



AWS DEVOPS BASICS WITH RE:POST

AWS Lambda

Lambda Basic

Hyoungjoo Lee

Cloud Support Engineer

AWS Korea



Agenda

- Serverless Architecture
- AWS Lambda
- Workshop



What is Serverless?



Conventional Deployment and Operations

- Configure an Instance
- Update OS
- Install App Platform
- Build and deploy apps
- Configure automatic scaling and load balancing
- Continuously secure and monitor instances
- Monitor and maintain apps

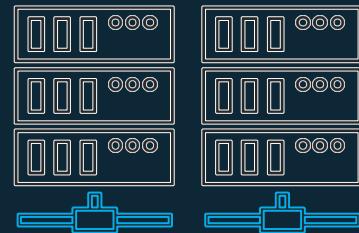


Serverless Deployment and Operations

- Configure an Instance
- Update OS
- Install App Platform
- Build and deploy apps
- Configure automatic scaling and load balancing
- Continuously secure and monitor instances
- Monitor and maintain apps



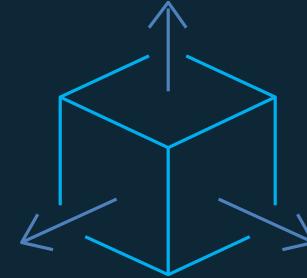
What is Serverless?



No server or container management



High availability



Flexible scaling



No idle capacity

What is AWS Lambda?



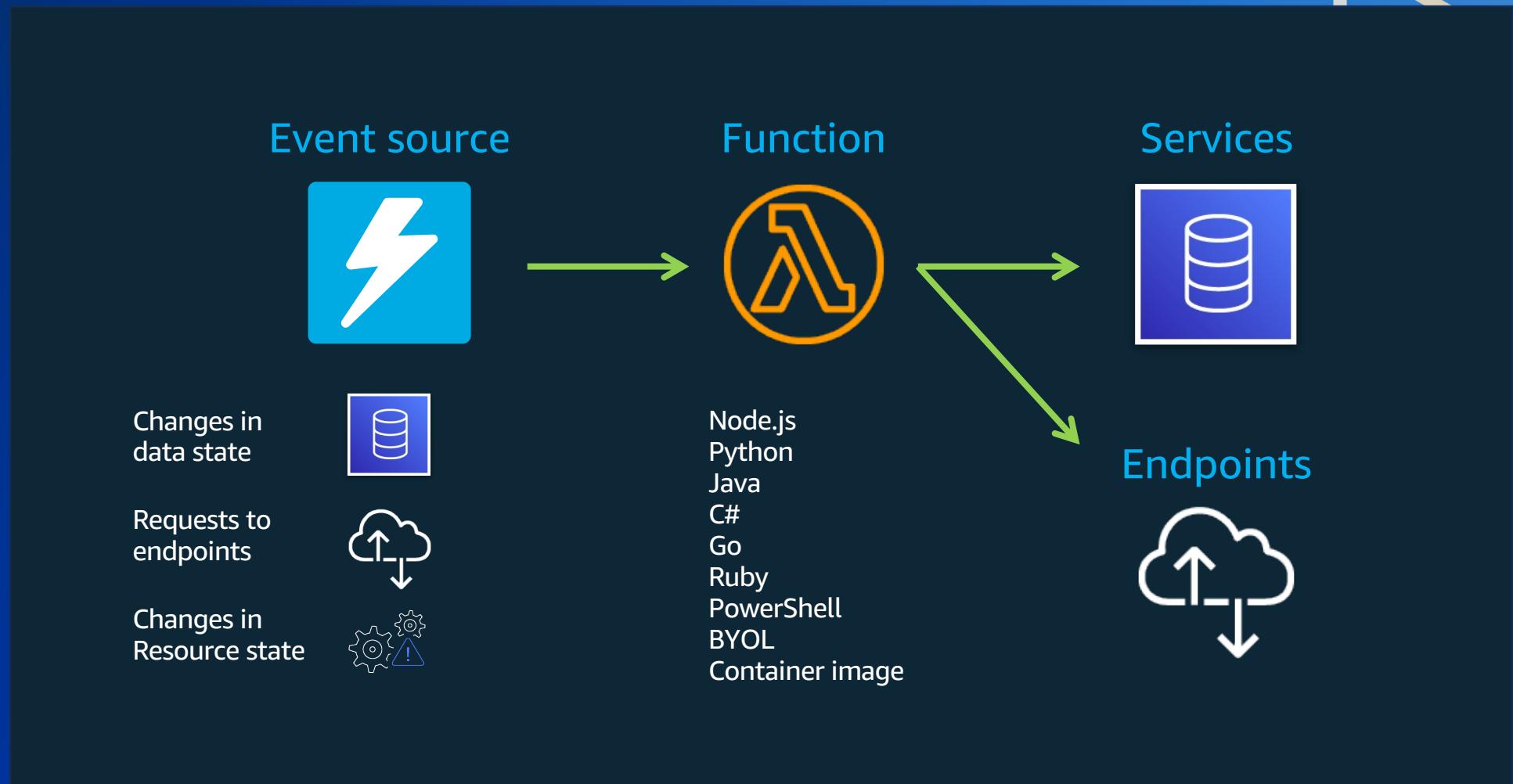
AWS Lambda



Free Tier
1M requests and 400,000 GBs of compute.
Every month, every customer.

- No servers to manage
- Continuous scaling
- Cost optimized with 1ms metering
- Consistent performance at scale
- “Always Free” 1 million req/month

Serverless Applications



Lambda Handler

Handler() function

Function to be executed upon invocation

Event object

Data sent during Lambda function Invocation

Context object

Methods and properties about the invocation, function, and execution environment

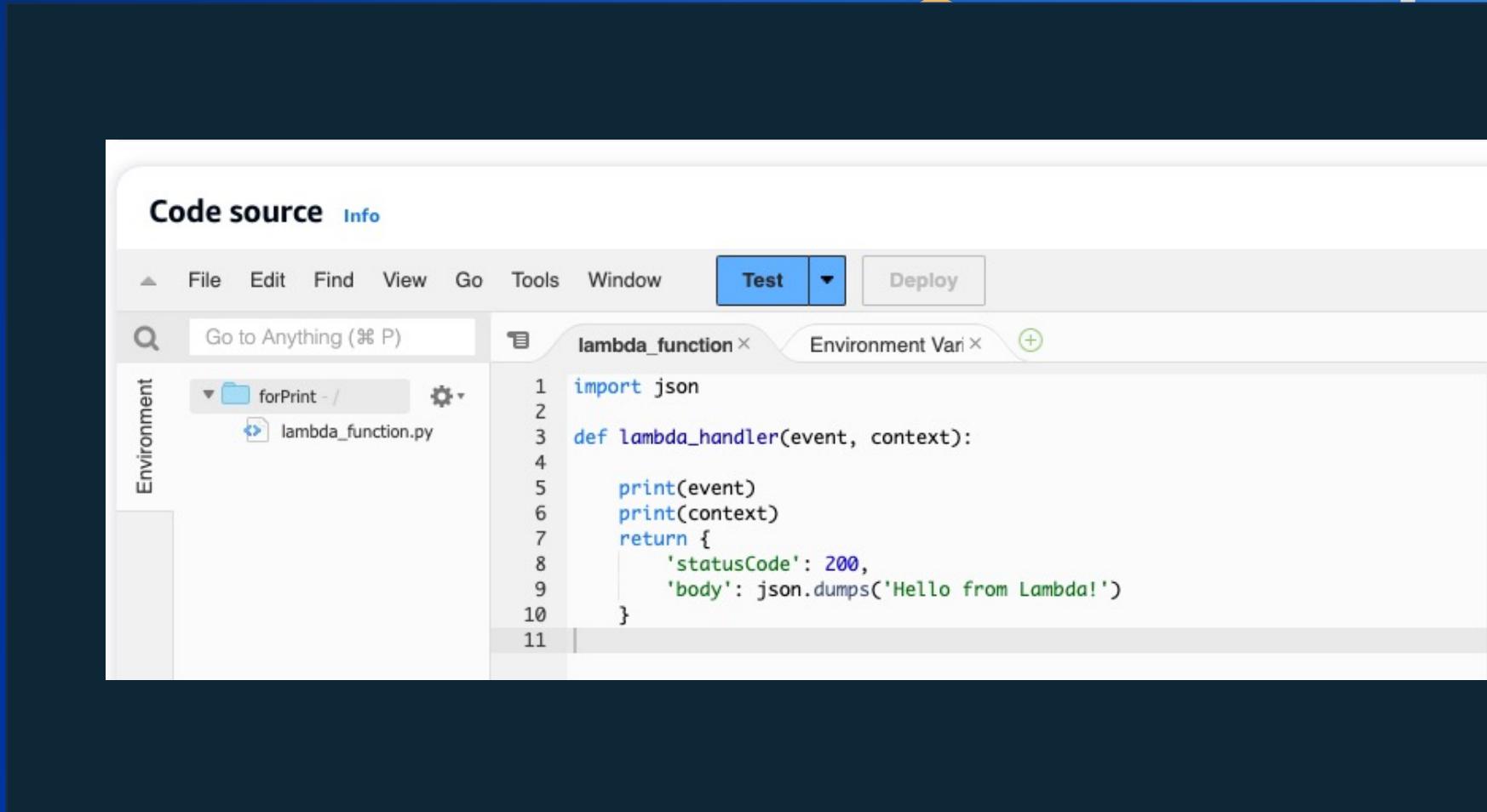
The screenshot shows the AWS Lambda code editor interface. On the left, there's a sidebar with 'Code source' and 'Info' tabs, file navigation, and deployment buttons ('Test', 'Deploy', 'Changes deployed'). Below the sidebar is a file tree with 'hello-world-python' and 'lambda_function.py'. The main area contains the Python code for the Lambda function:

```
1 import json
2
3 print('Loading function')
4
5 def lambda_handler(event, context):
6     #print("Received event: " + json.dumps(event, indent=2))
7     print("value1 = " + event['key1'])
8     print("value2 = " + event['key2'])
9     print("value3 = " + event['key3'])
10    return event['key1'] # Echo back the first key value
11    #raise Exception('Something went wrong')
```

Annotations with arrows point from the text labels to specific parts of the code:

- A blue arrow points from the 'Handler()' function header to the line `def lambda_handler(event, context):`.
- An orange arrow points from the 'Event object' definition to the variable `event` in the code.
- A yellow arrow points from the 'Context object' definition to the variable `context` in the code.

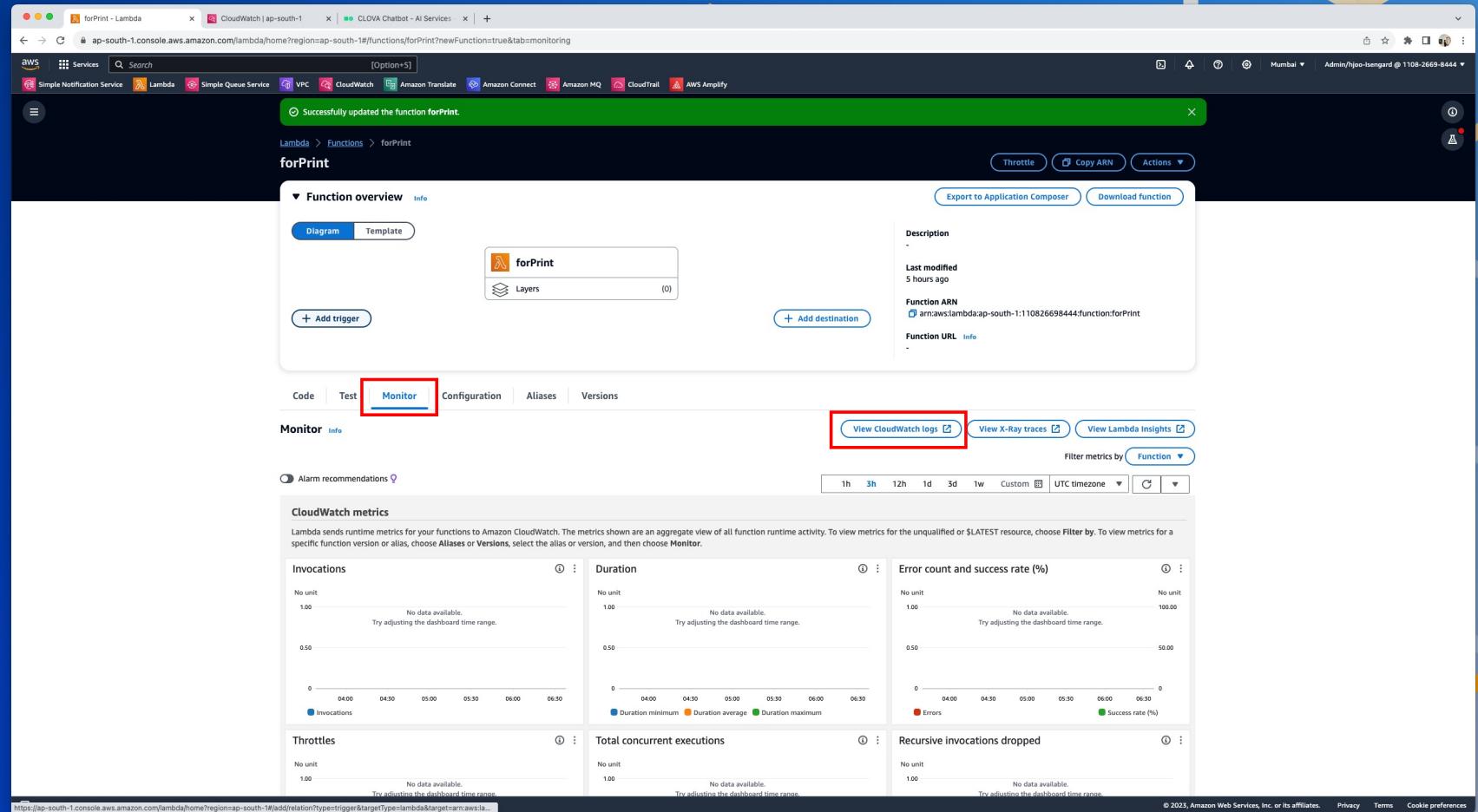
Lambda Log



The screenshot shows the AWS Lambda function editor interface. The title bar says "Code source" and "Info". The menu bar includes File, Edit, Find, View, Go, Tools, Window, Test (which is selected), and Deploy. A search bar says "Go to Anything (% P)". The left sidebar shows an "Environment" section and a file tree with "forPrint - /" expanded, showing "lambda_function.py". The main code editor window has the tab "lambda_function" and shows the following Python code:

```
1 import json
2
3 def lambda_handler(event, context):
4
5     print(event)
6     print(context)
7     return {
8         'statusCode': 200,
9         'body': json.dumps('Hello from Lambda!')
10    }
11
```

Lambda Log



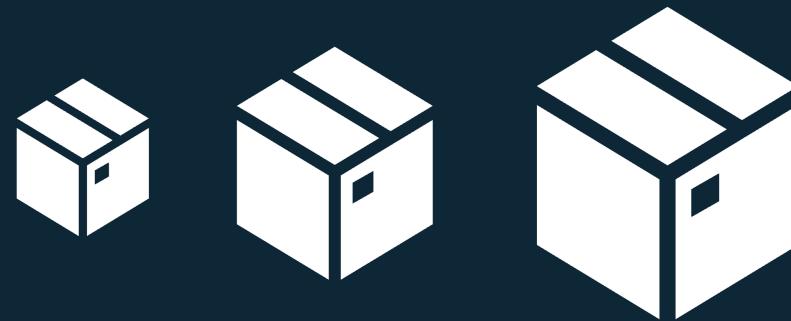
Lambda Log

Log events																			
		Filter events		Actions ▾		Start tailing		Create metric filter											
	Timestamp	Message																	
		No older events at this moment. Retry																	
▶	2023-11-16T15:39:27.316+09:00	INIT_START Runtime Version: python:3.11.v18 Runtime Version ARN: arn:aws:lambda:ap-south-1::runtime:6ebff6b58cf714d30879a40cc31554cd1bbc242da7bf75a000bc0c3052c6ebbc																	
▶	2023-11-16T15:39:27.432+09:00	START RequestId: 8d908beb-99a1-4bfe-946d-6c9926f945ed Version: \$LATEST																	
▼	2023-11-16T15:39:27.433+09:00	{'key1': 'value1', 'key2': 'value2', 'key3': 'value3'} {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}																	
▼	2023-11-16T15:39:27.433+09:00	LambdaContext([aws_request_id=8d908beb-99a1-4bfe-946d-6c9926f945ed,log_group_name=/aws/lambda/forPrint,log_stream_name=2023/11/16/[LATEST]f35d89d4c9f74baeb60d127366cc8650,function_name=forPrint,memory_limit_in_mb=128,function_version=\$LATEST,invoked_function_arn=arn:aws:lambda:ap-south-1:110826698444:function:forPrint,client_context=None,identity=CognitoIdentity([cognito_identity_id=None,cognito_identity_pool_id=None])])																	
▶	2023-11-16T15:39:27.435+09:00	END RequestId: 8d908beb-99a1-4bfe-946d-6c9926f945ed																	
▶	2023-11-16T15:39:27.435+09:00	REPORT RequestId: 8d908beb-99a1-4bfe-946d-6c9926f945ed Duration: 1.38 ms Billed Duration: 2 ms Memory Size: 128 MB Max Memory Used: 40 MB Init Duration: 115.02 ms																	
		No newer events at this moment. Auto retry paused. Resume																	

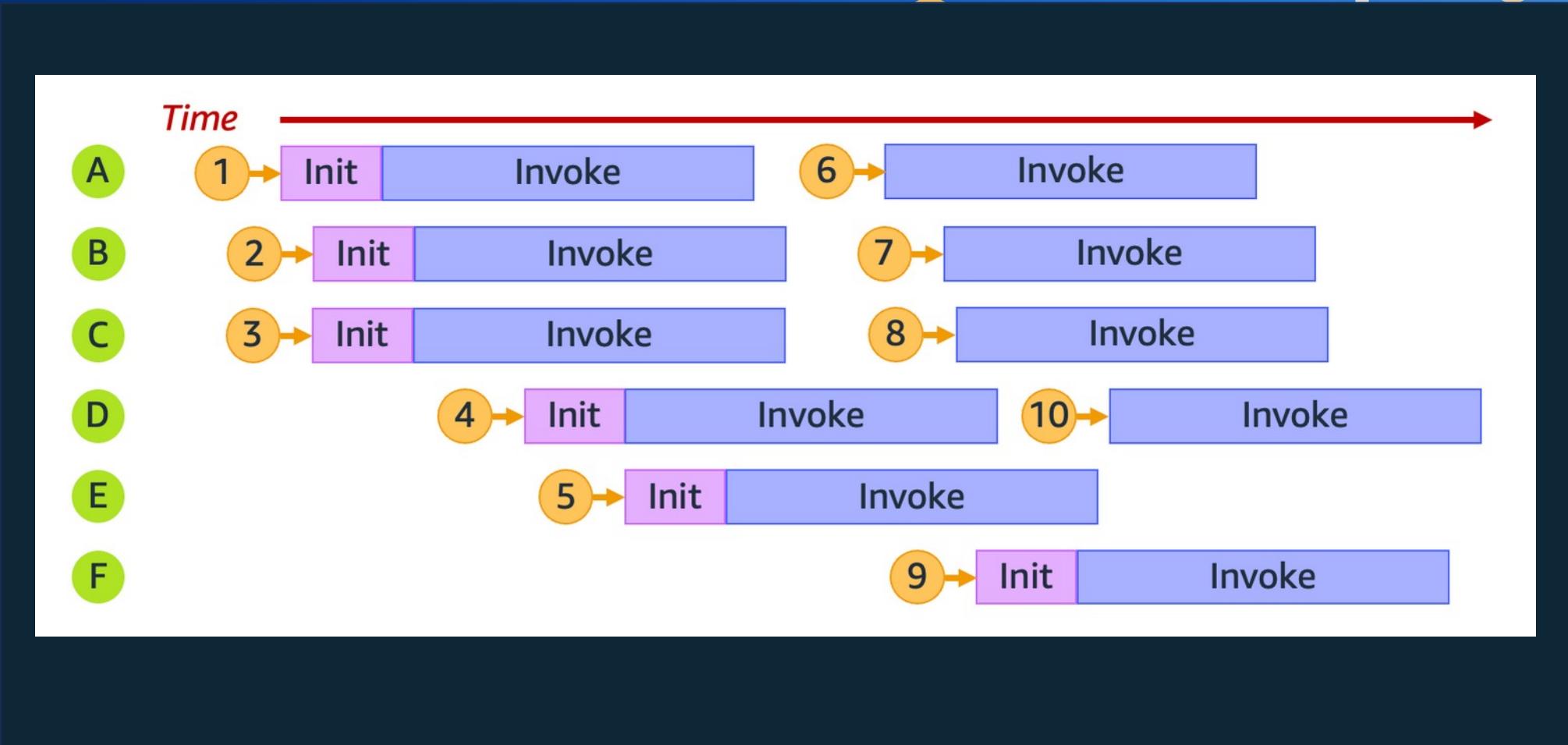


Scaling behavior

- Lambda dynamically scales in response to increased traffic
- Concurrent invocations are the number of invocations of the function code that happen in parallel at any given time
- Estimated concurrency (Capacity) required by the function = Invocations Per Second * Average Invocation Duration In Seconds



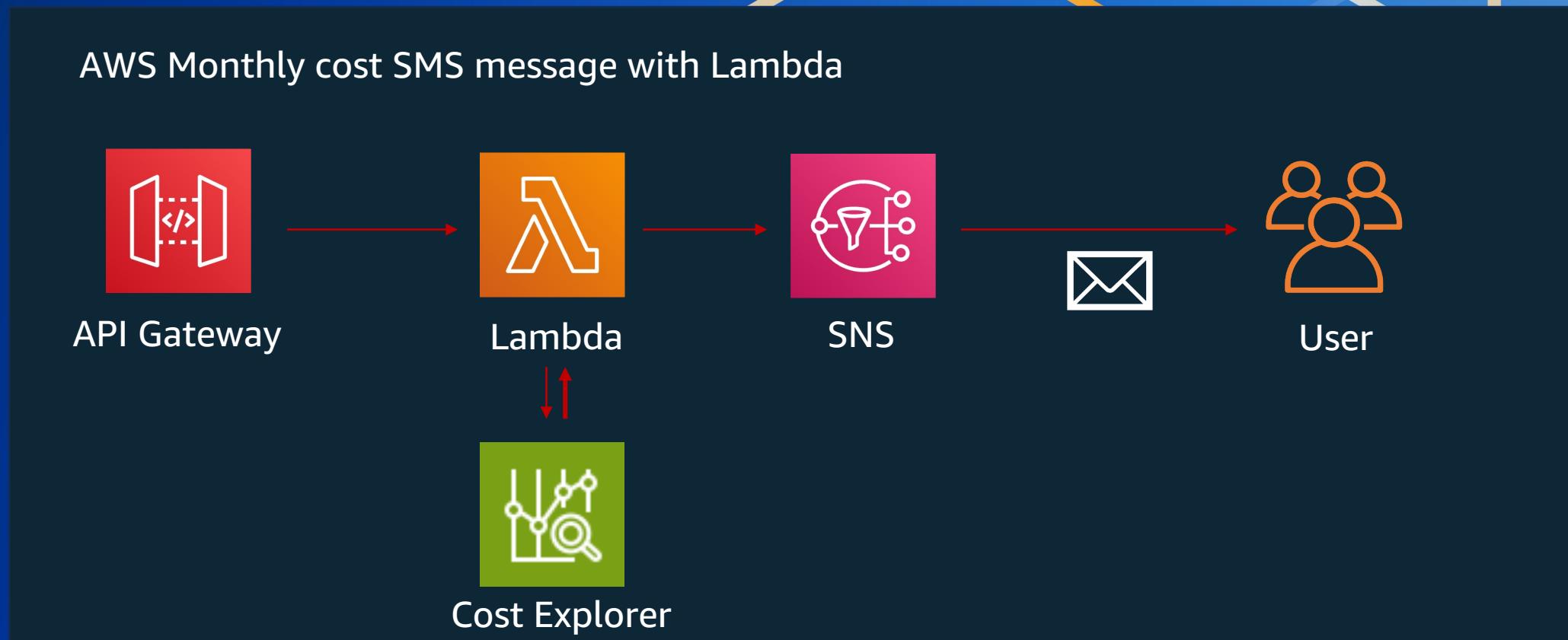
Concurrent Execution



Q&A



Workshop



Workshop

Screenshot of the AWS Cost Explorer Home dashboard for the us-east-1 region.

Cost summary:

- Current month costs: \$983.15 (Up 0% over last month)
- Forecasted month-end costs: \$1,686.71 (Up 5% over last month)

Daily unblended costs: Bar chart showing daily costs from Oct 01 to Nov 15. The chart shows significant fluctuations, with peaks around October 31 and November 03, and troughs around October 04 and November 12.

Recently accessed reports: No recently accessed reports.

November trends:

- Service usage: Amazon MQ costs are up \$98.11 (13%)
- Account usage: hjoo's account (110826698444) costs are up \$61.61 (6%)
- Region usage: us-east-1 costs are up \$366.51 (165%)

More resources:

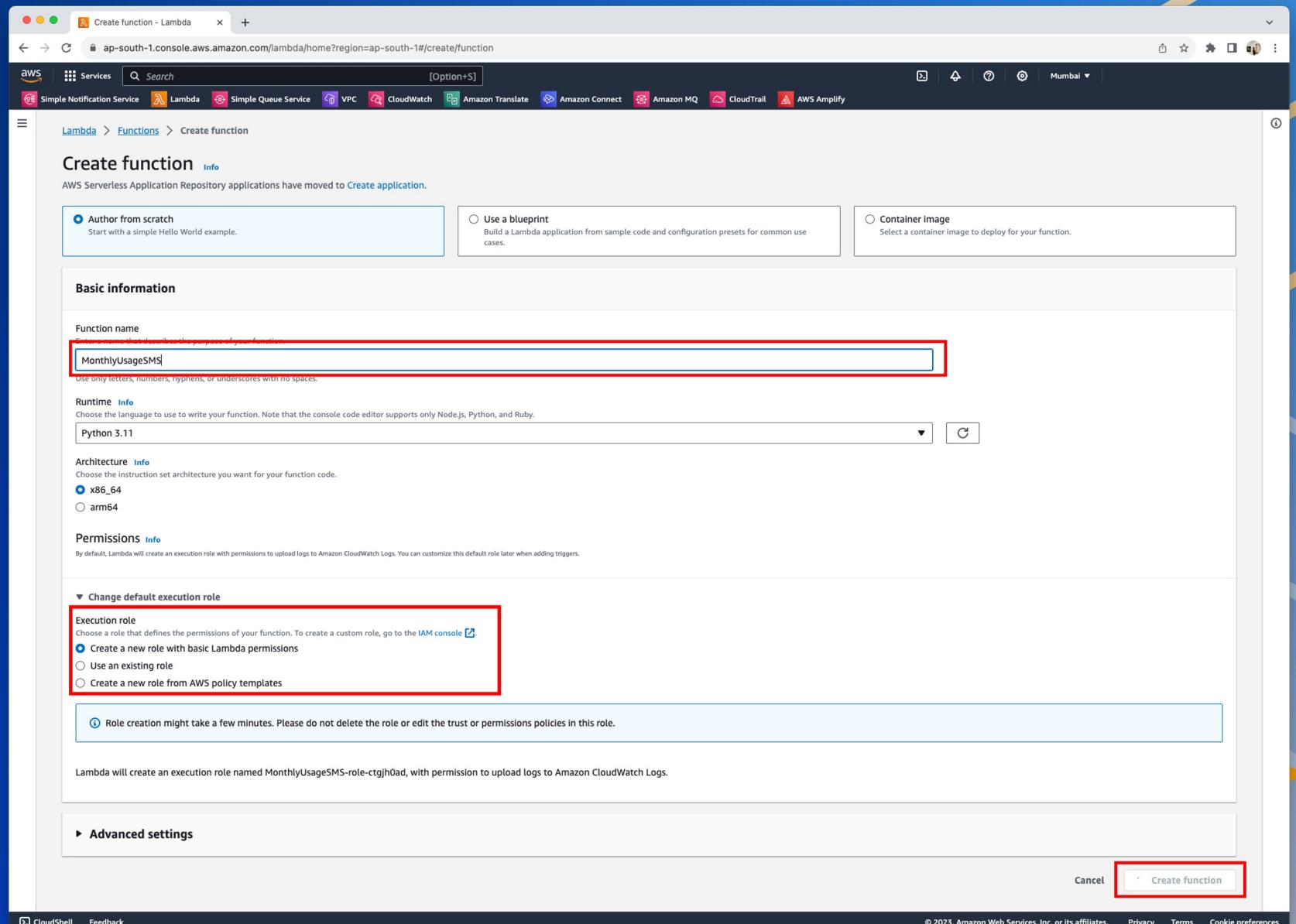
- What is AWS Billing and Cost Management?
- Documentation
- FAQ



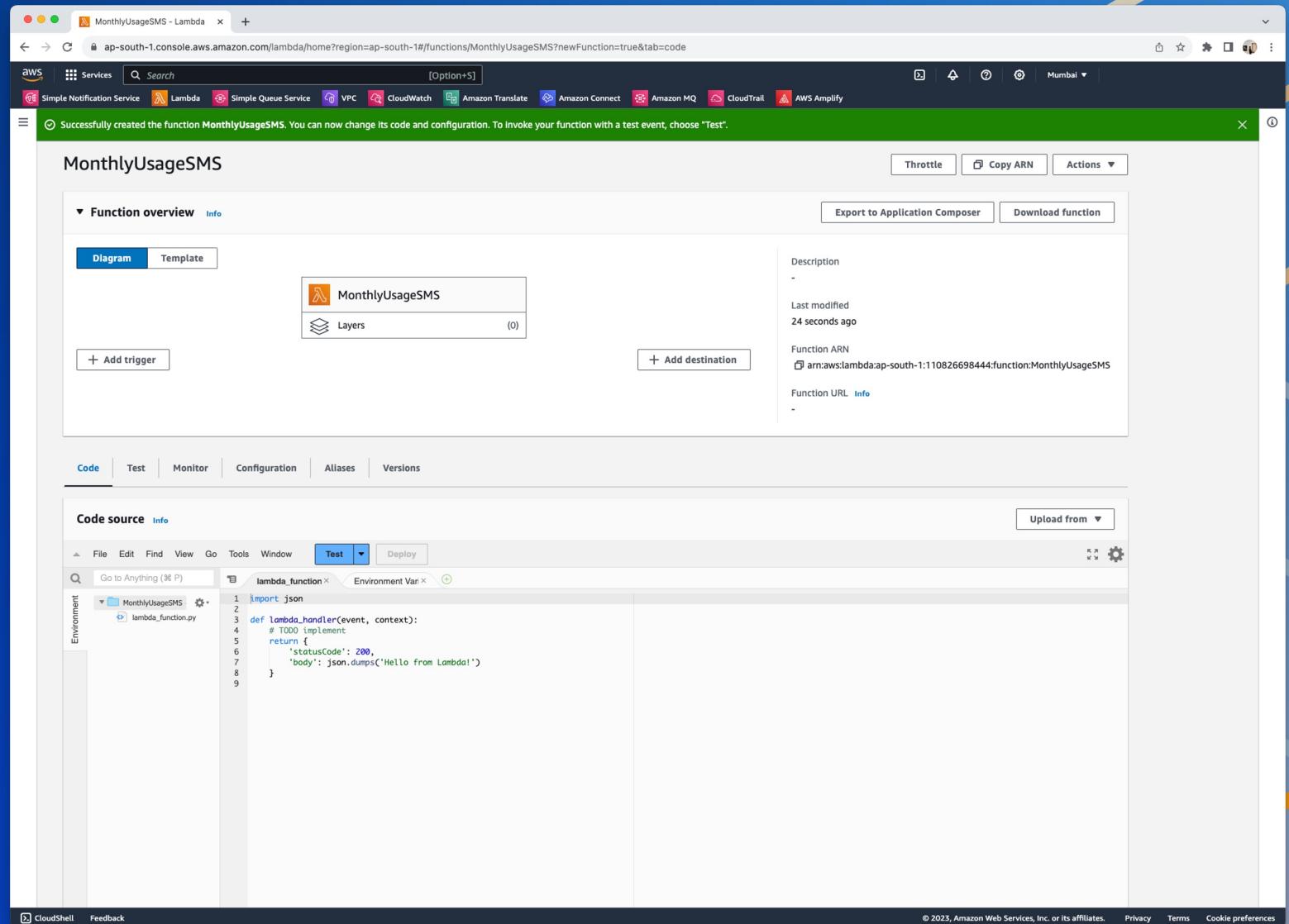
Workshop

The screenshot shows the AWS Lambda console interface. The left sidebar has a tree view with 'AWS Lambda' selected, under which 'Functions' is also selected. The main content area is titled 'Functions (0)' and contains a message: 'There is no data to display.' A search bar at the top says 'Filter by tags and attributes or search by keyword'. At the top right, there is a 'Create function' button highlighted with a red box. The bottom right corner of the page footer contains copyright information: '© 2023, Amazon Web Services, Inc. or its affiliates.'

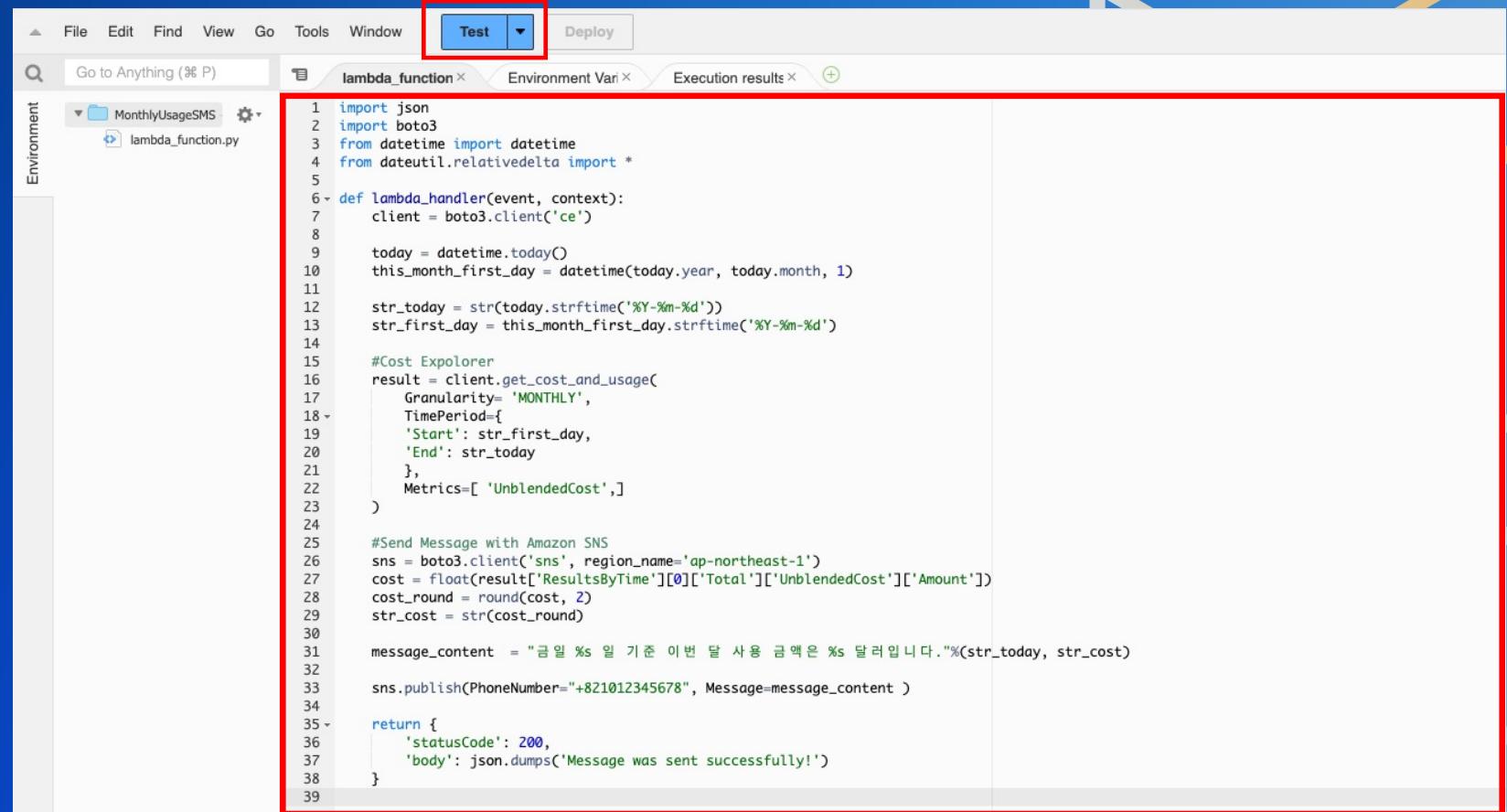
Workshop



Workshop



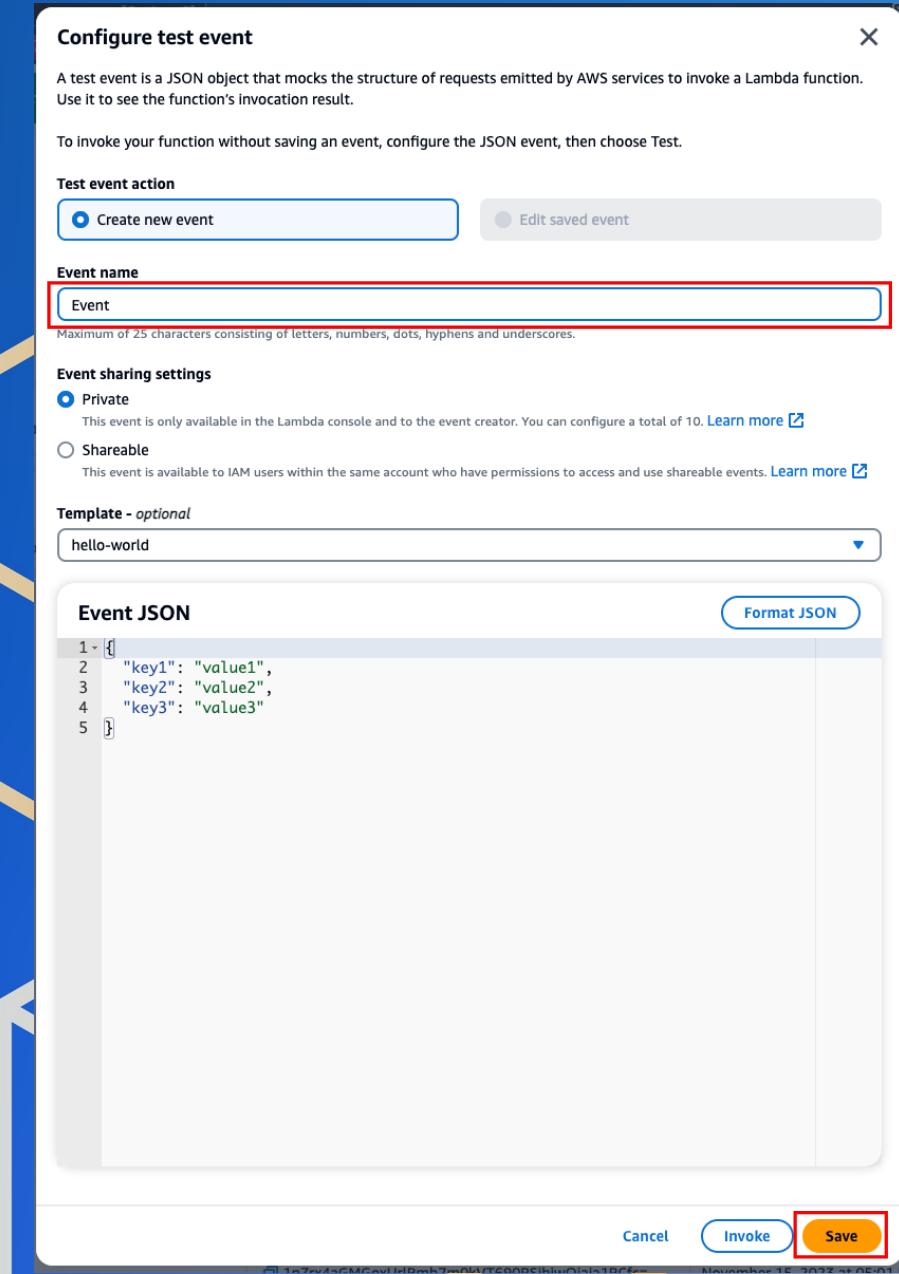
Workshop



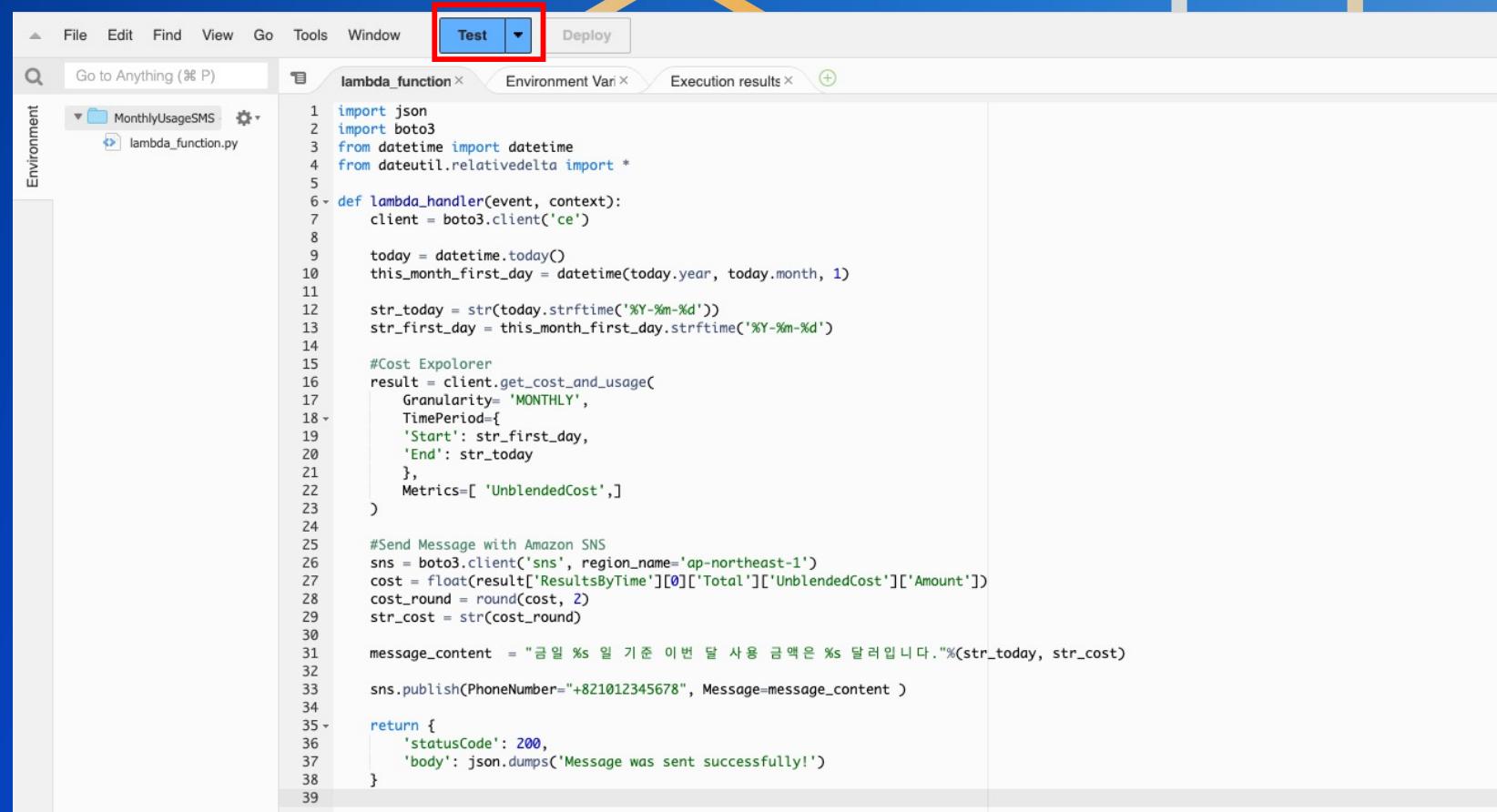
The screenshot shows the AWS Lambda function editor interface. At the top, there is a menu bar with File, Edit, Find, View, Go, Tools, Window, a Test button (which is highlighted with a red box), and Deploy. Below the menu is a search bar labeled "Go to Anything (% P)". The main workspace is titled "lambda_function" and contains tabs for Environment, lambda_function.py, Environment Var, and Execution results. On the left, there is an "Environment" sidebar with a "MonthlyUsageSMS" section containing a "lambda_function.py" file. The code editor displays the following Python script:

```
1 import json
2 import boto3
3 from datetime import datetime
4 from dateutil.relativedelta import *
5
6 def lambda_handler(event, context):
7     client = boto3.client('ce')
8
9     today = datetime.today()
10    this_month_first_day = datetime(today.year, today.month, 1)
11
12    str_today = str(today.strftime('%Y-%m-%d'))
13    str_first_day = this_month_first_day.strftime('%Y-%m-%d')
14
15    #Cost Explorer
16    result = client.get_cost_and_usage(
17        Granularity='MONTHLY',
18        TimePeriod={
19            'Start': str_first_day,
20            'End': str_today
21        },
22        Metrics=[ 'UnblendedCost',]
23    )
24
25    #Send Message with Amazon SNS
26    sns = boto3.client('sns', region_name='ap-northeast-1')
27    cost = float(result['ResultsByTime'][0]['Total']['UnblendedCost']['Amount'])
28    cost_round = round(cost, 2)
29    str_cost = str(cost_round)
30
31    message_content = "금 일 %s 일 기준 이번 달 사용 금액은 %s 달러입니다."%(str_today, str_cost)
32
33    sns.publish(PhoneNumber="+821012345678", Message=message_content )
34
35    return {
36        'statusCode': 200,
37        'body': json.dumps('Message was sent successfully!')
38    }
39
```

Workshop



Workshop



The screenshot shows the AWS Lambda function editor interface. At the top, there is a navigation bar with tabs: File, Edit, Find, View, Go, Tools, Window, Test (which is highlighted with a red box), and Deploy. Below the navigation bar, there are three tabs: lambda_function (selected), Environment Vari, and Execution results. On the left, there is an Environment sidebar with a search bar labeled "Go To Anything (%P)" and a dropdown menu for "MonthlyUsageSMS". Inside the dropdown, there is a file named "lambda_function.py". The main content area displays the Python code for the lambda function:

```
1 import json
2 import boto3
3 from datetime import datetime
4 from dateutil.relativedelta import *
5
6 def lambda_handler(event, context):
7     client = boto3.client('ce')
8
9     today = datetime.today()
10    this_month_first_day = datetime(today.year, today.month, 1)
11
12    str_today = str(today.strftime('%Y-%m-%d'))
13    str_first_day = this_month_first_day.strftime('%Y-%m-%d')
14
15    #Cost Explorer
16    result = client.get_cost_and_usage(
17        Granularity='MONTHLY',
18        TimePeriod={
19            'Start': str_first_day,
20            'End': str_today
21        },
22        Metrics=['UnblendedCost'],
23    )
24
25    #Send Message with Amazon SNS
26    sns = boto3.client('sns', region_name='ap-northeast-1')
27    cost = float(result['ResultsByTime'][0]['Total']['UnblendedCost']['Amount'])
28    cost_round = round(cost, 2)
29    str_cost = str(cost_round)
30
31    message_content = "금 일 %s 일 기준 이번 달 사용 금액은 %s 달러입니다."%(str_today, str_cost)
32
33    sns.publish(PhoneNumber="+821012345678", Message=message_content)
34
35    return {
36        'statusCode': 200,
37        'body': json.dumps('Message was sent successfully!')
38    }
39
```

Workshop

The screenshot shows the AWS Lambda Test interface. The top navigation bar includes tabs for Code, Test, Monitor, Configuration, Aliases, and Versions. The Code tab is selected. Below the tabs is a toolbar with File, Edit, Find, View, Go, Tools, Window, a Test dropdown, Deploy, and an Upload from button.

The main area displays the Code source and Info sections. The Environment sidebar shows a project named "MonthlyUsageSMS" containing a file "lambda_function.py". The Execution results section shows a test event named "Event". The Response pane contains the following JSON error message:

```
{  "errorMessage": "An error occurred (AccessDeniedException) when calling the GetCostAndUsage operation: User: arn:aws:sts::110826698444:assumed-role/MonthlyUsageSMS-ExecutionRole;Reason:User is not authorized to perform the requested action.",  "errorType": "ClientError",  "requestId": "67a00244-3dcf-4aa8-a5d4-fdc506e75d8e",  "stackTrace": [    "File \"/var/task/lambda_function.py\", line 16, in lambda_handler\n        result = client.get_cost_and_usage()\n",    "File \"/var/lang/lib/python3.11/site-packages/botocore/client.py\", line 534, in _api_call\n            return self._make_api_call(operation_name,\n",    "File \"/var/lang/lib/python3.11/site-packages/botocore/client.py\", line 976, in _make_api_call\n                raise error_class(parsed_response, operation_name)\n",  ]}
```

The Function Logs pane shows the full stack trace and logs corresponding to the error:

```
START RequestId: 67a00244-3dcf-4aa8-a5d4-fdc506e75d8e Version: $LATEST
[ERROR] ClientError: An error occurred (AccessDeniedException) when calling the GetCostAndUsage operation: User: arn:aws:sts::110826698444:assumed-role/MonthlyUsageSMS-ExecutionRole;Reason:User is not authorized to perform the requested action.
Traceback (most recent call last):
  File "/var/task/lambda_function.py", line 16, in lambda_handler
    result = client.get_cost_and_usage()
  File "/var/lang/lib/python3.11/site-packages/botocore/client.py", line 534, in _api_call
    return self._make_api_call(operation_name, kwargs)
  File "/var/lang/lib/python3.11/site-packages/botocore/client.py", line 976, in _make_api_call
    raise error_class(parsed_response, operation_name)
END RequestId: 67a00244-3dcf-4aa8-a5d4-fdc506e75d8e
REPORT RequestId: 67a00244-3dcf-4aa8-a5d4-fdc506e75d8e Duration: 715.35 ms Billed Duration: 716 ms Memory Size: 128 MB Max Memory Used: 75 MB
```

The Request ID is listed at the bottom of the logs.

Workshop

The screenshot shows the AWS Lambda console interface for the function "MonthlyUsageSMS". The "Configuration" tab is selected. A success message at the top states: "The test event Event was successfully saved." The main area displays the function's details:

- Function Name:** MonthlyUsageSMS
- Layers:** (0)
- Last modified:** 15 minutes ago
- Function ARN:** arn:aws:lambda:ap-south-1:110826698444:function:MonthlyUsageSMS
- Function URL:** [Info](#)

The left sidebar lists various configuration options: General configuration, Triggers, Permissions (which is highlighted with a red box), Destinations, Function URL, Environment variables, Tags, VPC, Monitoring and operations tools, Concurrency, Asynchronous invocation, Code signing, Database proxies, File systems, and State machines.

In the "Execution role" section, the role name is listed as "MonthlyUsageSMS-role-ctgjhOad". The "Resource summary" section shows permissions for Amazon CloudWatch Logs, allowing CreateLogGroup, CreateLogStream, and PutLogEvents actions. The "Resource-based policy statements" section is empty.

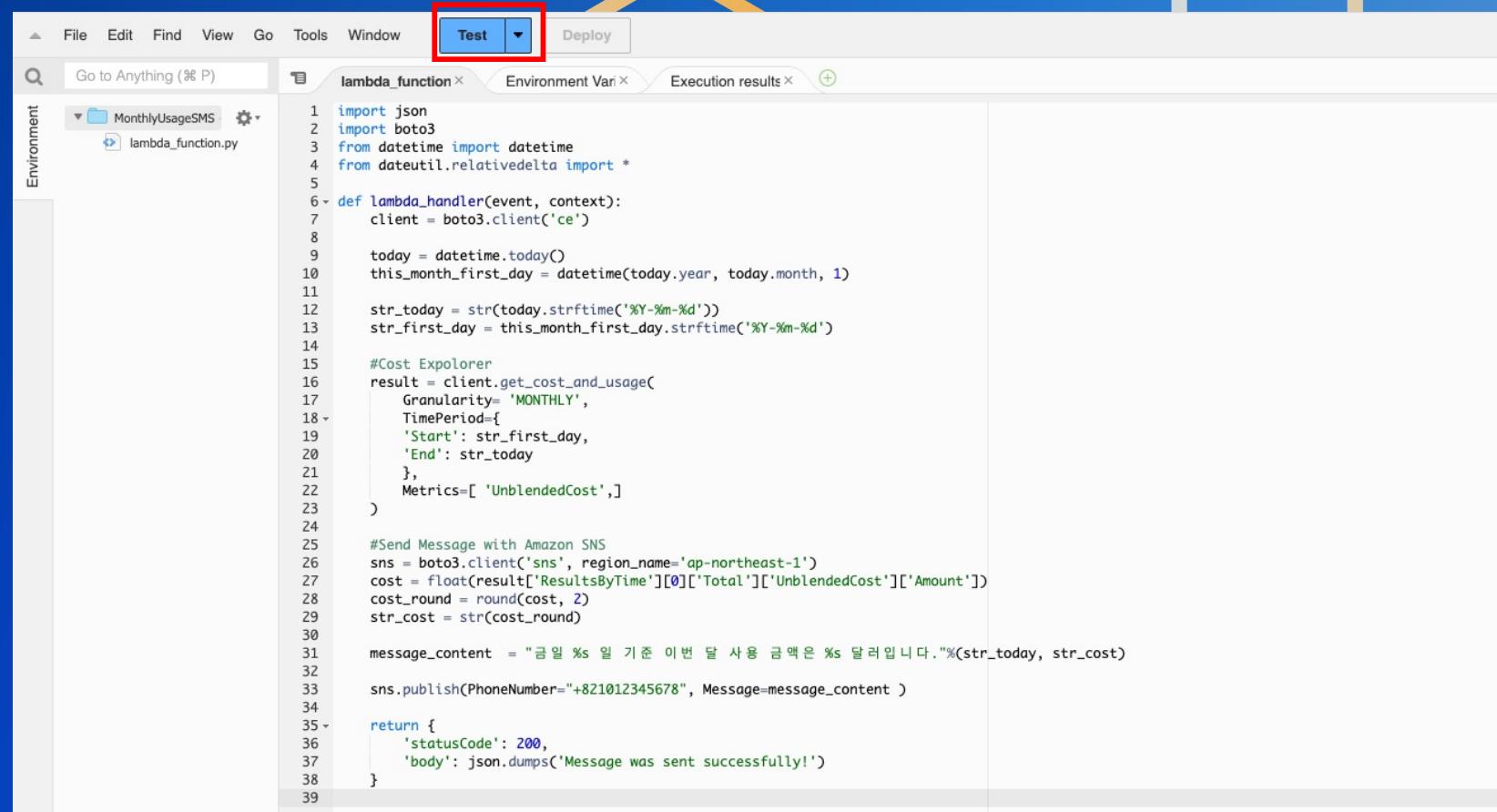


Workshop

The screenshot shows the AWS IAM Role Details page for the role `MonthlyUsageSMS-role-ctgjh0ad`. The left sidebar shows the IAM navigation menu with the `Roles` option selected. The main content area displays the `Summary` tab, which includes details like creation date (November 15, 2023, 17:01 UTC+09:00), ARN (`arn:aws:iam::110826698444:role/service-role/MonthlyUsageSMS-role-ctgjh0ad`), and maximum session duration (1 hour). Below the summary is the `Permissions` tab, which lists one attached policy: `AWSLambdaBasicExecutionRole-1f1f2922-461a-4a96-b78b-f...` (Customer managed). A red box highlights the `Add permissions`, `Attach policies`, and `Create inline policy` buttons. The bottom of the page includes CloudShell, Feedback, and footer links for Privacy, Terms, and Cookie preferences.



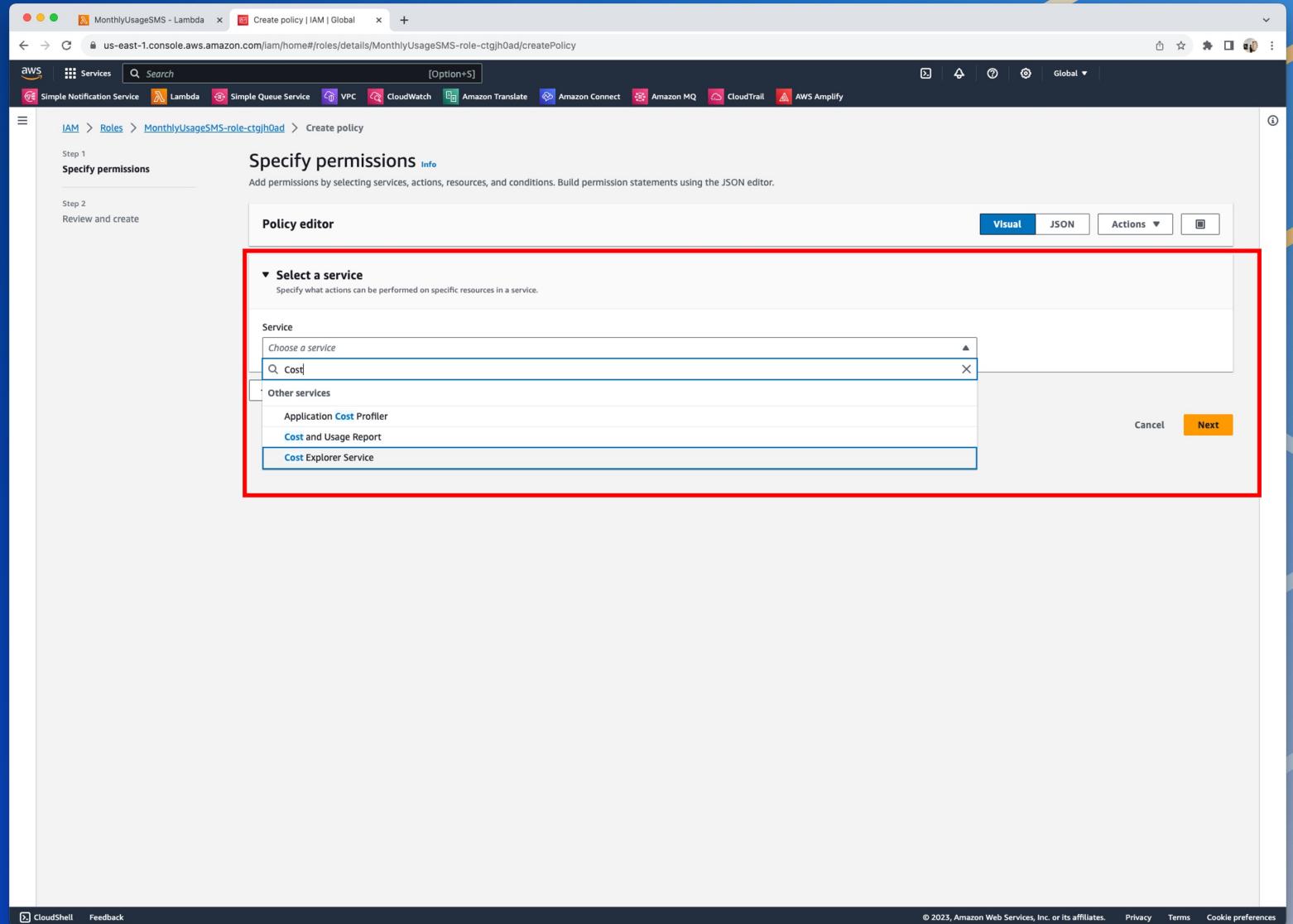
Workshop



The screenshot shows the AWS Lambda function editor interface. At the top, there is a toolbar with the following buttons from left to right: File, Edit, Find, View, Go, Tools, Window, a search bar labeled "Go To Anything (% P)", and three main buttons: "Test" (highlighted with a red box), "Deploy", and a dropdown arrow. Below the toolbar, there are tabs for "lambda_function", "Environment Vari", and "Execution results". The "lambda_function" tab is active. On the left, there is a sidebar titled "Environment" with a dropdown menu and a file tree showing a folder named "MonthlyUsageSMS" containing a file "lambda_function.py". The main content area displays the Python code for the lambda function:

```
1 import json
2 import boto3
3 from datetime import datetime
4 from dateutil.relativedelta import *
5
6 def lambda_handler(event, context):
7     client = boto3.client('ce')
8
9     today = datetime.today()
10    this_month_first_day = datetime(today.year, today.month, 1)
11
12    str_today = str(today.strftime('%Y-%m-%d'))
13    str_first_day = this_month_first_day.strftime('%Y-%m-%d')
14
15    #Cost Explorer
16    result = client.get_cost_and_usage(
17        Granularity='MONTHLY',
18        TimePeriod={
19            'Start': str_first_day,
20            'End': str_today
21        },
22        Metrics=['UnblendedCost'],
23    )
24
25    #Send Message with Amazon SNS
26    sns = boto3.client('sns', region_name='ap-northeast-1')
27    cost = float(result['ResultsByTime'][0]['Total']['UnblendedCost']['Amount'])
28    cost_round = round(cost, 2)
29    str_cost = str(cost_round)
30
31    message_content = "금 일 %s 일 기준 이번 달 사용 금액은 %s 달러입니다."%(str_today, str_cost)
32
33    sns.publish(PhoneNumber="+821012345678", Message=message_content)
34
35    return {
36        'statusCode': 200,
37        'body': json.dumps('Message was sent successfully!')
38    }
39
```

Workshop



Workshop

The screenshot shows the AWS IAM Policy editor interface. The title bar indicates it's for a role named "MonthlyUsageSMS - Lambda" and is creating a new policy. The main area is titled "Policy editor" and shows the "Cost Explorer Service" section. Under "Actions allowed", the "Read" section is expanded, showing a list of actions. The action "GetCostAndUsage" is selected and highlighted with a red box. Other actions listed include "DescribeCostCategoryDefinition", "GetAnomalies", "GetConsoleActionSetEnforced", "GetCostCategories", "GetPreferences", "GetReservationUtilization", "GetSavingsPlansCoverage", "GetSavingsPlansUtilizationDetails", "ListTagsForResource", "DescribeNotificationSubscription", "GetAnomalyMonitors", "GetCostForecast", "GetReservationCoverage", "GetRightsizingRecommendation", "GetSavingsPlansPurchaseRecommendation", "GetTags", "DescribeReport", "GetAnomalySubscriptions", "GetCostAndUsageWithResources", "GetDimensionValues", "GetReservationPurchaseRecommendationDetails", "GetSavingsPlanPurchaseRecommendationDetails", "GetSavingsPlansUtilization", and "GetUsageForecast". The "Effect" dropdown is set to "Allow".



Workshop

The screenshot shows the AWS IAM 'Create policy' interface for the 'MonthlyUsageSMS - Lambda' role. The policy document is titled 'Create policy | IAM | Global'. The 'Effect' is set to 'Allow'. The policy grants the 'sns:Publish' action on the 'sns:' resource. The 'sns:Publish' action is highlighted with a red box.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "sns:Publish",
            "Effect": "Allow",
            "Resource": "sns:*"
        }
    ]
}
```



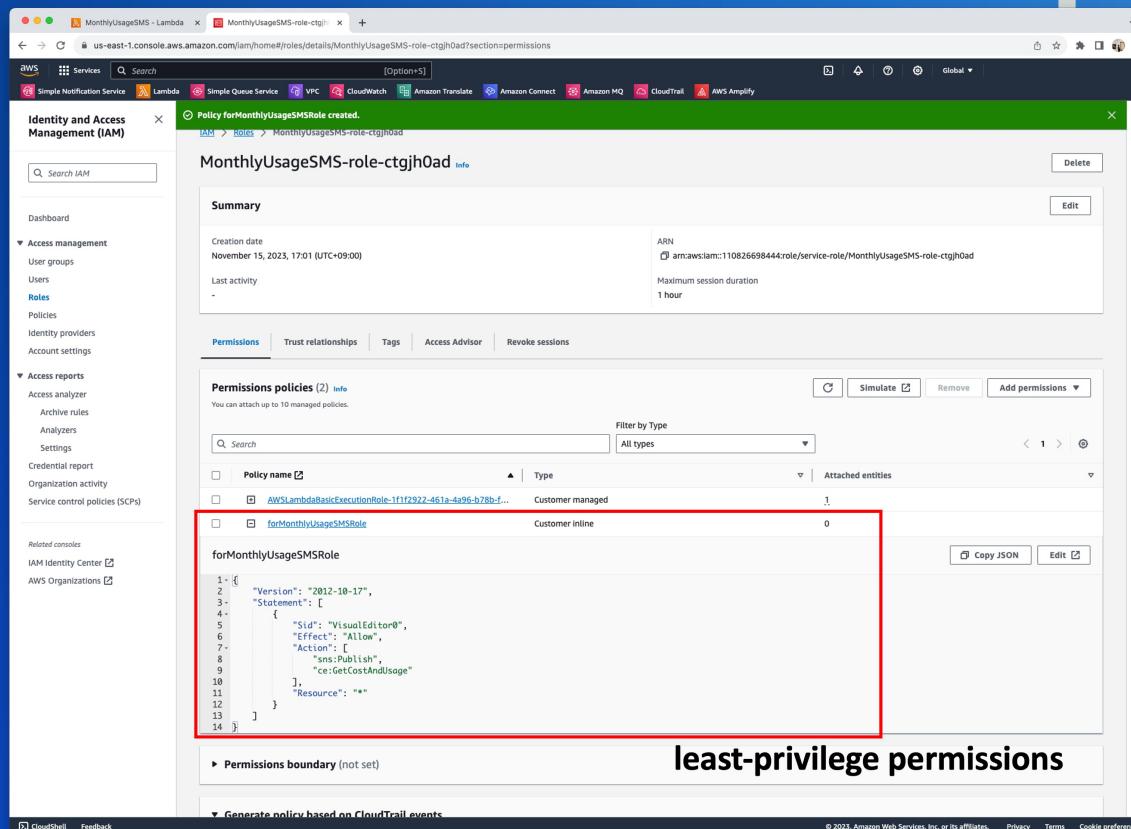
Workshop

The screenshot shows the 'Create policy' wizard in the AWS IAM console. The policy is named 'forMonthlyUsageSMSRole'. It has two permissions defined:

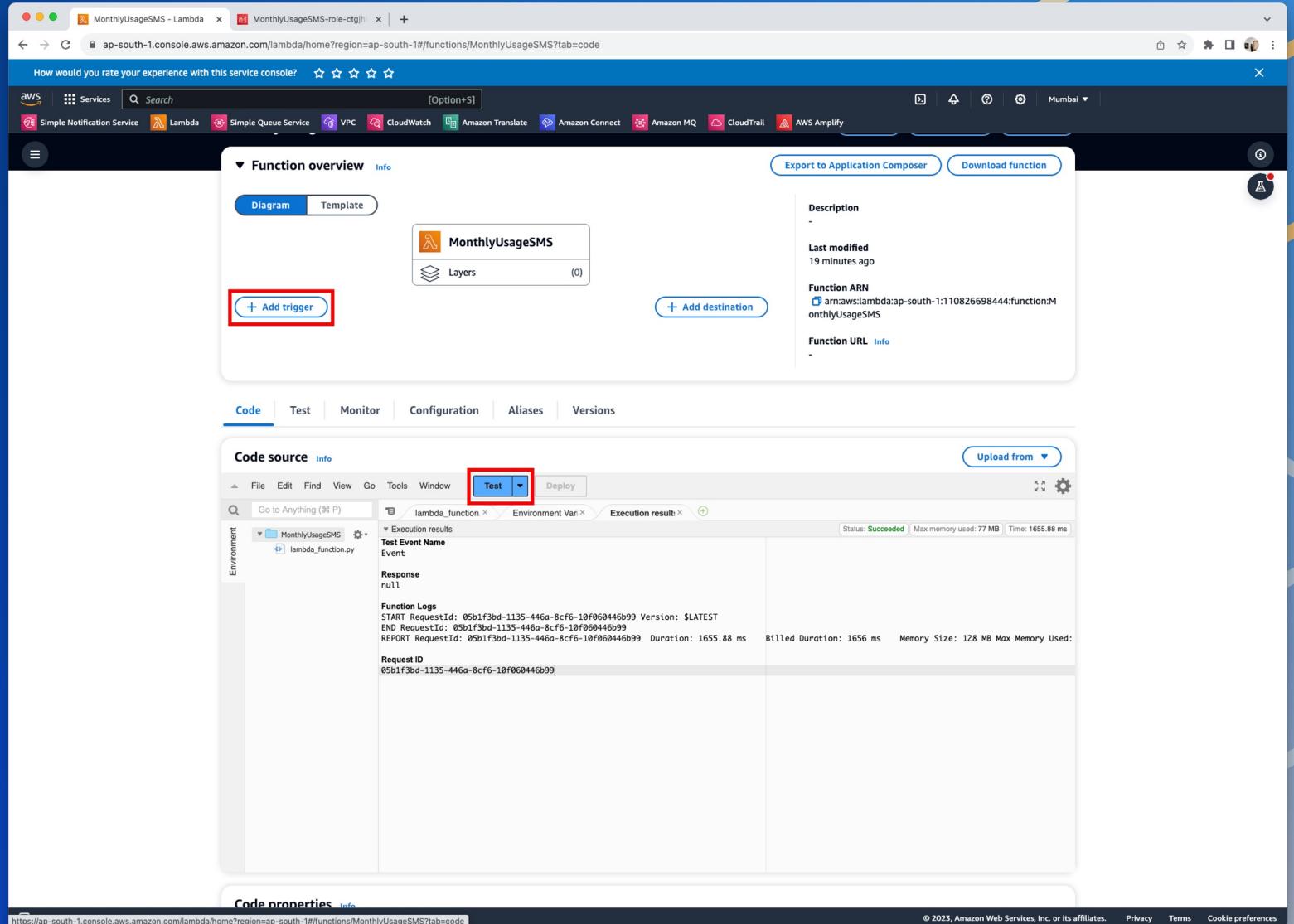
Service	Access level	Resource	Request condition
Cost Explorer Service	Limited: Read	All resources	None
SNS	Limited: Write	All resources	None

The 'Create policy' button at the bottom right is highlighted with a red box.

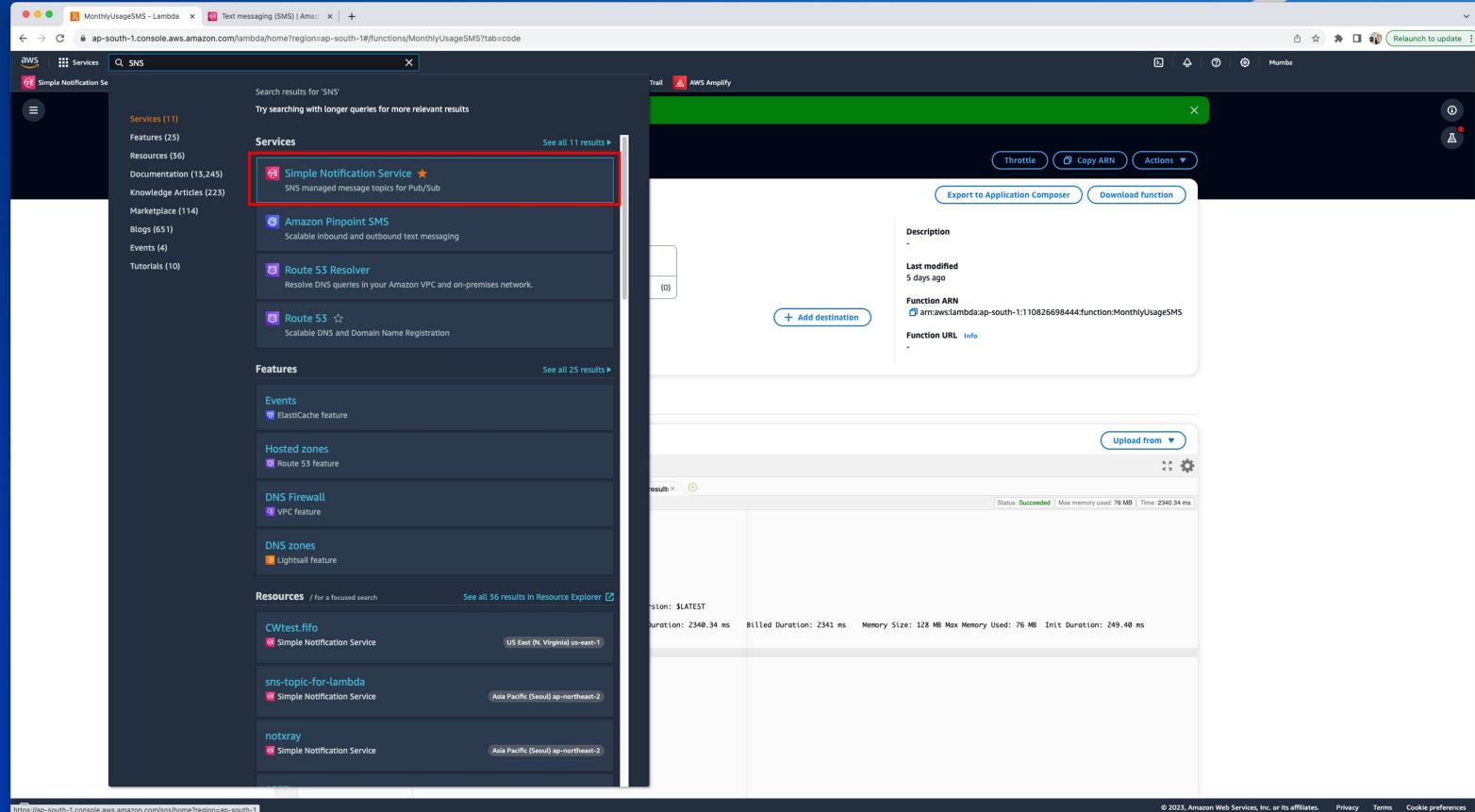
Workshop



Workshop



Workshop



Workshop

The screenshot shows the AWS Simple Notification Service (SNS) homepage. The left sidebar navigation includes 'Dashboard', 'Topics', 'Subscriptions', 'Mobile' (with 'Push notifications' and 'Text messaging (SMS)' listed, where 'Text messaging (SMS)' is highlighted with a red box), and 'Origination numbers'. The main content area features a 'New Feature' banner about in-place message archiving and replay for FIFO topics. Below it, the heading 'Amazon Simple Notification Service' is displayed, followed by the subtext 'Pub/sub messaging for microservices and serverless applications.' A detailed description of Amazon SNS follows, mentioning its availability, durability, and use cases like decoupling microservices and event-driven serverless applications. To the right, there are sections for 'Create topic', 'Pricing', 'Documentation', 'Explore AWS', and 'Use cases'.



Workshop

MonthlyUsageSMS - Lambda Text messaging (SMS) | Amazon SNS

ap-south-1.console.aws.amazon.com/sns/v3/home?region=ap-south-1#/mobile/text-messaging

AWS Services Search [Option+S]

Simple Notification Service Lambda Simple Queue Service VPC CloudWatch Amazon Translate Amazon Connect Amazon MQ CloudTrail AWS Amplify

Mumbai

Amazon SNS Mobile text messaging (SMS)

View origination numbers Subscribe number to topic Publish text message

Dashboard Topics Subscriptions

Mobile Push notifications Text messaging (SMS) Origination numbers

▼ Overview

Amazon SNS lets you send SMS text messages to any phone number. [Learn more](#)

Amazon SNS

Direct publish

Publish text messages

You can send SMS messages to millions of users through a topic or directly to their devices

SNS topic

Account information

Status

This account is in the SMS sandbox in Asia Pacific (Mumbai).

When in the sandbox, you can only deliver SMS to the sandbox destination phone numbers you have verified below. [Learn more](#)

Exit SMS Sandbox

Sandbox destination phone numbers (0)

Verify phone number Delete phone number Add phone number

Search

Verified phone number

No phone numbers added

To get started sending SMS, add a verified number.

Add phone number

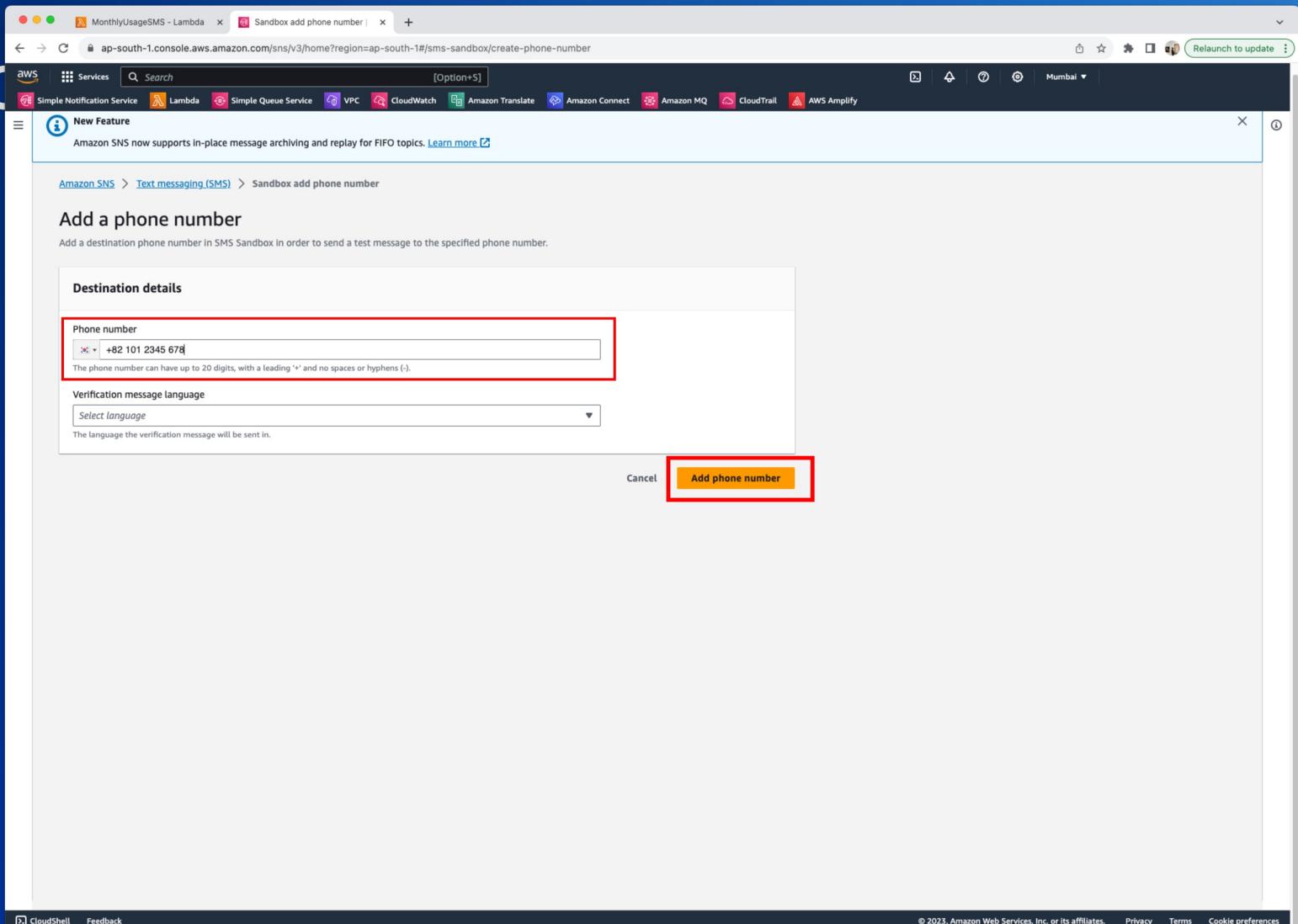
Delivery status logs

CloudShell Feedback

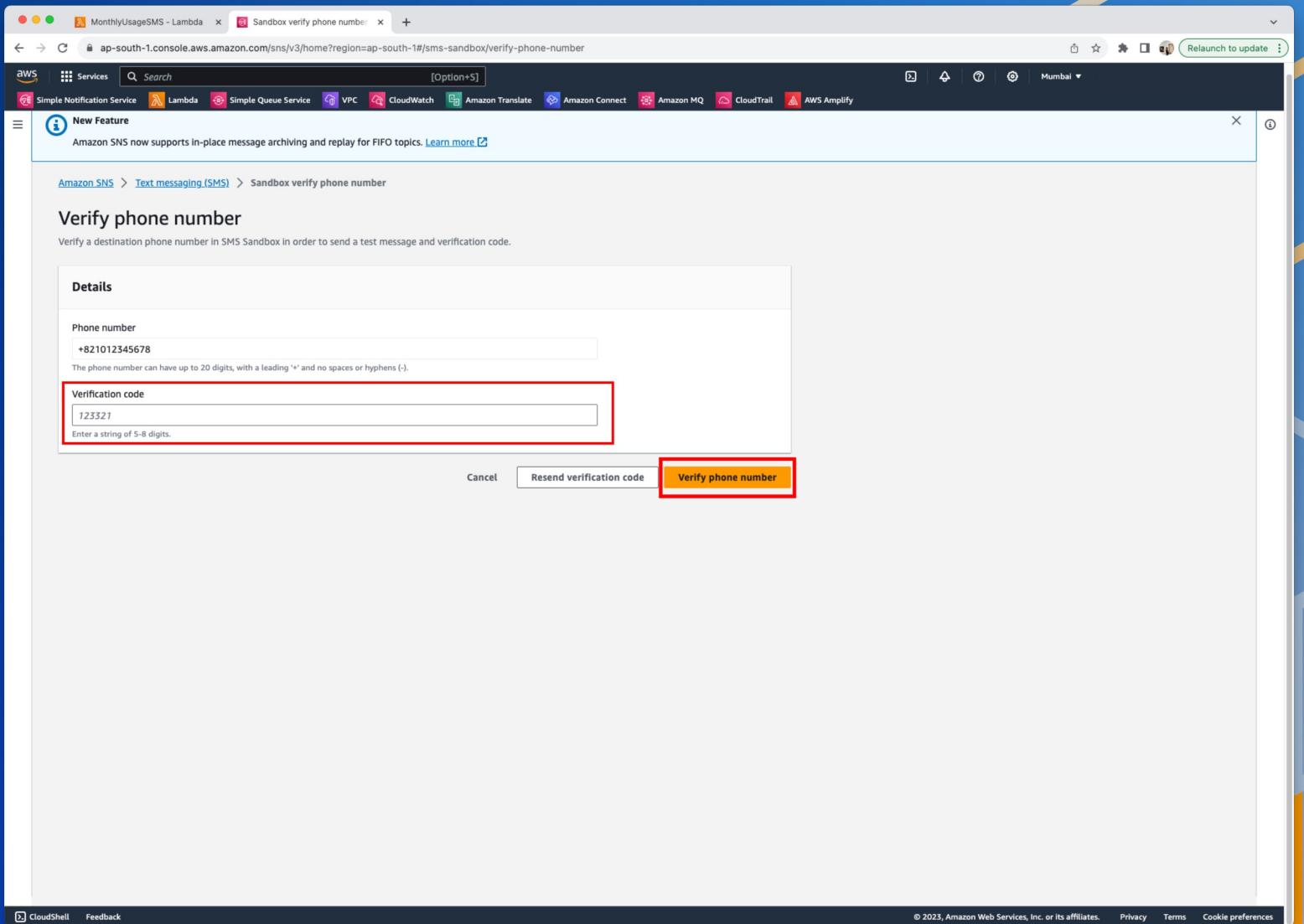
© 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

The screenshot shows the AWS SNS mobile text messaging (SMS) console. The left sidebar has a 'Mobile' section with 'Text messaging (SMS)' selected. The main area has a 'Overview' section with a diagram showing the flow from Amazon SNS to a SNS topic, which then branches into 'Direct publish' and 'Publish text messages' options. Below this is an 'Account information' section indicating the account is in the SMS sandbox in Asia Pacific (Mumbai). The 'Sandbox destination phone numbers' section shows 0 numbers, with a red box highlighting the 'Add phone number' button. The bottom navigation bar includes CloudShell, Feedback, and links to AWS terms and conditions.

Workshop



Workshop



Workshop

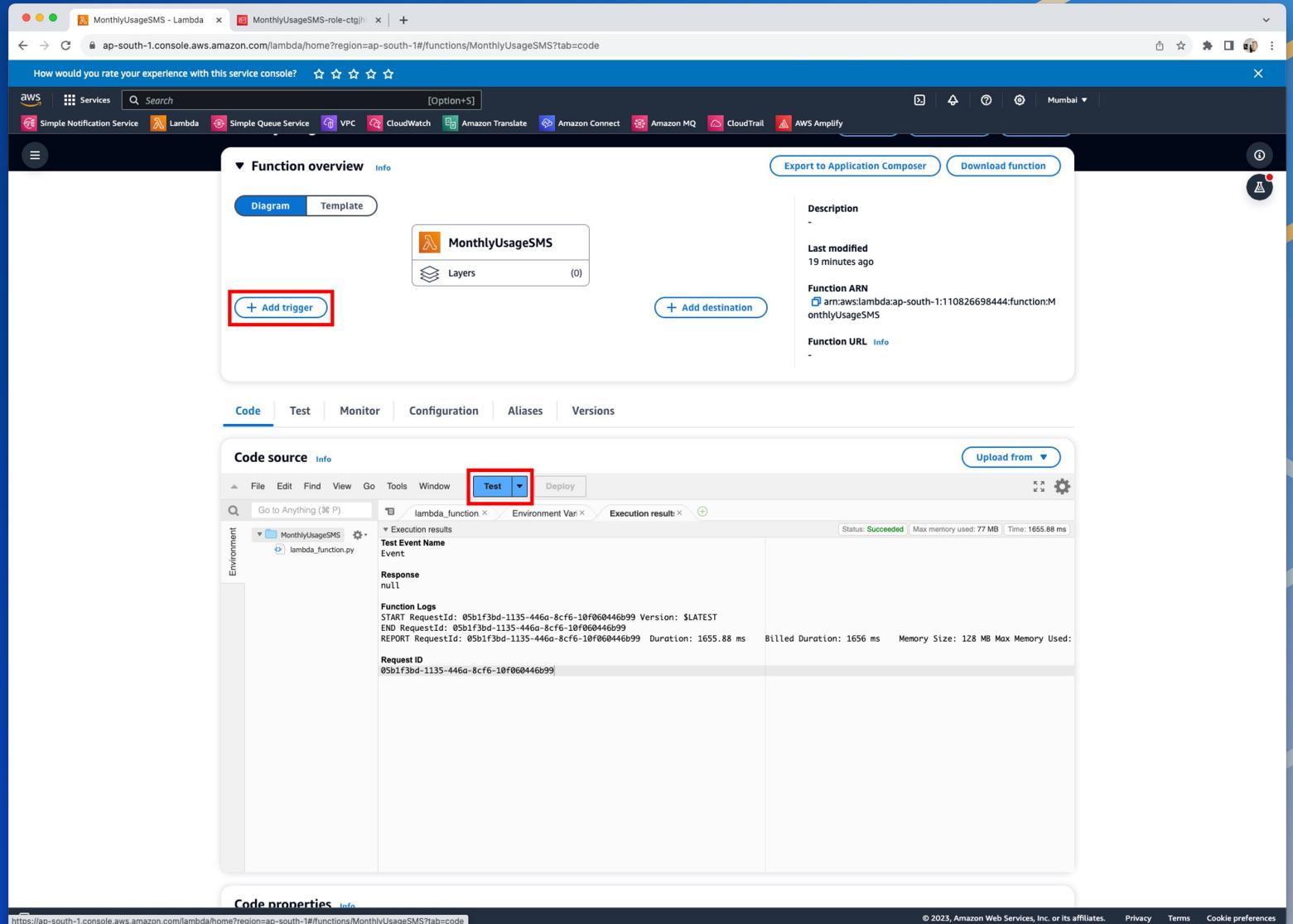
The screenshot shows the AWS SNS Text messaging (SMS) dashboard. The left sidebar has a 'Mobile' section with 'Text messaging (SMS)' selected. The main area displays 'Account information' with a note about being in the SMS sandbox. Below is a table for 'Sandbox destination phone numbers' containing two entries, both of which are highlighted with a red box:

Verified phone number	Verification status
+821012345678	Pending
+821012345678	Verified

Below the table are sections for 'Delivery status logs' (with a note to enable delivery status logging) and 'Delivery statistics (UTC)' showing promotional and transactional message metrics.



Workshop



Workshop

[국외발신]
금일 2023-11-15 일 기준 이번 달 사용 금액은 966.17 달러입니다.

[국외발신]
금일 2023-11-15 일 기준 이번 달 사용 금액은 966.17 달러입니다.

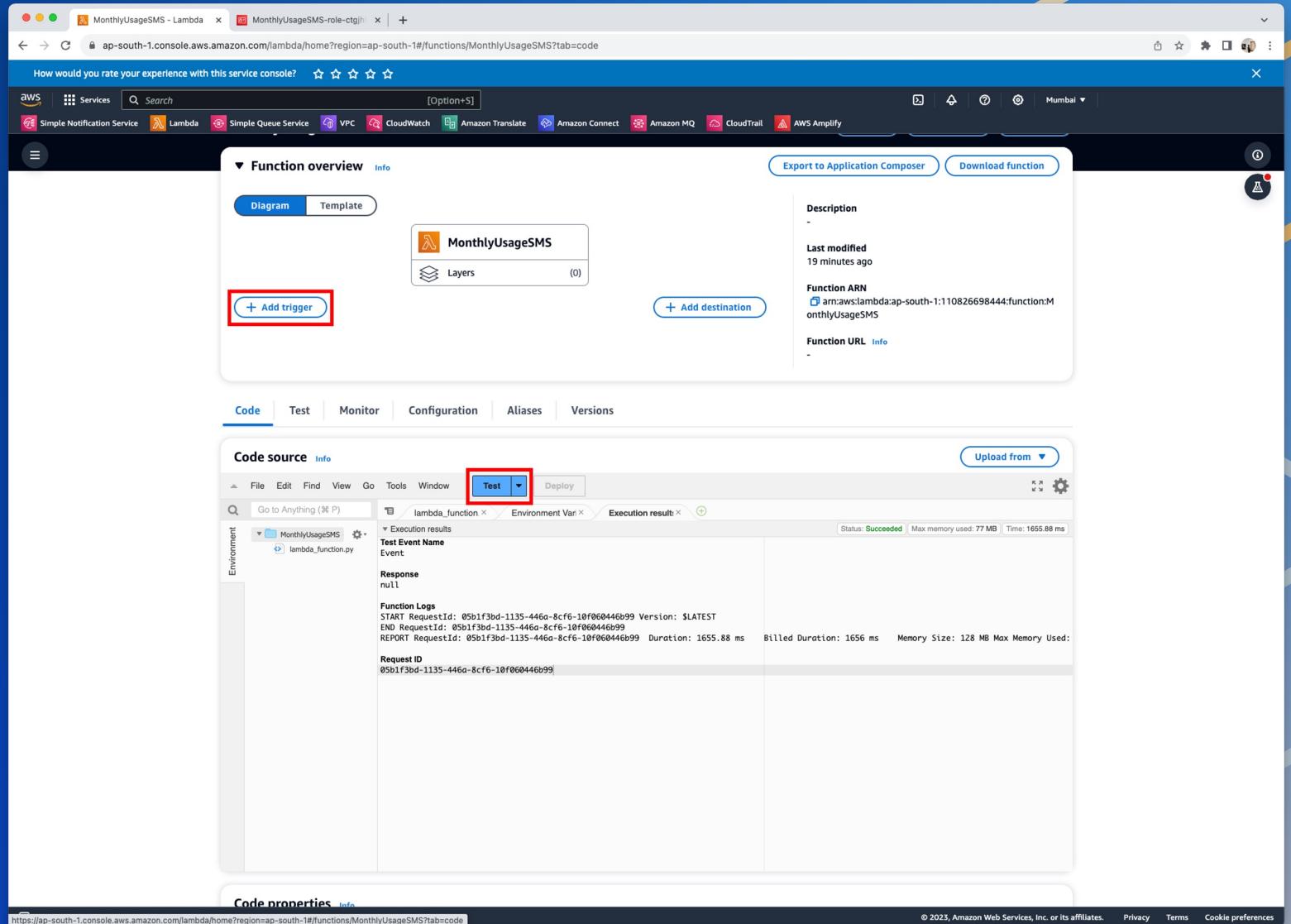
[국외발신]
금일 2023-11-15 일 기준 이번 달 사용 금액은 966.17 달러입니다.

[국외발신]
금일 2023-11-15 일 기준 이번 달 사용 금액은 966.17 달러입니다.

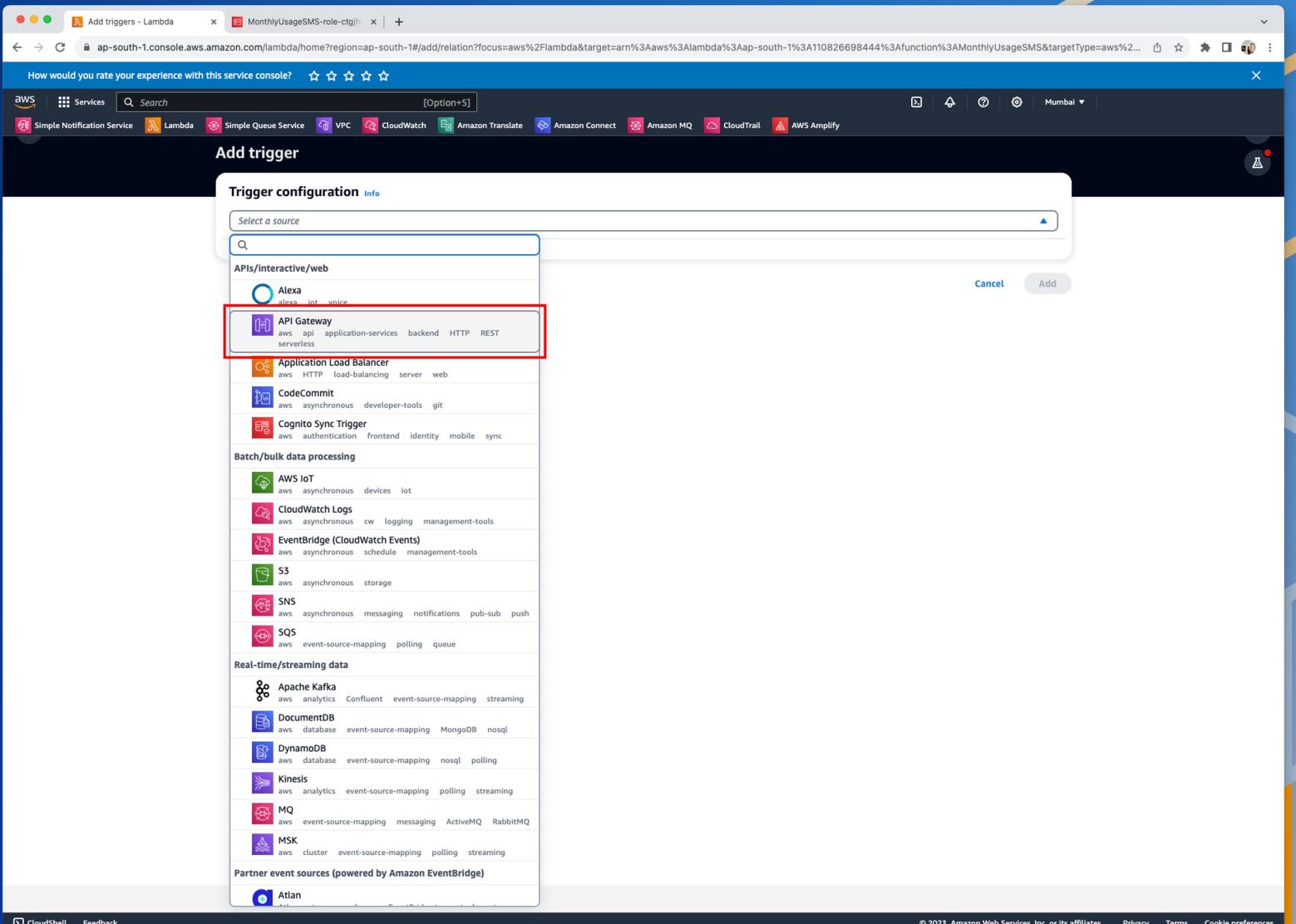
[국외발신]
금일 2023-11-15 일 기준 이번 달 사용 금액은 966.17 달러입니다.

[국외발신]
금일 2023-11-15 일 기준 이번 달 사용 금액은 966.17 달러입니다.

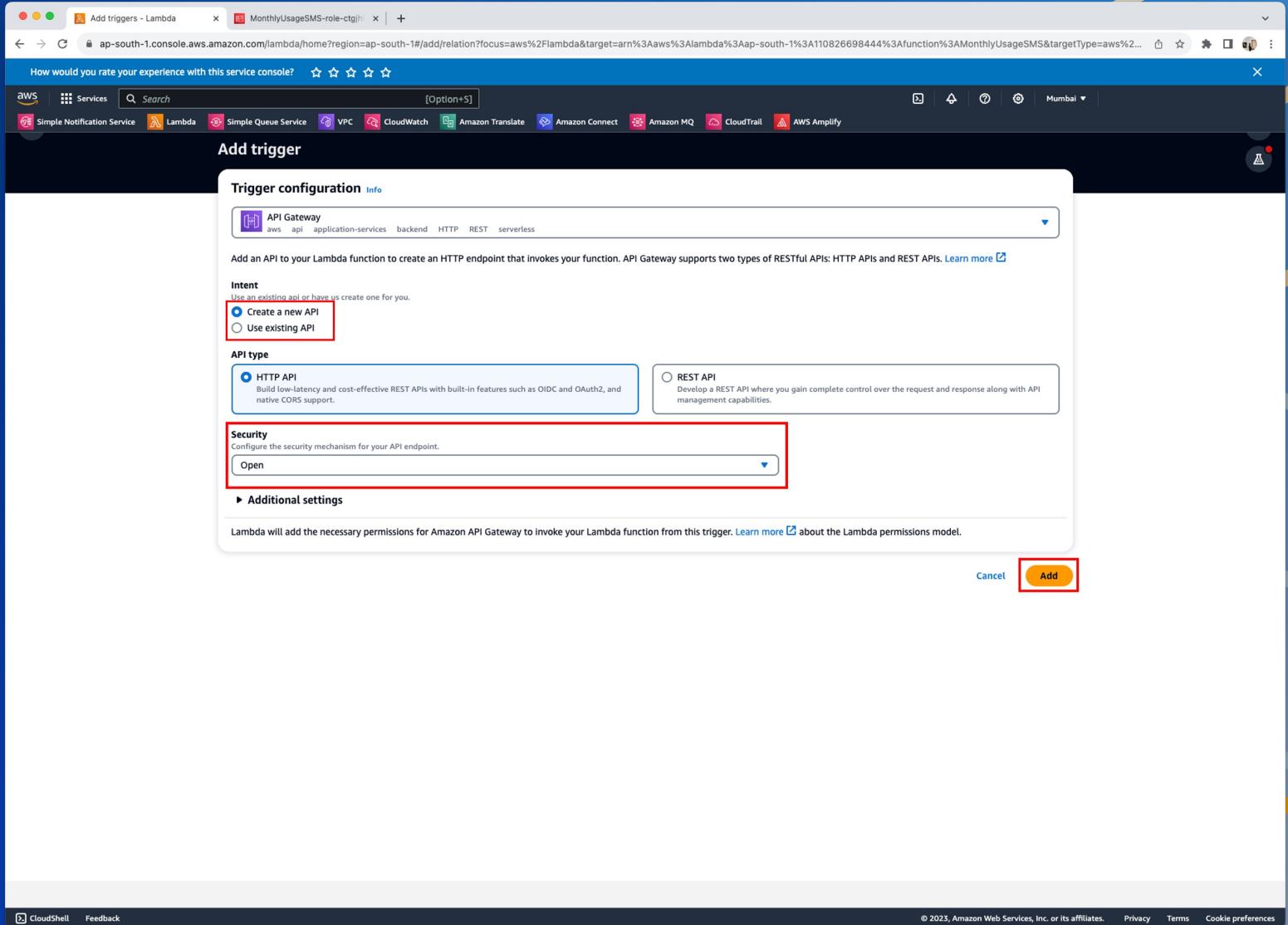
Workshop



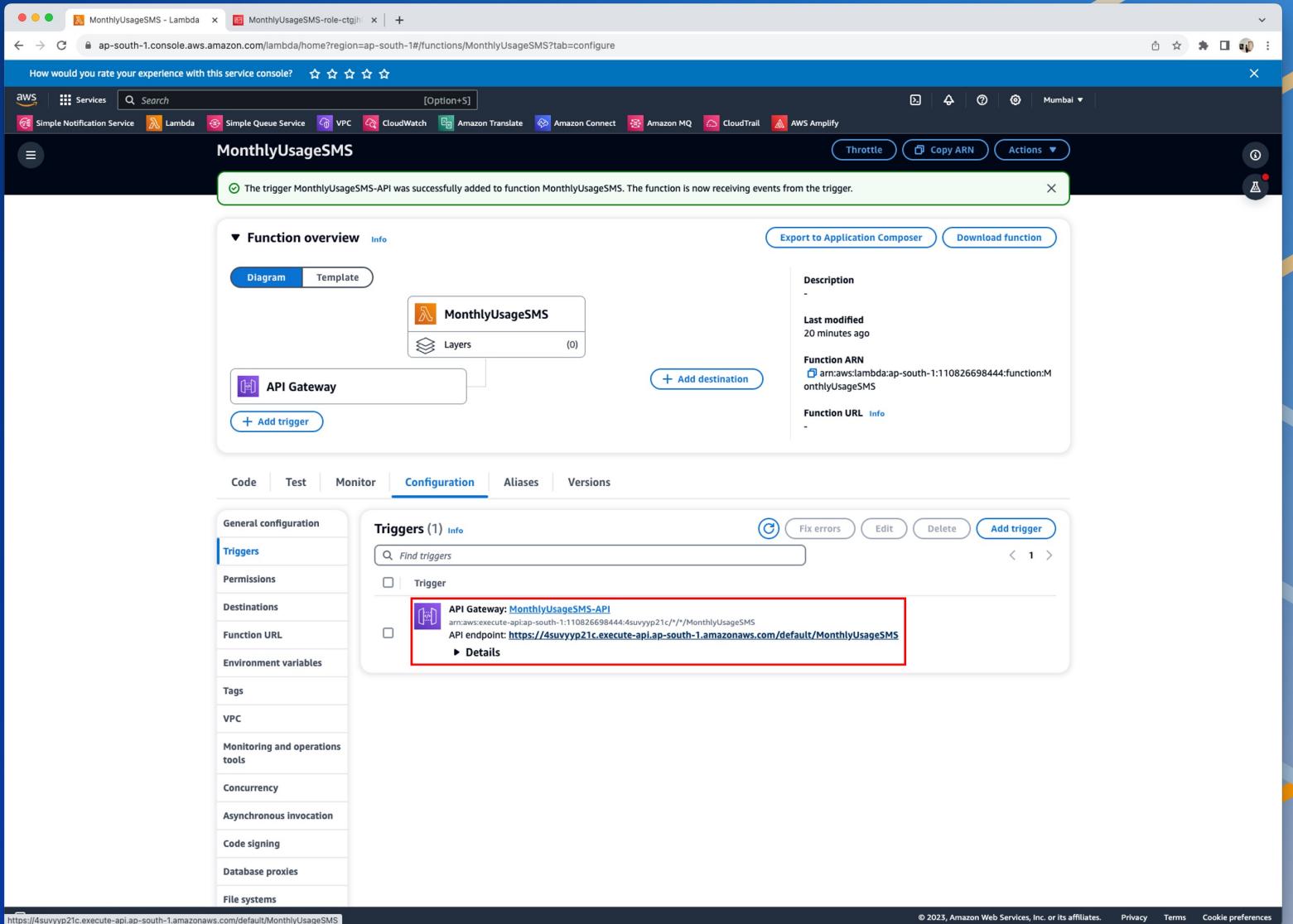
Workshop



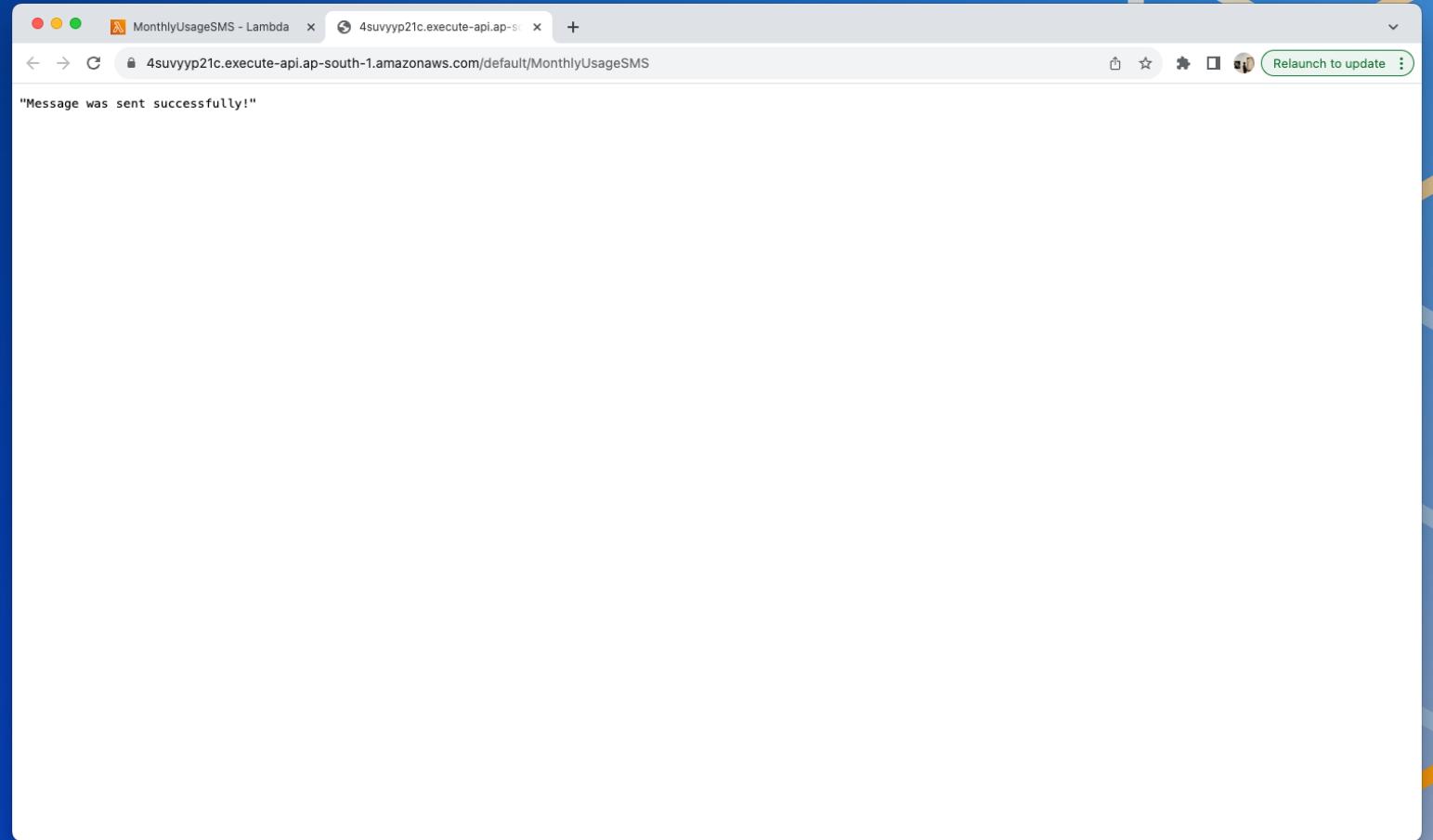
Workshop



Workshop



Workshop



Workshop

[국외발신]
금일 2023-11-15 일 기준 이번 달 사용 금액은 966.17 달러입니다.

[국외발신]
금일 2023-11-15 일 기준 이번 달 사용 금액은 966.17 달러입니다.

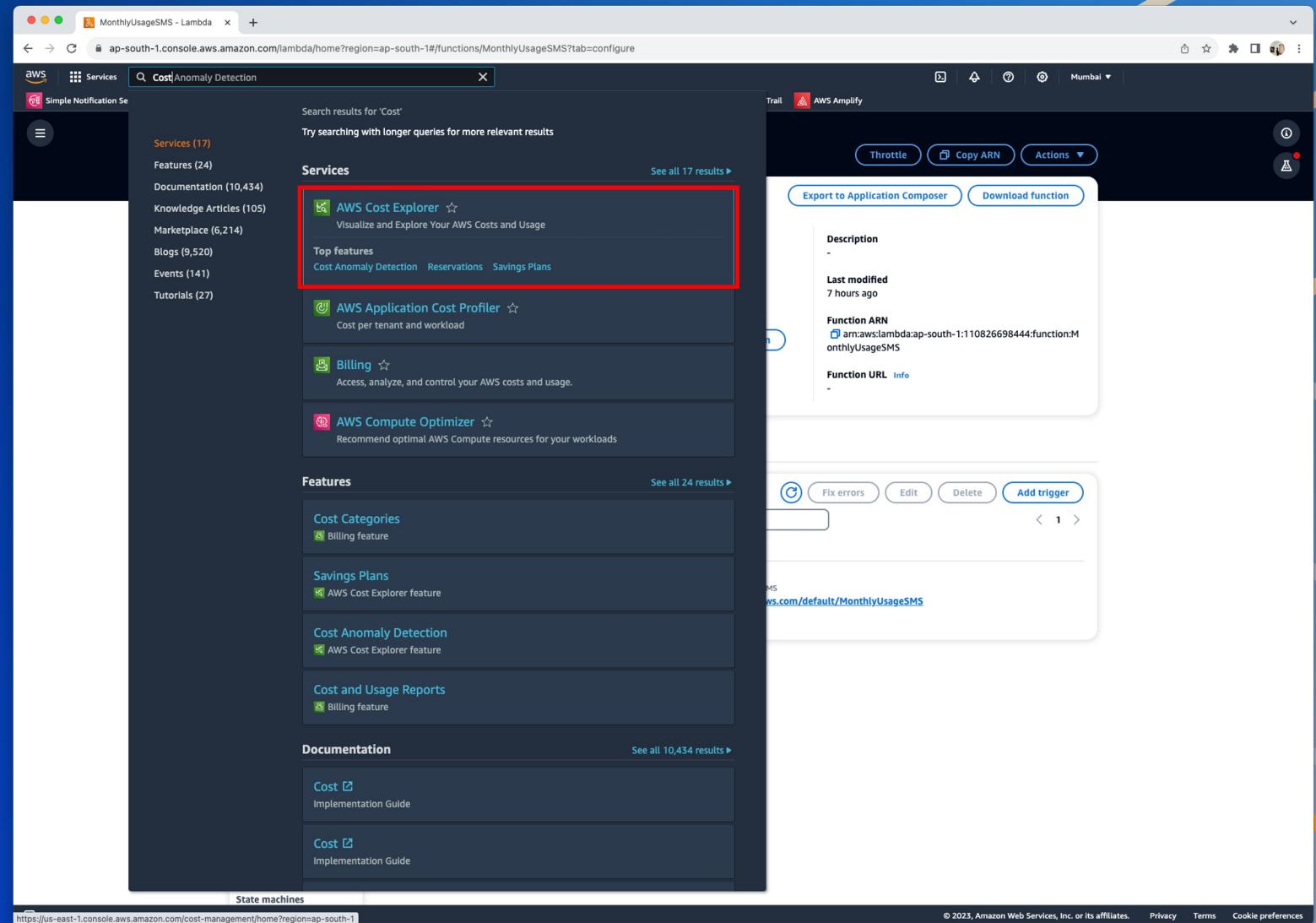
[국외발신]
금일 2023-11-15 일 기준 이번 달 사용 금액은 966.17 달러입니다.

[국외발신]
금일 2023-11-15 일 기준 이번 달 사용 금액은 966.17 달러입니다.

[국외발신]
금일 2023-11-15 일 기준 이번 달 사용 금액은 966.17 달러입니다.

[국외발신]
금일 2023-11-15 일 기준 이번 달 사용 금액은 966.17 달러입니다.

Workshop



Workshop

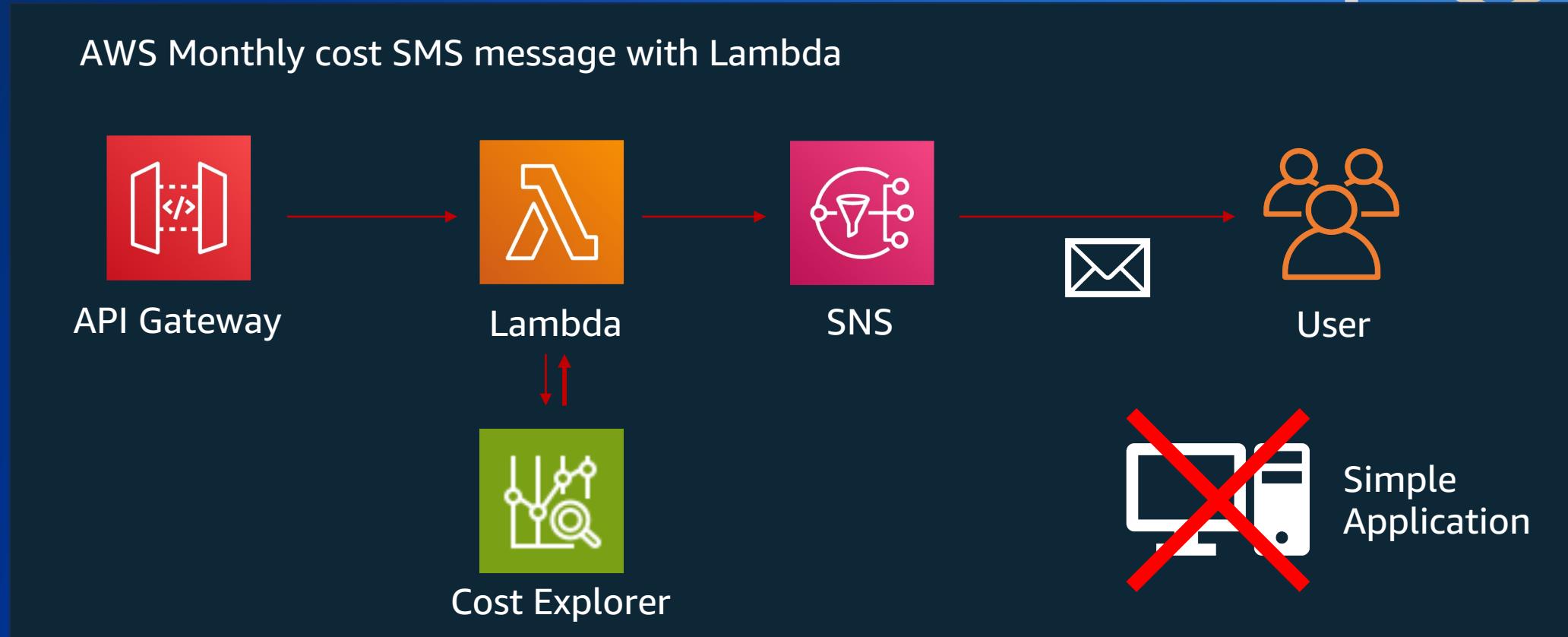
Screenshot of the AWS Cost Explorer Home page (us-east-1.console.aws.amazon.com/cost-management/home?region=ap-south-1#/dashboard).

The page displays the following information:

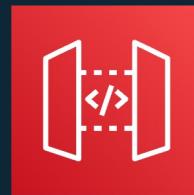
- Cost summary:**
 - Current month costs: **\$983.15** (Up 0% over last month)
 - Forecasted month-end costs: **\$1,686.71** (Up 5% over last month)
- Daily unblended costs:** A bar chart showing daily costs from Oct 01 to Nov 15. The Y-axis represents Cost (\$) from 0 to 100. The X-axis shows dates from Oct 01 to Nov 15. The chart shows a general downward trend with some fluctuations.
- Recently accessed reports:** No recently accessed reports.
- November trends:**
 - Service usage: Amazon MQ costs are up \$98.11 (13%).
 - Account usage: hjoo's account (11082669844) costs are up \$61.61 (6%).
 - Region usage: us-east-1 costs are up \$366.51 (165%).
- More resources:**
 - What is AWS Billing and Cost Management?
 - Documentation
 - FAQ



Workshop



Delete Resources



API Gateway



Lambda



SNS

Only pay for what you use



Thank you!

Hyoungjoo Lee

