JAVAWORLD

OPEN SOURCE JAVA TUTORIALS
By Steven Haines, Contributor, JavaWorld
JAN 11, 2018 10:26 AM PST

About
A working developer's guide to open source tools and frameworks for Java application development.

UPDATED

Serverless computing with AWS Lambda, Part 1

Get an overview of AWS Lambda's nanoservices architecture and execution model, then build your first Lambda function in Java

Click the **Save and Test** button and AWS Lambda will run your function with the sample event. If you did everything correctly, you should see a response similar to Figure 11.

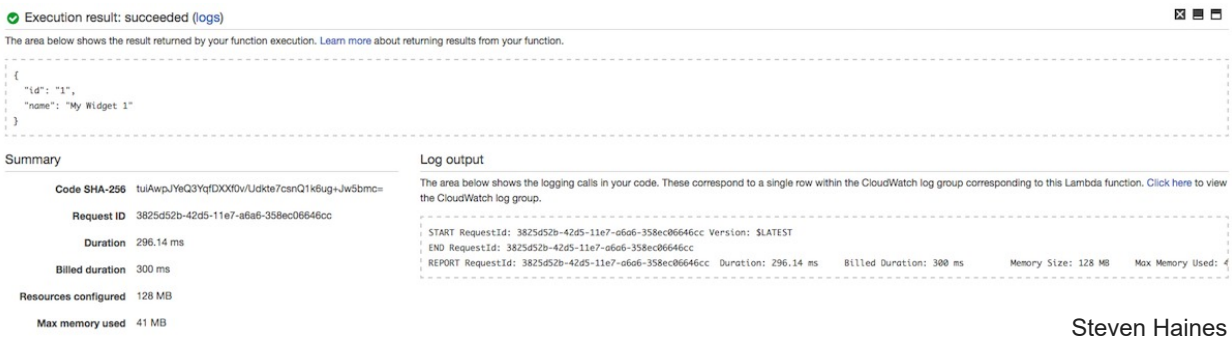

Figure 11. Lambda test execution

The top portion of the page shows the response body, which has an `id` and a `name`. The summary section shows the amount of time that the function took to execute. In this case, you should see `296.14 ms` followed by `300 ms`, which is the amount of time billed. You will also see the maximum amount of memory that was used, in this case `41 MB`. The log output shows any logs that you may have written through a `System.out.println()` statement, or by accessing the `Context` object's logger.

For fun you might want to test the function again and notice the change in duration. When I ran it a couple more times, the durations that I observed were, respectively, $4.4\ \text{ms}$ and $0.64\ \text{ms}$. The reason is that the first time the lambda runs, AWS needs to create a container with your JAR file and deploy it to an EC2 instance. Once it has been deployed on an EC2 instance, the function will run very quickly. Note, however, that if you do not access your function for an undetermined period of time, AWS Lambda will remove your container from the EC2 instance and you'll need to absorb that initial deployment overhead again.

If you've got all of this working so far, congratulations! You've built, deployed, and tested your first Lambda function.

Conclusion

In this article we've answered the question, "What is serverless computing, anyway?" You've learned how serverless architectures employ nanoservices (versus microservices) to increase application scalability, while lowering the price of delivery. I introduced the serverless computing execution model and the Function-as-a-Service concept, and explained the relationship of functions, as they are used in AWS Lambda, to functional programming. Finally, you built a Lambda function in Java, then deployed the function to AWS Lambda and tested it in the AWS Lambda console.

In Part 2, we'll add support for Amazon's DynamoDB. Using DynamoDB, you'll setup your Lambda function to manage widgets on the server, rather than creating them on-the-fly. You'll also leverage the AWS SDK to create a Java client application that can invoke your Lambda function from outside of AWS. Finally, you'll use the AWS Identity and Access Management service to create an IAM user, group, and custom policy for your example application, which we'll then build and run.

Steven Haines is an author, educator, architect and cloud expert with a passion for designing and architecting large-scale cloud-based applications. He is a principal software architect at Turbonomic, working with the team responsible for their cloud initiatives.

Follow