

PRACTICAL: 5

AIM:

Scenario:

The café is eager to launch a dynamic version of their website so that the website can access data stored in a database. Sofía has been making steady progress toward this goal. In a previous lab, you played the role of Sofía and created a DynamoDB database. The database table contains café menu details, and an index holds menu items that are flagged as specials. Then, in another lab, you created an API to add the ability for the website to receive mock data through REST API calls. In this lab, you will again play the role of Sofía. You will replace the mock endpoints with functional endpoints so that the web application can connect to the database. You will use Lambda to bridge the connection between the GET APIs and the data stored in DynamoDB. Finally, for the POST API call, Lambda will return an updated acknowledgment message.

Lab overview and objectives:

In this lab, you will use the AWS SDK for Python (boto3) to create AWS Lambda functions. Calls to the REST API that you created in the earlier Amazon API Gateway lab will initiate the functions. One of the Lambda functions will perform either an Amazon DynamoDB database table scan or an index scan. Another Lambda function will return a standard acknowledgment message that you will enhance later in a lab where implement Amazon Cognito.

After completing this lab, you should be able to:

- Create a Lambda function that queries a DynamoDB database table. · Grant sufficient permissions to a Lambda function so that it can read data from DynamoDB.
- Configure REST API methods to invoke Lambda functions using Amazon API Gateway.

THEORY:

This lab focuses on integrating AWS services—DynamoDB, Lambda, and API Gateway—to create a dynamic café website that can retrieve and display menu details stored in a database. **Amazon DynamoDB** serves as the backend database, storing menu items with attributes like name, description, price, and a flag for specials. A secondary index is used to efficiently query special menu items. **AWS Lambda** acts as the intermediary, running serverless functions to handle API calls and connect to DynamoDB. One Lambda function performs a database table scan or index scan to fetch menu data, while another function returns acknowledgment messages for POST requests.

Amazon API Gateway facilitates the communication between the web application and Lambda by exposing REST API endpoints. To achieve this, API methods (GET and POST) are configured to invoke the respective Lambda functions. The Lambda functions are granted **IAM permissions** to read data from DynamoDB. This architecture ensures a seamless, dynamic interaction between the web

application and the database, providing real-time data retrieval for menu items and a scalable solution for future enhancements like user authentication.

CODE:

update_config.py

```
import boto3
S3API = boto3.client("s3", region_name="us-east-1")
bucket_name = "c144426a373434419003220t1w816396249192-s3bucket-
dxyfssytdezp"

filename = "/home/ec2-user/environment/resources/website/config.js"
S3API.upload_file(filename, bucket_name, "config.js", ExtraArgs={'ContentType':
"application/js", "CacheControl": "max-age=0"})

print ("DONE")
```

get_all_products_code.py

```
import boto3, json
from boto3.dynamodb.conditions import Key
from boto3.dynamodb.conditions import Key, Attr, Not

TABLE_NAME_STR = 'FoodProducts'
INDEX_NAME_STR = 'special_GSI'
DDB = boto3.resource('dynamodb', region_name='us-east-1')

def lambda_handler(event, context):

    offer_path_str = event.get('path')
    if offer_path_str is not None:
        return scan_index(event, context)
    else:
        pass
    print("running scan on table")

    DDB = boto3.resource('dynamodb', region_name='us-east-1')

    TABLE = DDB.Table(TABLE_NAME_STR)

    response = TABLE.scan()
```

```
data = response['Items']

while 'LastEvaluatedKey' in response:
    response = TABLE.scan(ExclusiveStartKey=response['LastEvaluatedKey'])
    print("We needed to paginate and extend the response")
    data.extend(response['Items'])

#python return non standard JSON
#so we need a helper to convert Decimal('595') and special returned by dynamo
#to an integer like 595

for item in data:
    item['price_in_cents_int'] = item.pop('price_in_cents')
    if item.get('special') is not None:
        item['special_int'] = item.pop('special')
    item['tag_str_arr'] = item.pop('tags')
    item['description_str'] = item.pop('description')
    item['product_name_str'] = item.pop('product_name')
    item['product_id_str'] = item.pop('product_id')

    if item['price_in_cents_int']:
        item['price_in_cents_int'] = int(item['price_in_cents_int'])
    if item.get('special_int') is not None:
        item['special_int'] = int(item['special_int'])

return_me={"product_item_arr": data}

return return_me

def scan_index(event, context):

    print("running scan on index")
    ## event and context not used
    TABLE = DDB.Table(TABLE_NAME_STR)

    response = TABLE.scan(
        IndexName=INDEX_NAME_STR,
        FilterExpression=Not(Attr("tags").contains("out of stock"))
    )

    data = response['Items']

    while 'LastEvaluatedKey' in response:
```

```

response = TABLE.scan(
    ExclusiveStartKey=response['LastEvaluatedKey'],
    IndexName=INDEX_NAME_STR,
    FilterExpression=Not(Attr("tags").contains("out of stock"))
)
print("We needed to paginate and extend the response")
data.extend(response['Items'])

#python return non standard JSON
#so we need a helper to convert Decimal('595') and special returned by dynamo
#to an integer like 595
for item in data:
    item['price_in_cents_int'] = item.pop('price_in_cents')
    item['special_int'] = item.pop('special')
    item['tag_str_arr'] = item.pop('tags')
    item['description_str'] = item.pop('description')
    item['product_name_str'] = item.pop('product_name')
    item['product_id_str'] = item.pop('product_id')

    if item['price_in_cents_int']:
        item['price_in_cents_int'] = int(item['price_in_cents_int'])
    if item.get('special_int') is not None:
        item['special_int'] = int(item['special_int'])

return_me = {
    "product_item_arr": data
}
return return_me

#remove this line below once you have tested locally and wish to deploy
print(lambda_handler({}, None))

get_all_products_wrapper.py

import boto3
import subprocess

client = boto3.client('lambda', region_name='us-east-1')
ROLE = 'arn:aws:iam::816396249192:role/LambdaAccessToDynamoDB'
BUCKET = subprocess.getoutput('aws s3api list-buckets --query "Buckets[].Name" |
grep s3bucket | tr -d "," | xargs')

response = client.create_function(

```

```
FunctionName='get_all_products',
Runtime='python3.8',
Role=ROLE,
Handler='get_all_products_code.lambda_handler',
Code={
    'S3Bucket': BUCKET,
    'S3Key': 'get_all_products_code.zip'
}
)

print ("DONE")

create_report_wrapper.py

import boto3
import subprocess

client = boto3.client('lambda', region_name='us-east-1')
ROLE = 'arn:aws:iam::816396249192:role/LambdaAccessToDynamoDB'
BUCKET = subprocess.getoutput('aws s3api list-buckets --query "Buckets[].Name" |
grep s3bucket | tr -d "," | xargs')

response = client.create_function(
    FunctionName='create_report',
    Runtime='python3.8',
    Role=ROLE,
    Handler='create_report_code.lambda_handler',
    Code={
        'S3Bucket': BUCKET,
        'S3Key': 'create_report_code.zip'
    }
)

print ("DONE")
```

OUTPUT:

The screenshot shows a terminal window within a web browser interface. The browser tabs include 'Cc lab playlist - 22it056@charu...', 'Home', 'Lab 7.1: Creating Lambda Funct...', 'AWS Cloud9', and 'Cloud9 Instance - AWS Cloud9'. The address bar shows the URL: 'us-east-1.console.aws.amazon.com/cloud9/ide/83b53caec6c94a98b3cd65d18745ef2d?region=us-east-1'. The terminal window has a menu bar with 'File', 'Edit', 'Find', 'View', 'Go', 'Run', 'Tools', 'Window', 'Support', 'Preview', and 'Run'. The left sidebar shows the file explorer with 'Cloud9 Instance -', 'python_3', 'resources', 'code.zip', and 'README.md'. The terminal output shows the following commands and results:

```

voclabs:~/environment $ wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/05-lab-lambda/code.zip -P /home/ec2-user/environment
--2025-01-21 15:37:19-- https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/05-lab-lambda/code.zip
Resolving aws-tc-largeobjects.s3.us-west-2.amazonaws.com (aws-tc-largeobjects.s3.us-west-2.amazonaws.com)... 52.92.204.10, 52.92.130.26, 52.92.240.74, ...
Connecting to aws-tc-largeobjects.s3.us-west-2.amazonaws.com (aws-tc-largeobjects.s3.us-west-2.amazonaws.com)|52.92.204.10|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5827765 (5.6M) [application/zip]
Saving to: '/home/ec2-user/environment/code.zip'

100%[=====>] 5,827,765  5.78MB/s  in 1.0s

2025-01-21 15:37:20 (5.78 MB/s) - '/home/ec2-user/environment/code.zip' saved [5827765/5827765]

voclabs:~/environment $ unzip code.zip
Archive: code.zip
  extracting: python_3/create_report_code.py
  extracting: python_3/get_all_products_code.py
  extracting: python_3/get_all_products_wrapper.py
  extracting: python_3/create_report_wrapper.py
  extracting: python_3/update_config.py
  extracting: resources/seed.py
  extracting: resources/setup.sh
  extracting: resources/public_policy.json
  extracting: resources/permissions.py
  extracting: resources/website/callback.html
  extracting: resources/website/all_products.json
  extracting: resources/website/all_products_on_offer.json
  extracting: resources/website/config.js
  extracting: resources/website/beans.json
  extracting: resources/website/spint.md
  extracting: resources/website/index.html
  extracting: resources/website/favicon.ico
  extracting: resources/website/scripts/main.js
  extracting: resources/website/scripts/pastries.js
  
```

The terminal window also shows the status bar at the bottom with 'CodeWhisperer', 'AWS: profile: default', and system information like '24°C Smoke', 'Search', and the time '21:07:47 21-01-2025'.

Figure 1: Download a zip file and extract it

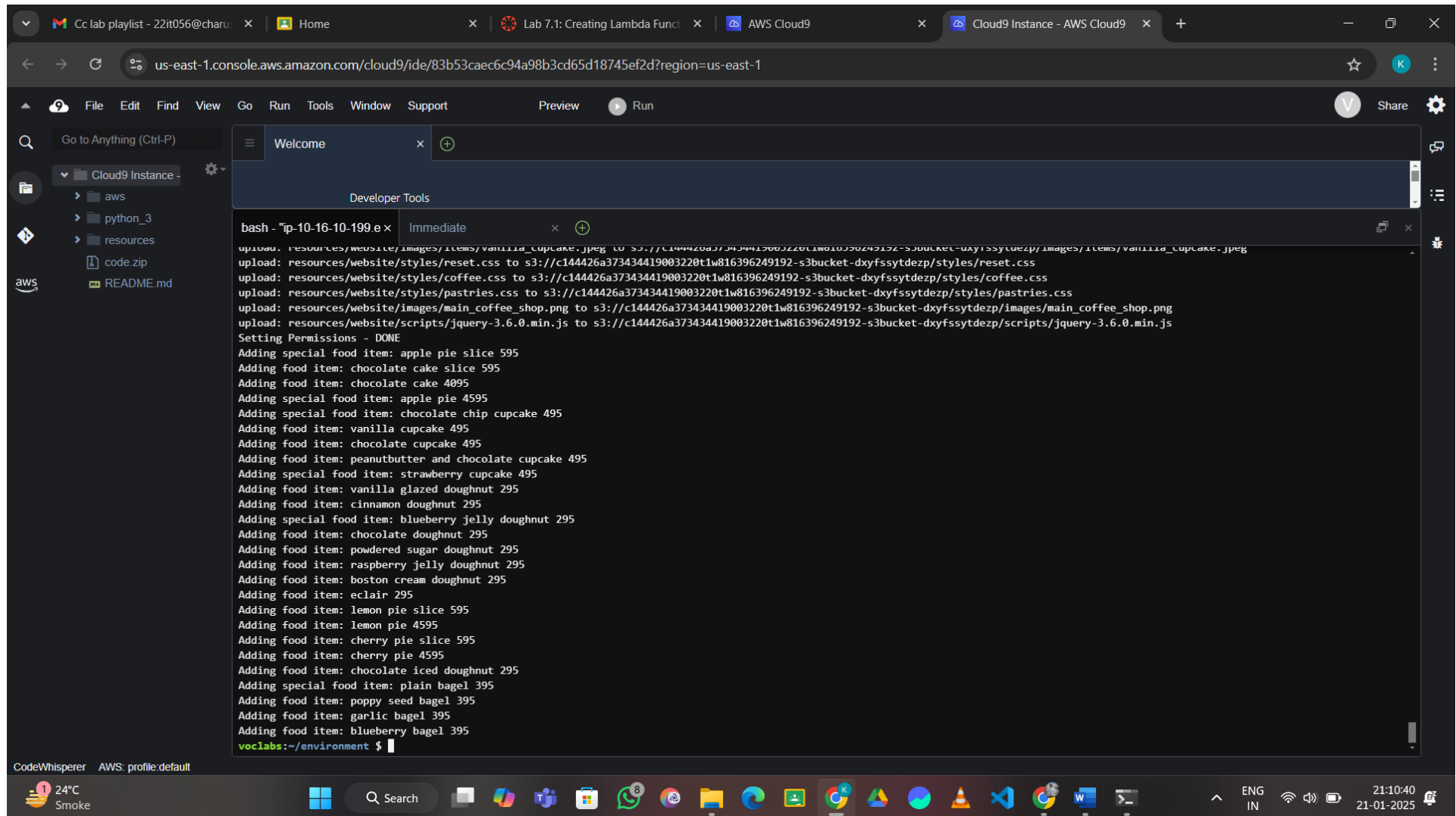


Figure 2: Set up resources with our ip address

The screenshot shows the AWS Management Console for the 'FoodProducts' table in the 'us-east-1' region. The 'Indexes' tab is active, showing a table with one global secondary index. The table is named 'FoodProducts' and has a partition key of 'special (Number)'. The index is named 'special_GSI' and is currently 'Active'. The read and write capacities are both set to 1, and auto scaling is turned off. The interface includes a left-hand navigation menu with options like 'Dashboard', 'Tables', 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', 'Integrations', 'Reserved capacity', and 'Settings'. The top navigation bar shows the AWS logo, a search bar, and the user's profile. The bottom taskbar displays various application icons and the system clock.

Global secondary indexes (1)

Name	Status	Partition key	Sort key	Read capacity	Write capacity	Proje
special_GSI	Active	special (Number)	-	1 Auto scaling is off	1 Auto scaling is off	All

Figure 3: Check table in DynamoDB

The screenshot displays the AWS Cloud9 IDE interface. The top navigation bar shows the current session is in the 'us-east-1' region. The left sidebar contains a file explorer with a tree view of the project structure, including folders like 'aws', 'python_3', and 'resources'. The main editor area shows the 'update_config.py' file with the following Python code:

```

1  ...
2  You must replace <FMI_1> with your bucket name
3  ...
4  import boto3
5  S3API = boto3.client("s3", region_name="us-east-1")
6  bucket_name = "c144426a373434419003220t1w816396249192-s3bucket-dxyfssytdezp"
7
8  filename = "/home/ec2-user/environment/resources/website/config.js"
9  S3API.upload_file(filename, bucket_name, "config.js", ExtraArgs={'ContentType': "application/js", "CacheControl": "max-age=0"})
10
11
12 print("DONE")

```

Below the editor, a terminal window is open, showing the output of the script execution:

```

bash - "ip-10-16-10-199.e x Immediate
Adding food item: lemon pie 4595
Adding food item: cherry pie slice 595
Adding food item: cherry pie 4595
Adding food item: chocolate iced doughnut 295
Adding special food item: plain bagel 395
Adding food item: poppy seed bagel 395
Adding food item: garlic bagel 395
Adding food item: blueberry bagel 395
voclabs:~/environment $ pip show boto3
Name: boto3
Version: 1.36.2
Summary: The AWS SDK for Python
Home-page: https://github.com/boto/boto3
Author: Amazon Web Services
Author-email: None
License: Apache License 2.0
Location: /home/ec2-user/.local/lib/python3.8/site-packages
Requires: jmespath, s3transfer, botocore
Required-by:
voclabs:~/environment $ aws s3 ls
2025-01-21 15:33:59 c144426a373434419003220t1w816396249192-s3bucket-dxyfssytdezp
2024-10-22 18:10:37 destinationbucket310803
voclabs:~/environment $ cd ~/environment/python_3
voclabs:~/environment/python_3 $ python update_config.py
DONE
voclabs:~/environment/python_3 $

```

The bottom status bar shows the temperature as 23°C and the time as 21:24:25 on 21-01-2025.

Figure 4: Update a confirmation in update_config.py file and run it

us-east-1.console.aws.amazon.com/cloud9/ide/83b53caec6c94a98b3cd65d18745ef2d?region=us-east-1

File Edit Find View Go Run Tools Window Support Preview Run

Go to Anything (Ctrl-P)

Cloud9 Instance -

aws

python_3

create_report_code.py

create_report_wrap.py

get_all_products_code.py

update_config.py

resources

website

images

scripts

styles

all_products.json

all_products_on_c

beans.json

callback.html

JS config.js

favicon.ico

index.html

spint.md

permissions.py

public_policy.json

seed.py

setup.sh

code.zip

README.md

CodeWhisperer AWS: profile:default

23°C Smoke

Search

21:28:47 21-01-2025

```

1 import boto3, json
2 from boto3.dynamodb.conditions import Key
3 from boto3.dynamodb.conditions import Key, Attr, Not
4
5 TABLE_NAME_STR = 'FoodProducts'

python3 - "ip-10-16-10-19" x Immediate x +
voclabs:~/environment/python_3 $ python get_all_products_code.py
running scan on table
{'product_item_arr': [{'price_in_cents_int': 295, 'special_int': 1, 'tag_str_arr': ['doughnut', 'on offer'], 'description_str': 'A doughnut with blueberry jelly filling.', 'product_name_s
tr': 'blueberry jelly doughnut', 'product_id_str': 'a455'}, {'price_in_cents_int': 295, 'tag_str_arr': ['doughnut', 'on offer'], 'description_str': 'so good!', 'product_name_str': 'vanill
a glazed doughnut', 'product_id_str': 'a453'}, {'price_in_cents_int': 295, 'tag_str_arr': ['doughnut', 'on offer'], 'description_str': 'Boston's favorite doughnut, done right.', 'product_
name_str': 'boston cream doughnut', 'product_id_str': 'a458'}, {'price_in_cents_int': 495, 'tag_str_arr': ['cupcakes', 'on offer'], 'description_str': 'Chocolate and peanut butter togethe
r.', 'product_name_str': 'peanutbutter and chocolate cupcake', 'product_id_str': 'a451'}, {'price_in_cents_int': 595, 'special_int': 1, 'tag_str_arr': ['pie slice', 'on offer'], 'descript
ion_str': 'A delicious slice of Frank's homemade pie.', 'product_name_str': 'apple pie slice', 'product_id_str': 'a444'}, {'price_in_cents_int': 395, 'tag_str_arr': ['bagel', 'on offer'],
'description_str': 'A fresh homemade bagel, with poppy seeds.', 'product_name_str': 'poppy seed bagel', 'product_id_str': 'a466'}, {'price_in_cents_int': 395, 'special_int': 1, 'tag_str_
arr': ['bagel', 'on offer'], 'description_str': 'Boiled in salt water, then baked. As it should be.', 'product_name_str': 'plain bagel', 'product_id_str': 'a465'}, {'price_in_cents_int':
4595, 'tag_str_arr': ['whole pie', 'on offer'], 'description_str': 'A generously sized homemade cherry pie.', 'product_name_str': 'cherry pie', 'product_id_str': 'a463'}, {'price_in_cents_
int': 495, 'special_int': 1, 'tag_str_arr': ['cupcakes', 'on offer'], 'description_str': 'A fresh strawberry on top!', 'product_name_str': 'strawberry cupcake', 'product_id_str': 'a452'},
{'price_in_cents_int': 295, 'tag_str_arr': ['doughnut', 'on offer'], 'description_str': 'A fluffy doughnut in a cinnamon sugar mix.', 'product_name_str': 'cinnamon doughnut', 'product_id_
str': 'a454'}, {'price_in_cents_int': 495, 'tag_str_arr': ['cupcakes', 'on offer'], 'description_str': 'A chocolate cupcake with chocolate frosting.', 'product_name_str': 'chocolate cup
cake', 'product_id_str': 'a450'}, {'price_in_cents_int': 495, 'special_int': 1, 'tag_str_arr': ['cupcakes', 'on offer'], 'description_str': 'A vanilla cupcake with chocolate chips.', 'pro
duct_name_str': 'chocolate chip cupcake', 'product_id_str': 'a448'}, {'price_in_cents_int': 4595, 'special_int': 1, 'tag_str_arr': ['whole pie', 'on offer'], 'description_str': 'Frank's h
omemade pie with flakey crust - yum!', 'product_name_str': 'apple pie', 'product_id_str': 'a447'}, {'price_in_cents_int': 4595, 'tag_str_arr': ['whole pie', 'on offer'], 'description_str'
: 'The whole lemon pie!', 'product_name_str': 'lemon pie', 'product_id_str': 'a461'}, {'price_in_cents_int': 295, 'tag_str_arr': ['doughnut', 'on offer'], 'description_str': 'The chocola
te makes it so good!', 'product_name_str': 'chocolate iced doughnut', 'product_id_str': 'a464'}, {'price_in_cents_int': 395, 'tag_str_arr': ['bagel', 'on offer'], 'description_str': 'A fre
sh homemade bagel, with toasted garlic.', 'product_name_str': 'garlic bagel', 'product_id_str': 'a467'}, {'price_in_cents_int': 595, 'tag_str_arr': ['cake slice', 'on offer'], 'descriptio
n_str': 'Chocolate heaven. What's not to like?', 'product_name_str': 'chocolate cake slice', 'product_id_str': 'a445'}, {'price_in_cents_int': 395, 'tag_str_arr': ['bagel', 'out of stock'
], 'description_str': 'A fresh homemade bagel, with blueberries.', 'product_name_str': 'blueberry bagel', 'product_id_str': 'a467'}, {'price_in_cents_int': 4095, 'tag_str_arr': ['whole ca
ke', 'on offer'], 'description_str': 'Chocolate cake with chocolate icing. A classic.', 'product_name_str': 'chocolate cake', 'product_id_str': 'a446'}, {'price_in_cents_int': 295, 'tag_s
tr_arr': ['doughnut', 'on offer'], 'description_str': 'A delicious doughnut coated in powdered sugar.', 'product_name_str': 'powdered sugar doughnut', 'product_id_str': 'a456'}, {'price_i
n_cents_int': 295, 'tag_str_arr': ['doughnut', 'on offer'], 'description_str': 'A doughnut make with rich milk chocolate.', 'product_name_str': 'chocolate doughnut', 'product_id_str': 'a4
55'}, {'price_in_cents_int': 595, 'tag_str_arr': ['pie slice', 'on offer'], 'description_str': 'Just the right amount of lemon. A cafe favorite.', 'product_name_str': 'lemon pie slice', '
product_id_str': 'a460'}, {'price_in_cents_int': 595, 'tag_str_arr': ['pie slice', 'on offer'], 'description_str': 'Deliciously tart bing cherries inside.', 'product_name_str': 'cherry pi
e slice', 'product_id_str': 'a462'}, {'price_in_cents_int': 295, 'tag_str_arr': ['doughnut', 'on offer'], 'description_str': 'A delicious french delicacy make with real cream.', 'product_
name_str': 'eclair', 'product_id_str': 'a459'}, {'price_in_cents_int': 295, 'tag_str_arr': ['doughnut', 'on offer'], 'description_str': 'A doughnut with raspberry jelly filling.', 'produc
t_name_str': 'raspberry jelly doughnut', 'product_id_str': 'a457'}, {'price_in_cents_int': 495, 'tag_str_arr': ['cupcakes', 'on offer'], 'description_str': 'Italian vanilla bean cupcake.'
, 'product_name_str': 'vanilla cupcake', 'product_id_str': 'a449'}}}]
voclabs:~/environment/python_3 $

```

Figure 5: Update get_all_products_code.py file and run it

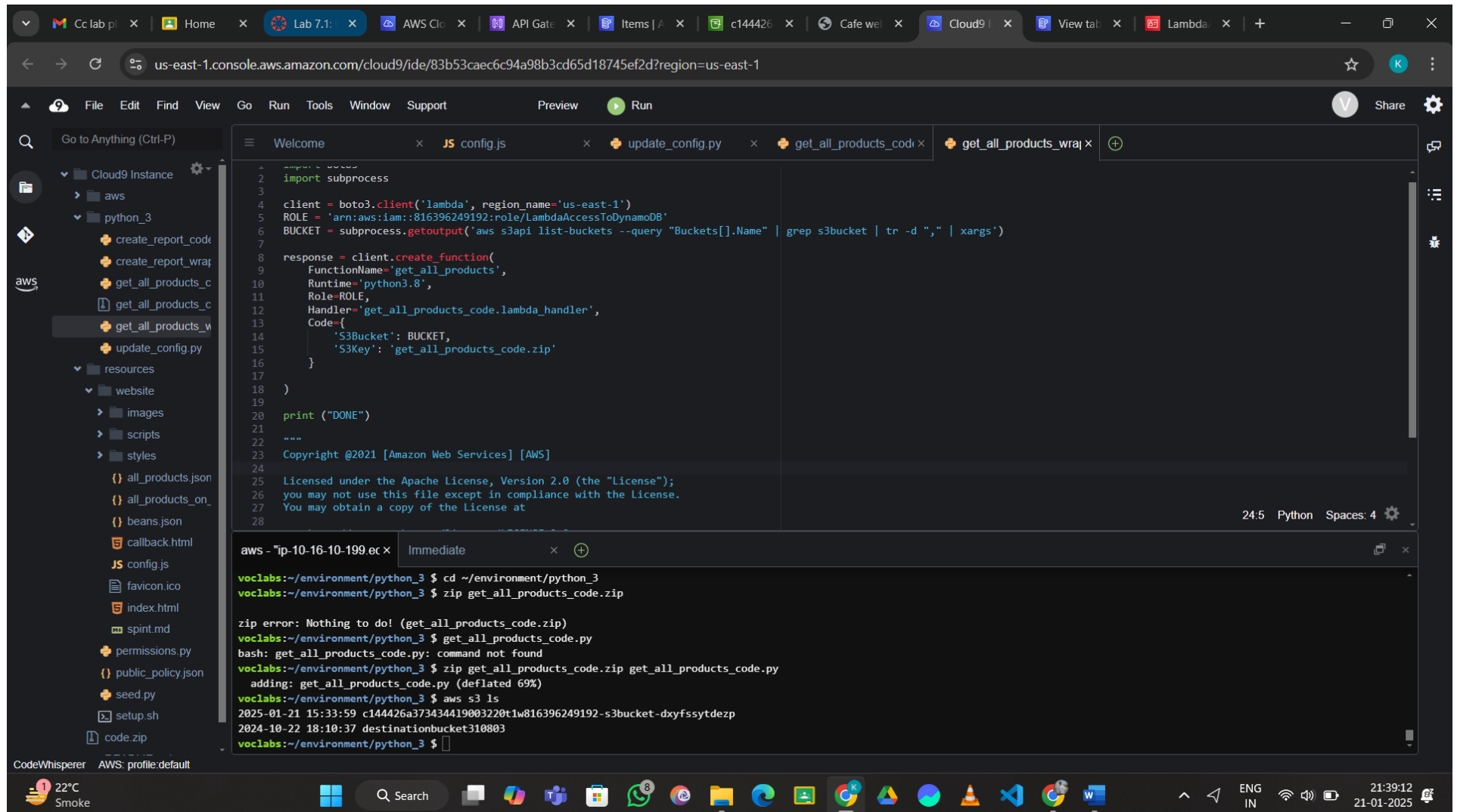


Figure 6: Download zip file in python3 directory and extract it

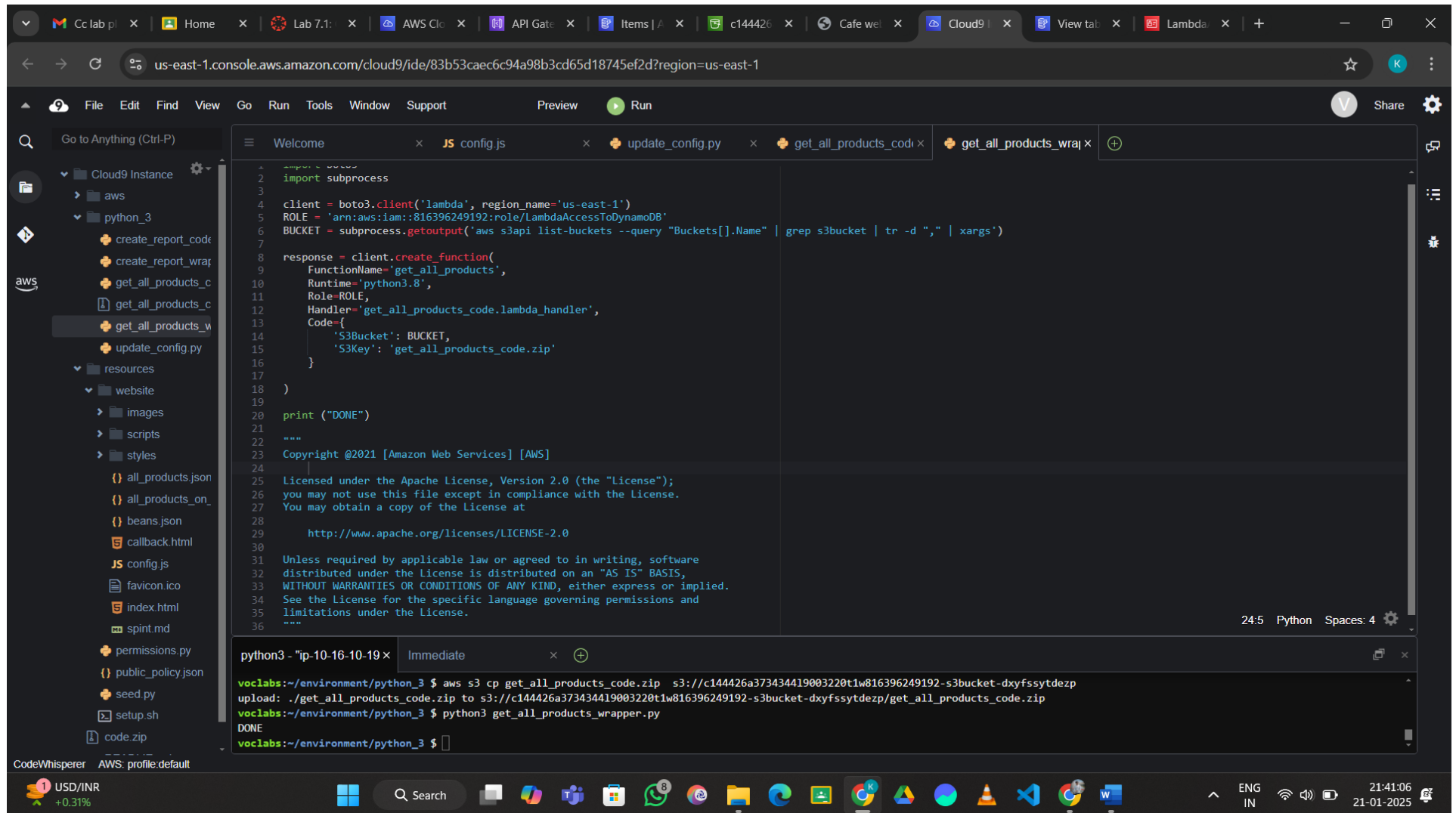


Figure 7: Update `get_all_products_wrape.py` file and run it

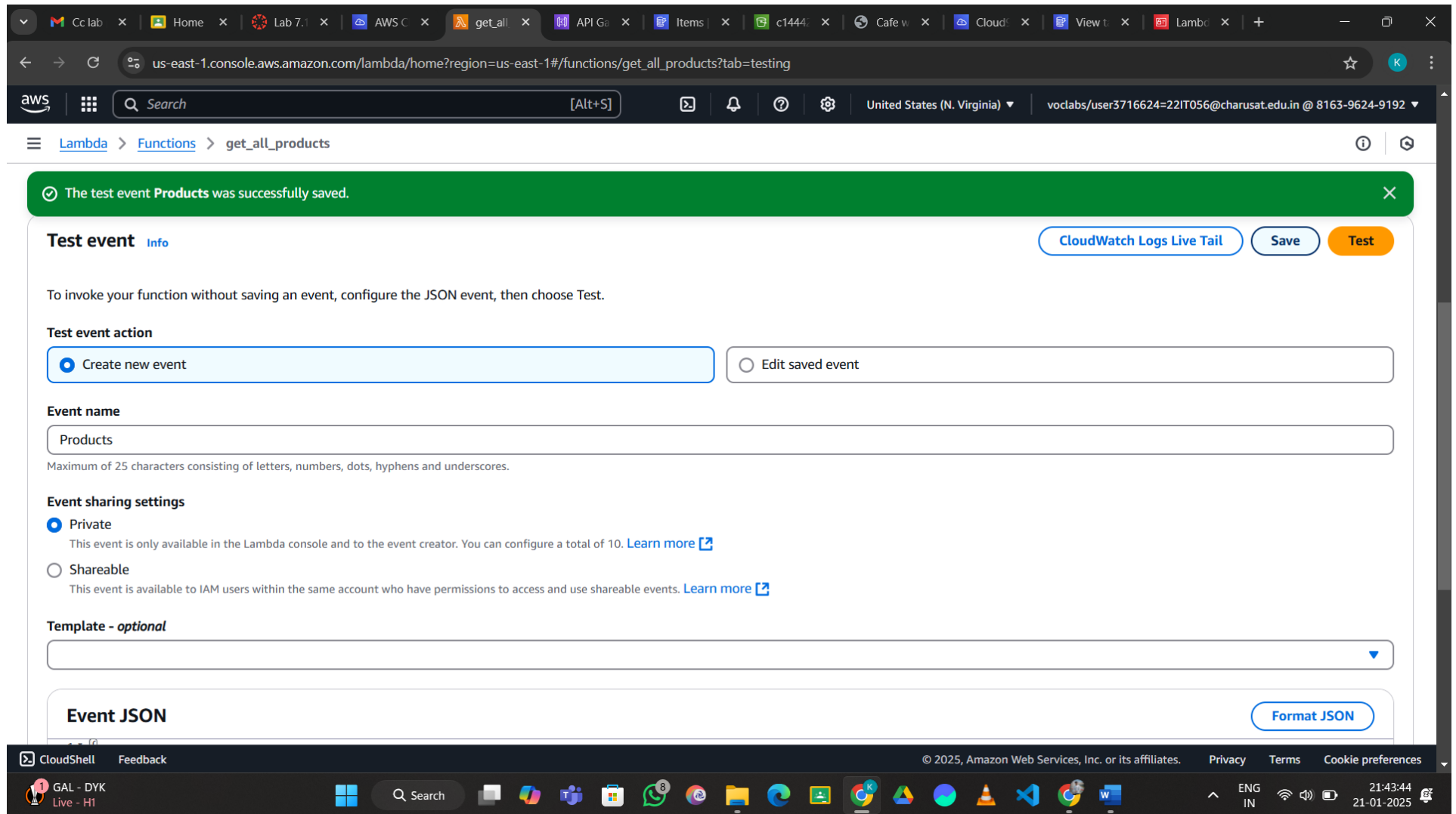


Figure 8: Test created product in lambda

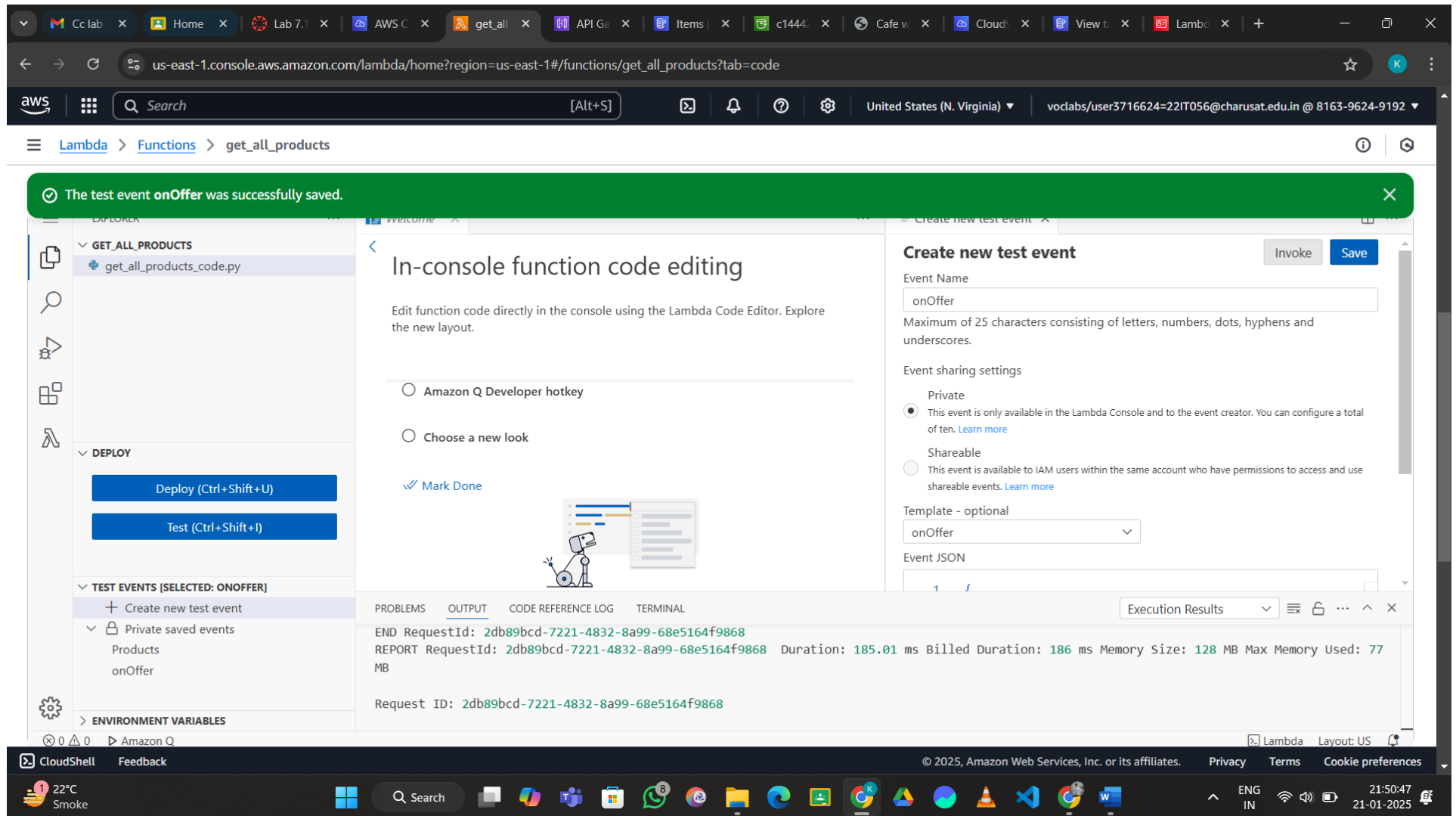


Figure 9: Create a new test onOffer for website

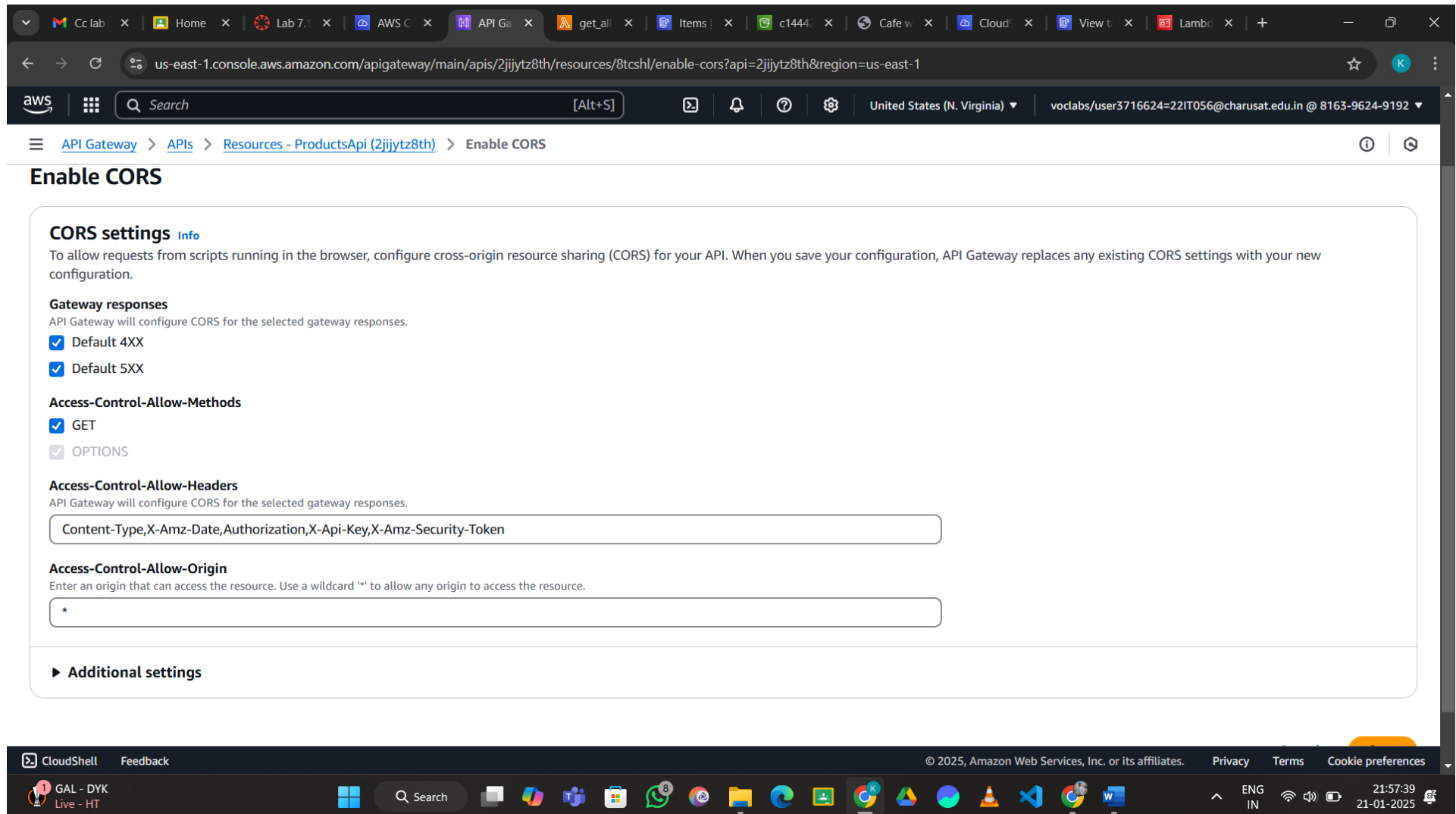


Figure 10: Enable CORS and update configuration

us-east-1.console.aws.amazon.com/apigateway/main/apis/2jijytz8th/resources/?api=2jijytz8th®ion=us-east-1

API Gateway > APIs > Resources - ProductsApi (2jijytz8th)

API Gateway

- APIs
- Custom domain names
- Domain name access associations
- VPC links

▼ **API: ProductsApi**

- Resources
- Stages
- Authorizers
- Gateway responses
- Models
- Resource policy
- Documentation
- Dashboard
- API settings
- Usage plans
- API keys

Successfully enabled CORS

▼ **Details**

- Created **OPTIONS** method.
- Added **200 Method Response** with **Empty Response Model** to **OPTIONS** method.
- Added **Mock Integration** to **OPTIONS** method.
- Added **200 Integration Response** to **OPTIONS** method.
- Added **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin Method Response Headers** to **OPTIONS** method.
- Added **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin Integration Response Header Mappings** to **OPTIONS** method.
- Added **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin Response Headers** to **Default 4XX Gateway response**.
- Added **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin Response Headers** to **Default 5XX Gateway response**.
- Added **Access-Control-Allow-Origin Method Response Header** to **GET** method.
- Added **Access-Control-Allow-Origin Integration Response Header Mapping** to **GET** method.

Resources

Create resource

/

- /create_report
- OPTIONS
- POST
- /products
- GET

Resource details

Path
/products/on_offer

Resource ID
i75gls

Delete Update documentation Enable CORS

Methods (2)

Delete Create method

Method type	Integration type	Authorization	API key
POST	Mock	None	
GET	Integration	None	

CloudShell Feedback

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

22:00:41 21-01-2025

Figure 11: Update a access method

The screenshot displays the AWS API Gateway console interface. At the top, a green banner states: "Successfully updated method request settings for 'GET' in 'on_off'. Redeploy your API for the update to take effect." The left sidebar shows the navigation menu for "API Gateway" and "APIs", with "Resources - ProductsApi (2jijytz8th)" selected. The main content area is titled "Resources" and shows a list of resources on the left, including "/on_off" with a "GET" method selected. The right pane displays the "Integration request settings" for the selected method, showing a diagram of the request flow: Client → Method request → Integration request → Lambda integration. The "Integration request settings" section is visible, showing the "Integration type" and "Region". The bottom of the screen shows a Windows taskbar with various application icons and a system tray with weather and time information.

Figure 12: Retest updated on_off request method

The screenshot displays the AWS API Gateway console interface. At the top, a green banner indicates a successful deployment for 'ProductsApi'. The left sidebar shows the navigation menu with 'API Gateway' selected. The main content area is titled 'Stages' and shows a list of stages with 'prod' selected. The 'Stage details' panel for 'prod' is expanded, showing the following information:

- Stage name:** prod
- Rate:** 10000
- Cache cluster:** Inactive
- Burst:** 5000
- Default method-level caching:** Inactive
- Web ACL:** -
- Client certificate:** -
- Invoke URL:** <https://2jijytz8th.execute-api.us-east-1.amazonaws.com/prod>
- Active deployment:** kcawby on January 21, 2025, 22:08 (UTC+05:30)

The bottom of the screen shows the Windows taskbar with various application icons and the system clock indicating 22:08:59 on 21-01-2025.

Figure 13: Deploy API

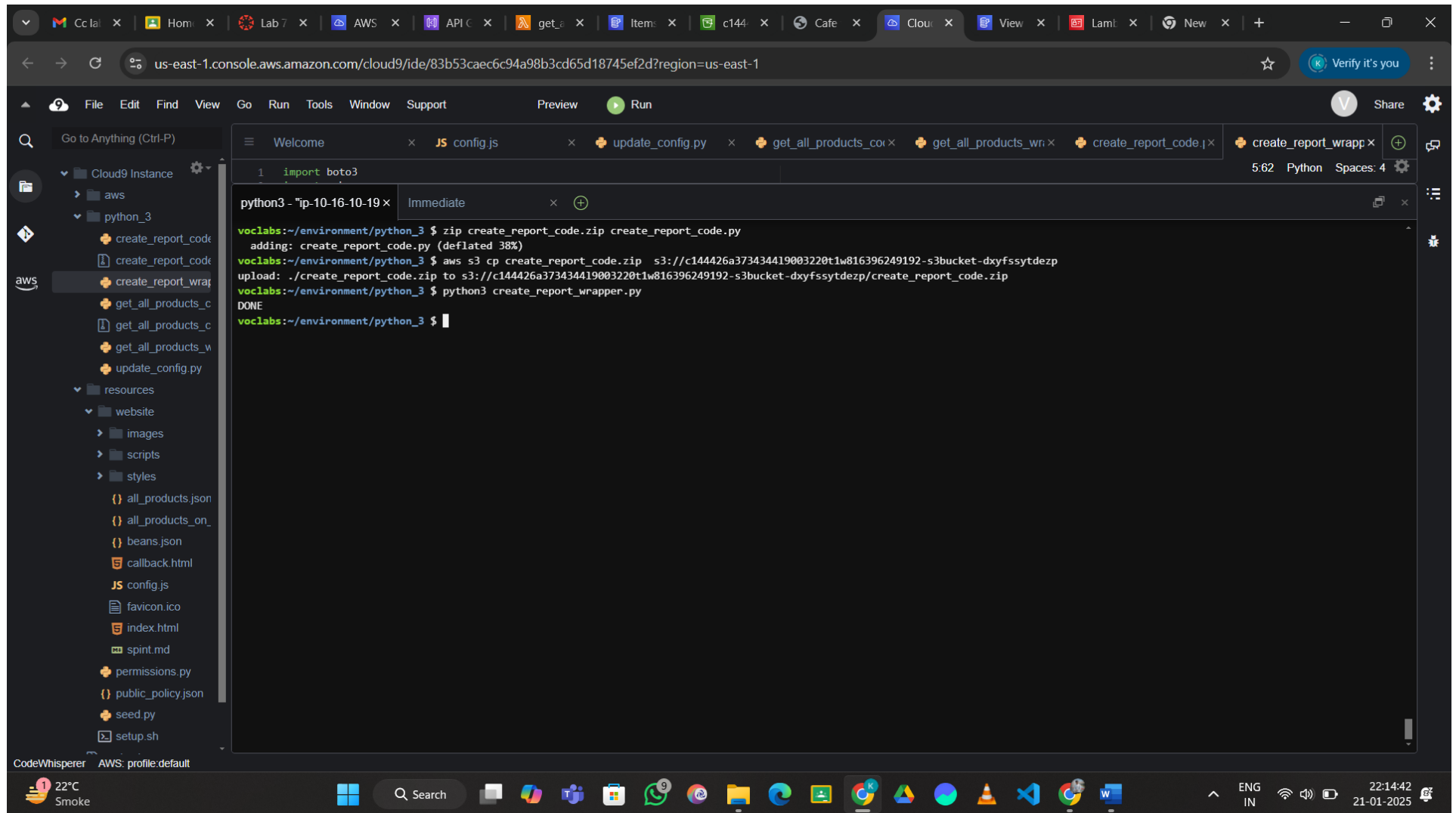


Figure 14: Update `create_report_wrapper.py` file and run it

The screenshot displays the AWS Lambda console for the 'create_report' function in the 'us-east-1' region. A green notification bar at the top states: 'The test event ReportTest was successfully saved.' The main interface is divided into several sections:

- Code source:** Shows the Python code for the lambda handler in 'create_report_code.py'. The code defines a 'lambda_handler' function that returns a dictionary with a 'msg_str' key containing the message 'Report processing, check your phone shortly'.
- Deploy:** Includes buttons for 'Deploy (Ctrl+Shift+U)' and 'Test (Ctrl+Shift+I)'.
- Test Events:** A section titled 'TEST EVENTS [SELECTED: REPORTTEST]' with options to 'Create new test event' and 'Private saved events'.
- Execution Results:** A panel on the right showing the 'Create new test event' form with 'ReportTest' as the event name. Below this, the 'Execution Results' are displayed, showing a 'Status: Succeeded' and a 'Response' object:

```
{  "msg_str": "Report processing, check your phone shortly"}
```
- Function Logs:** A section at the bottom showing the logs for the function execution, including the start and end request IDs, version (\$LATEST), and duration (2.54 ms).

The bottom of the screen shows a Windows taskbar with various application icons and a system tray displaying the temperature (22°C), time (22:16:58), and date (21-01-2025).

Figure 15: Test ReportTest

The screenshot displays the AWS API Gateway console interface. At the top, a navigation bar shows the breadcrumb path: **API Gateway** > **APIs** > **ProductsApi (2jijytz8th)** > **Stages**. The left-hand sidebar contains a menu with categories like **API Gateway**, **APIs**, **Custom domain names**, **Domain name access associations**, **VPC links**, and **API: ProductsApi** (which is expanded to show **Resources**, **Stages**, **Authorizers**, **Gateway responses**, **Models**, **Resource policy**, **Documentation**, **Dashboard**, **API settings**, **Usage plans**, and **API keys**). The main content area features three green notification banners at the top, each with a checkmark icon and a close button. Below these is a **Notifications** summary bar showing 0 errors, 0 warnings, 3 successes, 0 info, and 0 debug messages. The **Stages** section includes a list of stages with a **prod** stage selected. To the right of the stage list is a **Stage details** panel with an **Edit** button. This panel contains fields for **Stage name** (prod), **Rate** (10000), **Cache cluster** (Inactive), **Burst** (5000), **Default method-level caching** (Inactive), **Invoke URL** (https://2jijytz8th.execute-api.us-east-1.amazonaws.com/prod), and **Active deployment**. The bottom of the image shows a Windows taskbar with various application icons and a system tray displaying weather (22°C, Smoke) and time (22:19:57, 21-01-2025).

Figure 16: Update the request method and save it

LATEST APPLICATIONS:

1. AWS Lambda

- Dynamic content delivery for e-commerce, news, and media websites.
- Real-time data processing for IoT, sensor networks, and streaming applications.
- Event-driven microservices, reacting to changes in databases, file uploads, or queue messages.
- Automated workflows, such as image processing, file transformations, or notifications.
- Backend logic for chatbot and voice assistant integrations (e.g., Alexa skills).

2. Amazon DynamoDB

- High-performance NoSQL storage for gaming leaderboards, user sessions, or inventory management.
- Real-time analytics for streaming data and event tracking.
- Data storage for serverless applications with high read/write throughput requirements.
- Managing metadata and logs in media processing workflows.
- Supporting large-scale applications like ticketing systems and ride-sharing platforms.

3. Amazon API Gateway

- Backend for mobile and web applications, enabling secure REST/HTTP APIs.
- Real-time APIs for multiplayer gaming and collaborative tools.
- Integration with WebSocket APIs for chat apps and live data feeds.
- Exposing microservices for enterprise-scale distributed systems.
- Connecting serverless applications with external systems for hybrid architectures.

LEARNING OUTCOME:

By completing this lab, I will gain a comprehensive understanding of integrating AWS services to build a dynamic, serverless application. I will learn to create and deploy AWS Lambda functions to handle API requests and connect with Amazon DynamoDB for real-time data retrieval while designing and querying NoSQL databases and optimizing data retrieval using indexes. Additionally, I will develop skills in configuring and managing REST APIs with Amazon API Gateway to enable secure and scalable communication between the frontend and back end. This includes connecting API methods to Lambda functions, managing API security, and configuring AWS IAM roles and policies for controlled resource

access. Overall, this lab will equip me with practical knowledge to build event-driven architectures and implement real-world applications, such as dynamically updating a café menu on a website.

REFERENCE:

1. <https://awsacademy.instructure.com/courses/104050>