**Comparison with PaaS**

Given that Serverless FaaS functions are very similar to [Twelve-Factor applications](http://12factor.net/), are they just another form of ["Platform as a Service"](https://en.wikipedia.org/wiki/Platform_as_a_service) (PaaS) like [Heroku](http://www.heroku.com/)? For a brief answer I refer to Adrian Cockcroft - *If your PaaS can efficiently start instances in 20ms that run for half a second, then call it serverless*

In other words, most PaaS applications are not geared towards bringing entire applications up and down in response to an event, whereas FaaS platforms do exactly this.

If I’m being a good Twelve-Factor app developer, this doesn’t necessarily impact how I program and architect my applications, but it does make a big difference in how I operate them. Since we're all good DevOps-savvy engineers, we're thinking about operations as much as we’re thinking about development, right?

The key operational difference between FaaS and PaaS is *scaling*. Generally with a PaaS you still need to think about how to scale—for example, with Heroku, how many Dynos do you want to run? With a FaaS application this is completely transparent. Even if you set up your PaaS application to auto-scale you won’t be doing this to the level of individual requests (unless you have a very specifically shaped traffic profile), so a FaaS application is much more efficient when it comes to costs.

Given this benefit, why would you still use a PaaS? There are several reasons, but tooling is probably the biggest. Also some people use PaaS platforms like [Cloud Foundry](https://en.wikipedia.org/wiki/Cloud_Foundry) to provide a common development experience across a hybrid public and private cloud; at time of writing there isn’t a FaaS equivalent as mature as this.

#### Comparison with containers

One of the reasons to use Serverless FaaS is to avoid having to manage application processes at the operating-system level. PaaS services, like Heroku, also provide this capability, and I’ve described above how PaaS is different to Serverless FaaS. Another popular abstraction of processes are containers, with [Docker](https://www.docker.com/) being the most visible example of such a technology. Container hosting systems such as [Mesos](http://mesos.apache.org/) and [Kubernetes](http://kubernetes.io/), which abstract individual applications from OS-level deployment, are increasingly popular. Even further along this path we see cloud-hosting container platforms like [Amazon ECS](https://aws.amazon.com/ecs/) and [EKS](https://aws.amazon.com/eks/), and [Google Container Engine](https://cloud.google.com/container-engine) which, like Serverless FaaS, let teams avoid having to manage their own server hosts at all. Given the momentum around containers, is it still worth considering Serverless FaaS?

Principally the argument I made for PaaS still holds with containers - for Serverless FaaS **scaling is automatically managed, transparent, and fine grained**, and this is tied in with the automatic resource provisioning and allocation I mentioned earlier. Container platforms have traditionally still needed you to manage the size and shape of your clusters.

I’d also argue that container technology is still not mature and stable, although it is getting ever closer to being so. That’s not to say that Serverless FaaS is mature, of course, but picking which rough edges you’d like is still the order of the day.

#### #NoOps

Serverless doesn’t mean "No Ops"—though it might mean “No sysadmin” depending on how far down the Serverless rabbit hole you go.

“Ops” means a lot more than server administration. It also means—at least—monitoring, deployment, security, networking, support, and often some amount of production debugging and system scaling. These problems all still exist with Serverless apps, and you’re still going to need a strategy to deal with them. In some ways Ops is harder in a Serverless world because a lot of this is so new.

The sysadmin is still happening—you’re just outsourcing it with Serverless. That’s not necessarily a bad (or good) thing—we outsource a lot, and its goodness or badness depends on what precisely you’re trying to do. Either way, at some point the abstraction will likely leak, and you’ll need to know that human sysadmins somewhere are supporting your application.