SDEV400 6381: Secure Programming in the Cloud

Homework 3: Lambda Function Testing

Mark Kardash

Professor Nathan Braun

University of Maryland Global Campus

20 February 2024

In this document, I will present the results of testing all 3 of my newly created Lambda functions for SDEV400 Homework 3. Each function will be tested with at least three use cases. The first function I will test is one that calculates the area of a circle. But before I can test anything, I must set up a test configuration. The process of configuring and testing the first function is described in detail below, along with screenshots for visual demonstration.

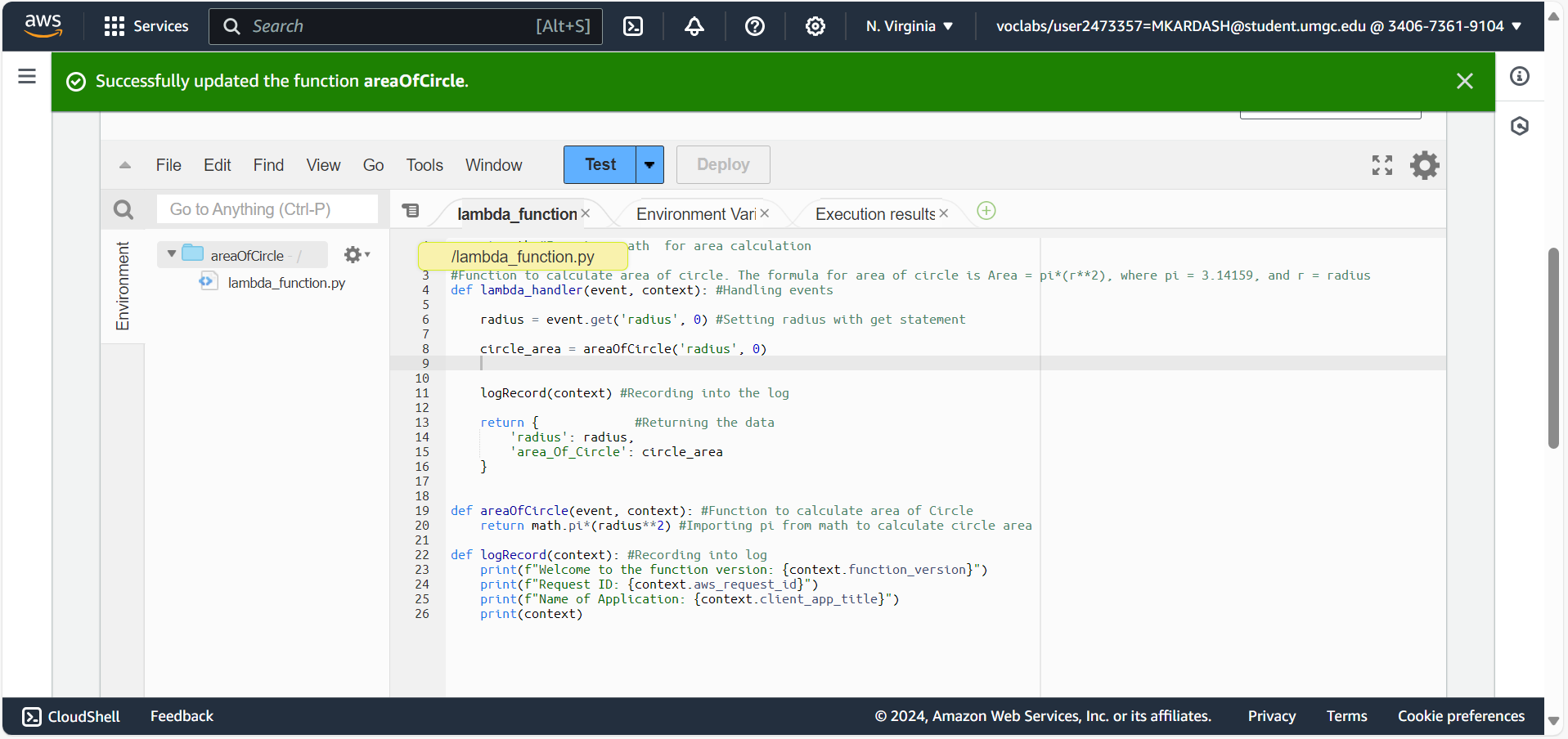


Figure 1: Function for area of circle

The simple function above calculates the area of a circle. The formula for this is Area = pi\*(radius\*\*2), where pi = 3.14159. The function has three components: An event handler, a calculator, and a log recorder. The event handler is responsible for the correct functionality of the other two. The calculator calculates the area, while the log recorder records everything into a log. It calls the version, ID, and name of the function, printing out the context at the end. The two remaining functions will follow the same exact structure.

Once the code has been typed, I deploy the changes by clicking on the “Deploy” button, and get a message that the operation was successful.

Now that the function has been created, a test configuration must be set up. To do this, I clicked on the blue “Test” button at the top. This will cause a test event configuration window to pop up.

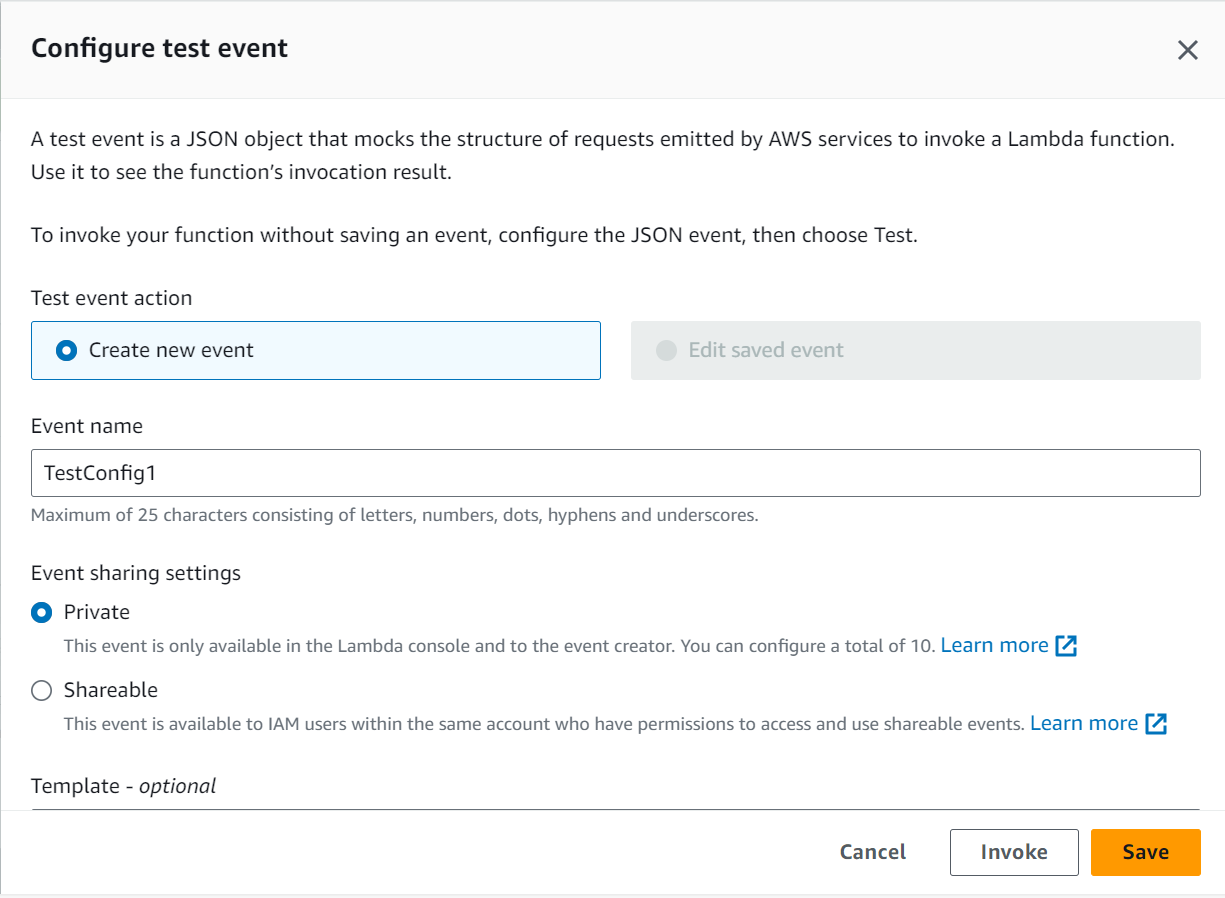


Figure 2: Test Event Configuration Window

In the configuration window, I entered the name of the configuration, and kept the sharing settings as “Private”.

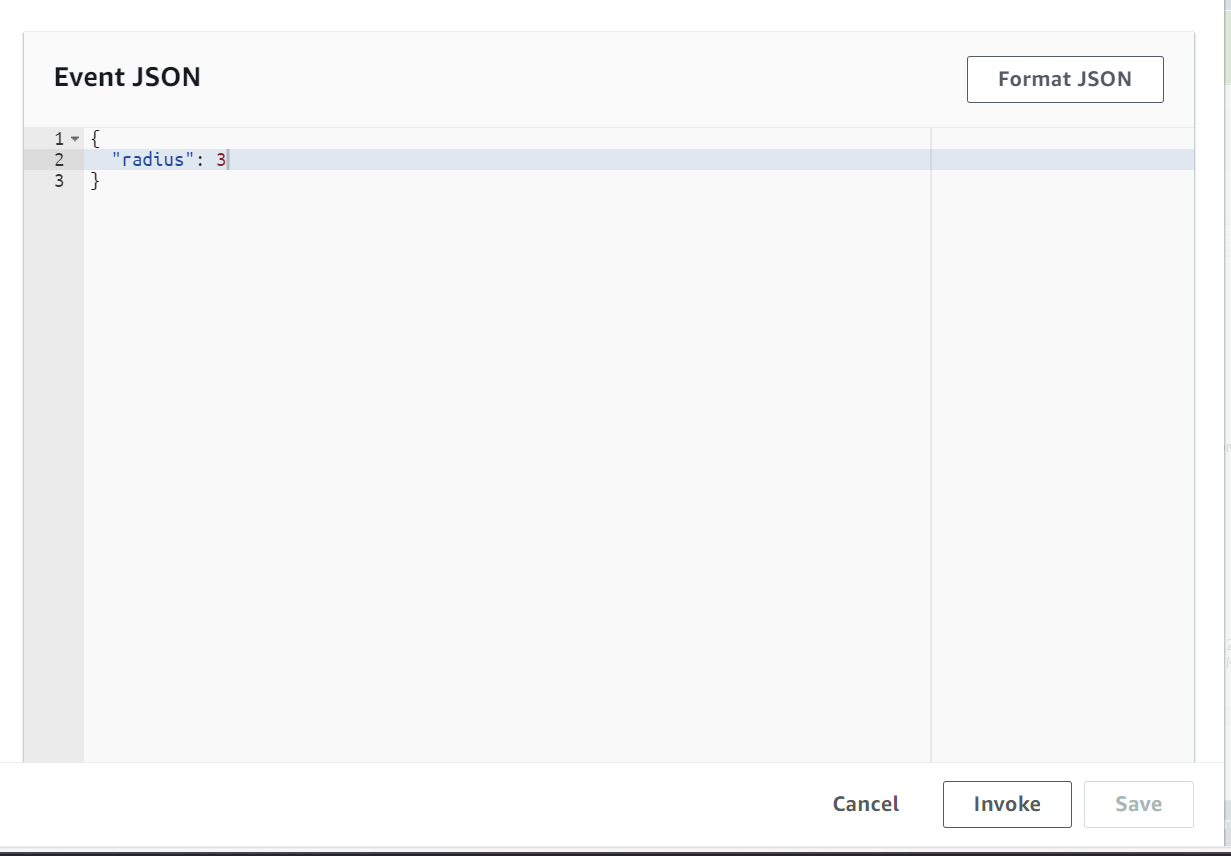


Figure 3: Event Template

Scrolling further down the configuration window, one can find the event template. This is essentially a demonstration of how the input values should be displayed. In my case, all I had to input was the radius. To put the test into action, one must click on the “Invoke” button.

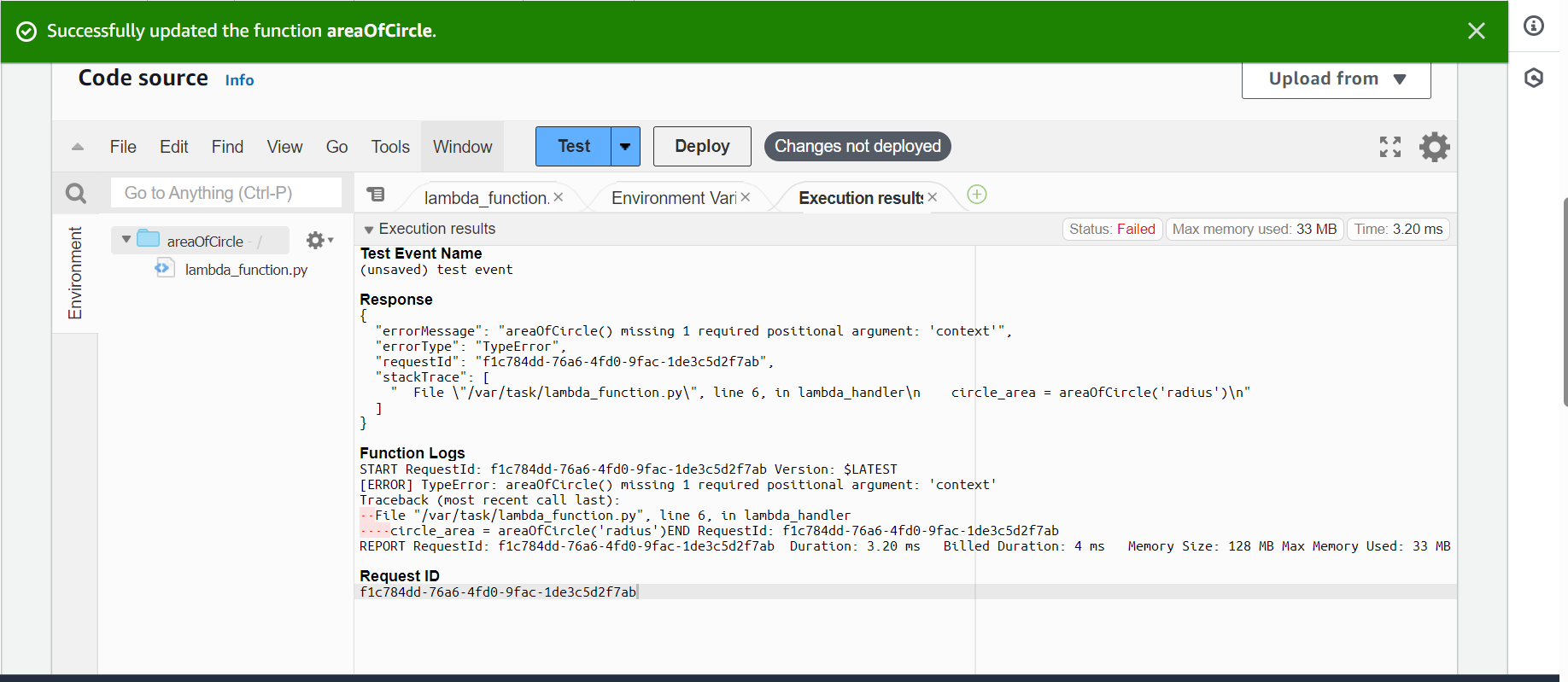


Figure 4: Missing argument error.

My first invocation resulted in a “Missing 1 required positional argument” error, which was a TypeError. I quickly realized the error came from line 6, where I had forgotten to place a 0 for “context” next to the radius. I quickly corrected it and redeployed the changes.



Figure 5: Source of missing argument error.



Figure 6: Corrected missing argument error

With the new change deployed, I was ready to try the test again.

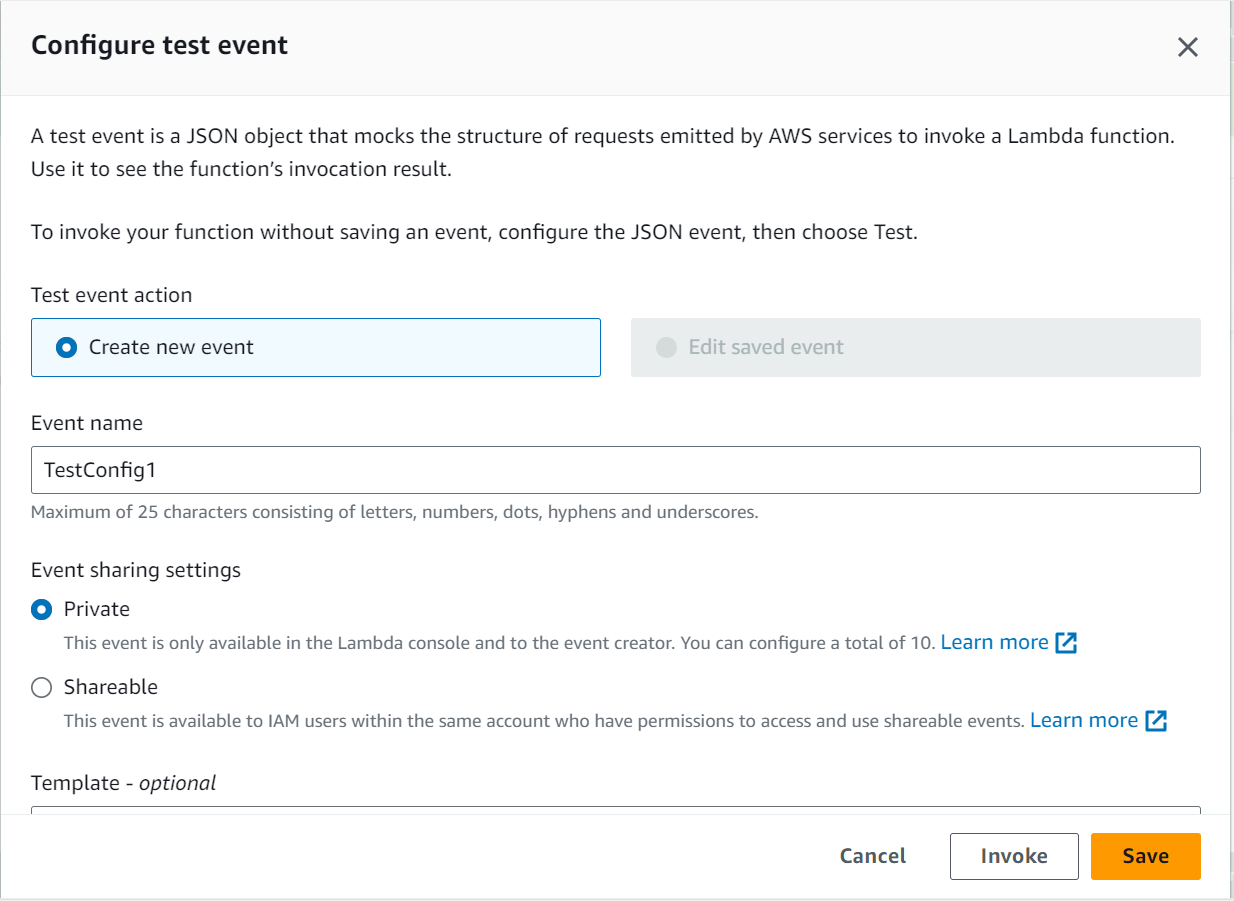


Figure 5: Re-configuring test event

I reconfigured the test event, giving it the exact same name as before, clicked “Invoke”.

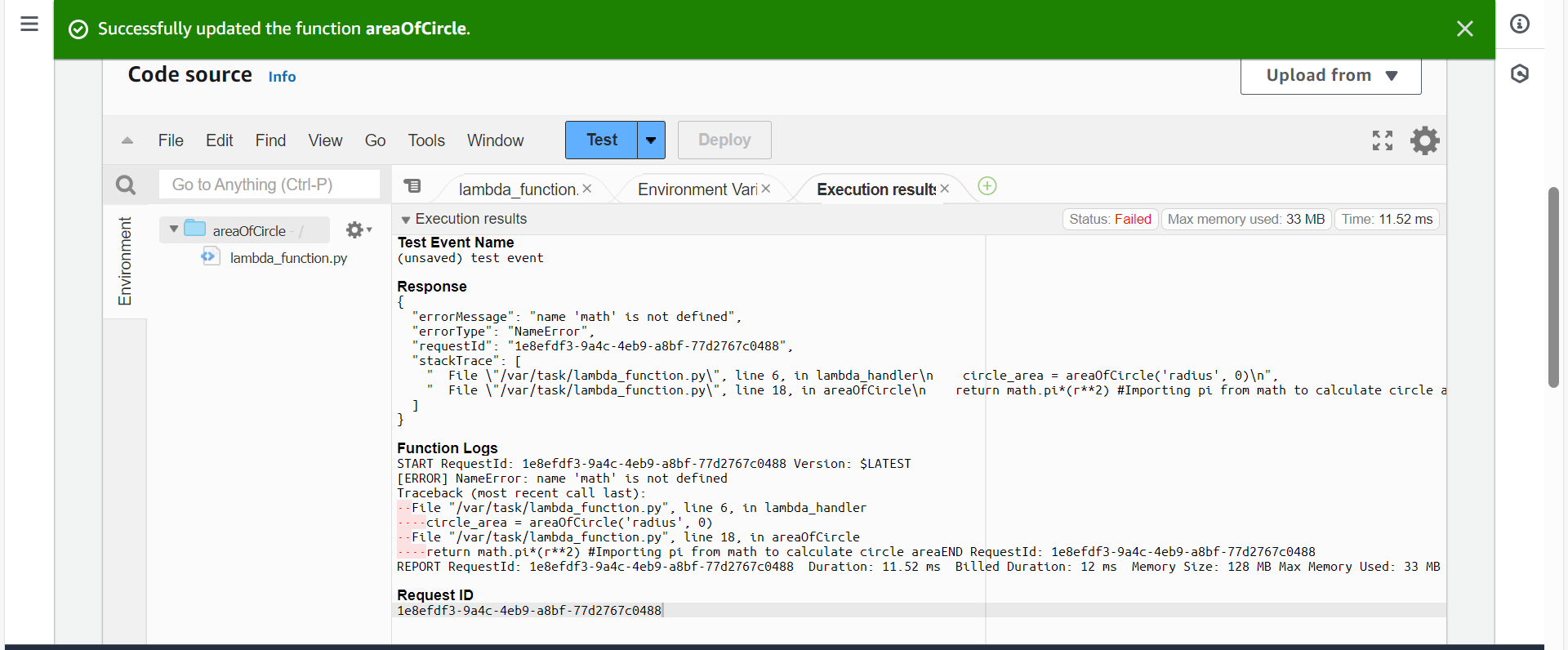
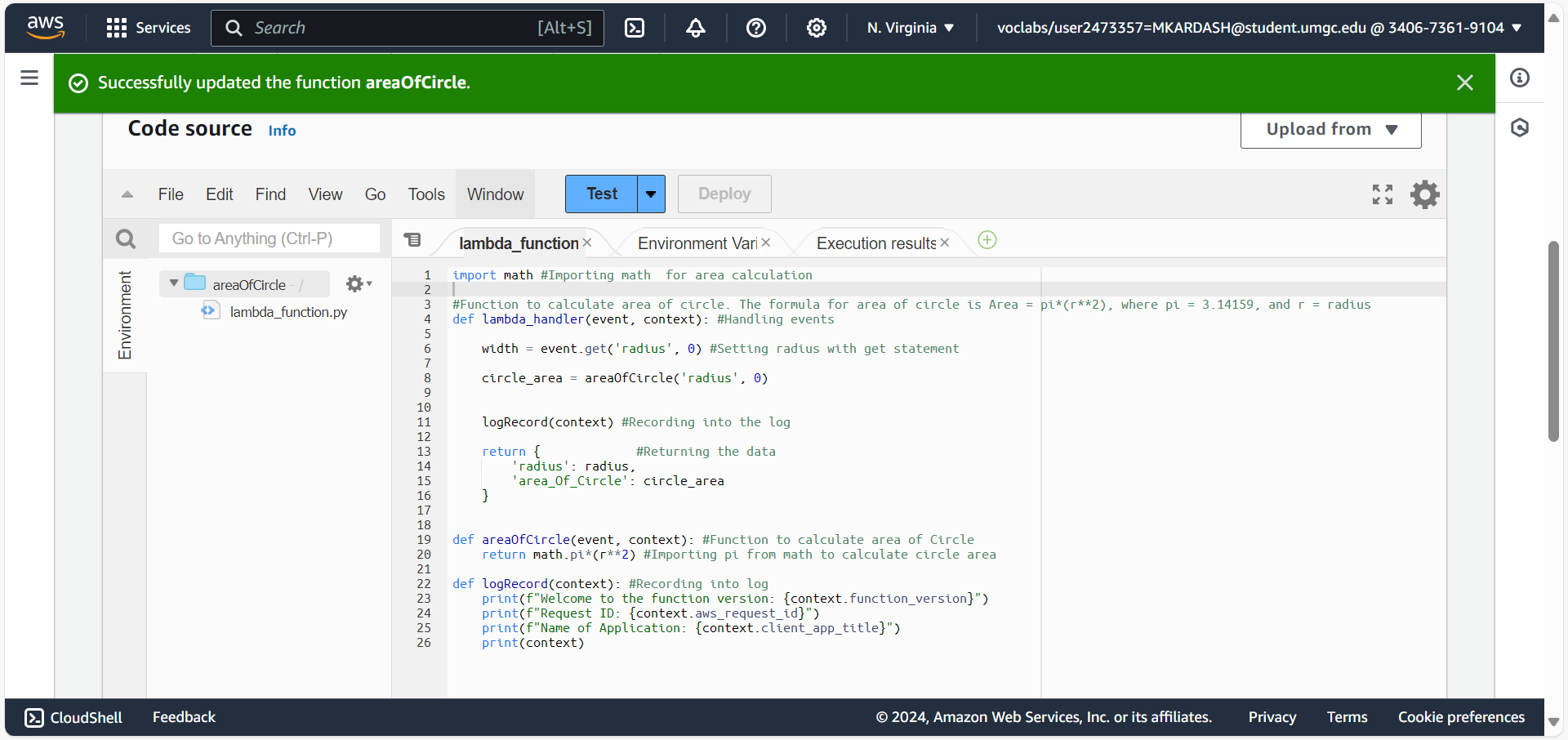


Figure 6: “math” not defined error

This time, I got an error message stating that the name “math”, which I had used to calculate the circle area. After some struggle, I realized I had simply forgotten to import it, so I did that, and tried yet again.

Figure 7: The code with the ”math” function imported.

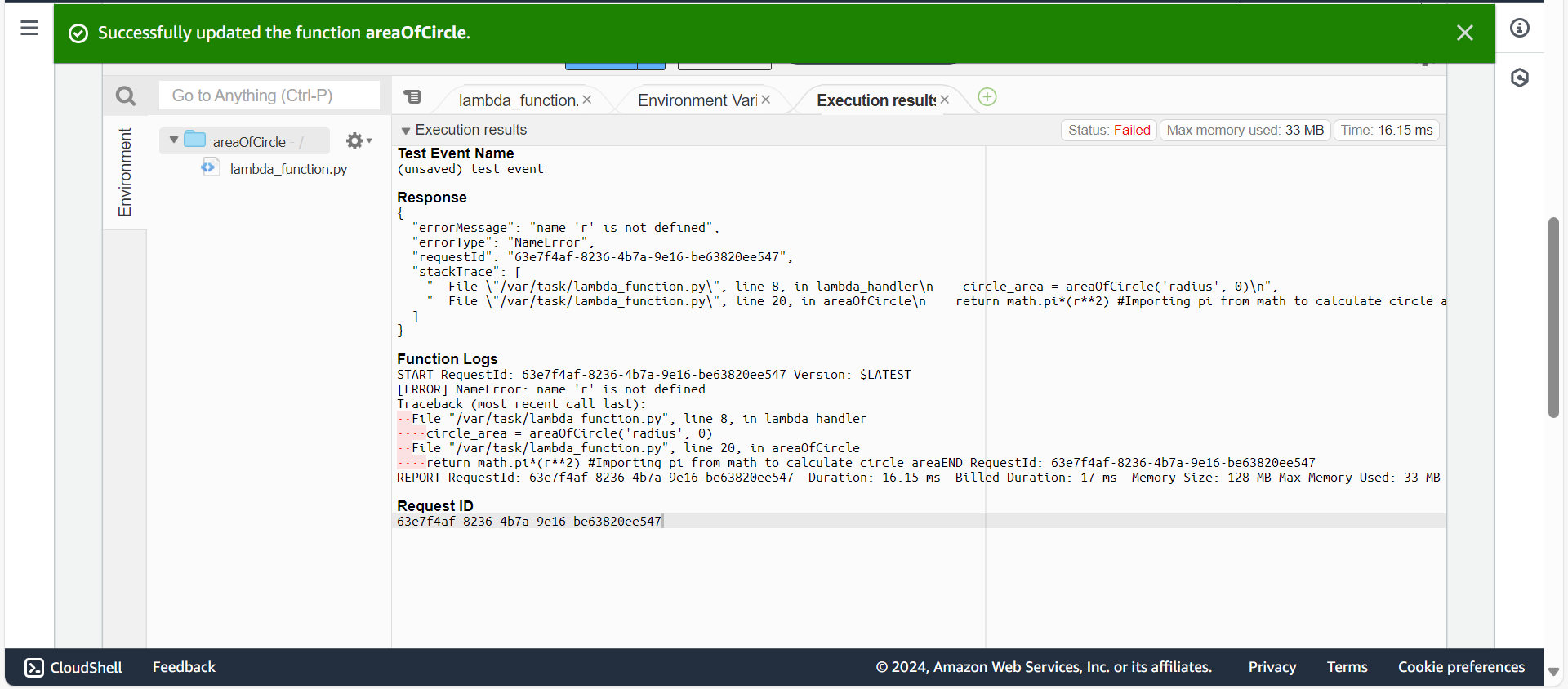


Figure 8: “r” not defined error.

Yet again, I received an error stating that the “r” variable in my formula was not defined. I figured this was probably because my code was not consistent, in some places referring to the radius as “radius”, and in others as “r”. To fix this, I simply spelled out the word “radius” in every single occasion.

While fixing this error, I also noticed I had the “event.get(‘radius’) section incorrectly defined as “width” (This is because originally, I was designing a whole different function). To correct this error, I defined it as “r” as well.

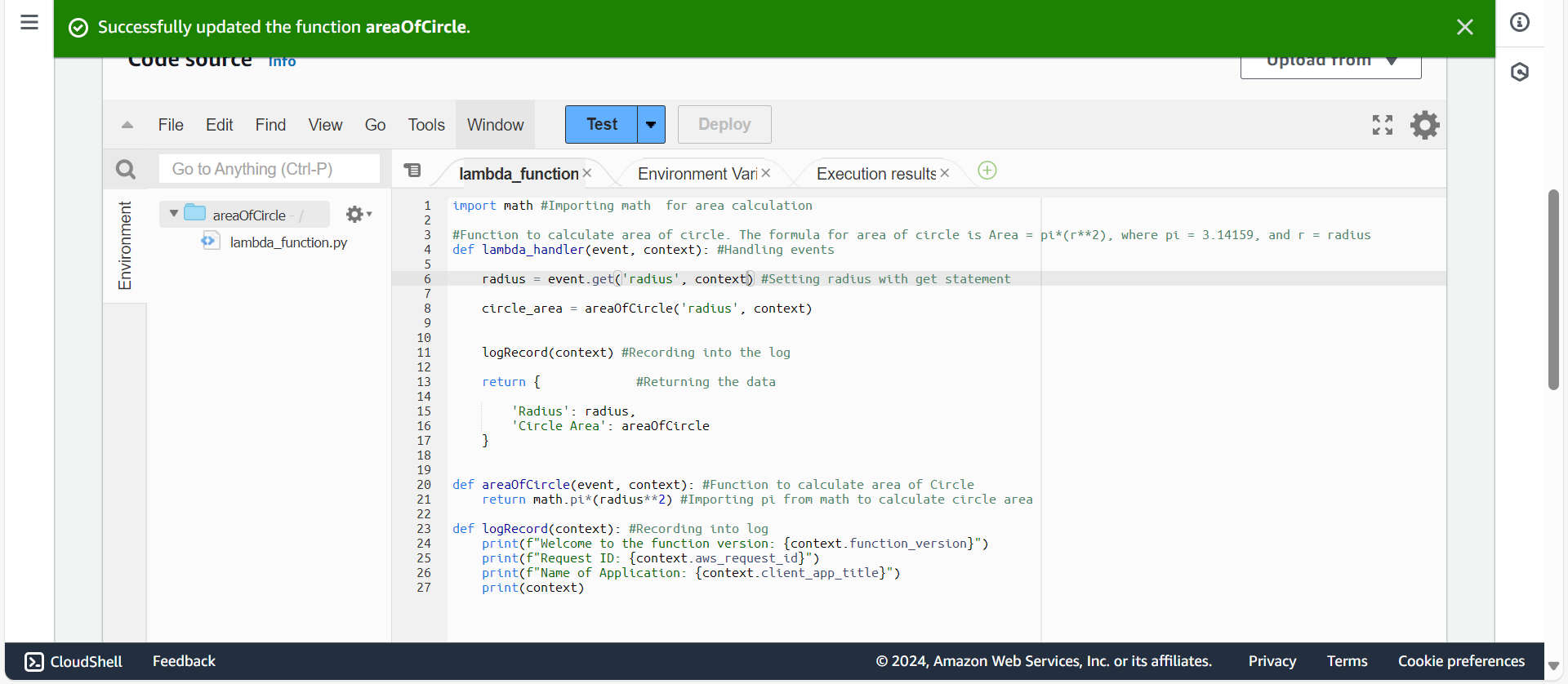


Figure 9: Look of the code after all the new changes

Now, I shall test the code once again, hoping that this time, the configuration finally works.

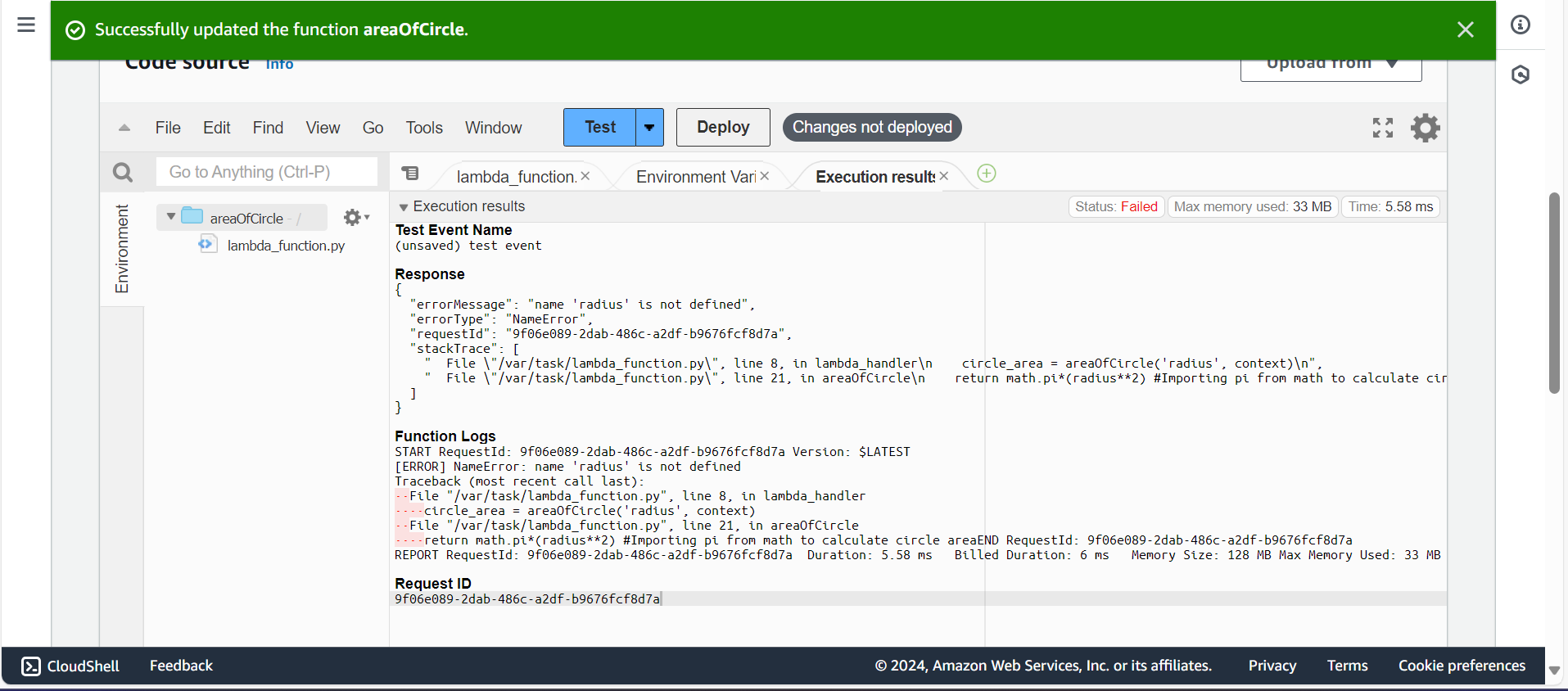


Figure 10: Second “radius not defined” error

After getting the “radius not defined” error yet again, I carefully looked through my code, and noticed that within the “circle\_area” variable, I had accidentally put “radius” in single quotes, while in this sort of function, it should be without any. The fixing of this error can be seen below.



Figure 11: Original line of code with error



Figure 12: Fixed code

In my desperation, I restructured the function once again, printing the radius and area in the event handler. I also asked for the ARN, instead of the function name, within the “logRecord” section.

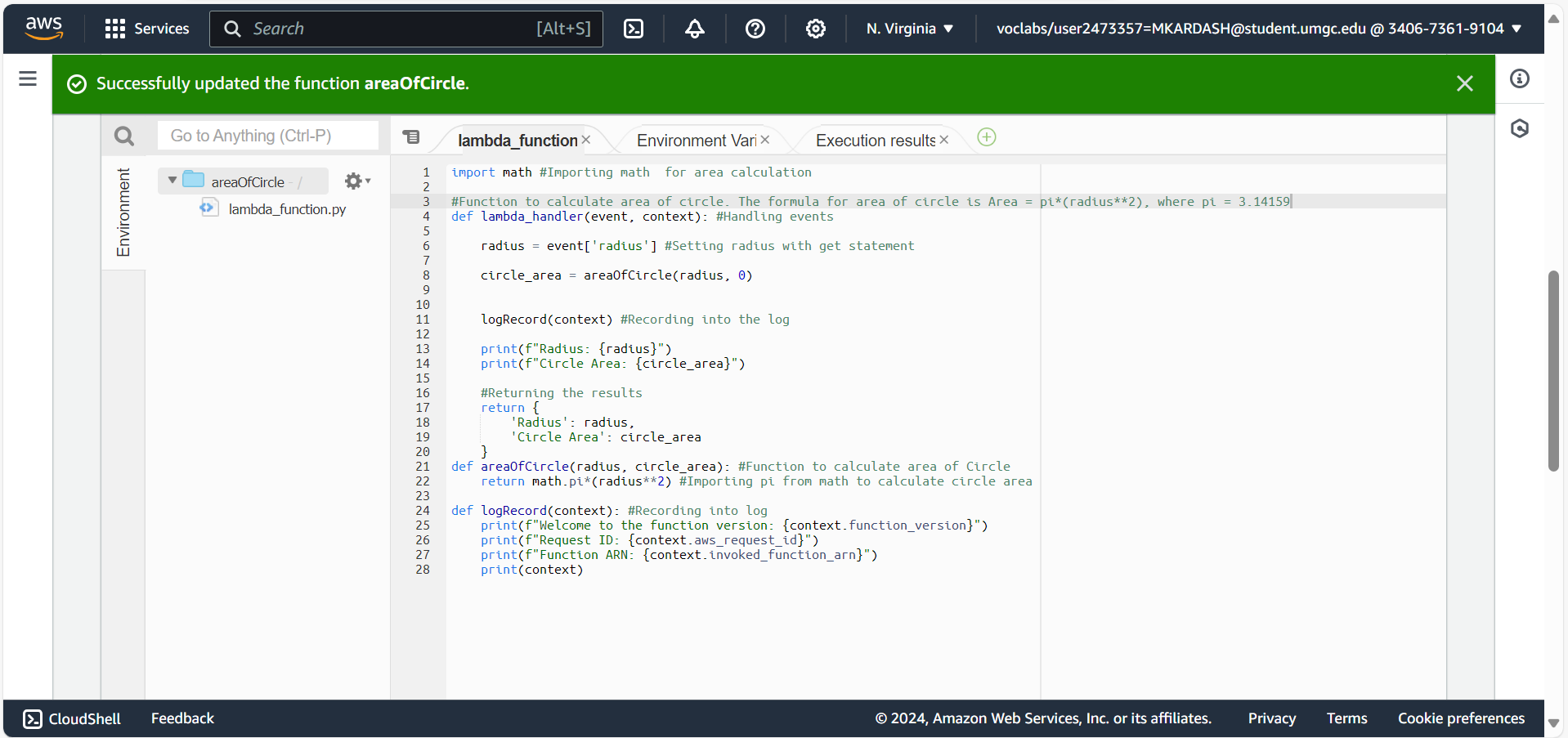


Figure 13: Restructured function

This time, I named my configuration “NewTestConfig1”.

Finally, after this last, complete overhaul, the function actually worked correctly, and calculated the area as expected. As such, I hereby present three use cases that calculate the area of a circle with a different radius. I have also decided that the two remaining functions will keep the format that was presently successful.

Use Cases – Area of Circle Function:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case #: | Input (Radius): | Expected Result (Circle Area): | Actual Result: | Pass/Fail: |
|  | 3 | Radius: 3  Area: 28.274 | Radius: 3  Area: 28.274333882308138 | Pass |
|  | 5.5 | Radius: 5.5  Area: 95.83 | Radius: 5.5  Area: 95.03317777109125 | Pass |
|  | 1.1 | Radius: 1.1  Area: 3.8 | Radius: 1.1  Area: 3.8013271108436504 | Pass |

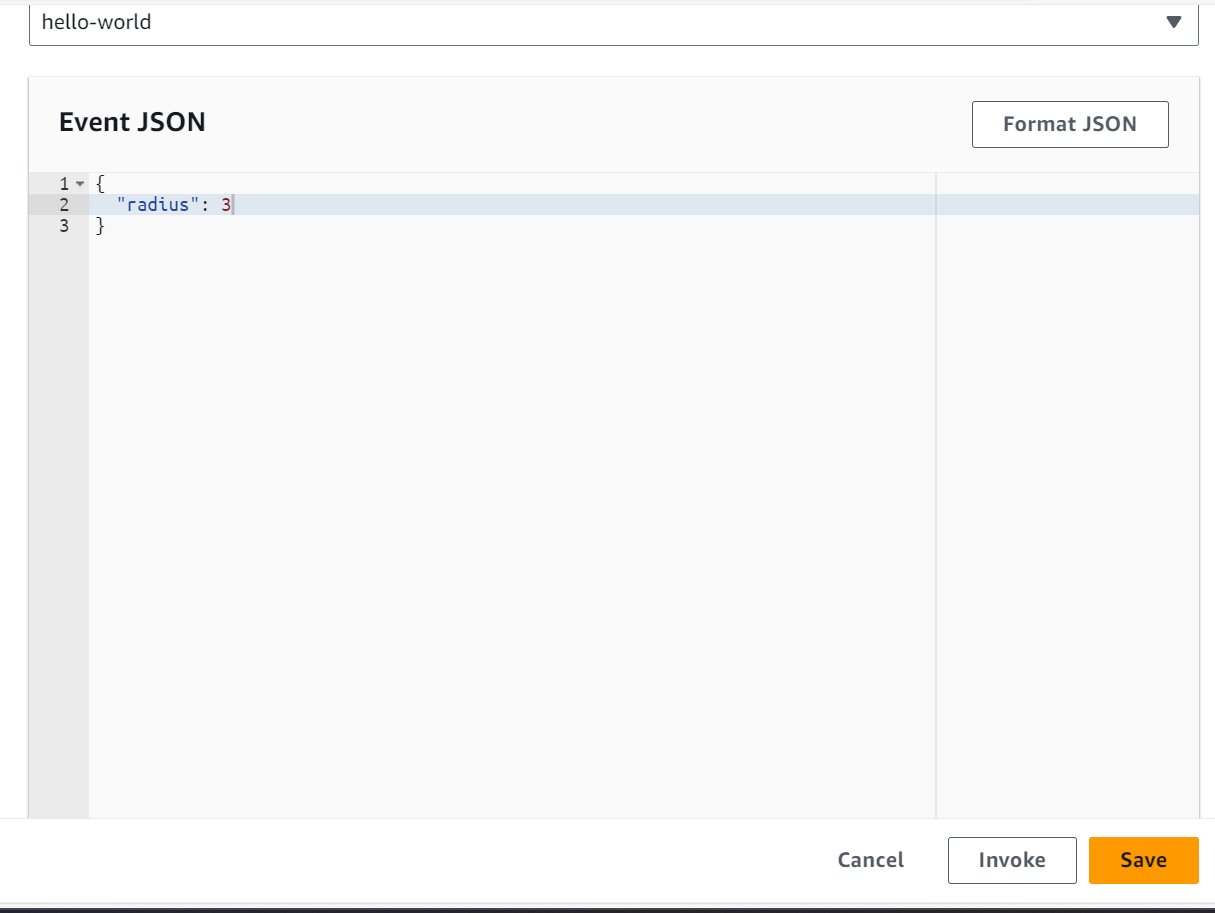


Figure 14: Function 1 Test Case 1 Input

For the first test, we input the very simple number 3, to see how the program handles it.

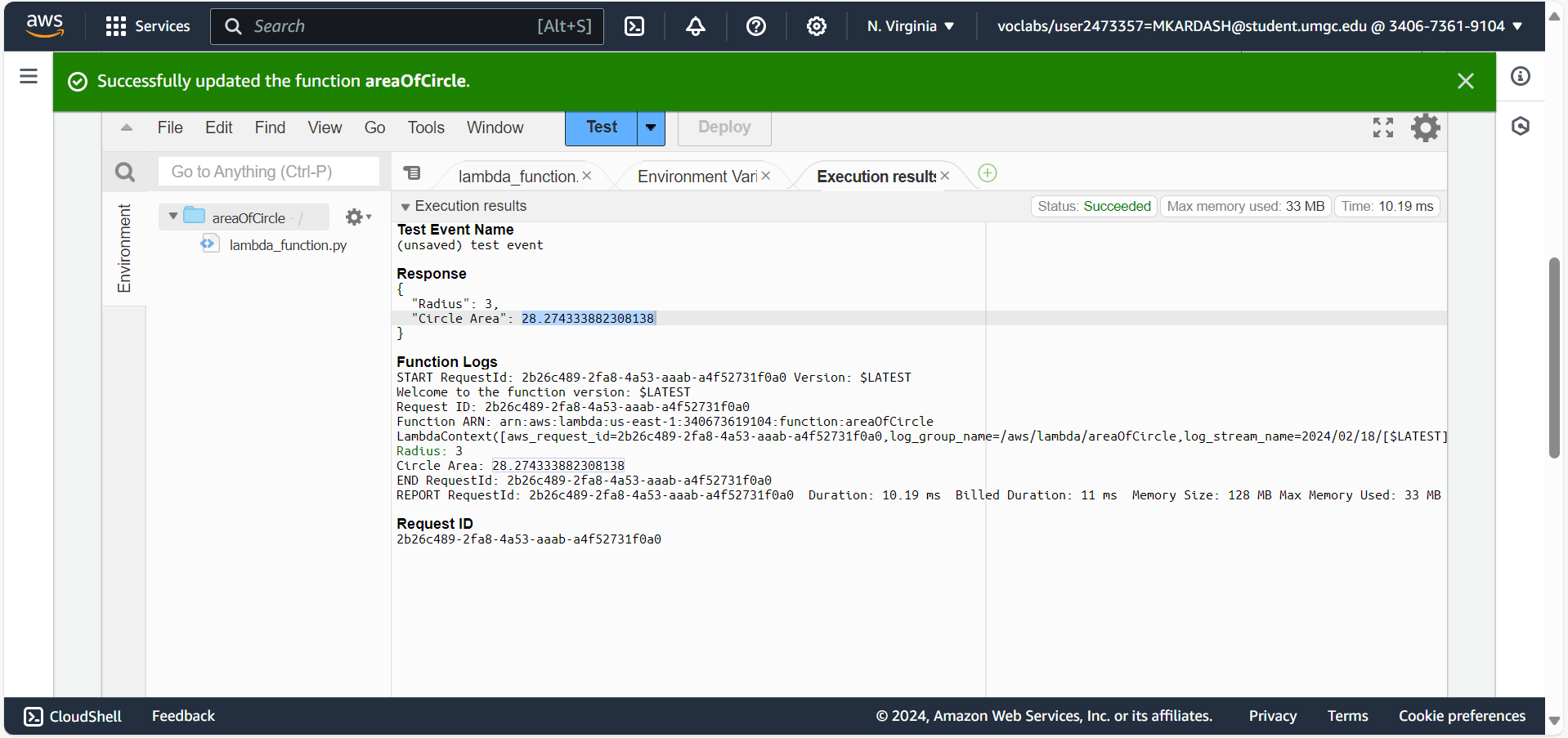


Figure 15: Function 1 Test Case 1 Result

As can be seen in the figure above, the program worked successfully, printing the input, the correct output, and the log data.

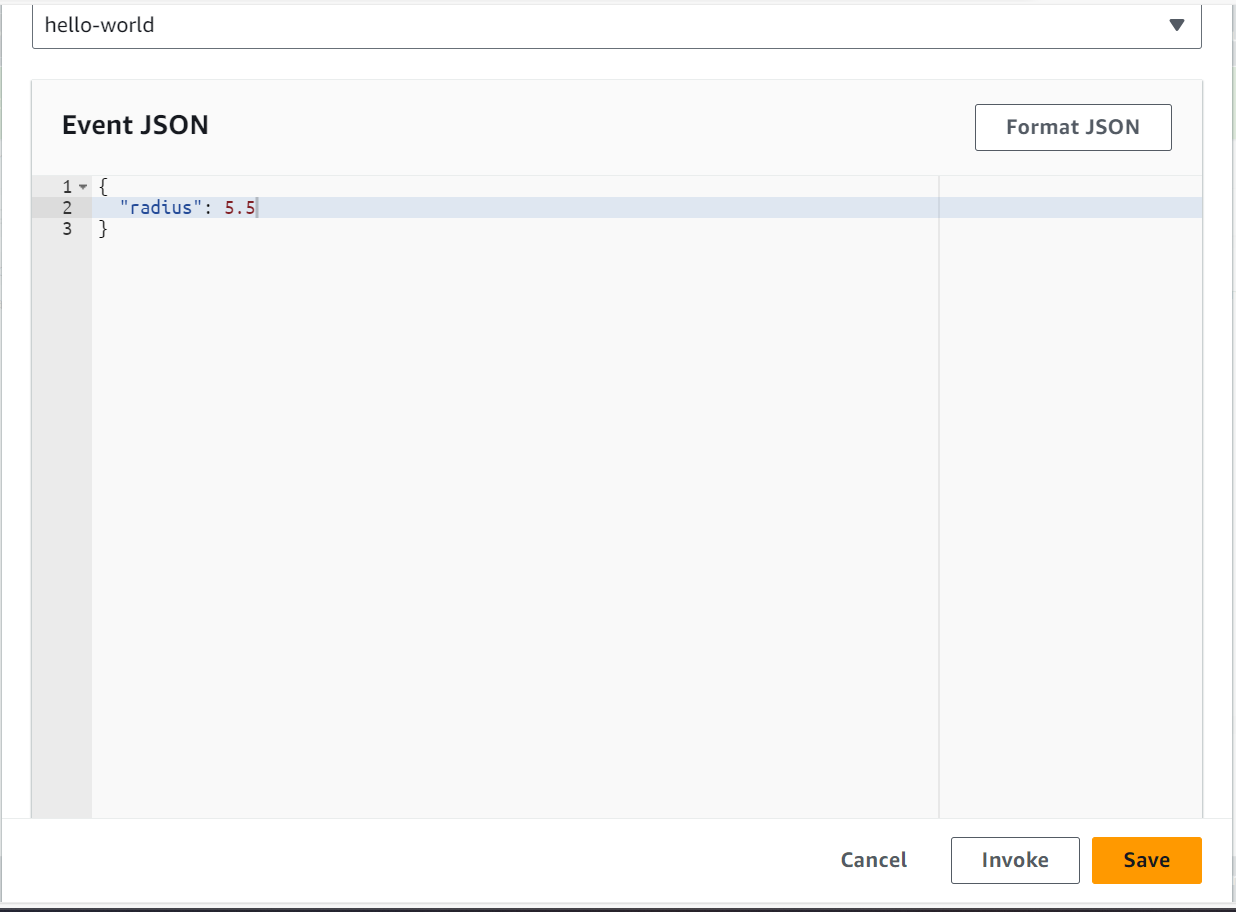


Figure 16: Function 1 Use Case 2 Input

For the second use case, I complicated things a bit by inputting a radius that is a decimal.

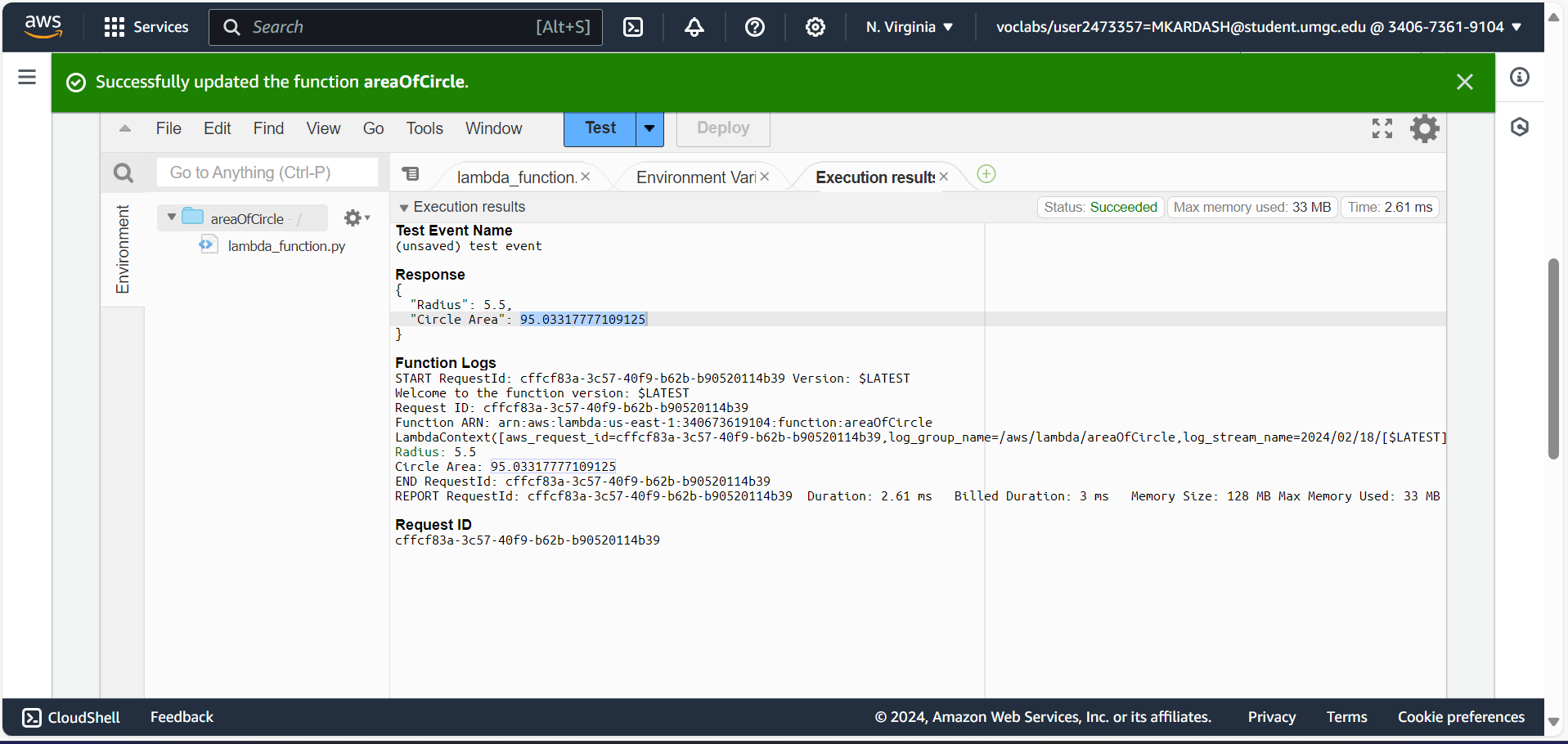


Figure 17: Function 1 Use Case 2 Result

Once again, the response from the software was splendid, calculating everything correctly, and outputting all requested data.

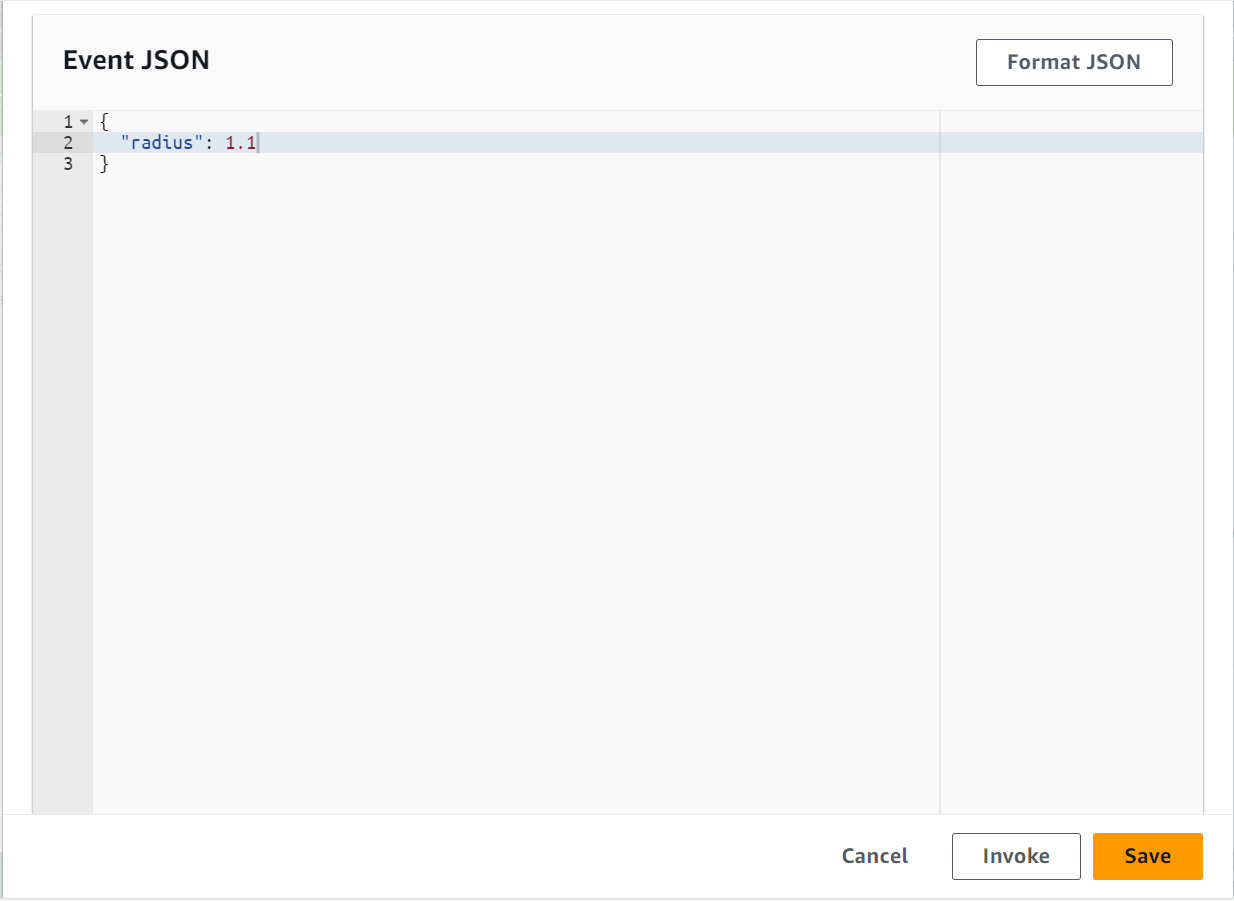


Figure 18: Function 1 Use Case 3 Input

For the final test of this function, I inputted a very small radius that was also a decimal.

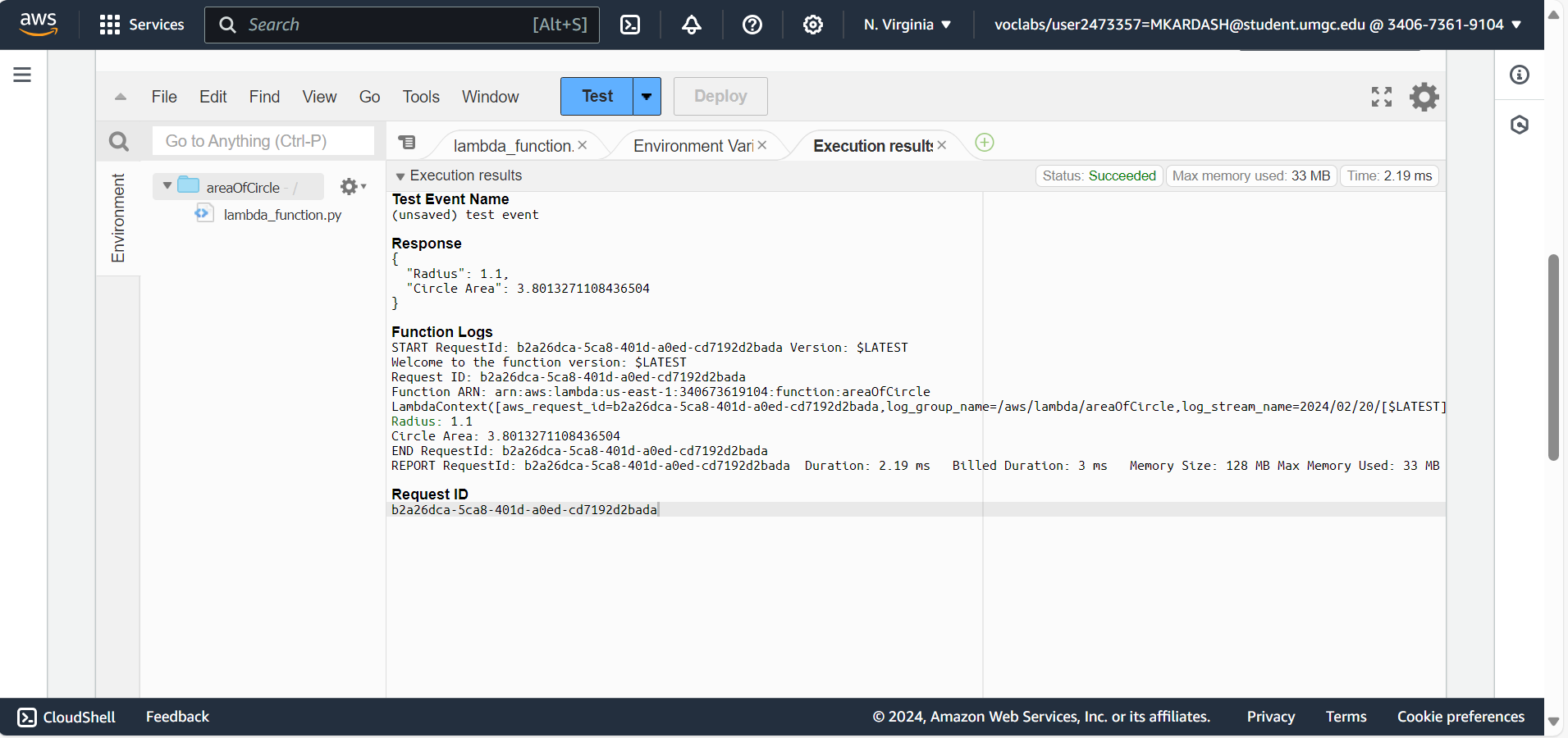


Figure 19: Function 1 Use Case 3 Result

The software, once again, worked perfectly.

Description/Explanation:

For each test, I entered a new number for the radius of the circle (3, 5.5, and 1.1, respectively). When the test is invoked, the function takes the radius from the event via the lambda handler. It inputs it into the area calculator, which performs its calculations with the help of the imported “math” function (Pi is brought in automatically). The handler then prints out both the inputted radius and the calculated area, to let the user know of the result. Meanwhile, the “logRecord” function documents the function version, ID, and ARN. Finally, the results are printed on the screen, the input and calculation from the calculator, and the other data from the log.

The next function to be tested is one that calculates the perimeter of parallelogram. The formula for this is Perimeter = 2\*(a+b), where a = side length, and b = base length. For better understanding, within the function, the formula will be written as Perimeter = 2\*(side+base).

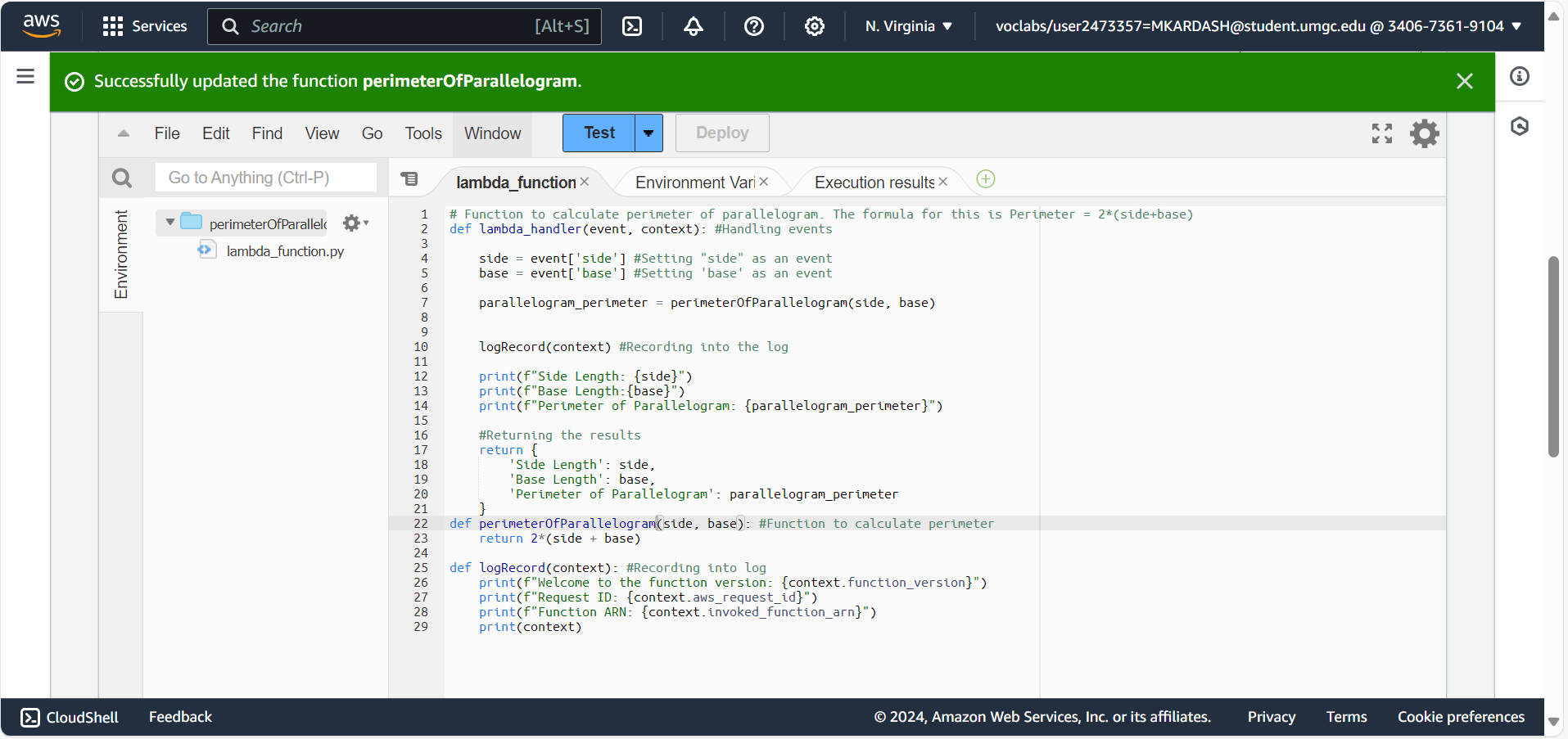


Figure 20: Function for perimeter of a parallelogram

As before, I opened the configuration window and created a new test event, this time naming it “TestConfig2”, pushing the “Invoke” button.

Once again, I performed three tests on the software, documented in three use cases below.

Use Cases: Perimeter of Parallelogram Function

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Use Case # | Input: | Expected Output: | Actual Output: | Pass/Fail: |
|  | Side: 8  Base: 5 | Side: 8  Base: 5  Perimeter of Parallelogram: 26 | Side: 8  Base: 5  Perimeter of Parallelogram: 26 | Pass |
|  | Side: 7.7  Base: 4.9 | Side: 7.7  Base: 4.9  Perimeter of Parallelogram: 25.2 | Side: 7.7  Base: 4.9  Perimeter of Parallelogram: 25.200000000000003 | Pass |
|  | Side: 19  Base: 11 | Side: 19,  Base: 11,  Perimeter of Parallelogram: 60 | Side: 19,  Base: 11,  Perimeter of Parallelogram: 60 | Pass |

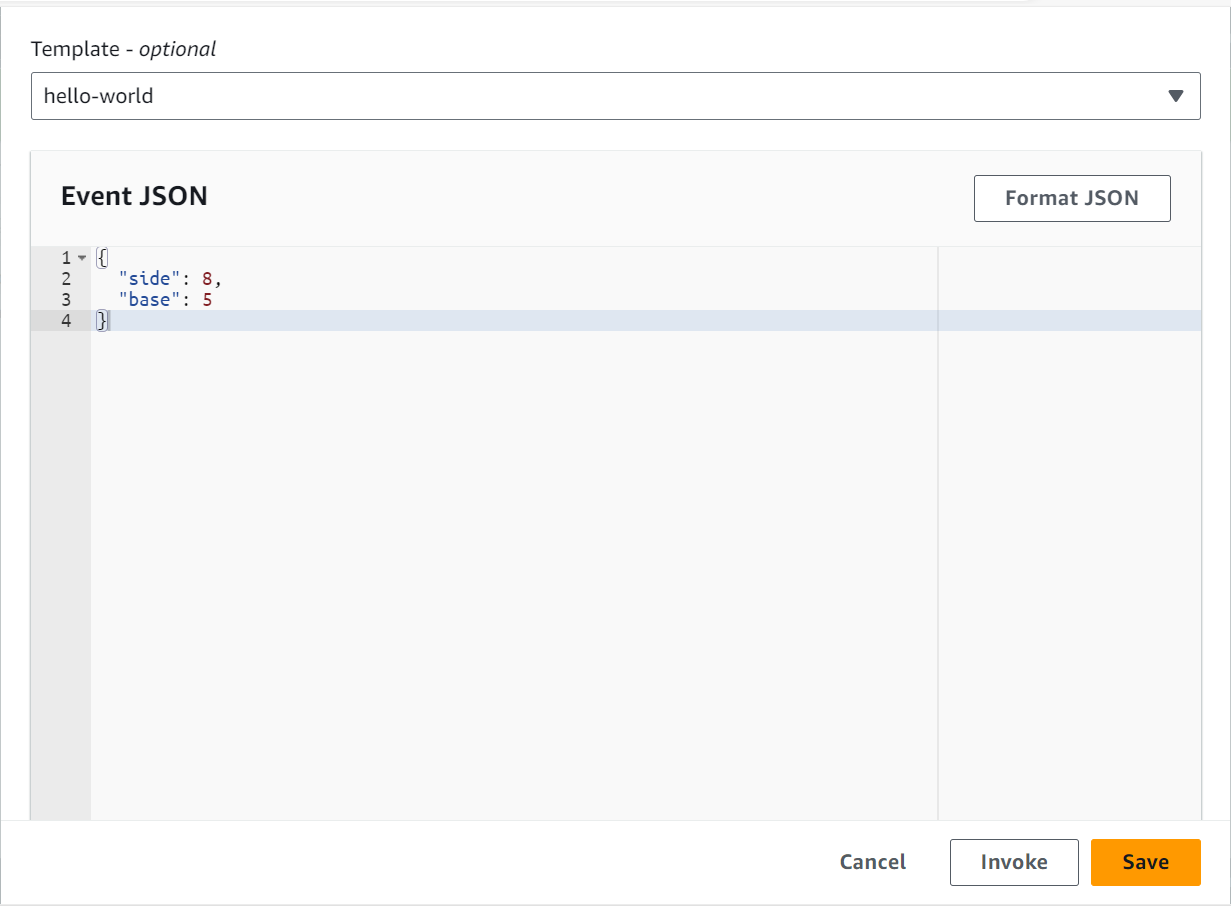


Figure 21: Function 2 Use Case 1 Input

For my first input, I chose the numbers 8 and 5, which should be quite easy for the function to handle.

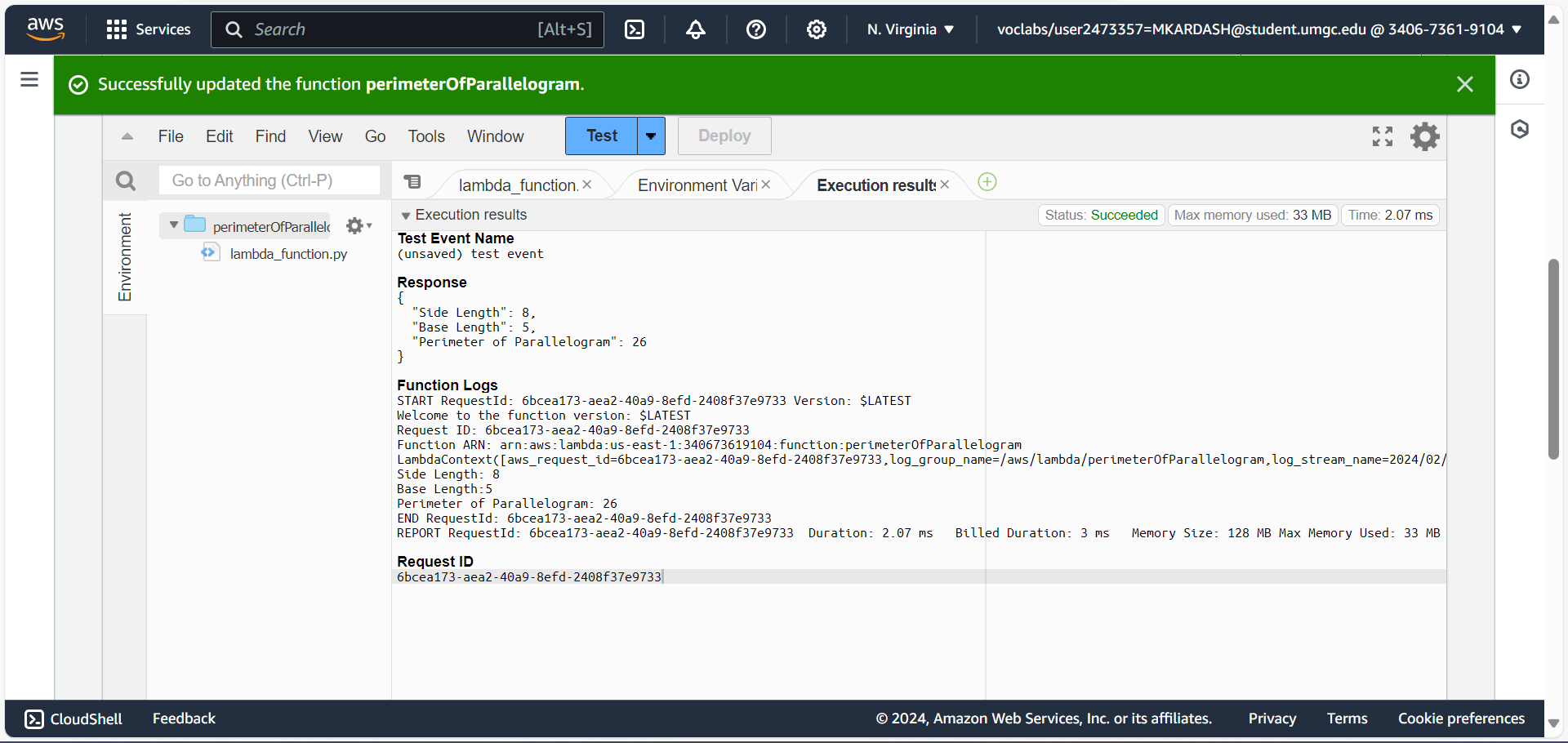


Figure 22: Function 2 Use Case 1 Result

As expected, the function quickly calculated them and gave the correct answer, while also returning the log data.

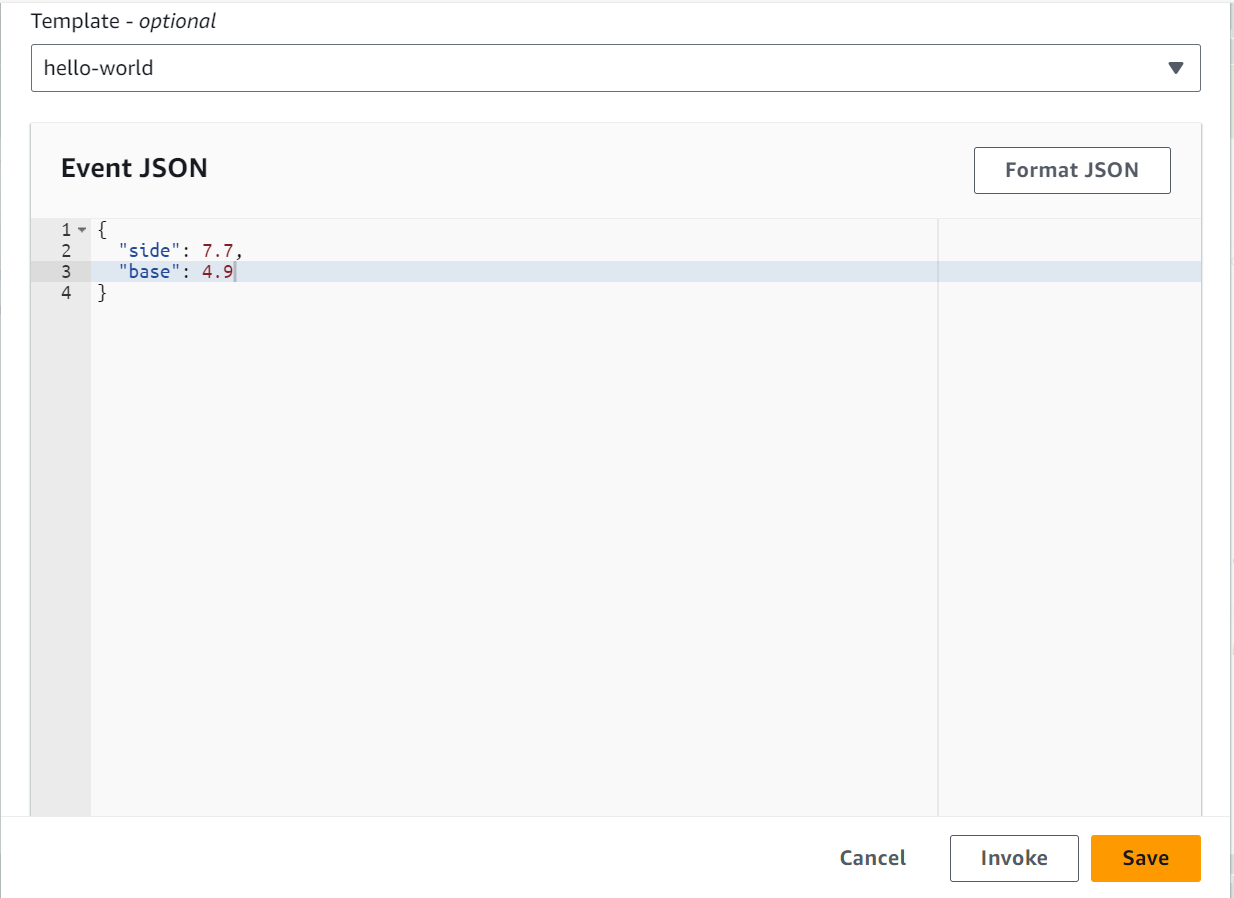


Figure 23: Function 2 Use Case 2 Input

For the second use case, both inputs were decimal numbers.

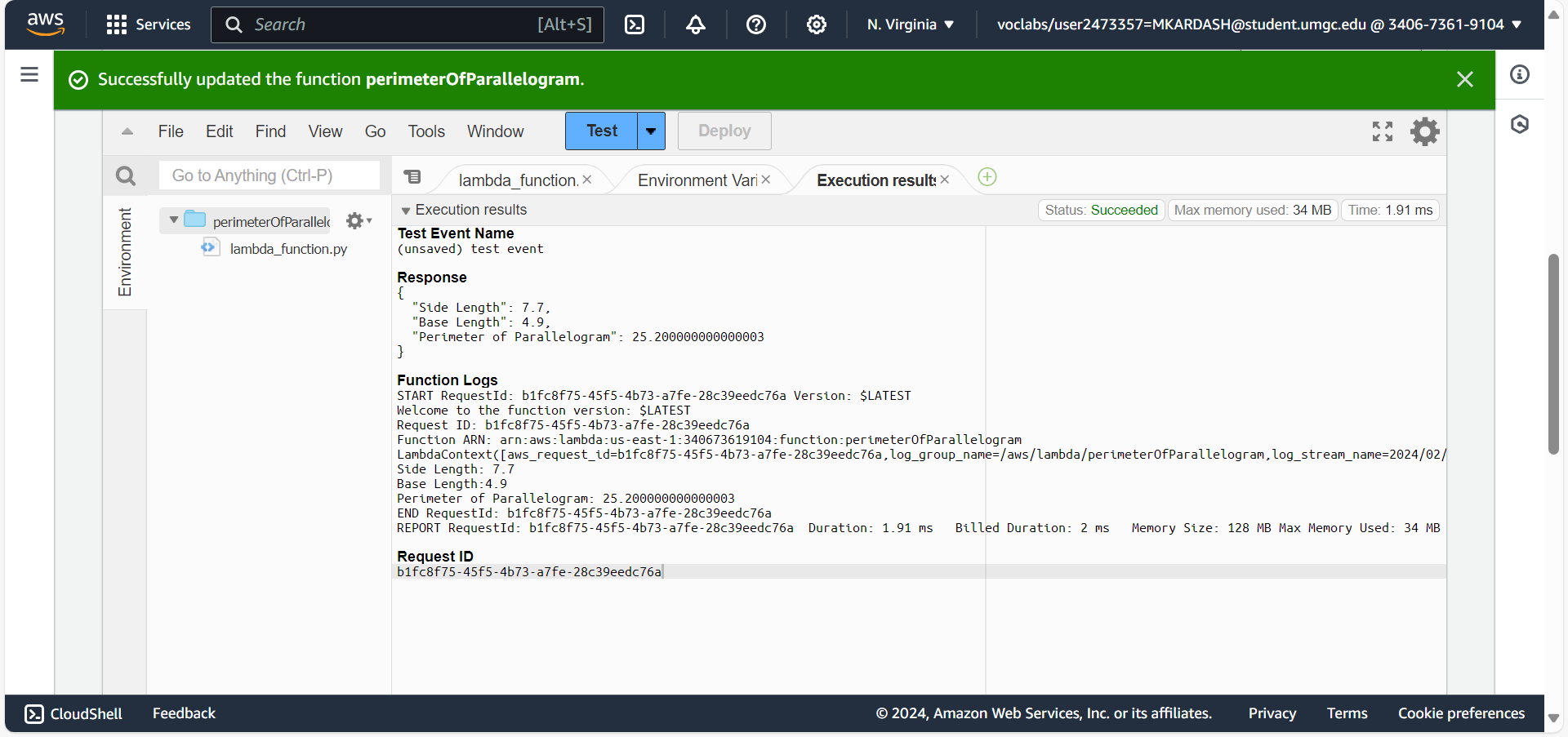


Figure 24: Function 2 Use Case 2 Result

The inputs resulted in a very precise output.

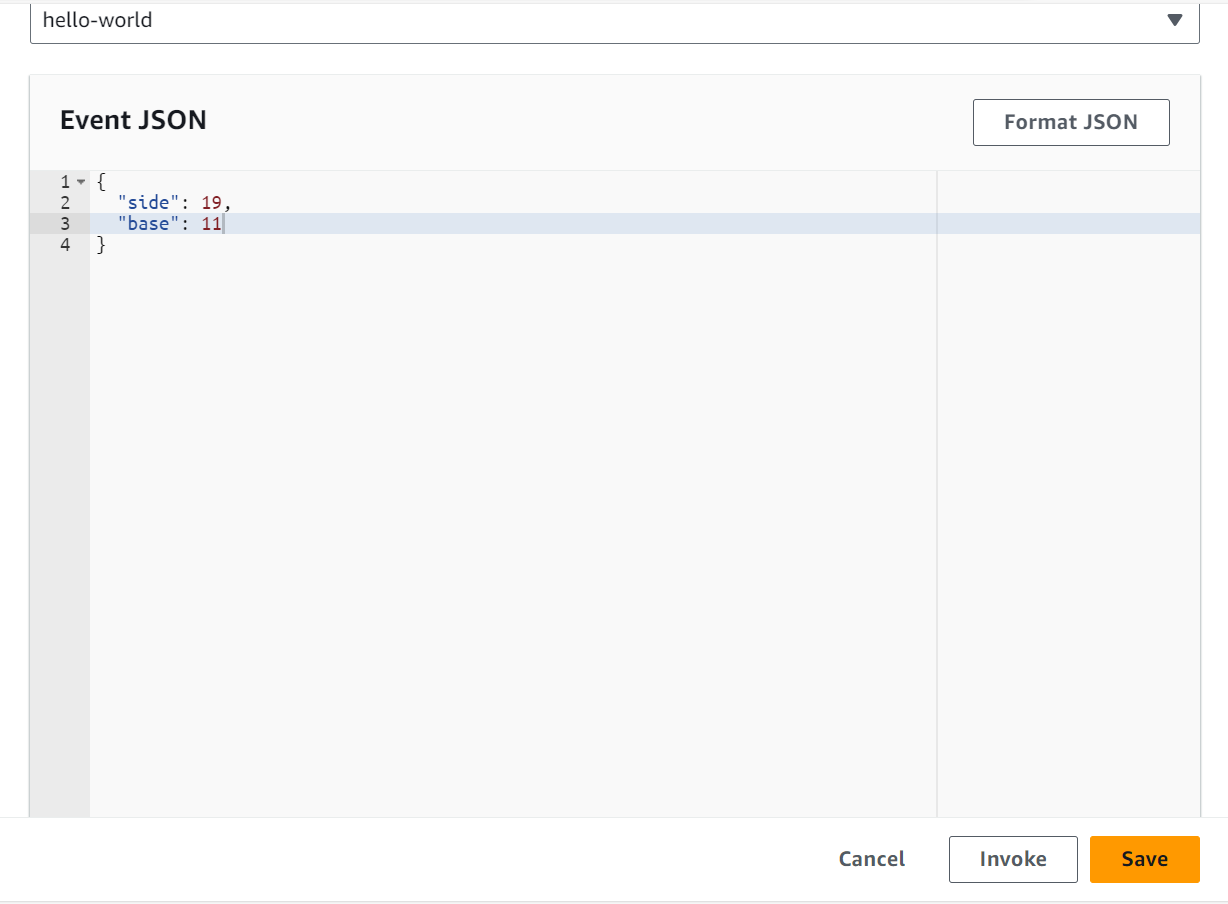


Figure 25: Function 2 Use Case 3 Input

The final use case did not involve decimals, but the input numbers were slightly higher than usual.

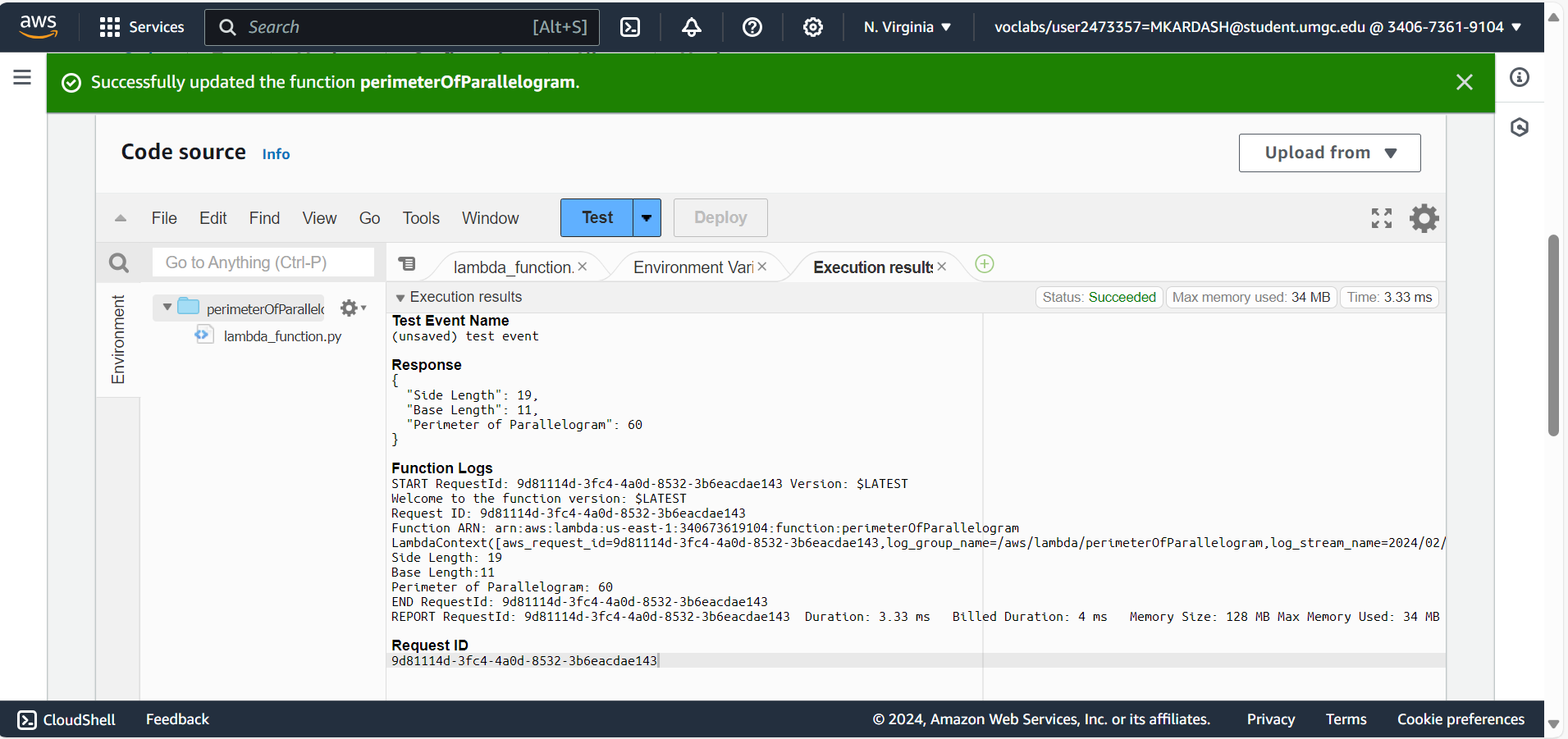


Figure 26: Function 2 Use Case 3 Result

I again received a correct answer.

Description/Explanation:

As with the previous function, the event handler takes in the data, side and base, as events. They are then used within the “perimeterOfParallelogram” function to calculate the perimeter. The inputs are subsequently printed out and returned, along with the calculated output, as they are also recorded into a log via “logRecord”. Meanwhile, “logRecord” additionally records the function version, ARN, and ID as log data.

The last function I have created is one to calculate the volume of a cube. The formula for this is Volume = side^3, where “side” is the length of the cube’s sides.

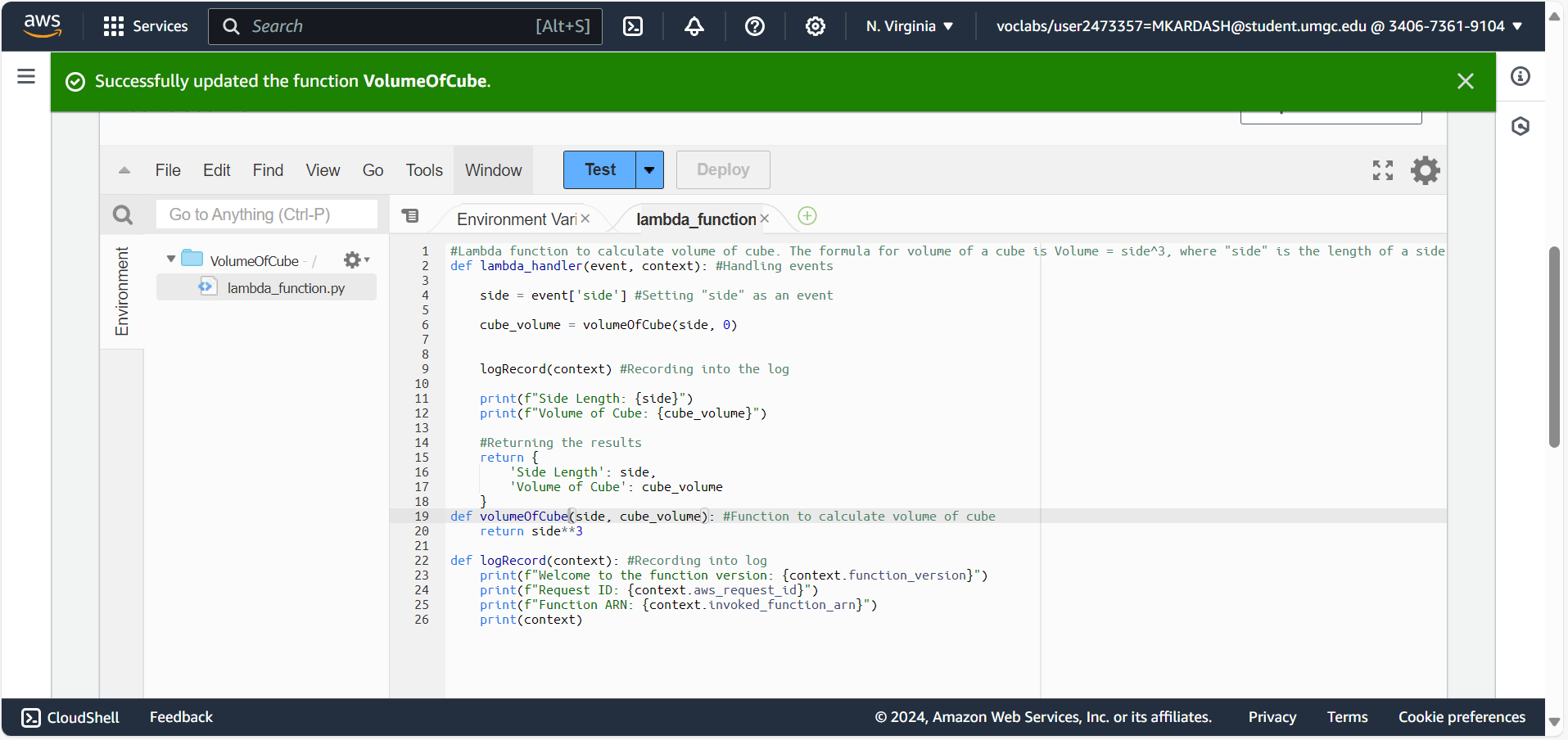


Figure 27: Function to calculate the volume of a cube

As with the previous functions, I created a test configuration that I named “TestConfig3” and pressed the “Invoke” button. Below are my 3 use cases.

Use Cases: Volume of Cube Function

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Use Case # | Input (Side): | Expected Output: | Actual Output: | Pass/Fail: |
|  | 3 | Side: 3  Volume: 27 | Side: 3  Volume: 27 | Pass |
|  | 4.3 | Side: 4.3  Volume: 79.5 | Side: 4.3  Volume: 79.5069999 | Pass |
|  | 7 | Side: 7  Volume: 343 | Side: 7  Volume: 343 | Pass |

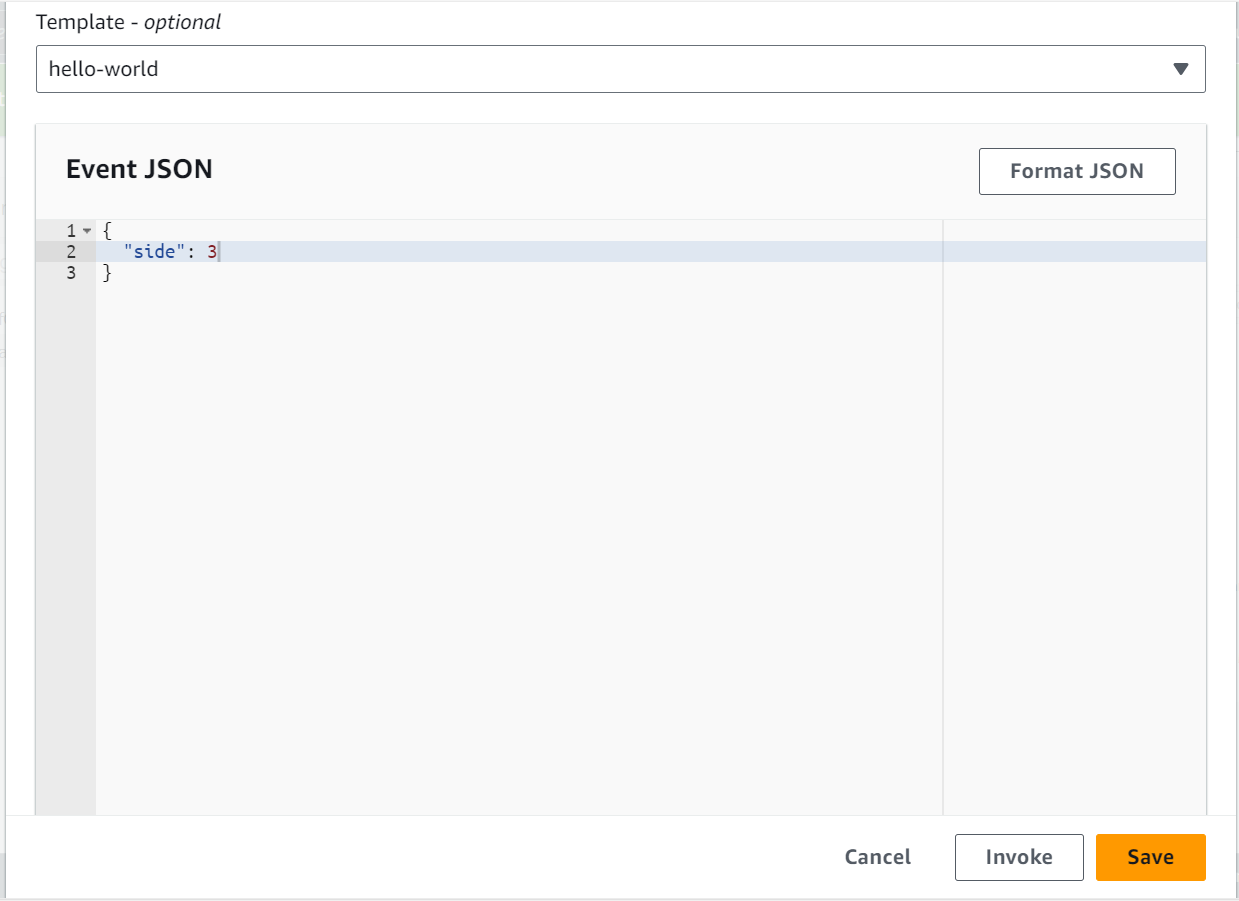


Figure 28: Function 3 Use Case 1 Input

Because of the function’s relative simplicity, I decided to again use an input of 3 for the first case.

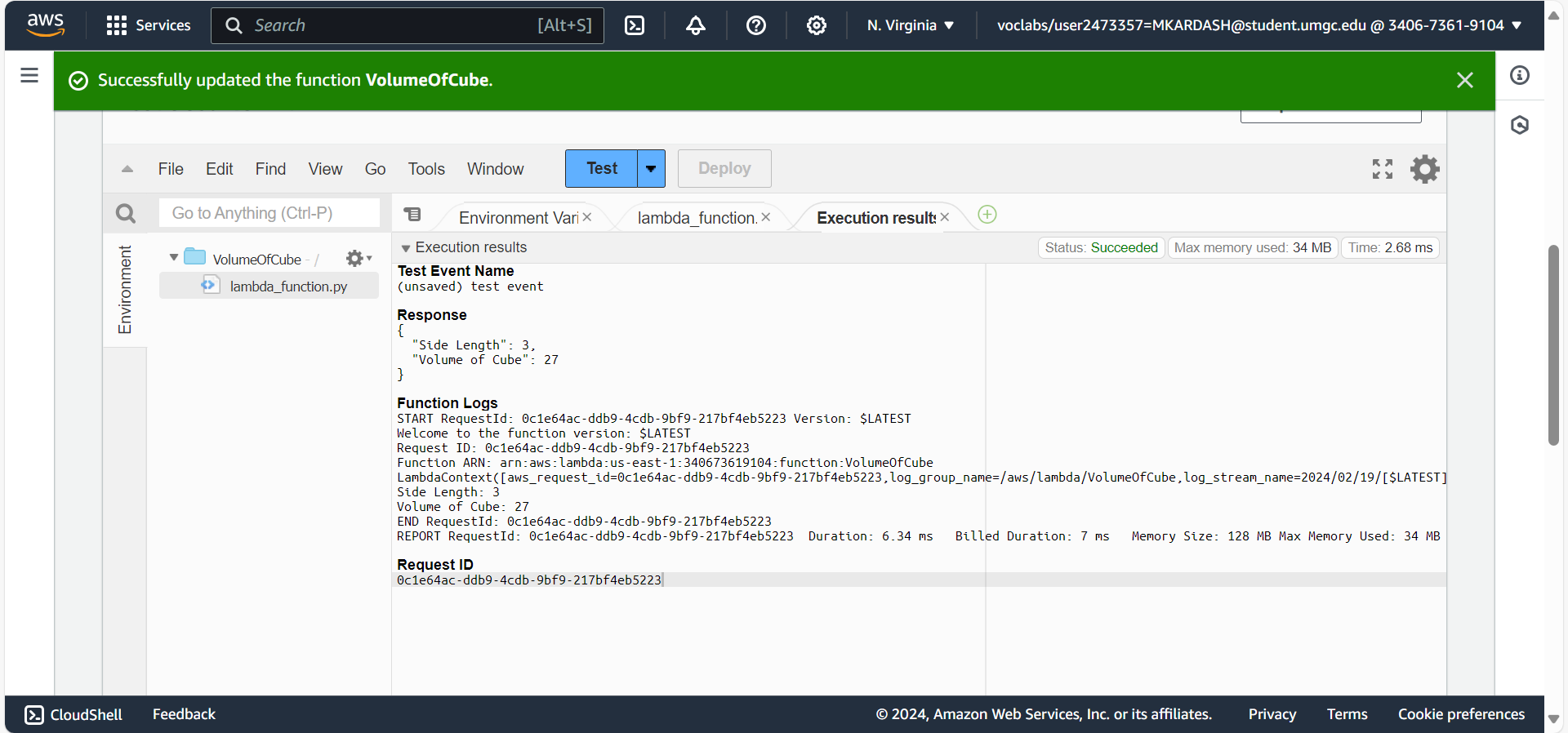


Figure 29: Function 3 Use Case 1 Result

The function returned both the input and the cube of the input (A.K.A The calculated volume of a cube).

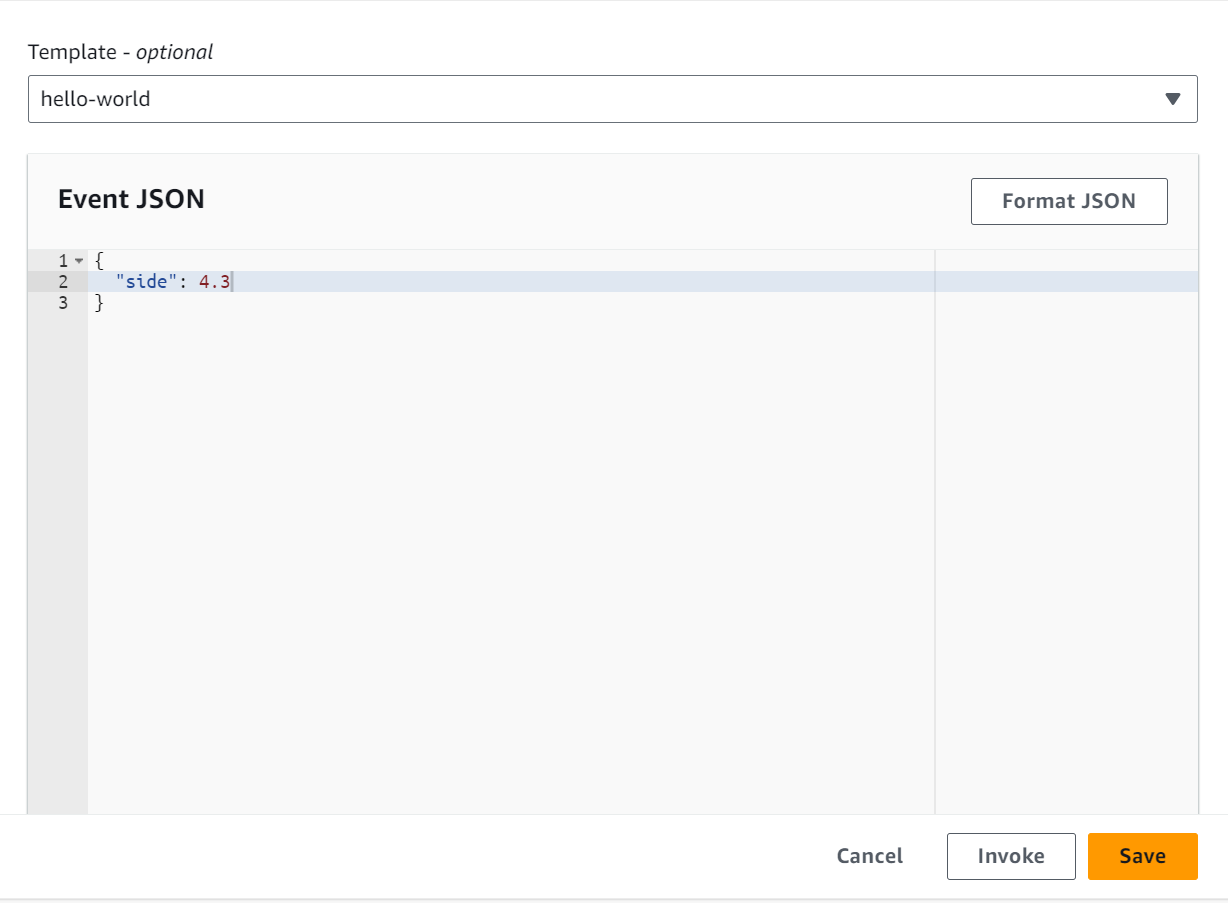


Figure 30: Function 3 Use Case 2 Input

The second case involved decimals again.

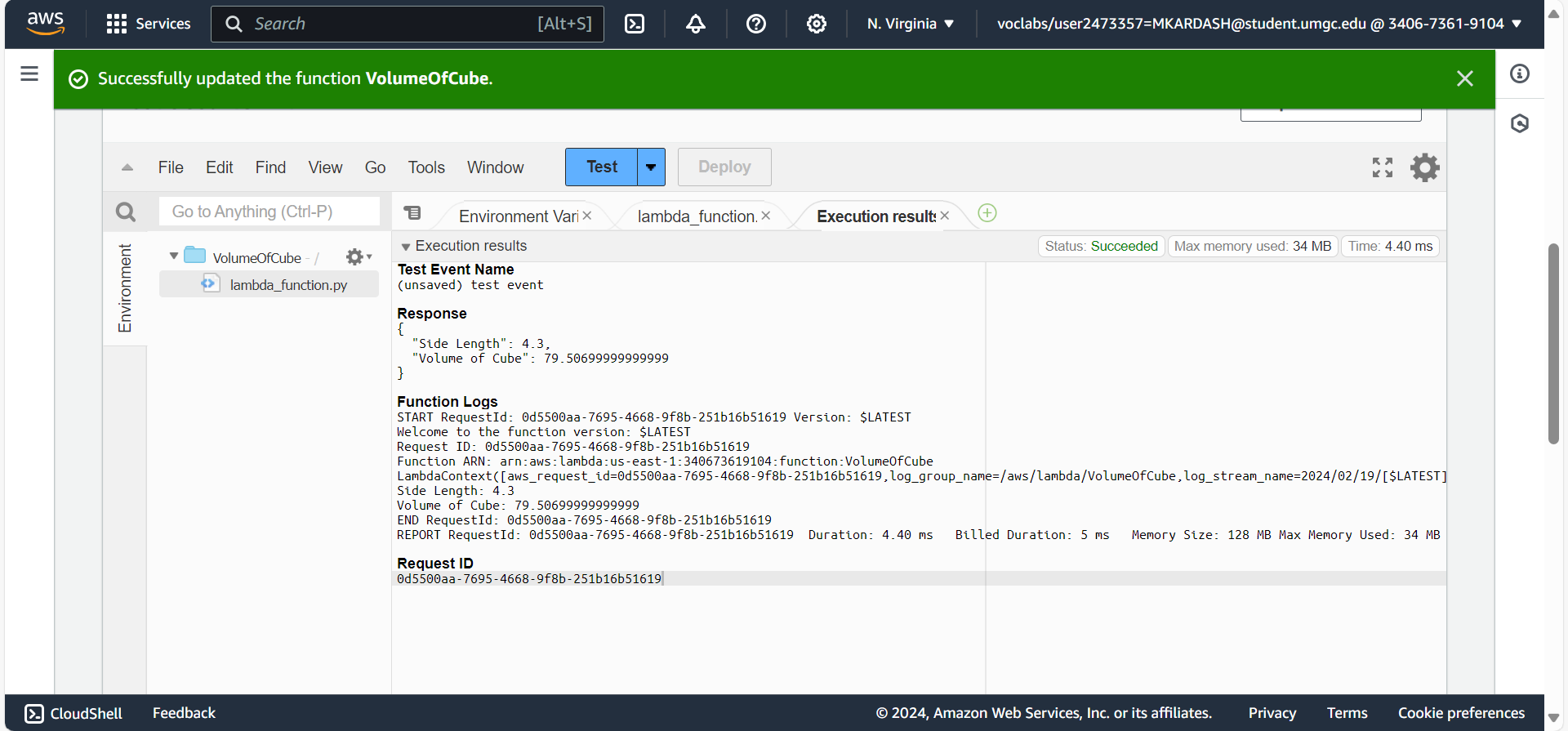


Figure 31: Function 3 Use Case 2 Result

And returned the input, output, and log data, meeting my expectations.

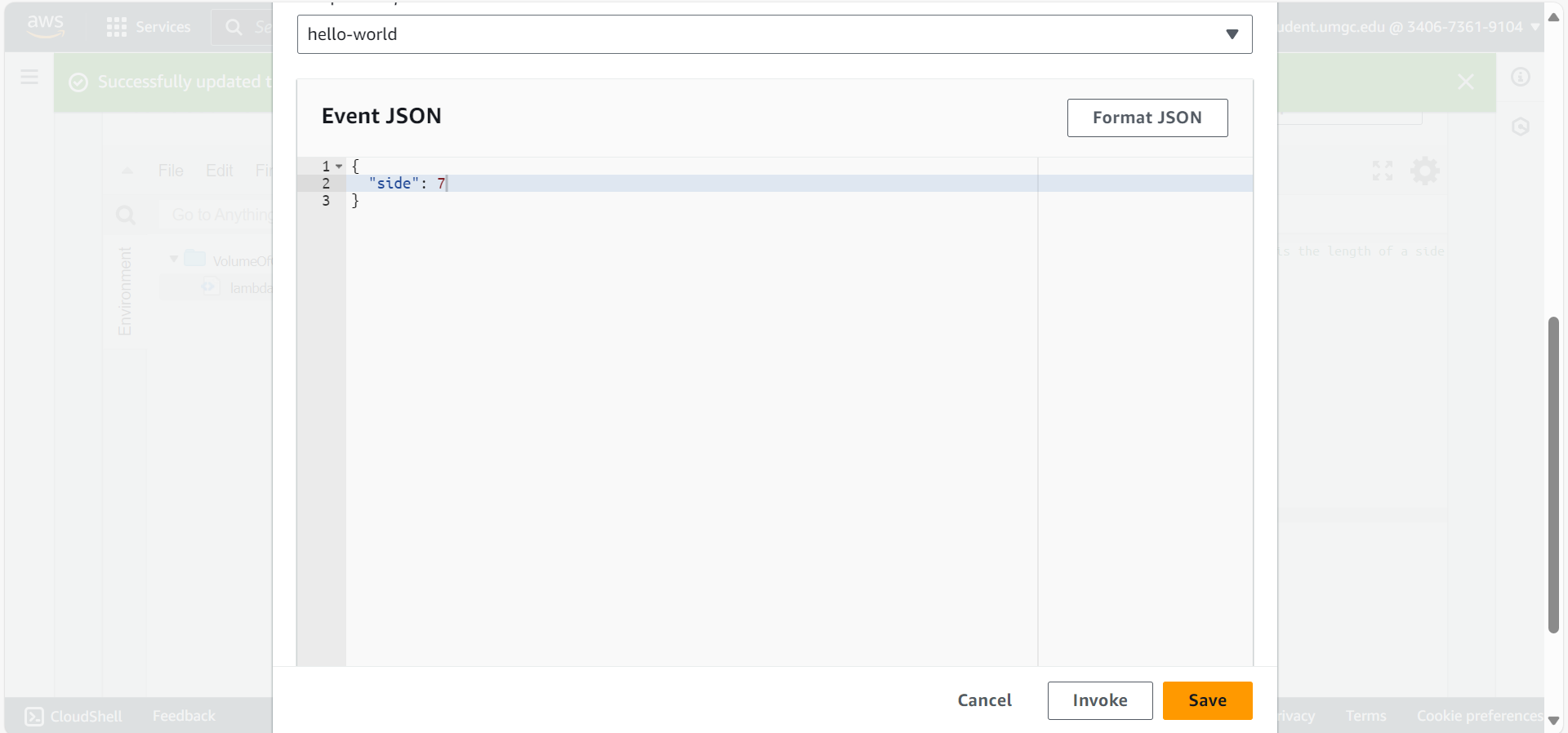


Figure 32: Function 3 Use Case 3 Input

For the final case, I used a number that would produce a high output if cubed.

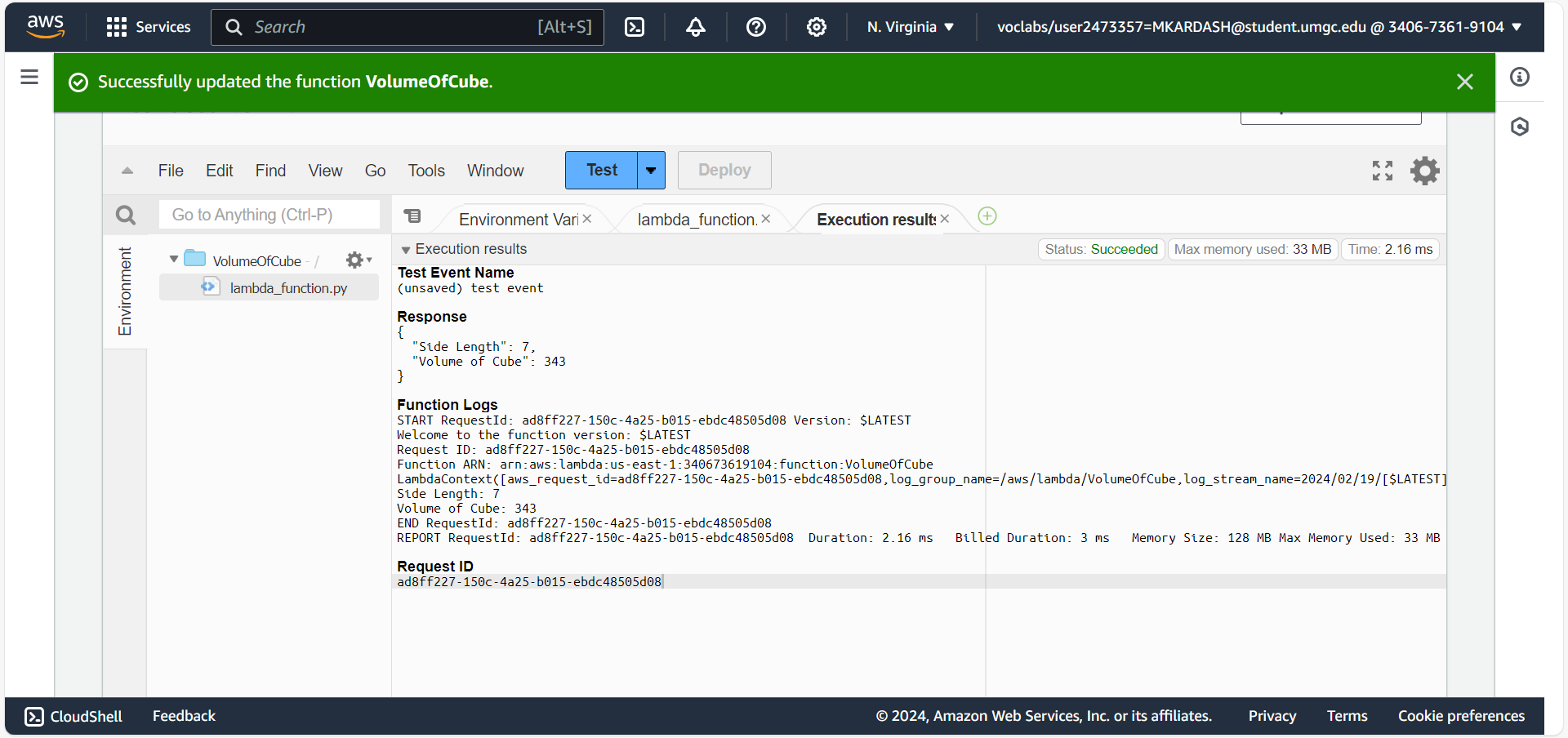


Figure 33: Function 3 Use Case 3 Result

The number was calculated correctly, meaning all the use cases have officially been passed.

Description/Explanation:

For each use case, the function was given different values for the side of the cube (3, 4.3, and 7 respectively). The event handler would “read” the side from the test event, eventually inputting it into the calculation function, while also recording it into a log. The function “volumeOfCube” would calculate the volume, after which it would be printed and returned by the evet handler, along with the initial input value of the side. Meanwhile, the log recorder would record and print out the function version, ARN, and ID.

Conclusion:

I feel like I made this assignment way more complicated than it could have been. The most challenging part was getting the function structure and variables right. I was very confused as to where each variable goes (for example, whether it should be areaOfCircle(radius, 0) or areaOfCircle(radius, circleArea)). However, after much fiddling, I finally managed to get the basic structure right, and decided to apply the same structure to each function. Instead of the perimeter of a parallelogram, I first intended a function to calculate triangle perimeter. However, after much struggle, I realized that a function like that could not take in the three parameters needed for that (Side A, Side B, and Side C of the triangle). Therefore, I switched to something that had only 2 input parameters (Side and Base). I probably could have finished the assignment much earlier, had I paid more attention to this and my other errors. Either way, all three functions eventually passed their use cases with flying colors, signifying success.