Which Deployment Strategy Should We Choose?

This lesson describes the use cases for each of the deployment strategies and also lists the rules to follow when choosing a deployment strategy.

We'll cover the following

We saw some of the deployment strategies. There are others, and this chapter does not exclude you from exploring them. Please do that. The more you learn, the more educated decisions you'll make. Still, until you figure out all the other strategies and variations you can do, we accumulated enough material to summarize what we learned so far.

Can we conclude that *canary deployments are the best and that everyone should use them for all their applications?* **Certainly not.** To begin with, if an application is not eligible for rolling updates (e.g., a single replica app), it is almost certainly not suitable for canary deployments. If we think of canaries as extensions of rolling updates, if an app is not a good candidate for the latter, it will also not be fit for the former.

Use case for `Recreate` strategy

In other words, there is a good use case for using the `Recreate` strategy in some

cases, specifically for those applications that cannot (or shouldn't) use one of the other strategies.

*So, if both `Canary` and `Recreate` strategies have their use cases, can we discard **rolling updates** and **serverless**?*

Use case for rolling updates strategy

Rolling updates should, in most cases, be replaced with canary deployments. The applications eligible for one deployment strategy qualify for the other, and canaries provide so much more. If nothing else, they give us a bigger safety net. The exceptions would be tiny clusters serving small teams. In those cases, the resource overhead added by a service mesh (e.g., **Istio**) and metrics collector and database (e.g., **Prometheus**) might be too much. Another advantage of rolling updates is simplicity. There are no additional components to install and manage, and there are no additional `YAML` files. Long story short, canary deployments could easily replace all your rolling updates, as long as the cost (on resources and operations) is not too high for your use case.

Use case for serverless strategy

That leaves us with serverless (e.g., **Knative**). It would be hard for me to find a situation in which there is no use for serverless deployments. It has fantastic scaling capabilities on its own that can be combined with HorizontalPodAutoscaler. It saves us money by shutting (almost) everything down when our applications are not in use.

Knative ticks all the boxes, and the only downside is the deployment process itself, which is more elaborated with canaries. The more important potential drawback is that scaling from nothing to something can introduce a delay from the user's perspective. Nevertheless, that is rarely a problem. If an application is unused for an extended period, users rarely complain when they need to wait for an additional few seconds for the app to wake up.

Canary vs Serverless

So, we are in a situation where one solution (`Canary`) provides better capabilities for the deployment process itself, while the other (serverless) might be a better choice as a model for running applications. Ultimately, you'll need to make a choice.

What matters more? Is it **operational cost (use serverless)** or **deployment safety net (use canary)**?
You might be able to combine the two but, at the time of this writing (August 2019), that is not that easy since the integration is not available in **Flagger** or other similar tools.

What is essential, though, is that it is not the winner-takes-all type of a decision. We can use `Recreate` with some applications, `RollingUpdate` with others, and so on. But it goes beyond choosing a single deployment strategy for a single application. Deployment types can differ from one environment to the other. The canaries, for example, are not a good choice for preview environments. All we'd get is increased time required to terminate the deployment process and potential failures due to the lack of metrics.

Rules for choosing the deployment strategy

Let's make a quick set of rules when to use one deployment strategy over the other. Bear in mind that what follows is not a comprehensive list but rather an elevator pitch for each deployment type.

Recreate strategy

Use the **recreate** strategy when working with legacy applications that often do not scale, that are stateful without replication, and are lacking other features that make them not cloud-native.

Rolling update strategy

Use the **rolling update** strategy with cloud-native applications which, for one reason or another, cannot use canary deployments.

Canary strategy

Use the **canary** strategy instead of **rolling update** when you need the additional control when to roll forward and when to roll back.

Serverless strategy

Use **serverless** deployments in permanent environments when you need excellent scaling capabilities or when an application is not in constant use.

For preview environments

Finally, use **serverless** for all the deployments to preview environments, no matter which strategy you're using in staging and production.

>  Finally, remember that your Kubernetes cluster might not support all those choices, so choose among those that you can use.

Let's wrap up the discussion regarding deployment strategies and remove any used resources in the next lesson.