Functions as a Service FaaS - Part 3

This lesson discusses the trade-offs of FaaS model's and provides use case examples.

We'll cover the following

Things to be aware of when choosing FaaS service model

Cold start problem

Since *FaaS* is an event-driven model, the server only spins up when an event occurs; it's not always running. There is a boot time associated, every time the server spins up, and this eventually adds a bit of latency to the response. This behavior is known as the *cold start problem*.

The server's boot time varies depending on several factors such as the service provider, environment configuration, memory footprint of the code, the technology the service is built with, and so on.

So, if you are trying to build a low latency application like a stock trading app using *FaaS*, you should be aware of the cold start problem.

If the service running on *FaaS* keeps receiving requests continually, the cold start won't be an issue because the server that spun up for the first request will keep processing the queued requests. However, think of an application that receives intermittent traffic. In this scenario, every time the event occurs the platform has to spin up a new server instance. Every user will feel that additional latency when interacting with the service.

Giving up control

The second trade-off when using *FaaS* is that you have to give up a lot of control. Although all kinds of applications can be deployed using a *FaaS* service model, it doesn't fit best with every use case. It's not a one size fits all service model. It has its pros and cons and should be picked wisely. I'll discuss more on this in the *FaaS* use case section.

Also, though the developer is not expected to manage the servers, they still have to have the knowledge of how their code runs in the system, how it interacts with the other services, and so on. They have to have an idea of the overall system architecture. Imagine having scalable functions with a bottleneck in the data persistence flow. Auto-scaling functions wouldn't be any good in this scenario.

Not having to manage anything might sound pretty cool from *35K* feet, but when you hit the ground, you start seeing the little details of everything. Then to tweak stuff, you would need control.

Complexity of managing so many functions

When you have so many functions running on the cloud, this adds up to the complexity of the service. *Monoliths* are simple to manage but difficult to scale. *Stateless functions* are easy to scale but too many of them add up to the complexity of the project. Individual functions may be simple to scale, but the project as a whole becomes complex.

Also, since *FaaS* is a relatively new service model the tools available for scaling are limited. The developers have to rely on the tools provided by the platform to manage and monitor their code.

Alright, with this being said, let's move on to the use cases of *Functions as a Service*.

FaaS – Use cases

- FaaS is perfect for services that are triggered only for a certain duration, say for a few hours in a day or a few days in a week or a month. Examples of these services would be modules that execute tasks initiated by a background batch job, including a medical bill processing module, an automated tax processing module, chatbots, etc. Since these services only run for a certain duration, it's best to spin up the servers only when required as opposed to

keeping them running continually. FaaS enables us to do just that.

- Stateless processes, like image compression, video analysis, data processing, and high volume big data analytics, can be isolated from the system and can be built using FaaS.
- If your application has an *event-driven architecture FaaS* can be a good fit for your application. Then again, if low latency is important for you, you have to consider other options. A lot of thought should go into the process of designing your application, picking the right cloud service model, and the technology stack.

> **Note:** In my other course *Web Application & Software Architecture 101*, discuss concepts such as event-driven architecture, how to pick the right architecture for your service, technology stack for your application, the trade-offs involved, and so on in detail. Check it out.

These are some of the use cases that fit best with the *FaaS* cloud service model. In the next lesson, we will discuss *Serverless*. What does *Serverless* really mean? Can the terms *Functions as a Service* & *Serverless* be used interchangeably?

Without further ado, let's get on with it.