Engineer

Canary release

at frigive en ændring til en lille del af brugerne først

Hvad det er

En canary release sender en ny version af en tjeneste ud til en lille del af den rigtige trafik først, mens den version, den skal afløse, bliver ved med at betjene alle andre. Holder den nye, vokser andelen trin for trin, indtil den bærer det hele, og den gamle version kan udfases. Holder den ikke, går trafikken tilbage til den gamle version, som aldrig holdt op med at køre, og kun den lille andel nåede at mærke fejlen. Danilo Sato beskrev teknikken på Martin Fowlers bliki i juni 2014 og nævner, at den også kaldes phased rollout eller incremental rollout. I Kubernetes-verdenen siger man oftere canary deployment, og på dansk ses canary-udrulning også. Argo Rollouts, en Kubernetes-controller, der tilføjer udrulningsstrategier af den slags, kalder indstillingen `canary`.

Navnet kommer fra kulminerne. Minearbejderne tog en kanariefugl i et bur med ned, fordi udsivende gas ville påvirke fuglen, før den påvirkede menneskene, og fuglens tegn på ubehag var deres varsel om at komme ud. Den lille andel trafik spiller samme rolle: den møder problemet først, mens der endnu er tid til at handle.

Hvem der havner i andelen, er et valg. Sato nævner en tilfældig stikprøve af brugerne, virksomhedens egne medarbejdere før resten af verden og brugere udvalgt efter profil eller demografi. Store systemer deler ofte langs skillelinjer, de allerede har: én region, ét bestemt sted eller ét brand først. Ifølge Sato brugte Facebook flere canaries, hvor den første kun var synlig for virksomhedens egne medarbejdere og havde alle feature flags slået til, så problemer med nye funktioner viste sig tidligt.

Forskellen fra blue-green deployment ligger i, hvordan trafikken flyttes. Blue-green bygger en komplet ekstra kopi og flytter al trafik i ét skift, mens den gamle kopi står klar som vejen tilbage. En canary flytter trafikken gradvist, så to

versioner i en periode betjener rigtige brugere side om side. En rullende opdatering har også to versioner i drift på samme tid, men den skifter instanser ud, så hurtigt de bliver klar, uden en pause til at vurdere, hvordan den nye version klarer sig. En canary stopper ved hvert trin og venter på et svar.

Brug begrebet i en beslutning

Brug »Hvad er en canary release?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Testet, udgivet og til at rulle tilbage, så ændringer når kunderne en tirsdag eftermiddag i stedet for sent en fredag aften.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har automatiseret GIS-SaaS-udrulning med CI/CD — udrulning, data og rapportering, alt sammen uden manuelt arbejde

Har automatiseret GIS-SaaS-udrulning med CI/CD Kontakt os →

Hvorfor det betyder noget

Nogle fejl viser sig kun i produktion: under rigtig belastning, med rigtige data og med den rigtige blanding af klienter. Et stagingmiljø kan komme tæt på, men bliver aldrig identisk. En canary accepterer det og begrænser i stedet skaden. Når hver tyvende forespørgsel rammer den nye version, går en dårlig release ud over hver tyvende forespørgsel, så længe det tager at opdage den, og løsningen er at sende den andel tilbage.

Sato peger på en anden gevinst: kapacitetstest. Når andelen skrues langsomt op, kan man måle, hvordan den nye version påvirker produktionsmiljøet under stigende belastning, med en sikker vej tilbage, i stedet for at bygge et separat miljø alene til belastningstest. Intet testmiljø kommer tættere på produktion end produktion selv.

Hagen er, at en canary aldrig er bedre end det signal, der bedømmer den. Fejl, svartider og de forretningstal, der betyder noget, som tilmeldinger, betalinger eller ordrer, skal kunne ses pr. version og tidligt nok til at standse næste trin. Uden det er en canary bare en langsom udrulning med ekstra ventetid. Sato fortæller, at IMVU var pionerer inden for det, der kaldes et »cluster immune system«: man følger forretningstallene og ruller automatisk en release tilbage ved en statistisk signifikant regression.

Der er også løbende omkostninger. To eller flere versioner er i drift samtidig, og alt, hvad de deler (databasen, en cache, en kø, det API, klienterne kalder), skal fungere med dem begge, så længe det står på. Sato råder til at holde antallet af

samtidige versioner nede på et minimum. Teknikken passer også dårligt til software, der er installeret på brugernes egne computere eller telefoner, hvor leverandøren ikke bestemmer, hvornår opgraderingen sker. Dér foreslår han, at backenden understøtter begge versioner, at man holder øje med, hvilke klientversioner der stadig er i brug, og at den gamle først fjernes, når dens andel er faldet tilstrækkeligt.

Sådan ser det ud i praksis

I sin enkleste form er en canary på Kubernetes to Services, én, der vælger de stabile pods, og én, der vælger canary-podsene, plus en rute, der vægter trafikken mellem dem. Med Gateway API ligger vægtene 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
```

Vægte er forholdstal, ikke procenter: hver backend får sin vægt divideret med summen af vægtene i reglen, så 95 og 5 sender hver tyvende forespørgsel til canary-versionen. En udeladt vægt tæller som 1, og en vægt på 0 sender ingenting. Gateway API's vejledning i trafikopdeling viser desuden en regel, der matcher en request-header, `traffic: test` i eksemplet, og kun sender de forespørgsler til den nye version, så testere kan nå den, før nogen kunde gør. At rykke canary-versionen frem er at rette to tal; at trække den tilbage er at sætte dens vægt til 0.

Argo Rollouts automatiserer trinnene. En `Rollout` træder i stedet for en `Deployment` med den samme pod-template og angiver de trin, controlleren går igennem, hver gang templaten ændres:

```
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: {}
```

Uden en trafikrouter tilnærmer Argo vægten med antallet af pods: ti replikaer ved 10 procent giver én canary-pod og ni stabile. Den første pause varer en time; den anden varer, indtil nogen beslutter sig. En release og dens to mulige udfald ser sådan ud:

```
kubectl argo rollouts set image web web=registry.example.com/web:1.5.0
# ved den åbne pause: enten videre ...
kubectl argo rollouts promote web
# ... eller stop og sæt den stabile version foran alle igen
kubectl argo rollouts abort web
```

Alt i det forløb afhænger af, at man under pausen kan se, om canary-versionen klarer sig dårligere end den, den afløser. Det er arbejdet bag ydelsen Site reliability & overvågning, der handler om at få besked, mens en fejl stadig er lille. Den hjemmebyggede fejlovervågning og tracing fra 2025 viser, hvordan sådan et signal ser ud: fejl samles i tilbagevendende problemer frem for en flad liste, og spikes og regressioner opdages med en cooldown, så én dårlig udrulning ikke får

nogen kaldt ud fyrre gange. Det er præcis, hvad et canary-trin har brug for: en regression, der melder sig selv inden for pausen, og ikke én, en bruger falder over næste morgen.

Hvor det går galt

Canary-versionen er for lille til at sige noget. Én pod ud af ti i et stille tidsrum betjener måske nogle hundrede forespørgsler i løbet af sin pause, og en fejl, der rammer hver tusinde forespørgsel, viser sig aldrig. Uret er det forkerte mål; antallet af forespørgsler, canary-versionen har betjent, er det rigtige. Afstem andel og pause, så en sjælden fejl får chancen for at dukke op flere gange, og hold et trin længere om natten end ved middagstid.

Dashboardene kan ikke skelne canary fra stabil. En canary-version, hvis fejlrate ligger to procentpoint over den stabile versions, lægger ved en andel på 5 procent kun cirka én fejl pr. tusind forespørgsler oven i den samlede fejlrate. På et dashboard med én fejlrate for hele tjenesten er det usynligt. Giv canary-pods og stabile pods hver sin label, som Argo Rollouts tilføjer via `canaryMetadata` og `stableMetadata`, tag labelen med over i målingerne, og sammenlign canary-versionen med den stabile i stedet for med en fast grænse. En `analysis`, der kører i baggrunden på en Rollout, afbryder opdateringen, hvis den fejler.

Begge versioner skriver til de samme data. Så længe udrulningen står på, læser og skriver gammel og ny kode i den samme database, og med tilfældig fordeling kan én brugers forespørgsler ramme begge inden for et minut. En ny kolonne, som den gamle version aldrig udfylder, eller et format, den ikke kan læse, bliver til fejl på den stabile side, altså den del, der skulle være sikker. Satos svar er parallel change: databasen understøtter begge versioner af applikationen under hele udrulningen, og den gamle struktur fjernes først, når den gamle version er væk.

En abort stopper trafikken, ikke ændringen. `kubectl argo rollouts abort` sender trafikken tilbage til det stabile ReplicaSet, men specifikationen i `Rollout`-objektet peger stadig på det nye image, og hvis objektet forlader den afbrudte tilstand, går det efter den version igen. Tilbagerulningen er først fuldført, når templaten er sat tilbage til den forrige version. Ligger specifikationen i et repository, skal rettelsen ske dér, så kilden og clusteret er enige om, hvad der skal køre.

Canary-versionen bliver også brugt som A/B-test. Routingen er den samme, så det er fristende at måle en ny funktions effekt på konverteringen med canary-trafikken. Sato råder til at holde de to adskilt: en canary jager problemer og bør være færdig på minutter eller timer, mens et eksperiment tester en hypotese og kan kræve flere dages data for at blive statistisk signifikant. Bedømmes canary-versionen på forretningstal, slører den effekt, eksperimentet skal fremkalde, også dommen. Udgiv med en canary, og kør derefter eksperimentet bag en opdeling for sig, for eksempel et feature flag, når versionen er ude.

Læs mere

- Canary Release — Danilo Satos beskrivelse fra 2014 på Martin Fowlers bliki: valg af brugere, kapacitetstest, A/B-test og databasen.
- Argo Rollouts: Canary — trin og pauser, vægte tilnærmet med antallet af pods og skalering med en trafikrouter.
- HTTP traffic splitting — Gateway API's vejledning i vægtede backends og en canary-rute, der matcher på en header.
- kubectl argo rollouts abort — hvad en abort gør ved trafikken, og hvad den efterlader i specifikationen.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er en canary release?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/canary-release/