Using Microservice Patterns in an increasingly Serverless world

Craig Godden-Payne
Jun 29 · 4 min read ★

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

Microservices have many advantages over monolithic stacks:

> *Highly maintainable and testable — enables rapid and frequent development and deployment*
>
> *Loosely coupled with other services — enables a team to work independently the majority of time on their service(s) without being impacted by changes to other services and without affecting other services*
>
> *Independently deployable — enables a team to deploy their service without having to coordinate with other teams*
>
> *Capable of being developed by a small team — essential for high productivity by avoiding the high communication head of large teams*

Hosting microservices

When it comes to hosting, it is possible to combine this pattern with a serverless framework, such as lambda to create serverless microservices.

> *Serverless architectures are application designs that run in managed, ephemeral containers on a "Functions as a Service" (FaaS) platform.*
>
> *Such architectures remove much of the need for a traditional always-on server component and may benefit from significantly reduced operational cost, complexity, and engineering lead time.*

. . .

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

Serverless microservices in the cloud

To use compute in AWS, it is possible to take advantage of Fargate or a combination of Api Gateway and Lambda to host a service in a serverless way.

Lambda is supposed to be used for small functions, and maintaining this pattern is key to not over-engineering or complicating your code.

Because of the complexity of the flow of some microservices, small Lambdas can be tied together, to replicate more complex behaviour.

To effectively manage the problem now, and for the future addition of Lambda functions within your microservice, there are multiple ways in which you can do this.

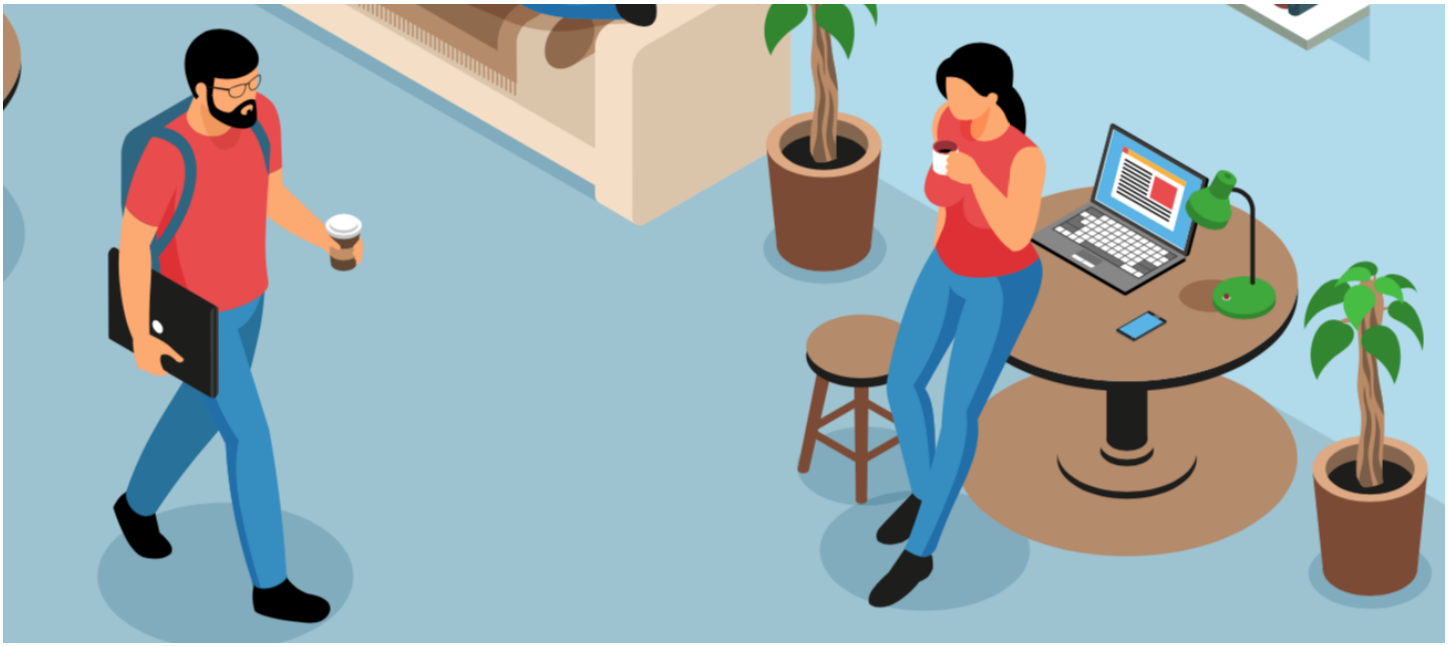
Scenario One — Use a serverless orchestrator.

You could create a master Lambda, which would primarily farm out requests to other Lambda functions. The solution is not the best, though.

Over time, the more this pattern grows, the more it will become challenging to maintain.

It becomes complicated to maintain synchronicity due to the fact each path the lambda farms out to, would need a success and failure scenario, and possibly others.

This kind of pattern works well when the code is contained in its entirety but starts to fall apart when it has multiple outward routes. If you currently use this pattern, it is worth looking at the other options available.



Scenario Two — Use a Queue for sharing messages.

You could consider using SQS to coordinate the execution of the other Lambda functions.

Again if your service is split over multiple lambdas, it is great to be able to use a message queuing system to join lambdas together. It also has the added benefit of being able to replay messages in a failure scenario.

But again this solution is not ideal, it is more geared towards asynchronous event-driven workflows, rather than the microservice pattern we are trying to achieve.

What can happen with this architecture is that although the code is split into small pieces, the overall flow can become complex and difficult to follow. There are better solutions than this, although it is better than scenario one.

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...

medium.com

Scenario Three — Use Step Functions to orchestrate.

You could consider using Step Functions to coordinate the execution of the other Lambda functions.

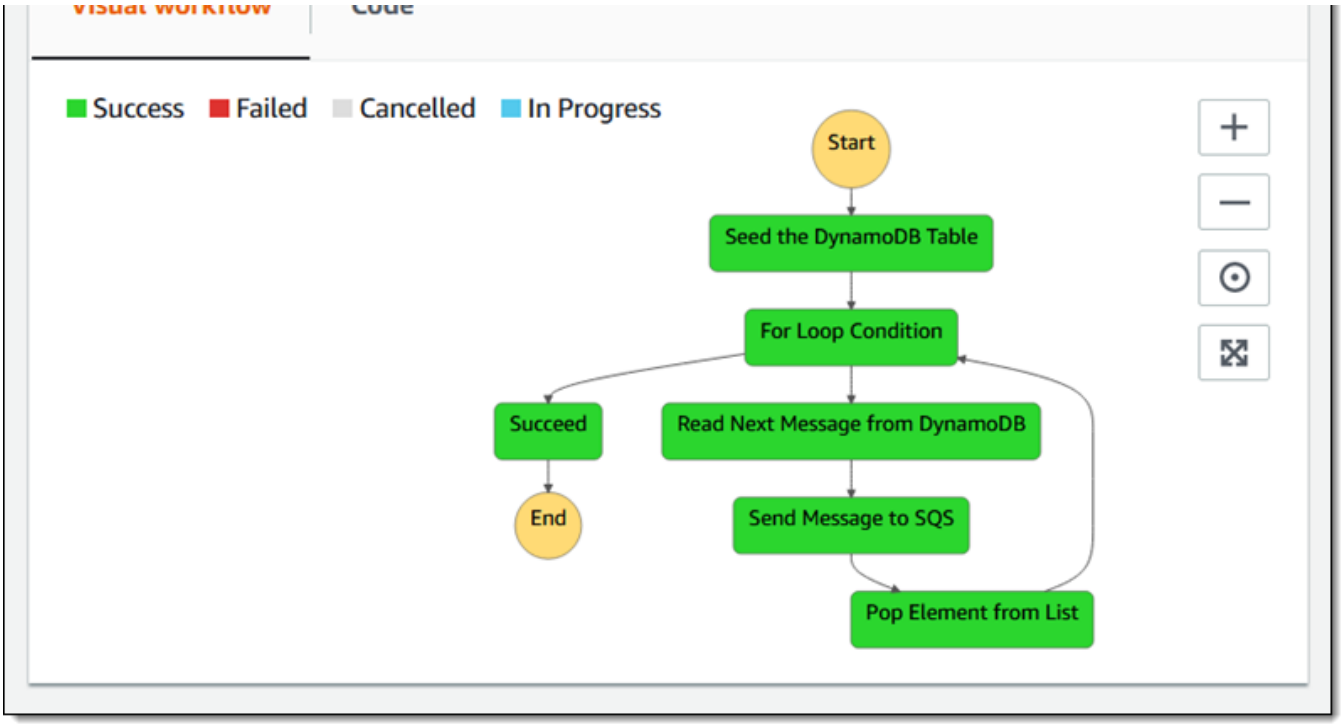
As your service grows, it is easy to use step functions as the glue between your lambda code. The is probably the best way to manage the workflow while maintaining synchronicity.

> *AWS Step Functions lets you coordinate multiple AWS services into serverless workflows so you can build and update apps quickly. Using Step Functions, you can design and run workflows that stitch together services, such as AWS Lambda, AWS Fargate, and Amazon SageMaker, into feature-rich applications.*

You can build applications from individual components that each perform a discrete function, or *task*, allowing you to scale and change applications quickly.

The reason this is a preferred approached in a microservice serverless architecture is that step-functions provide a reliable way to coordinate components and step through the functions of your application.

You also can use the UI view within the console to get a visual view of how your application flow will function.



Graphic Attributions:
https://www.freepik.com/free-photos-vectors/coffee
macrovector
