

Advanced DevOps Exp-11

Hitesh Rohra

D15A 47

Aim: To understand AWS Lambda, its workflow, various functions and create your first Lambda functions using Python / Java / Nodejs.

Theory:-

AWS Lambda

AWS Lambda is a serverless computing service provided by Amazon Web Services

(AWS). Users of AWS Lambda create functions, self-contained applications written in one

of the supported languages and runtimes, and upload them to AWS Lambda, which executes those functions in an efficient and flexible manner. The Lambda functions can

perform any kind of computing task, from serving web pages and processing streams of data to call APIs and integrate with other AWS services.

The concept of “serverless” computing refers to not needing to maintain your own servers to run these functions. AWS Lambda is a fully managed service that takes care of all the infrastructure for you.

Features of AWS Lambda

- AWS Lambda easily scales the infrastructure without any additional configuration. It reduces the operational work involved.
- It offers multiple options like AWS S3, CloudWatch, DynamoDB, API Gateway, Kinesis, CodeCommit, and many more to trigger an event.
- You don't need to invest upfront. You pay only for the memory used by the lambda function and minimal cost on the number of requests hence cost-efficient.
- AWS Lambda is secure. It uses AWS IAM to define all the roles and security policies.

- It offers fault tolerance for both services running the code and the function. You do not have to worry about the application down.

Packaging Functions

Lambda functions need to be packaged and sent to AWS. This is usually a process of compressing the function and all its dependencies and uploading it to an S3 bucket. And letting AWS know that you want to use this package when a specific event takes place.

To help us with In this process we use the Serverless Stack Framework (SST). We'll go over this in detail later on in this guide.

Execution Model

The container (and the resources used by it) that runs our function is managed completely by AWS. It is brought up when an event takes place and is turned off if it is not being used. If additional requests are made while the original event is being served, a new container is brought up to serve a request. This means that if we are undergoing a usage spike, the cloud provider simply creates multiple instances of the container with our function to serve those requests.

This has some interesting implications. Firstly, our functions are effectively stateless. Secondly, each request (or event) is served by a single instance of a Lambda function. This means that you are not going to be handling concurrent requests in your code. AWS brings up a container whenever there is a new request. It does make some optimizations here. It will hang on to the container for a few minutes (5 - 15 mins depending on the load) so it can respond to subsequent requests without a cold start.

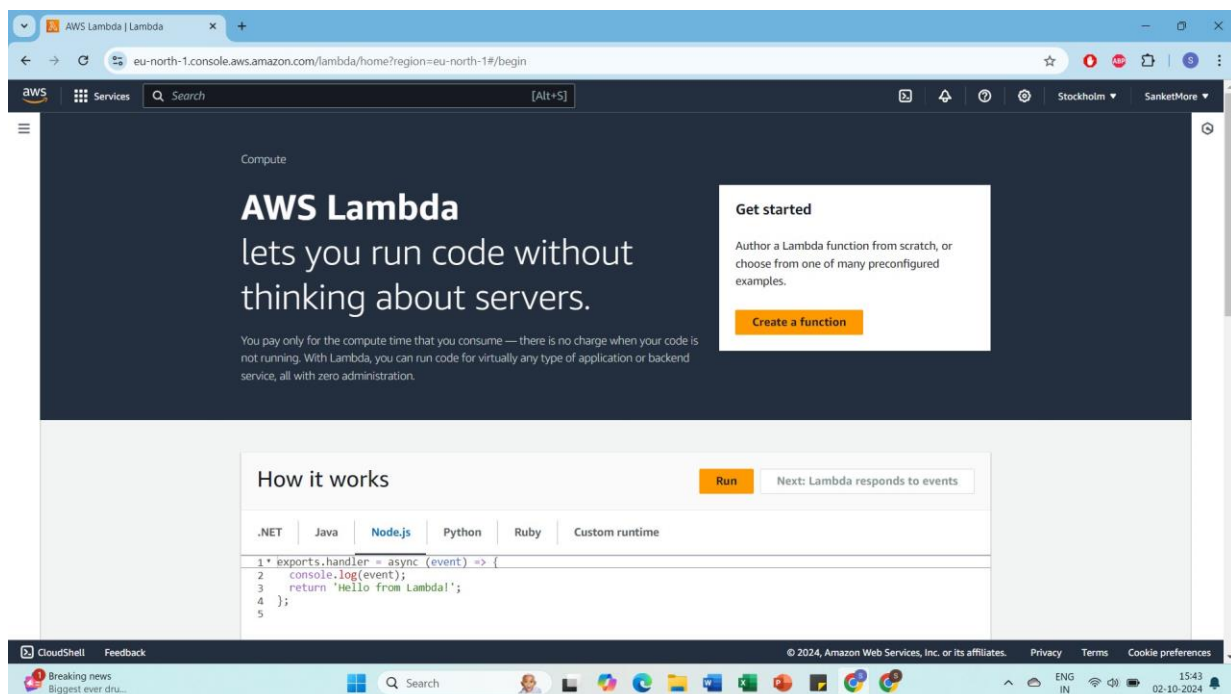
Stateless Functions

The above execution model makes Lambda functions effectively stateless. This means that every time your Lambda function is triggered by an event it is invoked in a completely new environment. You don't have access to the execution context of the previous event. However, due to the optimization noted above, the actual Lambda function is invoked only once

per container instantiation. Recall that our functions are run inside containers. So when a function is first invoked, all the code in our handler function gets executed and the handler function gets invoked. If the container is still available for subsequent requests, your function will get invoked and not the code around it.

Procedure:-

1. Open up the Lambda Console and click on the Create button. Be mindful of where you create your functions since Lambda is region-dependent.



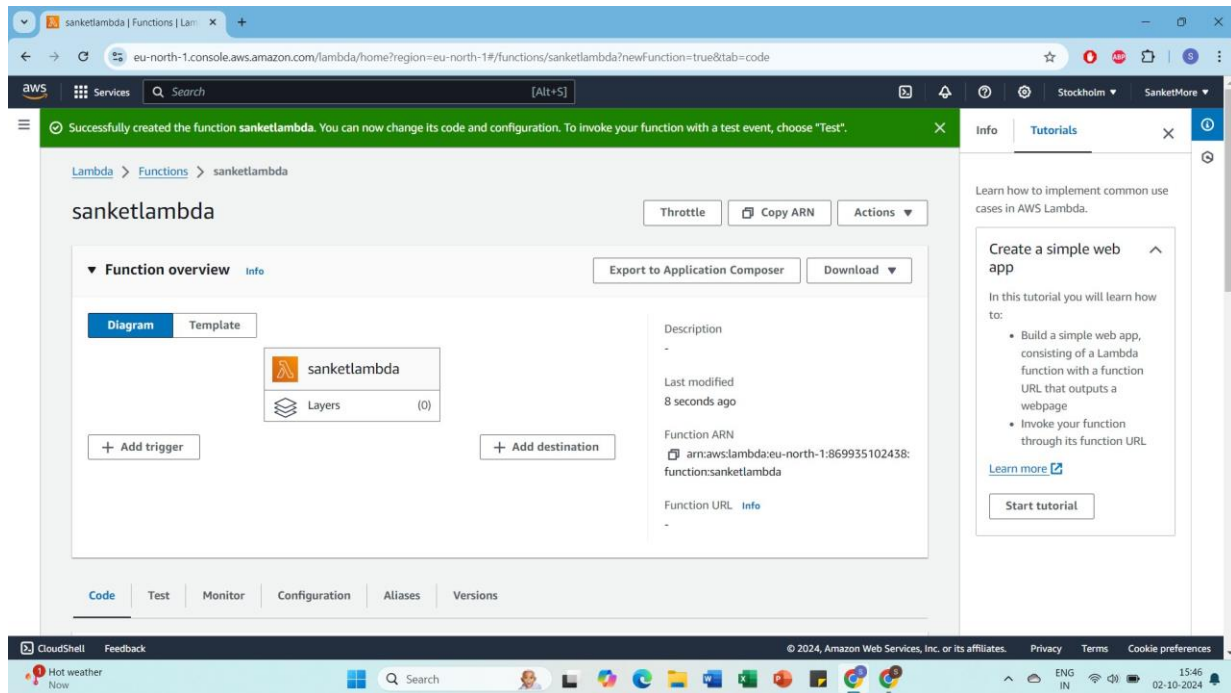
2. Choose to create a function from scratch or use a blueprint, i.e templates defined by AWS for you with all configuration presets required for the most common use cases.

Then, choose a runtime env for your function, under the dropdown, you can see all the options AWS supports, Python, Nodejs, .NET and Java being the most popular ones. After that, choose to create a new role with basic Lambda permissions if you don't have an existing one.

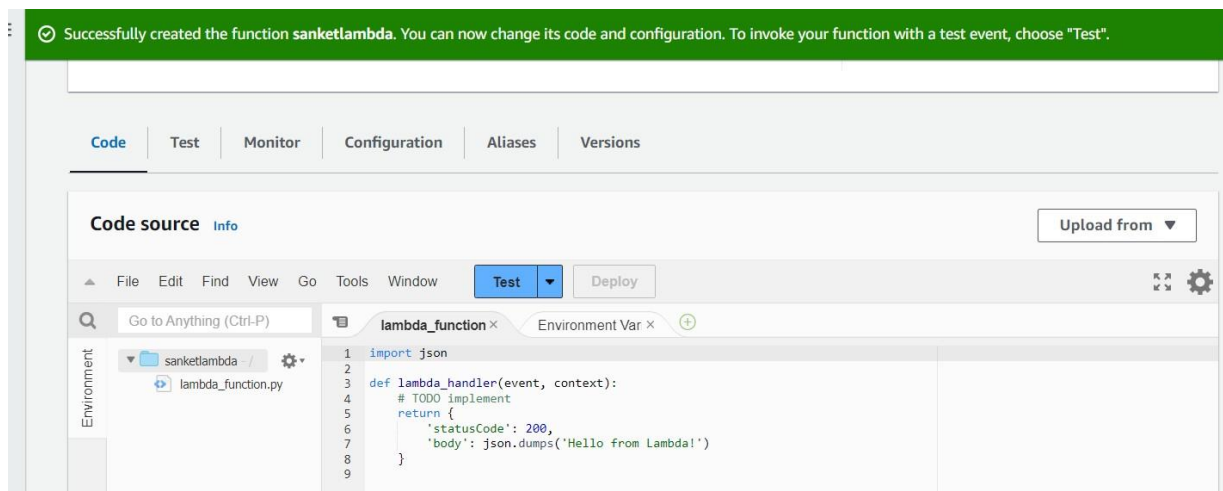
The screenshot shows the AWS Lambda 'Create function' page. At the top, there's a navigation bar with the AWS logo, 'Services', a search bar, and '[Alt+S]'. Below the navigation bar, the breadcrumb trail is 'Lambda > Functions > Create function'. The main heading is 'Create function' with an 'Info' link. Below the heading, it says 'Choose one of the following options to create your function.' There are four options, each in a box with a radio button: 'Author from scratch' (selected), 'Use a blueprint', 'Container image', and 'Browse serverless app repository'. The 'Author from scratch' box has a sub-heading 'Start with a simple Hello World example.' Below the options is a section titled 'Basic information'. It contains three fields: 'Function name' with a text input containing 'sanketlambda' and a note 'Enter a name that describes the purpose of your function. Use only letters, numbers, hyphens, or underscores with no spaces.'; 'Runtime' with a dropdown menu showing 'Python 3.12' and a note 'Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.'; and 'Architecture' with a dropdown menu showing 'x86_64' and a note 'Choose the instruction set architecture you want for your function code.'

The screenshot shows the 'Change default execution role' section. It has a dropdown arrow and the title 'Change default execution role'. Below the title is the section 'Execution role' with the text 'Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console [link]'. There are three radio button options: 'Create a new role with basic Lambda permissions' (selected), 'Use an existing role', and 'Create a new role from AWS policy templates'. Below these options is a light blue information box with an 'i' icon and the text 'Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.' At the bottom of the section, it says 'Lambda will create an execution role named sanketlambda-role-aqbjlc1, with permission to upload logs to Amazon CloudWatch Logs.'

3. This process will take a while to finish and after that, you'll get a message that your function was successfully created.

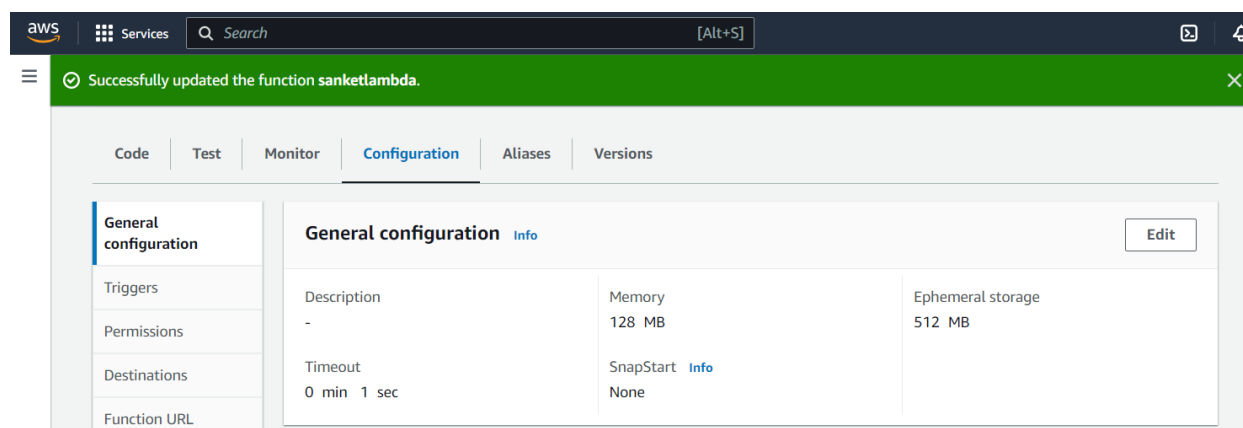
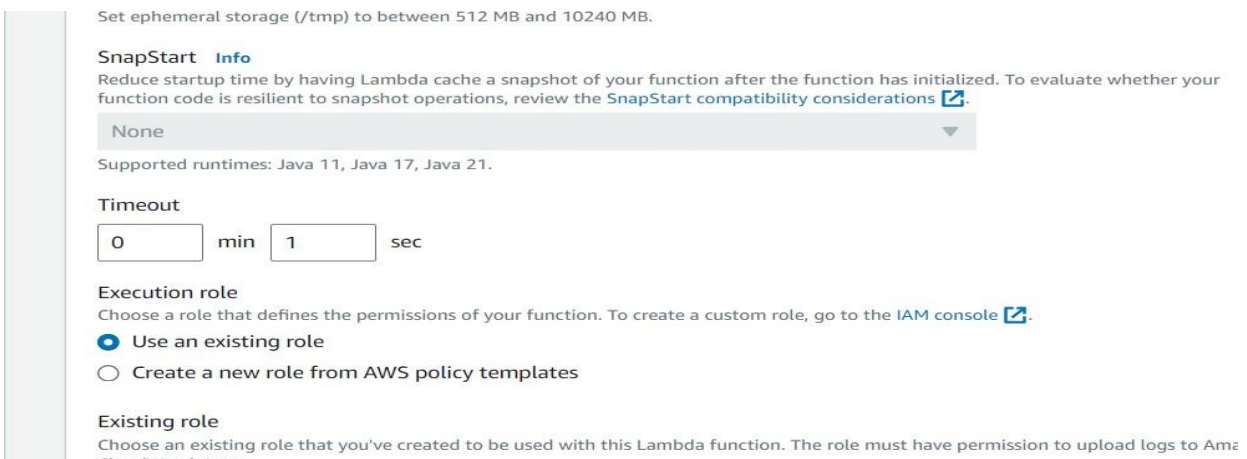
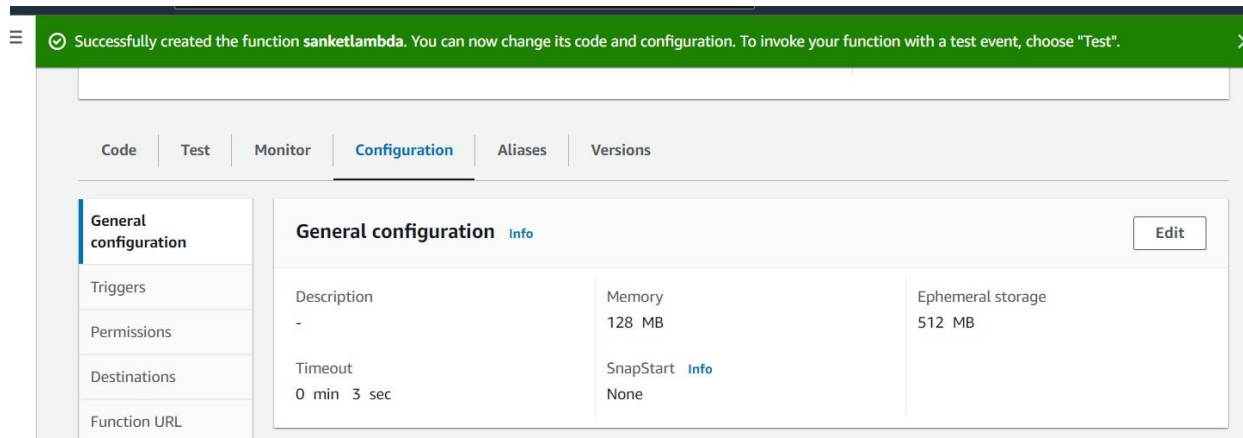


4. You can make changes to your function inside the code editor. You can also upload a zip file of your function or upload one from an S3 bucket if needed. Press Ctrl + S to save the file and click Deploy to deploy the changes.

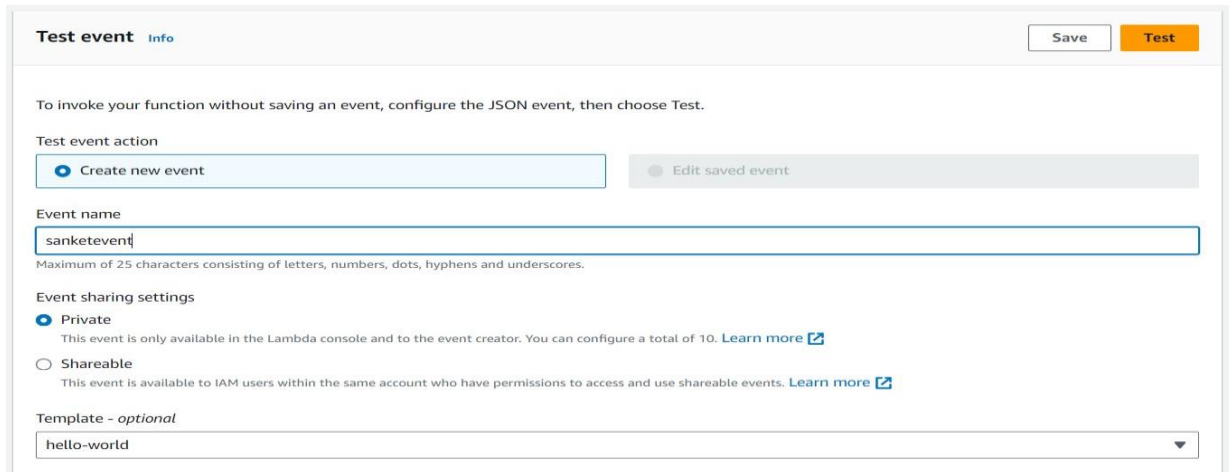


5. To change the configuration, open up the Configuration tab and under General Configuration, choose Edit.

Here, you can enter a description and change Memory and Timeout. I've changed the Timeout period to 1 sec since that is sufficient for now.



6. Click on Test and you can change the configuration, like so. If you do not have anything in the request body, it is important to specify two curly braces as valid JSON, so make sure they are there.



The screenshot shows the 'Test event' configuration page in the AWS Lambda console. At the top, there are 'Save' and 'Test' buttons. Below, a message states: 'To invoke your function without saving an event, configure the JSON event, then choose Test.' The 'Test event action' section has two radio buttons: 'Create new event' (selected) and 'Edit saved event'. The 'Event name' field contains 'sanketevent' with a note below it: 'Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.' The 'Event sharing settings' section has two radio buttons: 'Private' (selected) and 'Shareable'. Below 'Private' is a note: 'This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)'. Below 'Shareable' is a note: 'This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)'. The 'Template - optional' dropdown menu is set to 'hello-world'.

Template - optional

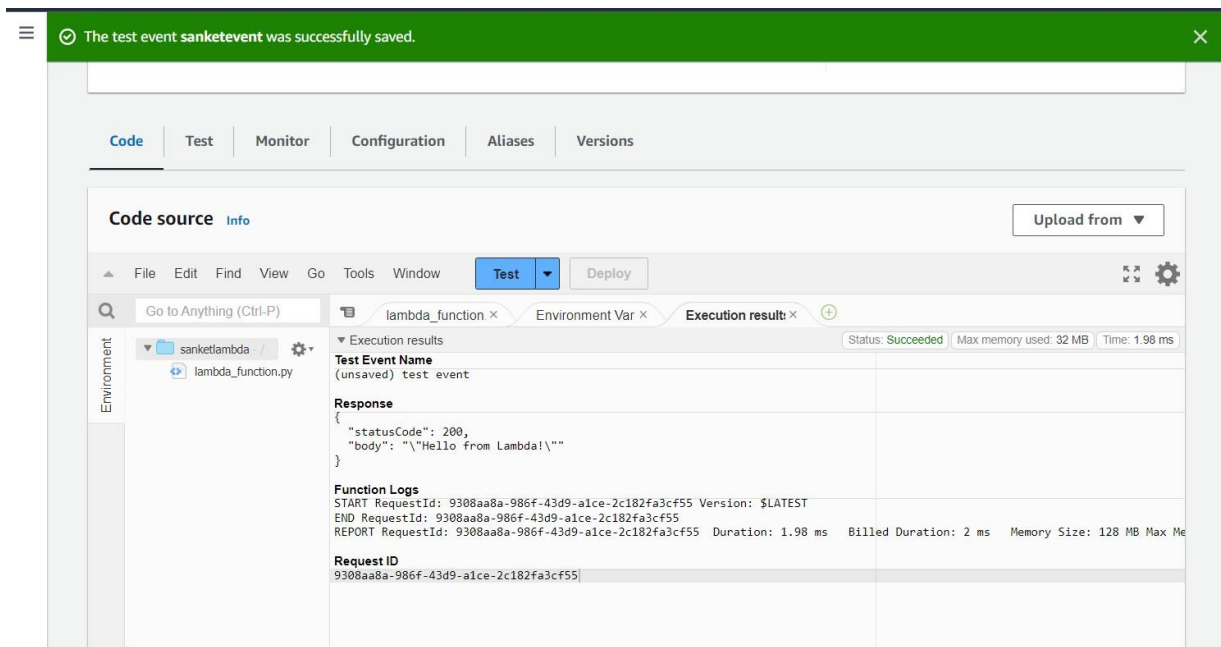
hello-world



The screenshot shows the 'Event JSON' editor. It has a title bar 'Event JSON' and a text area with the following JSON content:

```
1 {
2   "key1": "value1",
3   "key2": "value2",
4   "key3": "value3"
5 }
```

7. Now click on Test and you should be able to see the results.



The screenshot shows the AWS Lambda console with a green notification bar at the top stating: 'The test event sanketevent was successfully saved.' Below the notification bar, there are tabs: 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. The 'Test' tab is selected. The 'Code source' section shows 'Upload from' with a dropdown arrow. Below this, there is a toolbar with 'File', 'Edit', 'Find', 'View', 'Go', 'Tools', 'Window', 'Test' (highlighted), and 'Deploy'. The 'Environment' pane on the left shows 'sanketlambda' and 'lambda_function.py'. The 'Execution results' pane on the right shows the following details:

- Test Event Name** (unsaved) test event
- Response**:

```
{  "statusCode": 200,  "body": "\"Hello from Lambda!\""}
```
- Function Logs**:

```
START RequestId: 9308aa8a-986f-43d9-a1ce-2c182fa3cf55 Version: $LATEST
END RequestId: 9308aa8a-986f-43d9-a1ce-2c182fa3cf55
REPORT RequestId: 9308aa8a-986f-43d9-a1ce-2c182fa3cf55  Duration: 1.98 ms   Billed Duration: 2 ms   Memory Size: 128 MB Max Me
```
- Request ID**: 9308aa8a-986f-43d9-a1ce-2c182fa3cf55