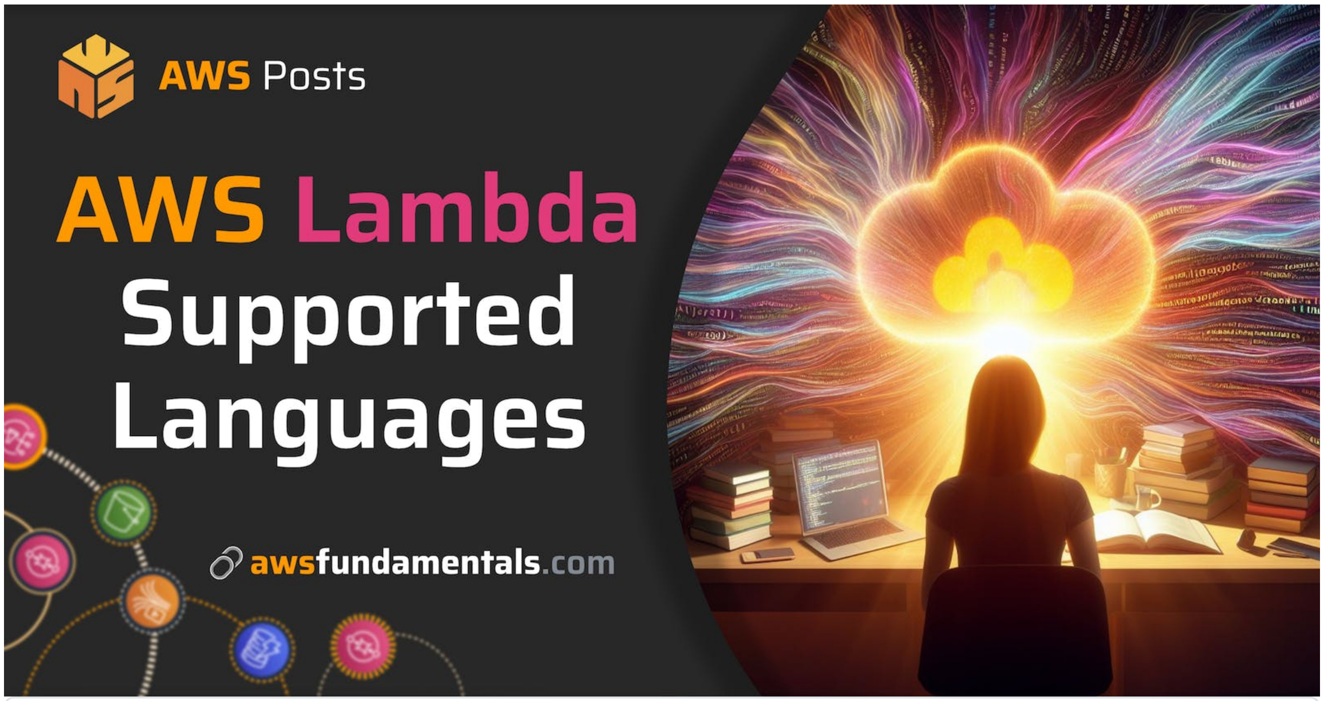
AWS Lambda Supported Languages and How They Compare

blog.awsfundamentals.com/supported-languages-at-aws-lambda

Tobias Schmidt



Play this article

0:00 / 7:18

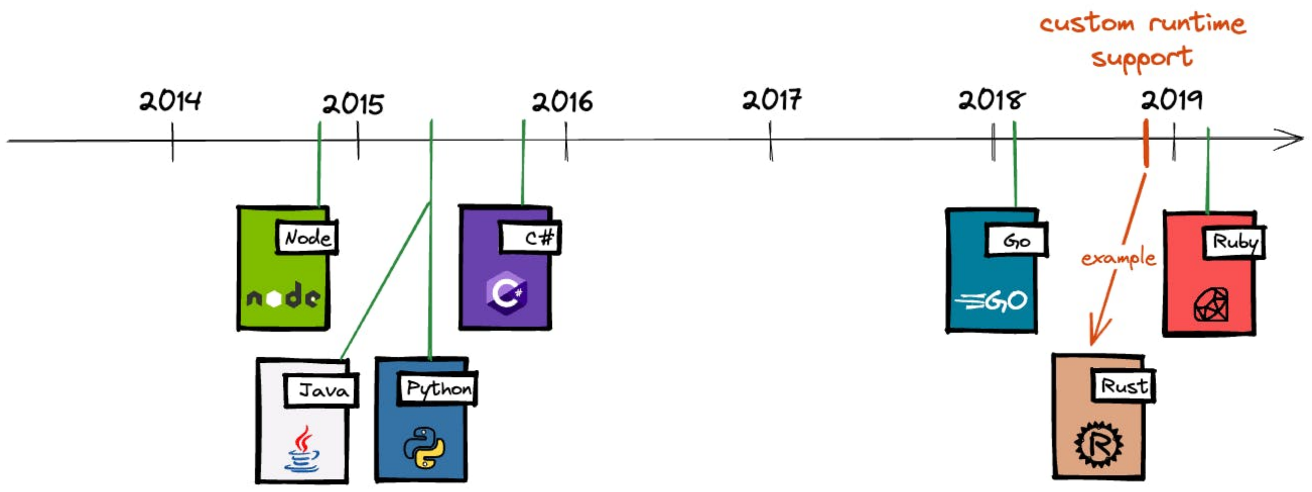
One of the key benefits of using AWS Lambda is the ability to choose from a variety of programming languages for writing and deploying functions. However, each language comes with its own cold and warm start performance.

In this article, we will explore the pros and cons of different programming languages supported by AWS Lambda to help make an informed decision when choosing a language for a project.

Supported Runtimes

While Lambda started only supporting Node.js in 2014, it now natively supports a variety of prominent programming languages. Among them are:

- **.NET (C#)** - a modern object-oriented programming language developed by Microsoft for building Windows desktop applications, web applications, and games using the .NET framework.
- **Go** - a statically typed programming language designed to be simple, fast, and efficient, ideal for developing large-scale, high-performance networked services and applications.
- **Java** - a popular general-purpose programming language used for developing cross-platform applications, mobile apps, web applications, and games.
- **Node.js** - an open-source server-side JavaScript runtime built on Google's V8 engine, used for building scalable, fast, and real-time network applications.
- **PowerShell** - a task automation and configuration management framework built on .NET, used for scripting and managing Windows environments and Microsoft products.
- **Python** - a high-level, interpreted, and dynamically typed programming language, used for web development, scientific computing, artificial intelligence, machine learning, and data analysis.
- **Ruby** - a dynamic, open-source, and object-oriented programming language. Its design aims to make it simple to read and write, and it's commonly utilized for web development, server-side scripting, and automation tasks.

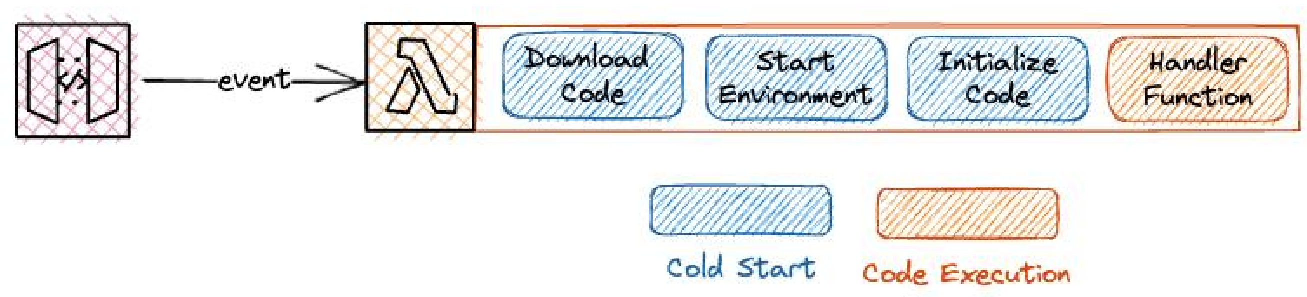


Additionally, you can bring a custom runtime and therefore use any possible language. This means you could also use PHP or Rust by building a Lambda-compatible Linux image.

Considerations when using Lambda

Choosing the right language for your application may depend on Lambda's cold and warm start performance.

When a Lambda function is invoked, AWS needs to initialize the underlying infrastructure and the code to be ready for execution.



This necessary time to initialize the environment is called cold start. Cold starts happen regularly, as AWS only keeps a Lambda micro-container available for a limited amount of time, even with a steady number of consecutive requests.

The cold start times drastically vary based on the used runtime. But performance should not be the only consideration when looking to build applications on AWS Lambda. There may be other crucial factors worth taking into account.

Node.js

Node.js has one of the fastest startup times and a low memory footprint, which makes it a good choice for customer-facing functions that require low latencies.

Node.js comes with the largest community around AWS Lambda and provides a great set of frameworks and tools for running Serverless applications. Just have a look at how many NPM packages are tagged with `aws-lambda` - there are several thousand.

It's a safe bet for running customer-facing Lambda functions with its fast spin-up and the fact that there will always be a solution online for every problem you'll ever experience.

Java

Java has a relatively slow startup time compared to Node.js, but it can achieve high performance and scale well for long-running functions with heavy workloads.

Java also provides dependable and extensively tested libraries that come with high reliability. Additionally, it facilitates the prediction of performance by enabling easy estimation of memory requirements.

Furthermore, it offers extensive tool support via widely-used choices like Eclipse, IntelliJ IDEA, Maven, and Gradle, and the most reliable frameworks out there like Spring.

Python

As another scripted language, Python also has fast startup times and a low memory footprint. To be precise: its startup times are even faster than with Node.js.

Same as with Node.js: Python comes with a large set of modules that help you to easily integrate with AWS and other platforms.

.NET

.NET comes with a slower startup time than scripted languages like Node.js and Python, but can be efficiently used for workloads that require high computation.

Due to its maturity, it's a safe bet for developers.

Go

Go has one of the fastest startup times, making it a good choice for functions that require fast response times and high throughput.

it also has a growing ecosystem of libraries and tools specifically designed for building serverless applications on AWS Lambda.

Furthermore, Go comes with static linking. This means all code is compiled into a single executable file that can be easily deployed without any external dependencies. The compiler will also just package necessary code that is actually within possible execution paths. You don't need third-party plugins like webpack for tree shaking in Node.js.

Ruby

has a moderate startup time and memory footprint, making it suitable for simple, lightweight functions.

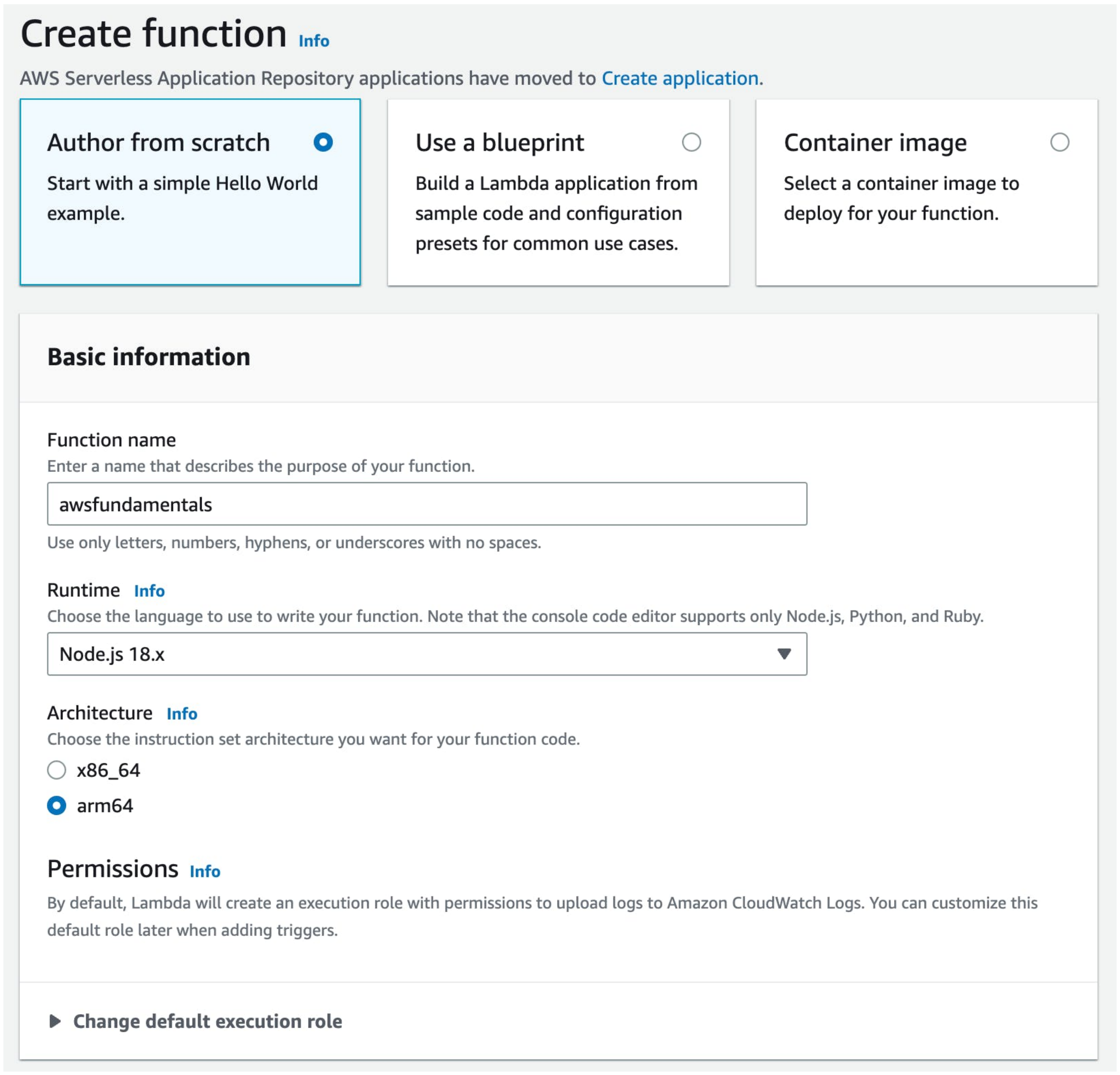
Ruby is known for its expressive and flexible syntax, which allows developers to write code that is both concise and readable. Its support for metaprogramming and functional programming also makes it a popular choice for building highly customizable and scalable applications.

Jumping into Action: Creating a Lambda Function

When creating a Lambda function you need to define many properties of the environment. Many can be changed afterward, but some are fixed and can’t be changed once the Lambda function is created. Let’s explore the most important settings and configurations.

Picking the Runtime and Configuring the Environment

Let's get started by logging into the AWS console and creating our first function. By selecting `Create function` in the function overview, we can easily set up a new function.


Create function Info

AWS Serverless Application Repository applications have moved to Create application.

Author from scratch [✓] — Start with a simple Hello World example.

Use a blueprint — Build a Lambda application from sample code and configuration presets for common use cases.

Container image — Select a container image to deploy for your function.

Basic information

Function name

Enter a name that describes the purpose of your function.

awsfundamentals

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Node.js 18.x

Architecture Info

Choose the instruction set architecture you want for your function code.

x86_64 [✓ arm64]

Permissions Info

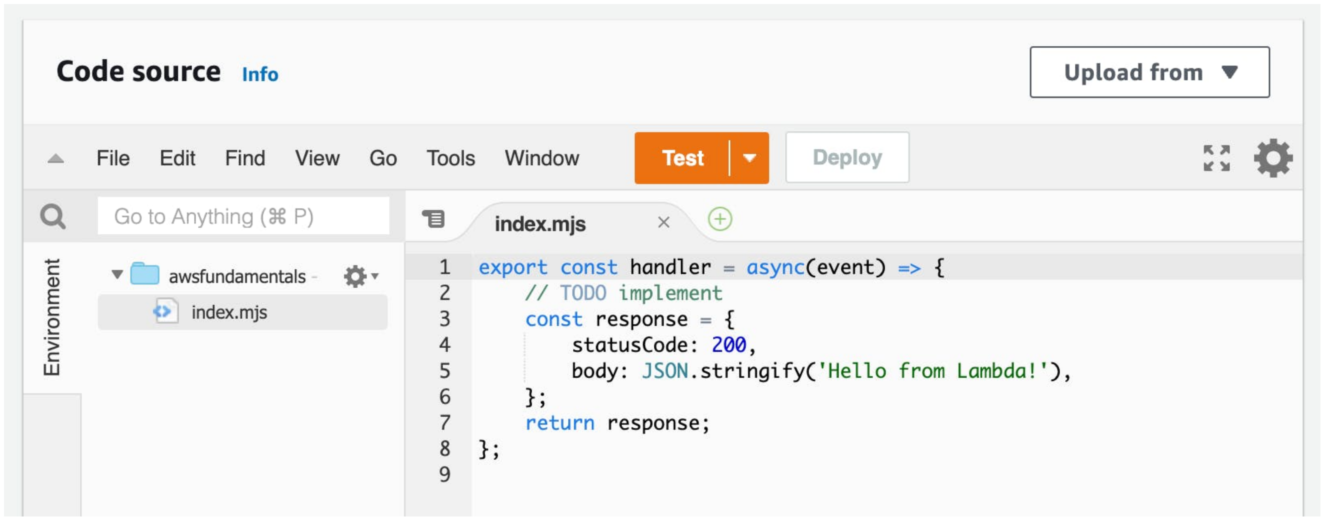
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▶ Change default execution role

All we have to do is choose a name, pick our desired architecture, and select the runtime we prefer - in this instance, we've opted for Node.js.

Once you click on `create`, you will be directed to an overview of the function.

You'll see that there's a live editor for the function's code. This editor is handy for testing, making updates to the function's code, and deploying the function.

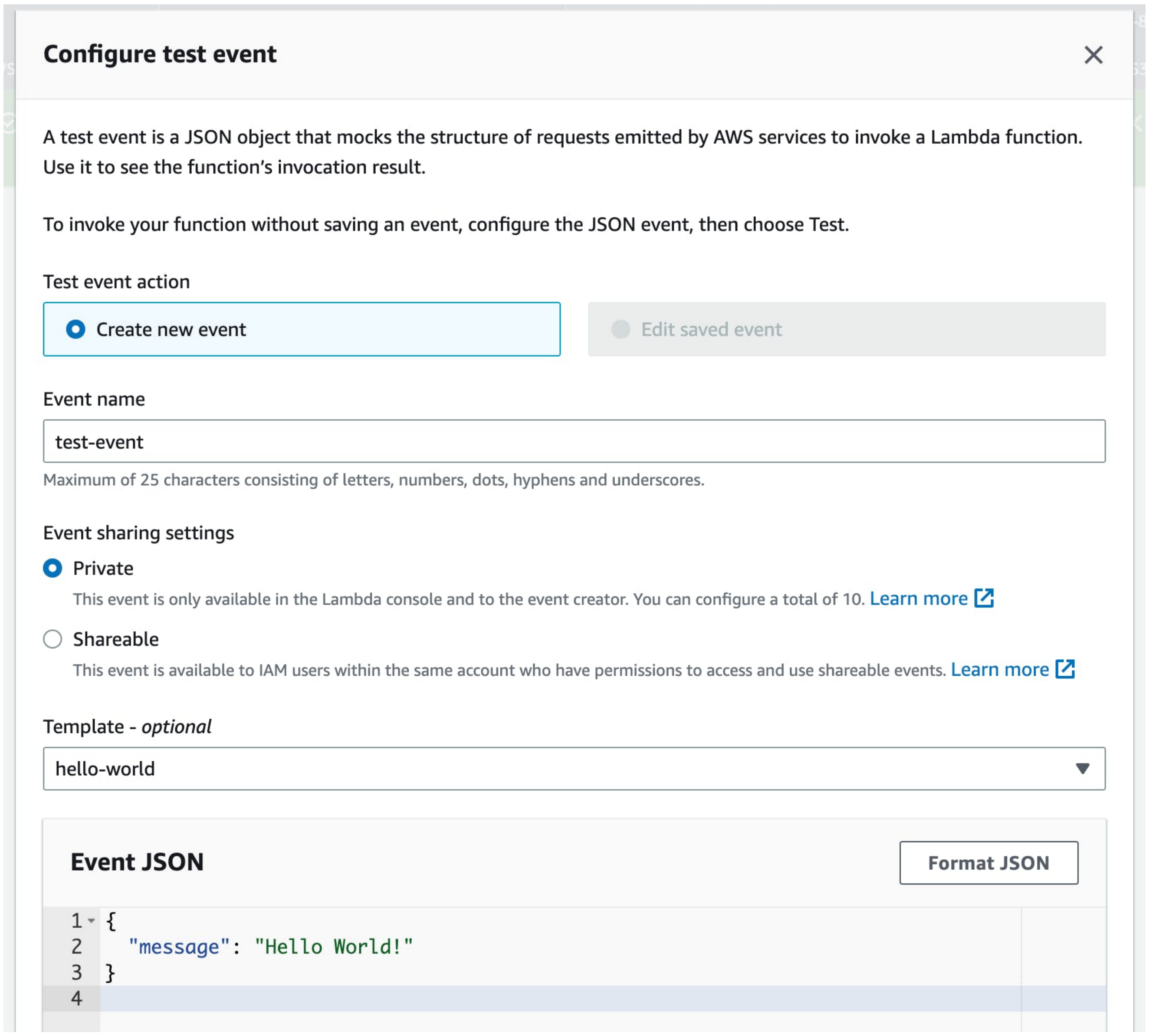


```
export const handler = async(event) => {
    // TODO implement
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
    return response;
};
```

Let's do exactly that by clicking on `Test`.

Executing and Testing our Function

This will open a modal where we can create our first test event. Let's pass a JSON object with a field `message` to our function.


Configure test event

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

[✓ Create new event] Edit saved event

Event name

test-event

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

[✓ Private]
This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn more

Shareable
This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Template - *optional*

hello-world

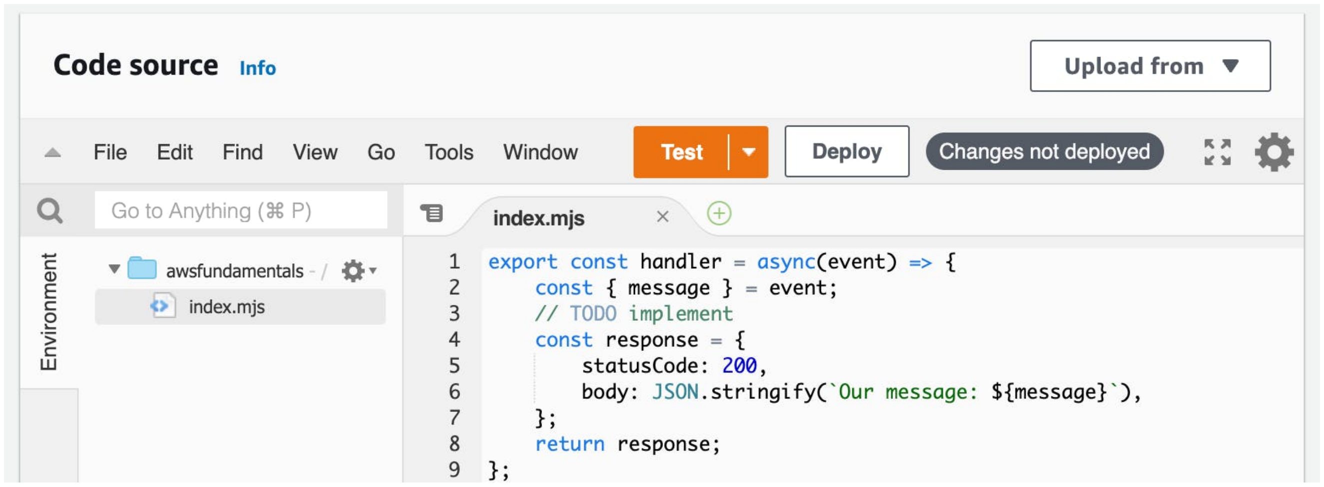
Event JSON

Format JSON

```
{
  "message": "Hello World!"
}
```

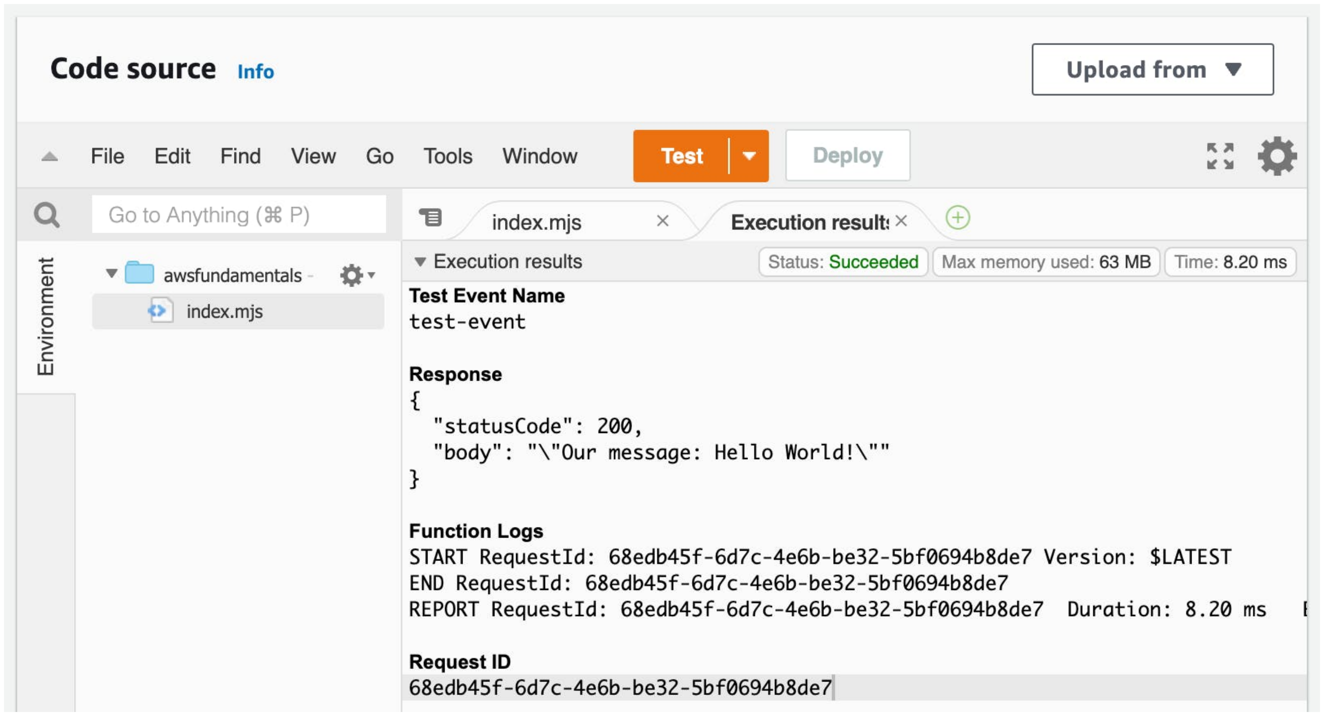
Adapting The Handler Function

Now let's adapt our function and return the message we pass in the function's response. After clicking `Deploy` the changes will be deployed to our function. We can now invoke it again with the test event.



```
export const handler = async(event) => {
    const { message } = event;
    // TODO implement
    const response = {
        statusCode: 200,
        body: JSON.stringify(`Our message: ${message}`),
    };
    return response;
};
```

We'll get the expected message back.



Test Event Name
```
test-event
```

Response
```
{
  "statusCode": 200,
  "body": "\"Our message: Hello World!\""
}
```

Function Logs
```
START RequestId: 68edb45f-6d7c-4e6b-be32-5bf0694b8de7 Version: $LATEST
END RequestId: 68edb45f-6d7c-4e6b-be32-5bf0694b8de7
REPORT RequestId: 68edb45f-6d7c-4e6b-be32-5bf0694b8de7  Duration: 8.20 ms
```

Request ID
```
68edb45f-6d7c-4e6b-be32-5bf0694b8de7
```

Conclusion

In conclusion, AWS Lambda supports a variety of programming languages, including C#, Go, Java, Node.js, PowerShell, Python, and Ruby. Additionally, users can bring their own runtime and use any language they prefer.

When choosing a language, it's important to consider the cold and warm start performance, as each language has its own startup time and memory footprint.

Node.js and Go have the fastest startup times, making them suitable for functions that require low latency and high throughput, while C# and Java are slower but can handle heavy workloads. Python and Ruby have fast startup times and low memory footprints, making them suitable for simple, lightweight functions. Ultimately, the choice of language will depend on the specific needs of the application.

Published on