Engineer

Canary release

att släppa en ändring till en liten andel användare först

Vad det är

En canary release skickar ut en ny version av en tjänst till en liten del av den riktiga trafiken först, medan versionen den ska ersätta fortsätter att betjäna alla andra. Håller den nya versionen, ökas andelen steg för steg tills den bär allt, och den gamla versionen kan fasas ut. Håller den inte, går trafiken tillbaka till den gamla versionen, som aldrig slutade köra, och bara den lilla andelen hann märka felet. Danilo Sato beskrev tekniken på Martin Fowlers bliki i juni 2014 och nämner att den också kallas phased rollout eller incremental rollout. I Kubernetesvärlden säger man oftare canary deployment, och på svenska förekommer också canary-driftsättning. Argo Rollouts, en Kubernetes-controller som lägger till driftsättningsstrategier av det här slaget, kallar inställningen `canary`.

Namnet kommer från kolgruvorna. Gruvarbetarna tog med sig en kanariefågel i bur ner i gruvan, eftersom gas som läckte in skulle påverka fågeln innan den påverkade människorna, och fågelns tecken på obehag var deras varning att ta sig ut. Den lilla andelen trafik spelar samma roll: den möter problemet först, medan det fortfarande finns tid att agera.

Vilka som hamnar i andelen är ett val. Sato nämner ett slumpmässigt urval av användarna, företagets egna anställda före resten av världen och användare utvalda efter profil eller demografi. Stora system delar ofta längs skiljelinjer de redan har: en region, en viss plats eller ett varumärke först. Enligt Sato använde Facebook flera canaries, där den första bara var synlig för företagets egna anställda och hade alla funktionsflaggor påslagna, så att problem med nya funktioner visade sig tidigt.

Skillnaden mot blå-grön driftsättning ligger i hur trafiken flyttas. Vid blå-grön driftsättning byggs en komplett extra kopia och all trafik flyttas i en enda växling, medan den gamla kopian står kvar som vägen tillbaka. En canary flyttar trafiken

gradvis, så att två versioner under en period betjänar riktiga användare sida vid sida. En rullande driftsättning har också två versioner i drift samtidigt, men den byter ut instanser så fort de blir klara, utan någon paus för att bedöma hur den nya versionen klarar sig. En canary stannar vid varje steg och väntar på ett svar.

Använd begreppet i ett beslut

Använd ”Vad är en canary release?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Testade, släppta och möjliga att backa, så att ändringar når kunderna en tisdagseftermiddag i stället för sent en fredagskväll.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Automatiserade GIS-driftsättning i GitHub Actions — driftsättning, data och rapportering, helt utan manuellt arbete

Automatiserade GIS-driftsättning i GitHub Actions Hör av er →

Varför det spelar roll

Vissa fel visar sig bara i produktion: under verklig belastning, med verkliga data och med den faktiska blandningen av klienter. En stagingmiljö kan komma nära men blir aldrig identisk. En canary accepterar det och begränsar i stället skadan. När vart tjugonde anrop når den nya versionen, drabbar en dålig release vart tjugonde anrop så länge det tar att upptäcka den, och lösningen är att skicka tillbaka den andelen.

Sato pekar på en annan vinst: kapacitetstestning. När andelen skruvas upp långsamt, ser man hur den nya versionen beter sig under växande produktionslast, med en säker väg tillbaka, i stället för att bygga en separat miljö bara för lasttester. Ingen testmiljö kommer närmare produktion än produktionen själv.

Haken är att en canary aldrig är bättre än signalen som bedömer den. Fel, svarstider och de affärssiffror som spelar roll, som registreringar, betalningar eller beställningar, måste synas per version och tidigt nog för att stoppa nästa steg. Utan det är en canary bara en långsam utrullning med extra väntan. Sato berättar att IMVU var pionjärer med det som kallas ett ”cluster immune system”: man följer affärssiffrorna och rullar automatiskt tillbaka en release vid en statistiskt signifikant regression.

Det finns också löpande kostnader. Två eller fler versioner är i drift samtidigt, och allt de delar (databasen, en cache, en kö, API:t som klienterna anropar) måste fungera med båda så länge det pågår. Sato rekommenderar att antalet samtidiga

versioner hålls till ett minimum. Tekniken passar också dåligt för programvara som är installerad på användarnas egna datorer eller telefoner, där leverantören inte bestämmer när uppgraderingen sker. Där föreslår han att backenden stöder båda versionerna, att man håller koll på vilka klientversioner som fortfarande används och att den gamla tas bort först när dess andel har sjunkit tillräckligt.

Så ser det ut i praktiken

I sin enklaste form är en canary på Kubernetes två Services, en som väljer de stabila poddarna och en som väljer canary-poddarna, plus en rutt som viktar trafiken mellan dem. Med Gateway API ligger vikterna 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
```

Vikter är proportioner, inte procent: varje backend får sin vikt delad med summan av vikterna i regeln, så 95 och 5 skickar vart tjugonde anrop till canary-versionen. En utelämnad vikt räknas som 1, och en vikt på 0 skickar ingenting. Gateway API:s guide till trafikdelning visar också en regel som matchar en header i anropet, `traffic: test` i exemplet, och bara skickar de anropen till den nya versionen, så att testare kan nå den innan någon kund gör det. Att flytta fram canary-versionen är att ändra två siffror; att dra tillbaka den är att sätta dess vikt till 0.

Argo Rollouts automatiserar stegen. En `Rollout` ersätter en `Deployment` med samma poddmall och anger de steg som controllern går igenom varje gång mallen ändras:

```
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: {}
```

Utan en trafikrouter approximerar Argo vikten med antalet poddar: tio repliker vid 10 procent ger en canary-podd och nio stabila. Den första pausen varar en timme; den andra varar tills någon bestämmer sig. En release och dess två möjliga utfall ser ut så här:

```
kubectl argo rollouts set image web web=registry.example.com/web:1.5.0
# vid pausen utan tidsgräns: antingen vidare ...
kubectl argo rollouts promote web
# ... eller stoppa och låt den stabila versionen ta all trafik igen
kubectl argo rollouts abort web
```

Allt i det förloppet hänger på att man under pausen kan se om canary-versionen klarar sig sämre än versionen den ersätter. Det är arbetet bakom tjänsten Tillförlitlig drift och övervakning, som handlar om larm som säger till medan ett fel fortfarande är litet. Den egenutvecklade felövervakningen och spårningen från 2025 visar hur en sådan signal ser ut: fel grupperas till återkommande problem i stället för en platt lista, och toppar och regressioner upptäcks med en

karenstid, så att en dålig driftsättning inte larmar någon fyrtio gånger. Det är precis vad ett canary-steg behöver: en regression som gör sig till känna inom pausen, och inte en som en användare snubblar över nästa morgon.

Var det går fel

Canary-versionen är för liten för att säga något. En podd av tio under en lugn timme betjänar kanske några hundra anrop under sin paus, och ett fel som drabbar vart tusende anrop syns aldrig. Klockan är fel mått; antalet anrop som canary-versionen har betjänat är det rätta. Anpassa andel och paus tillsammans, så att ett sällsynt fel får chansen att dyka upp flera gånger, och håll ett steg längre på natten än mitt på dagen.

Ingen dashboard skiljer canary från stabil. En canary-version vars felfrekvens ligger två procentenheter över den stabila versionens lägger, vid en andel på 5 procent, bara till ungefär ett fel per tusen anrop i den samlade felfrekvensen. På en dashboard med en enda felfrekvens för hela tjänsten är det osynligt. Ge canary-poddar och stabila poddar var sin label, som Argo Rollouts lägger till via `canaryMetadata` och `stableMetadata`, ta med labeln in i mätningarna och jämför canary-versionen med den stabila i stället för med en fast gräns. En `analysis` som körs i bakgrunden på en Rollout avbryter uppdateringen om den misslyckas.

Båda versionerna skriver till samma data. Så länge utrullningen pågår, läser och skriver gammal och ny kod i samma databas, och med slumpmässig fördelning kan en användares anrop hamna på båda inom en minut. En ny kolumn som den gamla versionen aldrig fyller i, eller ett format den inte kan läsa, blir fel på den stabila sidan, alltså den del som skulle vara säker. Satos svar är parallel change: databasen stöder båda versionerna av applikationen under hela utrullningen, och den gamla strukturen tas bort först när den gamla versionen är borta.

En abort stoppar trafiken, inte ändringen. `kubectl argo rollouts abort` skickar trafiken tillbaka till det stabila ReplicaSet, men specifikationen i `Rollout`-objektet pekar fortfarande på den nya avbildningen, och om objektet lämnar det avbrutna tillståndet går det mot den versionen igen. Återställningen är inte klar förrän mallen har satts tillbaka till föregående version. Ligger specifikationen i ett repo, ska ändringen göras där, så att källan och klustret är överens om vad som ska köras.

Canary-versionen får också tjäna som A/B-test. Routningen är densamma, så det är frestande att mäta en ny funktions effekt på konverteringen med canary-trafiken. Sato råder till att hålla isär de två: en canary letar efter problem och bör vara klar inom minuter eller timmar, medan ett experiment prövar en hypotes och kan behöva flera dagars data för att bli statistiskt signifikant. Om canary-versionen bedöms efter affärssiffror, grumlas domen dessutom av den effekt som experimentet ska framkalla. Släpp med en canary, och kör sedan experimentet bakom en egen uppdelning, till exempel en funktionsflagga, när versionen är ute.

Läs mer

- Canary Release — Danilo Satos beskrivning från 2014 på Martin Fowlers bliki: val av användare, kapacitetstestning, A/B-tester och databasen.
- Argo Rollouts: Canary — steg och pauser, vikter som approximeras med antalet poddar och skalning med en trafikrouter.
- HTTP traffic splitting — Gateway API:s guide till viktade backends och en canary-rutt som matchar på en header.
- kubectl argo rollouts abort — vad en abort gör med trafiken och vad den lämnar kvar i specifikationen.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är en canary release?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/canary-release/