Three misconceptions about Serverless, and why Serverless is often misunderstood?

The trend of serverless continues, but there are still misconceptions, and these often this can lead to decisions to not adopt these technologies.

Craig Godden-Payne
Jun 26 · 5 min read ★

Serverless technologies come in many guises.

Cloud platforms such as AWS offer many different kinds of ‘Serverless’ tech, such as Fargate, Lambda, Aurora, ApiGateway etc.

Misconception One — You will be vendor locked in.

Part of the power of serverless is that it can easily access additional cloud services from within the same platform. *

Because of this, there is a tendency to use multiple services from within the same cloud provider.

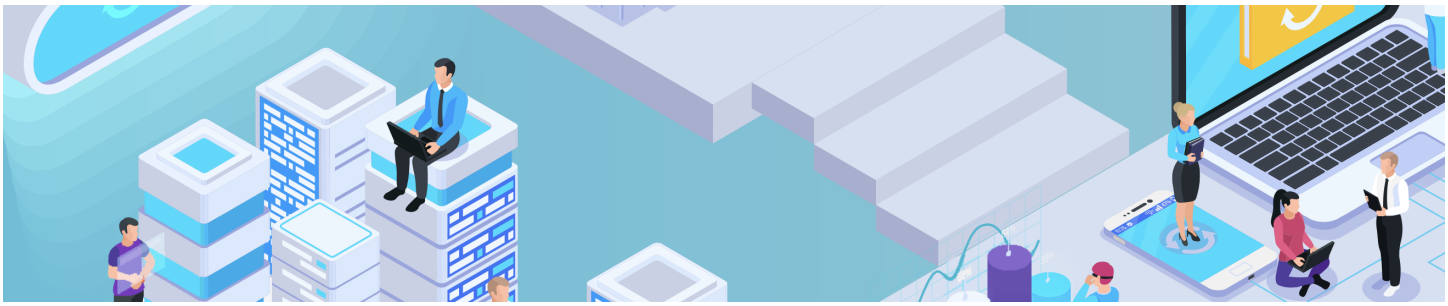
It is undoubtedly possible for say a function in AWS lambda to call out to a service in Azure or GCP, but you then have to start looking at things like security boundaries, authentication etc.

The reason people misconceive that you have to go with one cloud provider is that it is just so much simpler to do so.

The Only Step You Need to Progress in Your Career as a Developer or Engineer.

I don't usually like to write articles about myself, as it feels a bit self-indulgent, but I thought it would be useful...

medium.com



Using multiple cloud vendors also reduces the cost-effective nature of serverless.

The cost of developing a small piece of the puzzle, and the cost of running is so tiny in comparison to a full-blown application; it simply does not make sense cost-wise to then call another cloud service provider.

Cloud providers tend to use plug-and-play style hooks and make integration of services very straightforward, so if you want to avoid vendor lock-in, it is very much so possible, it just might cost you a bit more in time to do so.

Misconception two: Serverless == Functions as a Service

There is a common misconception that serverless is just a Function as a Service (FaaS), such as AWS Lambda or Azure Functions.

It is entirely false.

Although FaaS can make up the compute part of serverless, there are different ways to do this, and many other services out there which provide additional functionality than compute, such as database or gateways.

AWS Lambda has always been one of the great things to shout about, and arguably one of the more popular elements of serverless architecture, the real reason to adopt serverless is so that you:

- Don’t need to worry about managing infrastructure.
- Scaling is available out of the box.
- Cost is reduced due to only paying for what is used.

The tradeoff for serverless is usually worth the investment. It can cause abstracted designs, with more complicated workflows, but these are not so bad once you get used to the new world of serverless.

Creating a Simple, Low-Cost Twitter Bot, utilising Serverless

People have a love-hate relationship with twitter bots.

medium.com

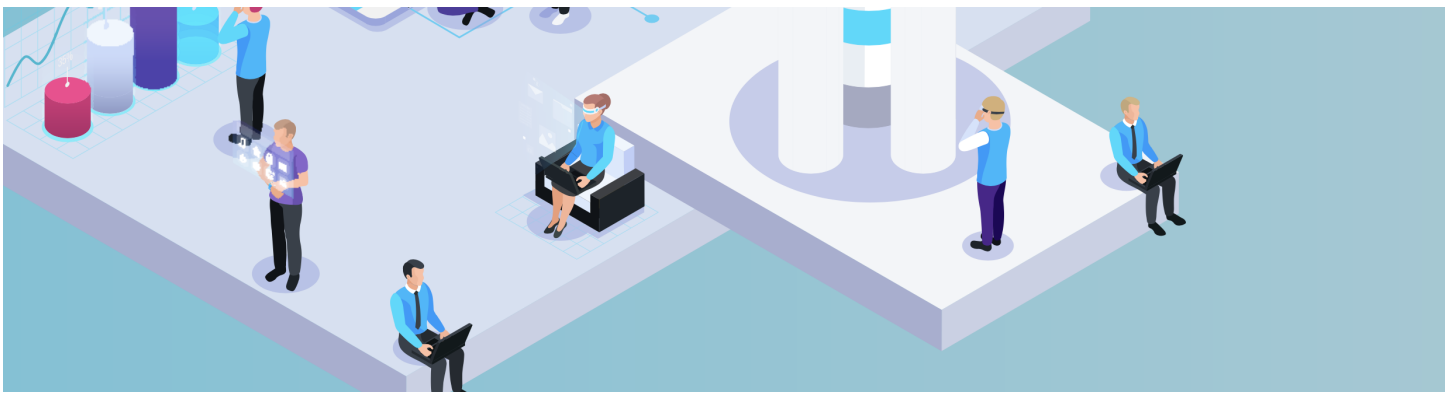
Misconception three: The time I save by not managing infrastructure will be replaced with supporting complexities.

With a serverless approach, the time working on managing updates, patching and security will undoubtedly decrease, that is a given.

The whole point of adopting the serverless approach is so more time can be spent impacting users of the infrastructure by helping to build more robust and reliable solutions.

It is likely applications will get more complicated, but this can (and should) be combated with breaking applications down into smaller pieces.

If it is difficult to get the application working in a serverless environment, it is likely too large and would need breaking down further. This problem tends to affect working on migrating an existing application to a serverless platform more.



Nearly all entry points into a serverless application are through some kind of API or gateway, via REST or message queues.

This common pattern would not need to change, except for abstracting the REST / Message queue logic away into serverless technologies, e.g. in AWS using API Gateway as the REST interface, or an SQS trigger as the invoker of a function.

By breaking down your application into smaller pieces, you are also creating a much more flexible infrastructure, where you can now develop independently of what once may have been the core application.

Suddenly the application you work on starts to have more straightforward integration points, and the number of dependencies it relies on starts the dwindle.

The simplicity of the integration of serverless pieces is what will make your platform thrive.

Using Microservice Patterns in an increasingly Serverless world

When working on an application domain, it is beneficial to use the microservices software design pattern.

medium.com

So why would you want serverless anyway?

When you build a serverless application, it's an evolution;

You do not tend to build with the thought in mind that this will one day go back to being a fat application.

When you pick a cloud vendor to run the serverless code, it is always important to consider the following points:

- Would a Function as a Service (FaaS) offering be the correct approach, would you need to use a container, or is your application just not ready to be split up and migrated to such a platform?
- What language would you like your application to be in? Cloud vendors tend to support many languages such as AWS Lambda supports Python, Node and dotnet.
- How would you manage your deployments using a CI/CD system? The lifecycle of a FaaS and Container may be very different from how your current 'fat' applications are running.
- Which security requirements do you have? You may have to combine multiple services to do this, for example with AWS you can connect a serverless API Gateway with AWS Lambda to create a REST entry point to your lambda, which can support some kind of security mechanism.

Graphics Attributions:
https://www.freepik.com
macrovector

Serverless Event Driven Architecture DevOps Software Development AWS

About Help Legal

Get the Medium app