Functions as a Service FaaS - Part 2

This lesson continues discussing the Functions as a Service cloud model.

We'll cover the following

Architecture of a simple application using FaaS

Here is a very basic architecture of an online service using *FaaS* to give you a better understanding of the service model.

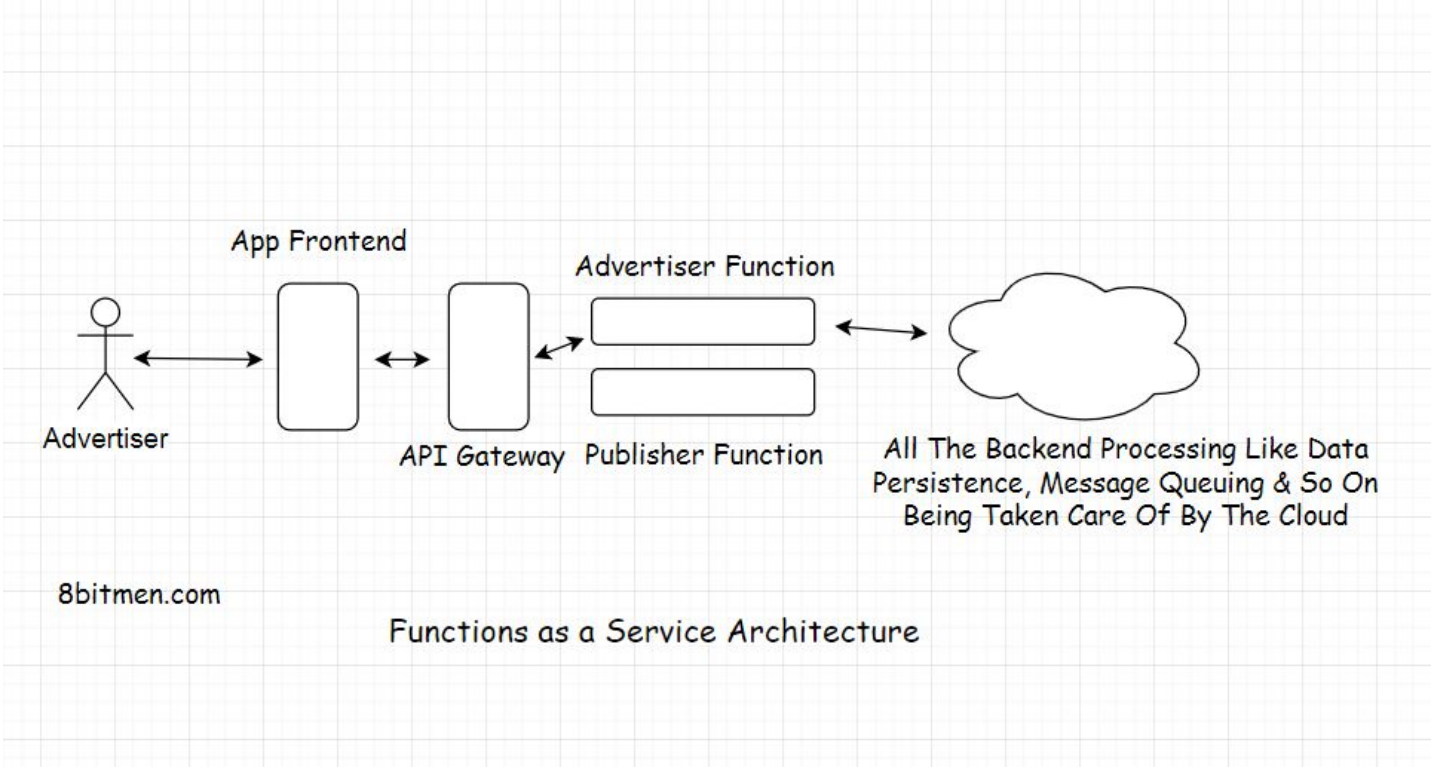
Imagine a simple social media marketing service that enables publishers and advertisers to work together by connecting them online.

To achieve this, it encourages both the parties, i.e., the advertisers and the publishers, to enter their business information via its web service. Once the user enters the information on the website and clicks submit, it is processed on the backend and stored in the database.

If we were building a regular application using a *PaaS*, for example, we would have two separate *APIs* – one for the advertiser and other for the publisher. Then, we would implement both our *APIs* on the backend. When building our service using *FaaS*, we have to focus on writing the business logic in functions. That's about it.

We will create two separate functions - one for the advertiser and the other for the publisher, containing the business logic. We are aware of the fact that we do not have access to the server on which our code will run; we just have to write the business logic and upload it to the platform, letting it take care of the rest.

Here is how the flow will look:



Our architecture will have an *API Gateway* to entertain the requests from the front end. An *API Gateway* is a managed service that takes care of all the *API* related tasks such as managing, publishing, monitoring the *APIs*, dealing with traffic bursts, and taking care of authorization, authentication, and so on. *AWS API Gateway* and *Google Cloud Endpoints* are examples of fully managed *API Gateways*.

The *API Gateway* will receive the requests from the front-end and will route the request to the respective function based on the handler configuration provided in the Gateway. In this scenario, the passing of a request to the appropriate function is an event. When the event occurs, the platform spins up the server, runs the function, and returns the response to the *API Gateway* to be returned to the client.

Now, this is a very simple use case with just two functions. Depending on the application size, as the number of functions increase, the complexity of managing them increases too. To manage this effectively, we have frameworks like *Serverless.*

The *Serverless Framework* is an open-source, end-to-end solution for building and operating Serverless applications.

So, you just saw how with *FaaS* the complete application development work is reduced to just coding functions. If you've ever worked on the first version of *Spring Framework*, you would know that the application required so much *XML* configuration and multiple project files just to run a hello world web page. Just a

configuration and multiple project files just to run a hello world web page. Just a

slight typo in configuration made us pull our hair, trying to figure out what really went wrong.

FaaS doesn't doesn't expect you to manage anything of that sort, making this service model amazing, right? Having the ability to focus on writing business logic and not worry about anything else. Having to pay based on the compute time only and not for the idle running server instances.

I wish I could say yes, but things with FaaS are not so simple as they appear. After all, there is no silver bullet; everything has trade-offs.

In the next lesson, we will find out what these trade-offs are.