test` i eksempelet, og bare sender de forespørslene til den nye versjonen, så testere kan nå den før noen kunde gjør det. Å flytte canary-versjonen fram er å endre to tall; å trekke den tilbake er å sette vekten dens til 0.

Argo Rollouts automatiserer trinnene. En `Rollout` trer i stedet for en `Deployment` med samme pod-mal og angir trinnene kontrolleren går gjennom hver gang malen endres:

```
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: {}
```

Uten en trafikkruter tilnærmer Argo vekten med antall pods: ti replikaer ved 10 prosent gir én canary-pod og ni stabile. Den første pausen varer en time; den andre varer til noen bestemmer seg. En utgivelse og de to mulige utfallene ser slik ut:

```
kubectl argo rollouts set image web web=registry.example.com/web:1.5.0
# ved pausen uten tidsgrense: enten videre ...
kubectl argo rollouts promote web
# ... eller stopp og sett den stabile versjonen foran alle igjen
kubectl argo rollouts abort web
```

Alt i det forløpet avhenger av at man under pausen kan se om canary-versjonen gjør det dårligere enn versjonen den erstatter. Det er arbeidet bak tjenesten Pålitelighet og overvåking, som handler om varsler som sier fra mens en feil fortsatt er liten. Den egenutviklede feilovervåkingen og sporingen fra 2025 viser hvordan et slikt signal ser ut: feil grupperes i gjentakende problemer i stedet for en flat liste, og topper og regresjoner oppdages med en avkjølingsperiode, slik at

én dårlig produksjonssetting ikke varsler noen førti ganger. Det er akkurat det et canary-trinn trenger: en regresjon som melder seg selv innenfor pausen, og ikke en som en bruker snubler over neste morgen.

Hvor det går galt

Canary-versjonen er for liten til å si noe. Én pod av ti i et stille tidsrom betjener kanskje noen hundre forespørsler i løpet av pausen, og en feil som rammer hver tusende forespørsel, viser seg aldri. Klokka er feil mål; antallet forespørsler canary-versjonen har betjent, er det riktige. Avstem andel og pause slik at en sjelden feil får sjansen til å dukke opp flere ganger, og hold et trinn lenger om natta enn midt på dagen.

Dashbordene kan ikke skille canary fra stabil. En canary-versjon med en feilrate som ligger to prosentpoeng over den stabile versjonens, legger ved en andel på 5 prosent bare omtrent én feil per tusen forespørsler til den samlede feilraten. På et dashbord med én feilrate for hele tjenesten er det usynlig. Gi canary-pods og stabile pods hver sin label, som Argo Rollouts legger til via `canaryMetadata` og `stableMetadata`, ta labelen med inn i målingene, og sammenlign canary-versjonen med den stabile i stedet for med en fast grense. En `analysis` som kjører i bakgrunnen på en Rollout, avbryter oppdateringen hvis den feiler.

Begge versjonene skriver til de samme dataene. Så lenge utrullingen pågår, leser og skriver gammel og ny kode i samme database, og med tilfeldig fordeling kan én brukers forespørsler treffe begge i løpet av ett minutt. En ny kolonne som den gamle versjonen aldri fyller ut, eller et format den ikke kan lese, blir til feil på den stabile siden, altså den delen som skulle være trygg. Satos svar er parallel change: databasen støtter begge versjonene av applikasjonen under hele utrullingen, og den gamle strukturen fjernes først når den gamle versjonen er borte.

En abort stopper trafikken, ikke endringen. `kubectl argo rollouts abort` sender trafikken tilbake til det stabile ReplicaSet, men spesifikasjonen i `Rollout`-objektet peker fortsatt på det nye imaget, og forlater objektet den avbrutte tilstanden, går det mot den versjonen igjen. Tilbakerullingen er først fullført når malen er satt tilbake til forrige versjon. Ligger spesifikasjonen i et repo, skal endringen gjøres der, slik at kilden og klyngen er enige om hva som skal kjøre.

Canary-versjonen brukes også som A/B-test. Rutingen er den samme, så det er fristende å måle en ny funksjons effekt på konverteringen med canary-trafikken. Sato råder til å holde de to fra hverandre: en canary jakter på problemer og bør være ferdig i løpet av minutter eller timer, mens et eksperiment tester en hypotese og kan trenge flere dager med data for å bli statistisk signifikant. Hvis canary-versjonen vurderes etter forretningstall, blir dommen dessuten forstyrret av den effekten eksperimentet skal framkalle. Gi ut med en canary, og kjør eksperimentet etterpå bak en egen oppdeling, for eksempel et funksjonsflagg, når versjonen er ute.

Les mer

- Canary Release — Danilo Satos beskrivelse fra 2014 på Martin Fowlers bliki: valg av brukere, kapasitetstesting, A/B-tester og databasen.
- Argo Rollouts: Canary — trinn og pauser, vekter tilnærmet med antall pods og skalering med en trafikkruter.
- HTTP traffic splitting — Gateway API-veiledningen om vektede backends og en canary-rute som matcher på en header.
- kubectl argo rollouts abort — hva en abort gjør med trafikken, og hva den etterlater i spesifikasjonen.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en canary release?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/canary-release/