Engineer

Canary release

releasing a change to a small share of users first

What it is

A canary release puts a new version of a service in front of a small share of its real traffic first, while the version it replaces keeps serving everyone else. If the newcomer holds up, the share grows in steps until it carries everything and the old version is retired. If it does not, the traffic goes back to the old version, which never stopped running, and only the small share ever saw the fault. Danilo Sato described the technique on Martin Fowler's bliki in June 2014 and noted that it also goes by phased rollout or incremental rollout. Kubernetes tooling tends to say canary deployment: Argo Rollouts, a Kubernetes controller that adds release strategies of this kind, names the setting `canary`.

The name comes from coal mining. Miners took a canary below ground in a cage, because leaking gas would overcome the bird before it reached the people, and the bird's distress was their warning to get out. The small share of traffic plays the same part: it meets the trouble first, while there is still time to act on it.

Who lands in that share is a choice. Sato lists a random sample of users, a company's own staff before the public, and users picked by profile or demographics. Large systems often split along lines they already have instead: one region, one location or one brand first. Facebook, in Sato's account, used several canaries, the first visible only to its own employees and with every feature flag turned on, so that problems with new features showed up early.

The contrast with blue-green deployment is the shape of the move. Blue-green builds a complete second copy and moves all traffic in one switch, keeping the old copy as the way back. A canary moves traffic gradually, so for a while two versions serve real users side by side. A rolling update also has two versions live

at once, but it replaces instances as fast as they become ready, with no pause to judge how the new version is doing. A canary stops at each step and waits for an answer.

Use the concept in a decision

Put “What is a canary release?” to work in a real decision. We can help relate the concept to the system you are building or buying. Tested, released and reversible, so changes reach customers on a Tuesday afternoon rather than late on a Friday night.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Automated GIS SaaS deployment with GitHub Actions — deployment, data and reporting, all off the hands

Automated GIS SaaS deployment with GitHub Actions Get in touch →

Why it matters

Some faults only show up in production: under real load, with real data, behind the real mix of clients. A staging environment can come close but is never identical. A canary accepts that and limits the damage instead. If one request in twenty reaches the new version, a bad release hurts one request in twenty for as long as it takes to notice, and the fix is to send that share back.

Sato points to a second use: capacity testing. Ramping the share up slowly shows how the new version behaves under growing production load, with a safe way back, instead of building a separate environment just for load tests, which can never be quite as production-like as production itself.

The catch is that a canary is only as good as the signal that judges it. Errors, latency and the business numbers that matter, such as sign-ups, payments or orders, have to be visible per version and early enough to stop the next step. Without that, a canary is a slow rollout with extra waiting. Sato notes that IMVU pioneered what is known as a “cluster immune system”: watching business metrics and rolling a release back automatically on a statistically significant regression.

There are running costs too. Two or more versions are in service at once, and everything they share, the database, a cache, a queue, the API that clients call, has to work with both for the duration. Sato advises keeping the number of versions live at the same time to a minimum. The technique also fits poorly with software installed on users’ own computers or phones, where the vendor does

not decide when an upgrade happens. There, he suggests keeping the backend able to serve both versions, watching which client versions are still in use, and dropping the old one only when its share has fallen far enough.

What it looks like in practice

At its plainest, a canary on Kubernetes is two Services, one selecting the stable pods and one selecting the canary pods, and a route that weights traffic between them. With the Gateway API, the weights live in an 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
```

Weights are proportions, not percentages: each backend gets its weight divided by the sum of the weights in that rule, so 95 and 5 send one request in twenty to the canary. An omitted weight counts as 1, and a weight of 0 sends nothing. The Gateway API guide to traffic splitting also shows a rule that matches a request header, `traffic: test` in its example, and sends only those requests to the new version, so testers can reach it before any customer does. Moving the canary forward is editing two numbers; pulling it back is setting its weight to 0.

Argo Rollouts automates the steps. A `Rollout` takes the place of a `Deployment`, with the same pod template, and lists the steps the controller walks through whenever that template changes:

```
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: {}
```

Without a traffic router, Argo approximates the weight with pod counts: ten replicas at 10 per cent means one canary pod and nine stable ones. The first pause waits an hour; the second waits until someone decides. A release and its two possible endings look like this:

```
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
```

Everything in that sequence hinges on knowing, during the pause, whether the canary is doing worse than what it replaced. That is the work behind the Site Reliability & Monitoring service, which is built around alerts that speak up while a fault is still small. The first-party error monitoring and tracing, from 2025, shows what such a signal looks like: errors are grouped into recurring problems rather than listed one by one, and spikes and regressions are detected with a cooldown,

so that one bad deploy does not page anyone forty times. A canary step needs exactly that: a regression that makes itself known inside the pause, not one a user stumbles on the next morning.

Where it goes wrong

The canary is too small to say anything. One pod in ten at a quiet hour may serve a few hundred requests during its pause, and a fault that hits one request in a thousand never shows. The clock is the wrong measure; the number of requests the canary has served is the right one. Size the share and the pause together, so that a rare error has the chance to appear several times, and hold a step longer at night than at noon.

The dashboards cannot tell canary from stable. A canary that fails one request in fifty more than the stable version does, at a 5 per cent share, adds about one failed request in a thousand to the total. On a dashboard with one error rate for the whole service, that is invisible. Give canary and stable pods different labels, which Argo Rollouts adds through `canaryMetadata` and `stableMetadata`, carry the label into the metrics, and compare the canary with the stable version rather than with a fixed threshold. A background `analysis` on the Rollout aborts the update when it fails.

Both versions write to the same data. For the length of the rollout, old and new code read and write the same database, and with random routing a single user's requests may land on both within a minute. A new column the old version never fills, or a format it cannot read, turns into errors on the stable side, the part that was supposed to be safe. Sato's answer is parallel change: the database supports both versions of the application for the whole rollout, and the old shape goes only once the old version is gone.

An abort stops the traffic, not the change. `kubectl argo rollouts abort` moves traffic back to the stable ReplicaSet, but the Rollout's spec still names the new image, and resuming the Rollout makes that image the target once more. The rollback is complete only when the template is set back to the previous version. If the spec is kept in a repository, make the change there, so that the source and the cluster agree on what should be running.

The canary doubles as an A/B test. The routing looks the same, so it is tempting to measure a new feature's effect on conversions with the canary's traffic. Sato advises keeping the two apart: a canary hunts for problems and should finish

within minutes or hours, while an experiment tests a hypothesis and may need days of data to be statistically significant. If the canary is judged on business metrics, the effect the experiment is meant to cause also muddies the verdict. Release with the canary, then run the experiment behind its own split, such as a feature flag, once the version is out.

Further reading

- Canary Release — Danilo Sato's 2014 description on Martin Fowler's bliki: choosing users, capacity testing, A/B tests and the database.
- Argo Rollouts: Canary — steps and pauses, weights approximated by pod counts, and scaling with a traffic router.
- HTTP traffic splitting — the Gateway API guide to weighted backends and a header-matched canary route.
- kubectl argo rollouts abort — what an abort does to traffic, and what it leaves in the spec.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is a canary release?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

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

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/canary-release/