Which Types of Applications Should Run As Serverless?

This lesson informs us about the characteristics of an application which make it suitable for serverless computing.

We'll cover the following

- The size and fast boot-up of the application
- Request traffic for the application
- Deployment to different environments
  - Temporary environments
  - Permanent environment

Initially, the idea was to have only *functions running as serverless loads*. Those would be single-purpose pieces of code that contain only a small number of lines of code. A typical example of a serverless application would be an image processing function that responds to a single request and can run for a limited period. Implementations of serverless computing impose restrictions like:

- The **size** of applications (functions).
- Their maximum **duration** in cloud providers.

But, if we adopt Kubernetes as the platform to run serverless deployments, those restrictions may not be valid anymore.

> **We can say that any application that can be packaged into a container image can run as a serverless deployment in Kubernetes.**

The size and fast boot-up of the application

That, however, does not mean that any container is as good a candidate as any other. The smaller the application, and the faster its boot-up time, the better the candidate for serverless deployments.

However, things are not as straightforward as they may seem. Not being a good candidate does not mean that one should not compete at all. **Knative**, like many other serverless frameworks, does allow us to fine-tune configurations. We can, for example, specify with **Knative** that there should never be less than one replica of an application. That would solve the problem of slow boot-up while still maintaining some of the benefits of serverless deployments. In such a case, there would always be at least one replica to handle requests while we would benefit from having the elasticity of serverless providers.

Request traffic for the application

The size and the boot-up time are not the only criteria we can use to decide whether an application should be serverless. We might want to consider **traffic** as well. If, for example, our app has high traffic and it receives requests throughout the entire day, we might never need to scale it down to zero replicas. Similarly, our application might not be designed in a way where a different replica processes every request. After all, most of the apps can handle a vast number of requests by a single replica. In such cases, serverless computing implemented by cloud vendors and based on function-as-a-service might not be the right choice. But, as we already discussed, there are other serverless platforms, and those based on Kubernetes do not follow those rules. Since we can run any container as a serverless, any type of application can be deployed as such, which means that a single replica can handle as many requests as the design of the app allows. Also, **Knative** and other platforms can be configured to have a minimum number of replicas so that they might be well suited even for the applications with a mostly constant flow of traffic since every application will need scaling sooner or later. The only way to avoid that need is to overprovision applications and give them as much memory and CPU as their peak loads require.

All in all, if it can run in a container, it can be converted into a serverless deployment, as long as we understand that smaller applications with faster boot-up times are better candidates than others. However, boot-up time is not the only rule, nor is it the most important. If there is a consideration we should follow when deciding whether to run an application as serverless, it is related to the state or, the lack thereof. If an application is stateless, it might be the right candidate for serverless computing.

Now, let us imagine that you have an application that is not the right candidate to be serverless. **Does that mean that we cannot reap any benefits from frameworks like Knative**? We can because there is still the question of deployments to different environments.

Deployment to different environments

Typically, we have **permanent** and **temporary** environments. The examples of the former would be staging and production. If we don't want our application to be serverless in production, we will probably not want it to be any different in staging. Otherwise, the behavior would be different, and we could not say that we tested precisely the same behavior as the one we expect to run in production. So, in most cases, if an application should not be serverless in production, it should not be serverless in any other permanent environment. But, that doesn't mean that it shouldn't be serverless in temporary environments.

Temporary environments

Let's take an environment in which we deploy an application as a result of making a pull request as an example. It would be a temporary environment since we'd remove it the moment that the pull request is closed. Its time span is relatively short. It could exist for a few minutes, but sometimes that could be days or even weeks. It all depends on how fast we are at closing pull requests.

Nevertheless, there is a high chance that the application deployed in such a temporary environment will have low traffic. We would typically run a set of automated tests when the pull request is created or when we make changes to it. That would certainly result in a traffic spike. But, after that, the traffic would be much lower and most of the time non-existent. We might open the application to have a look at it, we might run some manual tests, and then we would wait for the pull request to be approved or for someone to push additional changes if some issues or inconsistencies were found. That means that the deployment in question would be unused most of the time. Still, if it would be a "traditional" deployment, it would occupy resources for no particular reason. That might even discourage us from making temporary environments due to high costs.

Permanent environment

Given that deployments based on pull requests are not used for final validations before deploying to production as that's what permanent environments are for, we do not need to insist that they are the same as production. On the other hand, the applications in such environments are mostly unused. Those two facts lead us to conclude that temporary (often pull-request based) environments are a great candidate for serverless computing, no matter the deployment type we use in permanent environments (e.g., staging and production).

Now that we saw some of the use cases for serverless computing, there is still an important one that we will discuss the next lesson.