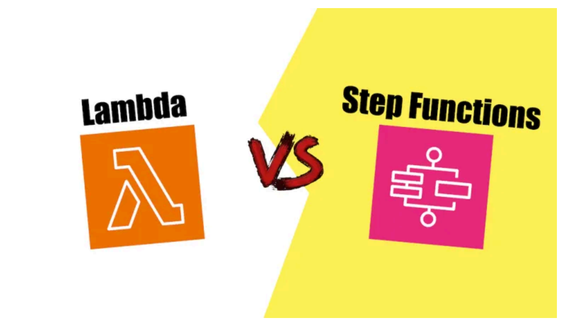
When to use Step Functions vs. doing it all in a Lambda function

Yan Cui · Follow
Published in theburningmonk.com · 6 min read · 4 days ago



I'm a big fan of AWS Step Functions. I use it to orchestrate all sorts of workflows, from payment processing to map-reduce jobs.

Why it's yet another AWS service you need to learn and pay for. And it introduces additional complexities, such as:

- It's hard to test [1].
- Your business logic is split between configuration and code.
- New decision points. Such as whether to use Express Workflows or Standard Workflows [2].

So it's fair to ask " *Why should we even bother with Step Functions?* " when you can do all the orchestration in code, inside a Lambda function.

Let's break it down.

Lambda pros

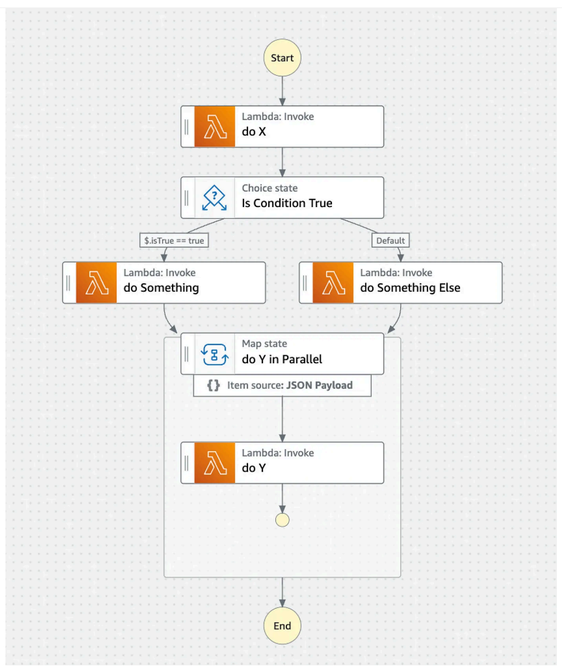
1. Doing all the orchestration in code is simpler. It's more familiar.

Everything you can do in Step Functions, you can do with just a few lines of code. Case in point:

```
module.exports.handler = async (event) => {
    // error handling
    try {
        await doX()
    } catch (err) {
        // handle errors
    }

    // branching logic
    if (condition === true) {
        await doSomething()
    } else {
        await doSomethingElse()
    }

    // parallelism
    const promises = event.input.map(x => doY())
    const results = await Promise.all(promises)

    return results
}
```

2. It's *likely* cheaper.

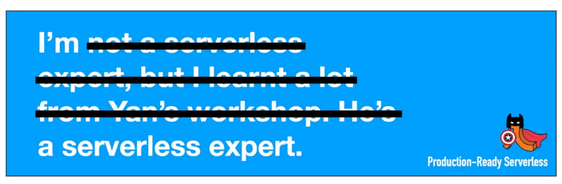
A Step Functions state machine would likely use Lambda for its `Task` states.



In which case, you'd end up paying for both:

- The Lambda invocations. There are multiple invocations instead of just one. However, the total billable execution time should be similar.
- Step Functions state transitions. At $25 per million state transitions, it's one of the more expensive services in AWS.

Paying for two services is likely more expensive than paying for just one.



3. It's *likely* more scalable.

When you use both Step Functions and Lambda functions (for the `Task` states) you are constrained by the throughput limits of both services.

Step Functions Standard Workflows have modest limits on the no. of state transitions and the no. of executions you can start per second.

Both of these limits can be raised. So with proper planning, they wouldn't be an issue.

Without Step Functions, you are limited only by the concurrent executions limit on Lambda. Similarly, Lambda has default throughput limits on the no. of concurrent executions.

Again, with proper planning, and given the recent scaling changes for Lambda [3], you will be OK.

Both the cost and scalability arguments are situational and depend on several architectural choices.

E.g. do you use Standard or Express workflows?

Do you use Lambda functions for `Task` states or direct service integrations?

Have you estimated your throughput needs and raised the soft limits accordingly?

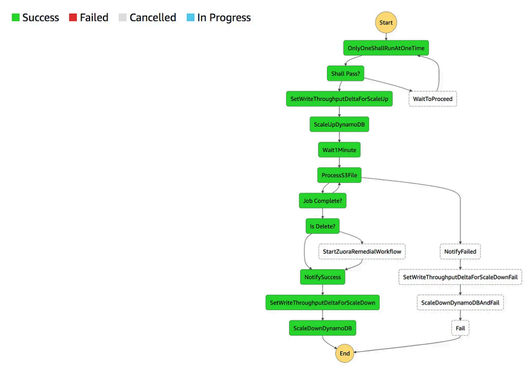
Because of these factors, they are only “likely” to be true based on what I think the average AWS customer is capable of.

Lambda cons

1. Cold starts.
2. 15 mins max duration.
3. Not good for waiting. Because you pay for execution time by the milliseconds. It’s a poor solution when it comes to waiting for something to happen because all that execution time (and money) is wasted.

Step Functions pros

1. The visual workflows make it very easy to debug problems. This is true even for non-technical team members, such as product owners and customer support teams.



2. Step Functions has a built-in audit history of everything that happened, including:

- When did a state start?
- What were the input and output?
- Error message and stack trace.

3. Step Functions have direct service integration with almost every AWS service. So it’s possible to implement an entire workflow without needing any Lambda functions.

4. No Lambda, no cold starts. No cold starts = more predictable performance.

5. Long execution time. A Standard Workflow can run for up to a year.

6. Callback patterns are a great way to support human decisions (e.g. approve a deployment request) in a workflow.

7. Standard Workflows are arguably the most cost-efficient way to wait. Because you don’t pay for the duration, only the state transition.

8. You can implement more robust error handling. This is important for business-critical workflows.

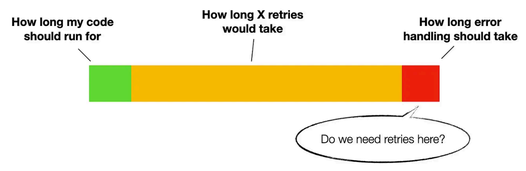
To make a workflow more robust, you need to have both:

- Retries. Preferably with exponential backoff and random jitters.
- A fallback or error handling logic. In case an operation fails after a reasonable number of retries.

With Lambda, this puts you between two opposing forces:

a) Security best practices recommend short timeouts. This mitigates the impact of denial-of-wallet attacks.

b) We need a long timeout to cater for the worst-case scenario. We need to allow enough time for the retries and the fallback. This means the timeout needs to be many times the expected execution time on a happy path. Oh, and you might also need retries in your fallback!



It’s difficult to guess the right timeout in these situations. Especially when your workflow might have different branches. And if you get it wrong, then your workflow would be killed off halfway and there’s no easy way to restart from the point of failure.

By lifting the error handling and retries out of your code and into the state machine itself, you alleviate this tension.

9. Step Functions lets you resume an execution from that point of failure [4].

Step Functions cons

1. Cost.
2. Learning curve.
3. More complexity. There are a lot more things to configure and manage.
4. Your business logic is split between the State Machine definition and your code (if using Lambda for `Task` states).

5. Hard to test. However, this is getting easier with the new TestState API [5].

Conclusion

In conclusion, Step Functions offer a plethora of capabilities. But they come with their own set of complexities and costs.

Whether it’s right for you depends on the demands of your use case.

Generally, I advocate for the path of least resistance: simple workflows call for simple solutions. And Lambda functions excel in these scenarios.

However, for many workloads, Step Functions can simplify the implementation with its built-in functionalities that would otherwise require custom solutions.

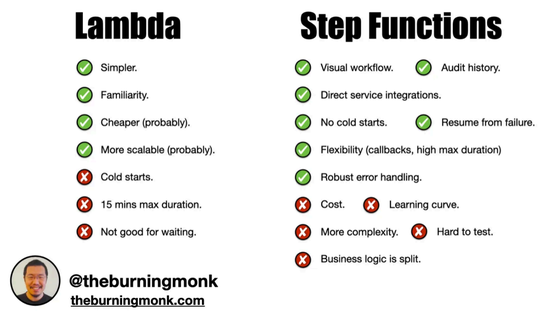
For example, when you need to incorporate human decisions into a workflow. You should use Step Functions and leverage its callback patterns instead of creating a bespoke solution.

Similarly, for workflows where resuming from a point of failure is important, you should go with Step Functions.

Personally, I heavily lean towards Step Functions for business-critical workflows. The advantages of visualization, audit trails, and robust error handling align with the high stakes involved. These workflows, like payment

processing, warrant the extra investment in Step Functions due to their critical nature.

I will leave you with this diagram with the pros & cons captured in one place. To learn more about testing Step Function, check out my course, **Testing Serverless Architectures** [6]. I have a whole chapter dedicated to Step Functions.



Links

[1] Why is Step Functions so hard to test?

[2] What is AWS Step Functions? An in-depth overview.

[3] AWS Lambda functions now scale up to 12X faster

[4] Introducing AWS Step Functions redrive to recover from failures more easily

[5] Does Step Function's new TestState API make end-to-end tests obsolete?

[6] Testing Serverless Architectures course

Originally published at https://theburningmonk.com on March 10, 2024.

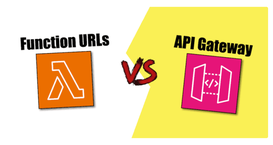
AWS · AWS Lambda · Serverless · Step Functions

Written by Yan Cui

5.8K Followers · Editor for theburningmonk.com

AWS Serverless Hero. Follow me to learn practical tips and best practices for AWS and Serverless.

More from Yan Cui and theburningmonk.com



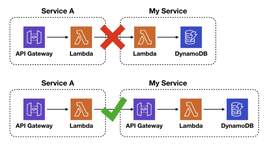
Yan Cui in theburningmonk.com

When to use API Gateway vs. Lambda Function URLs

"Lambdalith" is a monolithic approach to building serverless applications where a...

5 min read · Mar 3, 2024

17



Yan Cui in theburningmonk.com

Are Lambda-to-Lambda calls really so bad?

If you utter the words "I call a Lambda function from another Lambda function" you...

8 min read · Jul 12, 2020

90 2

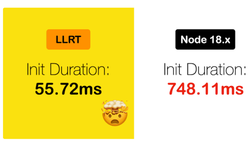
Yan Cui in theburningmonk.com

How to secure CI/CD roles without burning production to the ground

By now, most of us have moved away from using IAM users for CI/CD pipelines. Instead...

6 min read · Feb 16, 2024

5



Yan Cui in theburningmonk.com

First impressions of the fastest JavaScript runtime for Lambda

I thought Lambda needed a specialised runtime. One that works well with its...

5 min read · Feb 27, 2024

61

See all from Yan Cui · See all from theburningmonk.com

Recommended from Medium

The SaaS Enthusiast

Advanced Serverless Techniques III: Simplifying Lambda Functions...

Ever found yourself repeatedly typing the same lines of code across various functions...

7 min read · Feb 9, 2024

46

Mahtab Haider

Real-time Data Processing with AWS DynamoDB Streams and...

3 min read · 3 days ago

10

Lists

General Coding Knowledge

20 stories · 1017 saves

Leadership

48 stories · 266 saves

Natural Language Processing

1284 stories · 776 saves