To Canary or Not to Canary?

This lesson discusses the features of canary deployments and if they fulfill our needs or not.

We'll cover the following

We saw one possible implementation of canary deployments with **Flagger**, **Istio**, and **Prometheus**. As we discussed, we could have used other tools. We could have just as well created more complex formulas that would be used to decide whether to proceed or roll back a release. Now the time has come to evaluate canary deployments and see how they fit the requirements we defined initially.

Do canary deployments fulfill our needs?

> Canary deployments are, in a way, *an extension of rolling updates*. As such, all of the requirements fulfilled by one are fulfilled by the other.

High-availability

The deployments we did using the canary strategy were highly available. The process itself could be qualified as rolling updates on steroids. New releases are initially being rolled out to a fraction of users and, over time, we were increasing the reach of the new release until all requests were going there. From that perspective, there was no substantial difference between rolling updates and canary releases. However, the process behind the scenes was different.

While rolling updates are increasing the number of Pods of the new release and decreasing those from the old, canaries are leaving the old release untouched. Instead, a new Pod (or a set of Pods) is spun up, and service mesh (e.g., **Istio**) is making sure to redirect some requests to the new and others to the old release. While rolling updates are becoming available to users by changing how many Pods of the new release are available, canaries are accomplishing the same result through networking. As such, canaries allow us to have much greater control over the process.

Given that with canary deployments we are in control over who sees what, we can fine-tune it to meet our needs better. We are less relying on chance and more on instructions we're giving to **Flagger**. That allows us to create more complicated calculations. For example, we could let only people from one country see the new release, check their reactions, and decide whether to proceed with the rollout to everyone else.

All in all, with canary deployments, we have as much high-availability as with rolling updates.

Responsiveness

Given that our applications are running in multiple replicas and that we can just as easily use HorizontalPodAutoscaler or any other Kubernetes resource type, canary deployments also make our applications as responsive as rolling updates.

Progressive rollout

Where canary deployments genuinely shine and make a huge difference is in the progressive rollout. While, as we already mentioned, rolling updates give us that feature as well, the additional control of the process makes canary deployments true progressive rollout.

Rollback

On top of all that, canary deployments were the only ones that had a built-in mechanism to roll back. While we could extend our pipelines to run tests during and after the deployment and roll back if they fail, the synergy provided by canary deployments is genuinely stunning. The process itself decides whether to roll forward, to temporarily halt the process, or to roll back. Such tight integration provides benefits that would require considerable effort to implement with the other deployment strategies. I cannot say that only canary deployments allow us to

roll back automatically. But, it is the only deployment strategy that we explored

that has rollbacks integrated into the process. And that should come as no surprise given that canary deployments rely on using a defined set of rules to decide when to progress with rollouts. It would be strange if the opposite is true. If a machine can decide when to move forward, it can just as well decide when to move back.

Cost-effectiveness

Judging by our requirements, the only area where the `Canary` deployment strategy is not as good or better than any other is cost-effectiveness. If we want to save on resources, serverless deployments are in most cases the best option. Even rolling updates are cheaper than canaries. With rolling updates, we replace one Pod at the time (unless we specify differently). However, with `Canary` deployments, we keep running the full old release throughout the whole process, and add the new one in parallel. In that aspect, canaries are similar to blue-green deployments, except that we do not need to duplicate everything. Running a fraction (e.g., one) of the Pods with the new release should be enough.

All in all, canaries are expensive or, at least, more expensive than other strategies, excluding blue-green that we already discarded.

Conclusion

All in all, canary deployments, at least the version we used, provide **high-availability**. They are **responsive**, and they give us **progressive rollout** and **automatic rollbacks**. The major downside is that they are **not** as **cost-effective** as some other deployment strategies.

The summary of the fulfillment of our requirements for the `Recreate` deployment strategy is as follows.

Requirement	Fullfilled
High-availability	Fully
Responsiveness	Fully
Progressive rollout	Fully

Rollback	Fully
Cost-effectiveness	Not

Right now, we are left with the last and potentially the most important discussion; *“Which deployment strategy should we choose?”*. Let’s discuss this in the next lesson.