StepAhead
Way for excellence

Mini Project Documentation

Title: Serverless Feedback Form with AWS Lambda

Team Members: Naveen Kumar Varma & Shrihari Shinde

Tech Stack: Python, AWS Lambda, API Gateway, DynamoDB, IAM, Cloud Watch, GitHub Actions, CI/CD Pipelines

SDLC Model: Agile-Inspired Iterative Development

1. Requirement Analysis

1.1 Problem Statement

In today's digital-first era, user feedback is essential for improving products, enhancing user experience, and guiding future updates. To streamline the feedback collection process, we propose a **serverless feedback system** built using **AWS services**, offering a scalable, cost-effective, and maintenance-free solution. This system allows users to submit feedback through a **REST API**, eliminating the need for traditional server management. It leverages **AWS API Gateway** to expose the API endpoints, **AWS Lambda** for handling backend logic, and **DynamoDB** for secure and low-latency data storage. This architecture ensures seamless scalability and high availability, making it ideal for applications expecting dynamic user traffic. By adopting a serverless model, the system automatically scales based on demand, reduces operational overhead, and provides a pay-as-you-go pricing model. **IAM roles and policies** are used to enforce security, while **GitHub Actions** enables automated deployment and continuous integration, ensuring rapid delivery and updates.

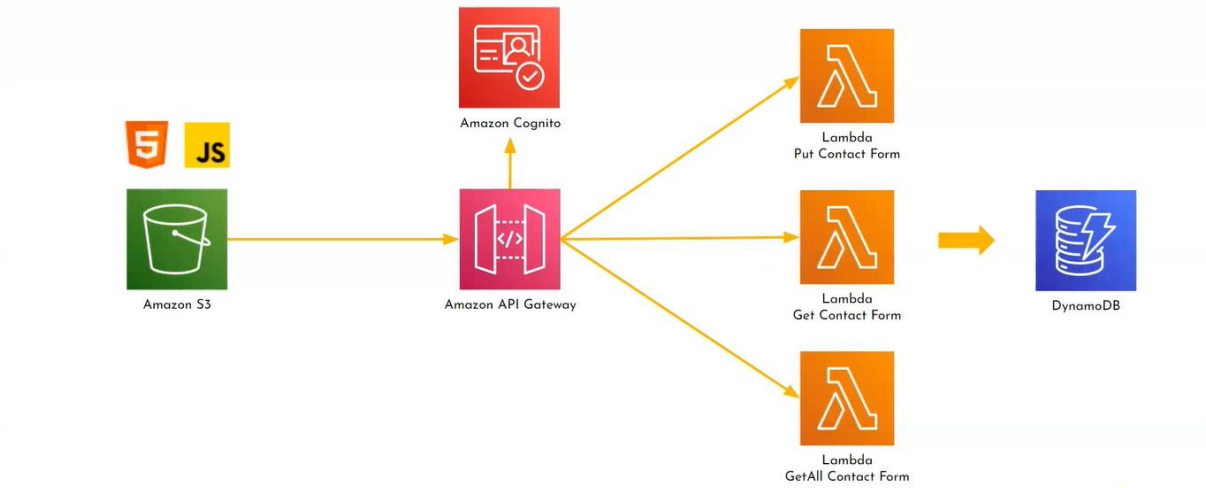
The feedback data collected can be used to identify user pain points, track feature requests, and support product enhancements. This system empowers organizations to make informed, data-driven decisions without investing in complex infrastructure.

1.2 Functional Requirements

- Accept user feedback via a **POST request** on a REST API
- Store feedback in **AWS DynamoDB** with attributes:
  - name, email, message, and auto-generated timestamp
- Return a proper **JSON response** indicating success or failure
- **Secure the system** with AWS IAM policies and roles
- Implement detailed **logging** using AWS Cloud Watch
- **Automate deployment** using GitHub Actions for CI/CD

2. System Design

2.1 Architecture Diagram



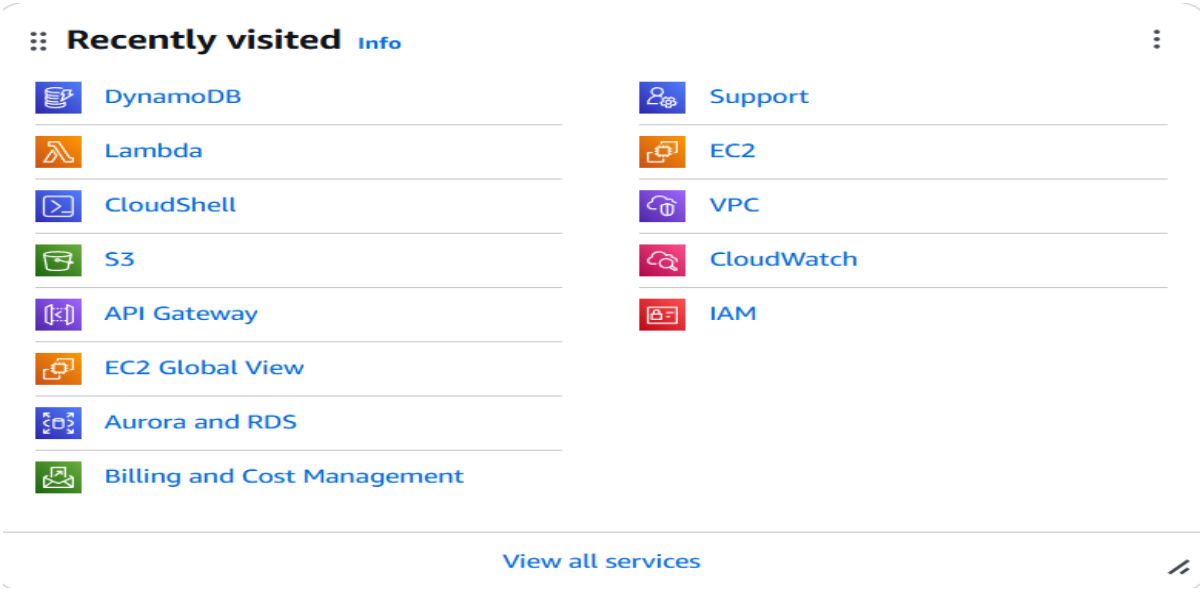
2.2 Component Summary

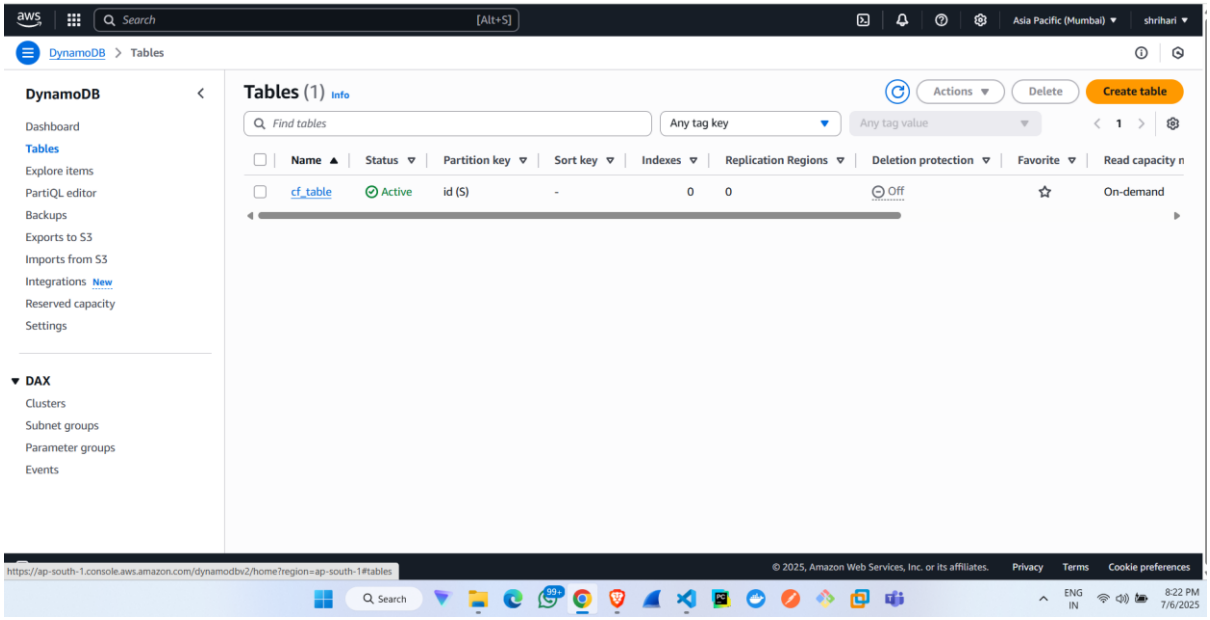
Component	Description
API Gateway	Exposes POST /feedback endpoint; handles request schema validation
AWS Lambda	Parses request, validates fields, adds feedback_id and timestamp, writes to DynamoDB
DynamoDB	Table storing feedback entries; schema supports feedback_id, timestamp, TTL
IAM	Role with permissions to put items into DynamoDB and write logs
Cognito & Cloud Watch	Captures Lambda logs, errors, performance metrics
GitHub Actions	CI/CD pipeline automates unit testing, packaging, and deployment to AWS

2.3 DynamoDB Table Schema

Attribute	Data Type	Description
feedback_id	String (UUID)	Unique identifier for each entry
name	String	Name of the feedback provider
email	String	Email address of the user
message	String	User's feedback or comment

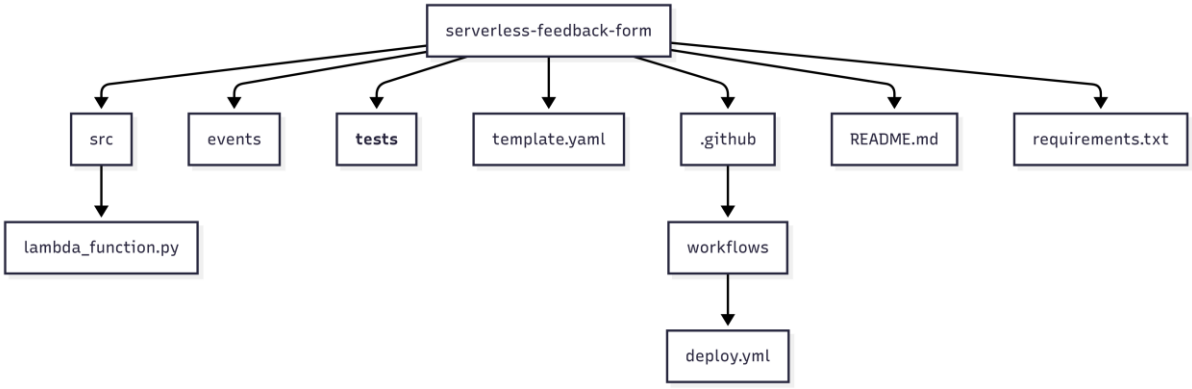
2.4 Snapshots





3. Implementation

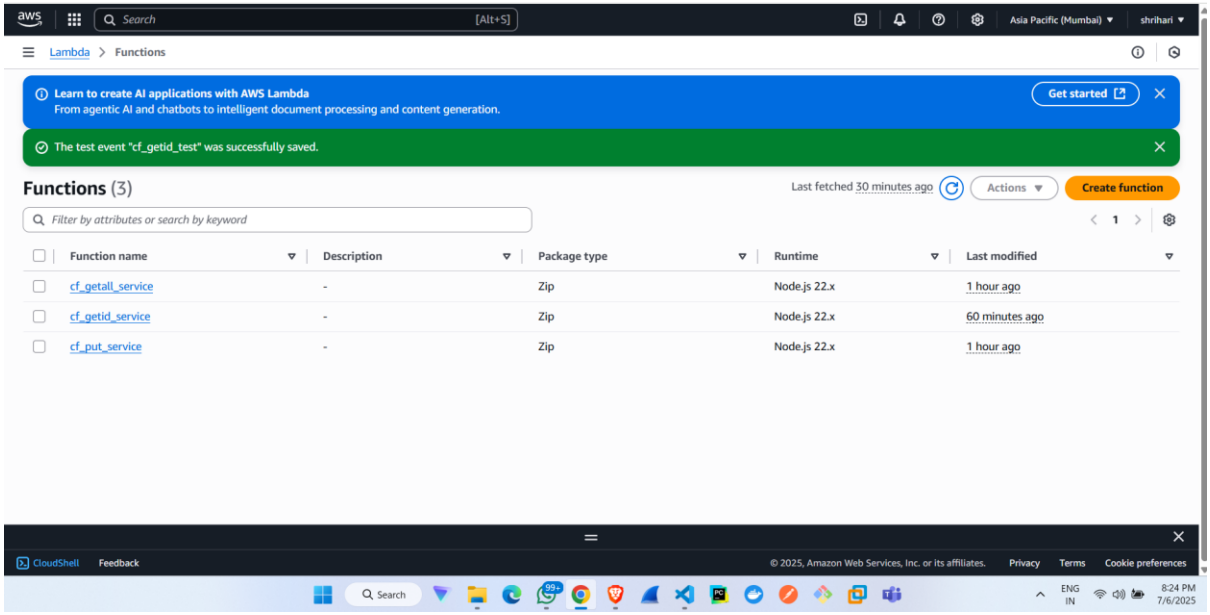
3.1 Project Folder Structure

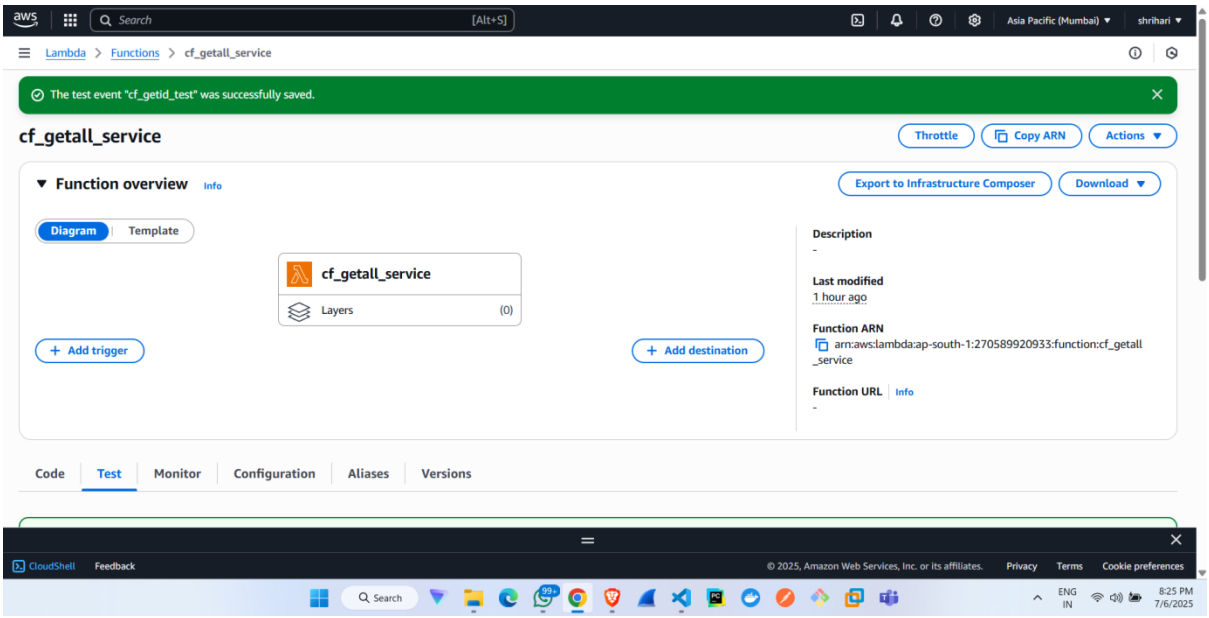


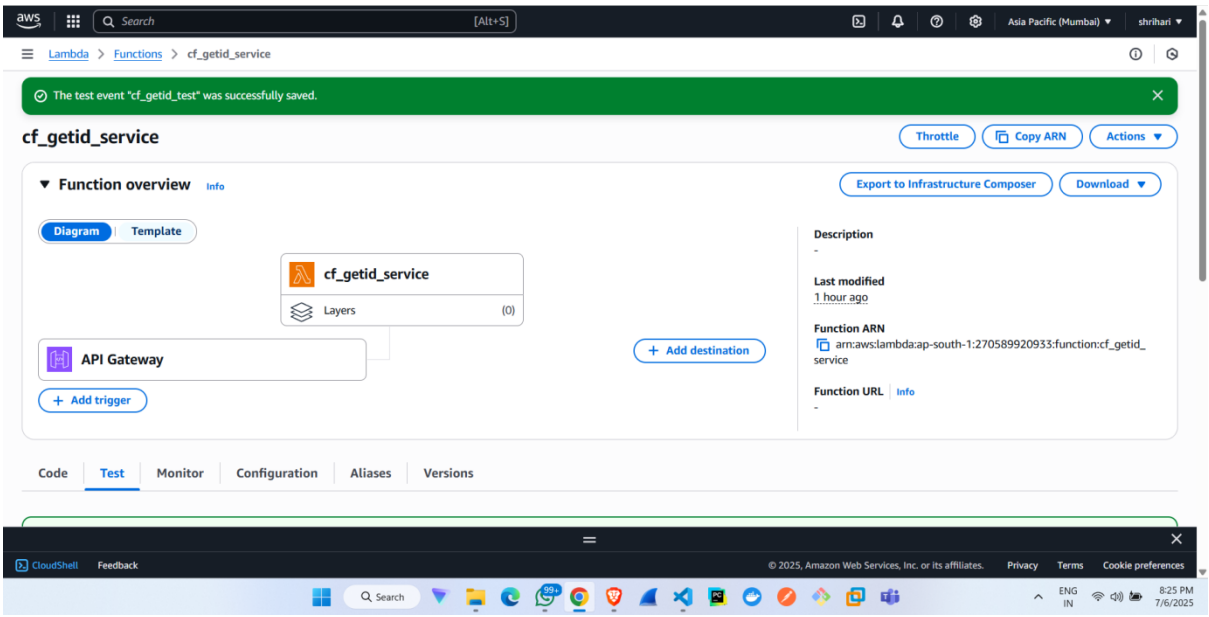
3.2 Lambda Function Logic

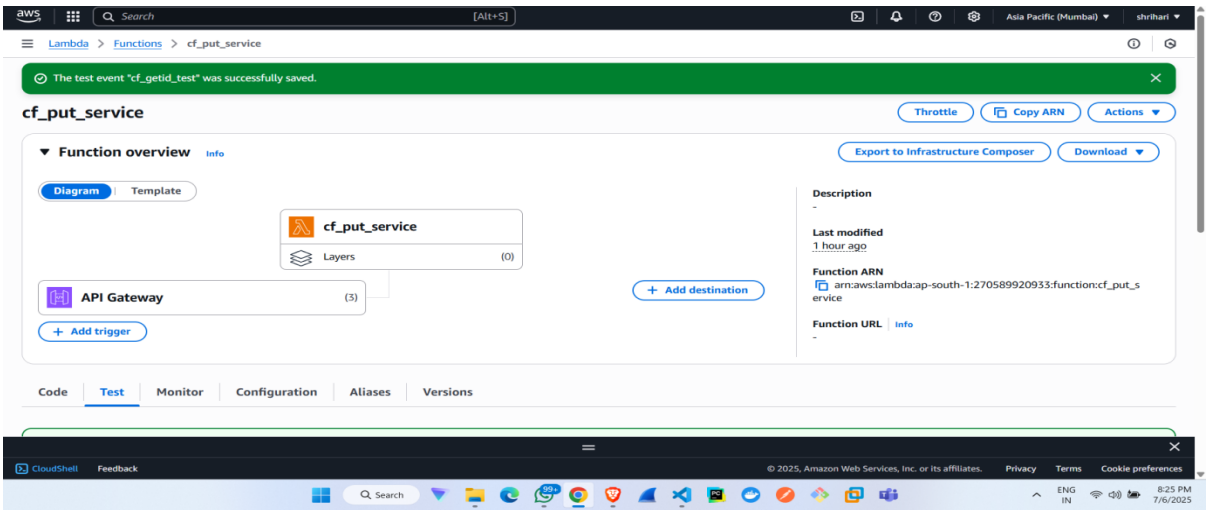
- Load JSON from event['body']
- Check presence and type of name, email, message
- Generate feedback_id using uuid4
- Write item to DynamoDB with put_item()
- Return 200 OK with JSON { "status": "success", "feedback_id": "…" }
- On error, return 400/500 with meaningful JSON error message

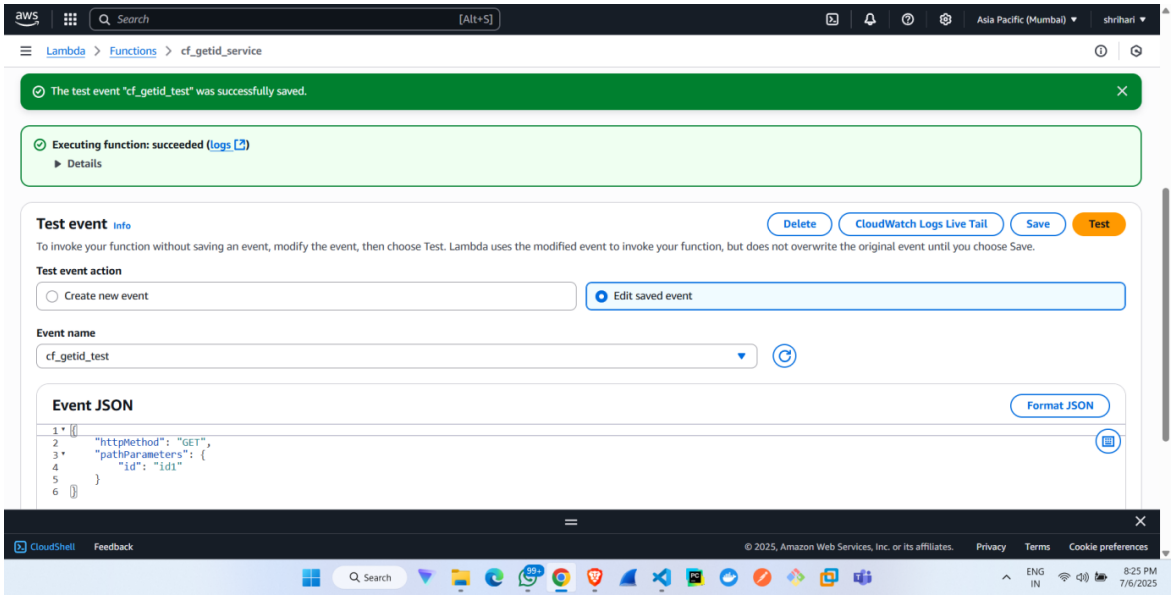
3.3 Snapshots




Lambda > Functions > cf_getall_service
The test event "cf_getid_test" was successfully saved.
cf_getall_service
Throttle
Copy ARN
Actions
Function overview
Export to Infrastructure Composer
Download
Diagram
Template
cf_getall_service
Layers (0)
Add trigger
Add destination
Description
-
Last modified
1 hour ago
Function ARN
arn:aws:lambda:ap-south-1:270589920933:function:cf_getall_service
Function URL
Code
Test
Monitor
Configuration
Aliases
Versions


Lambda > Functions > cf_getid_service
The test event "cf_getid_test" was successfully saved.
cf_getid_service
Throttle
Copy ARN
Actions
Function overview
Export to Infrastructure Composer
Download
Diagram
Template
cf_getid_service
Layers (0)
API Gateway
Add trigger
Add destination
Description
-
Last modified
1 hour ago
Function ARN
arn:aws:lambda:ap-south-1:270589920933:function:cf_getid_service
Function URL
Code
Test
Monitor
Configuration
Aliases
Versions


Lambda > Functions > cf_put_service
The test event "cf_getid_test" was successfully saved.
cf_put_service
Throttle
Copy ARN
Actions
Function overview
Export to Infrastructure Composer
Download
Diagram
Template
cf_put_service
Layers (0)
API Gateway (3)
Add trigger
Add destination
Description
-
Last modified
1 hour ago
Function ARN
arn:aws:lambda:ap-south-1:270589920933:function:cf_put_service
Function URL
Code
Test
Monitor
Configuration
Aliases
Versions



3.4 Sample Request/Response

Request (Postman example):

```
http
POST /feedback
Content-Type: application/json
{
  "name": "Naveen",
  "email": "nawin@example.com",
  "message": "Great app!"
}
```

Response:

```
json
{
  "status": "success",
  "feedback_id": "123e4567-e89b-12d3-a456-426614174000" }
```

4. Testing

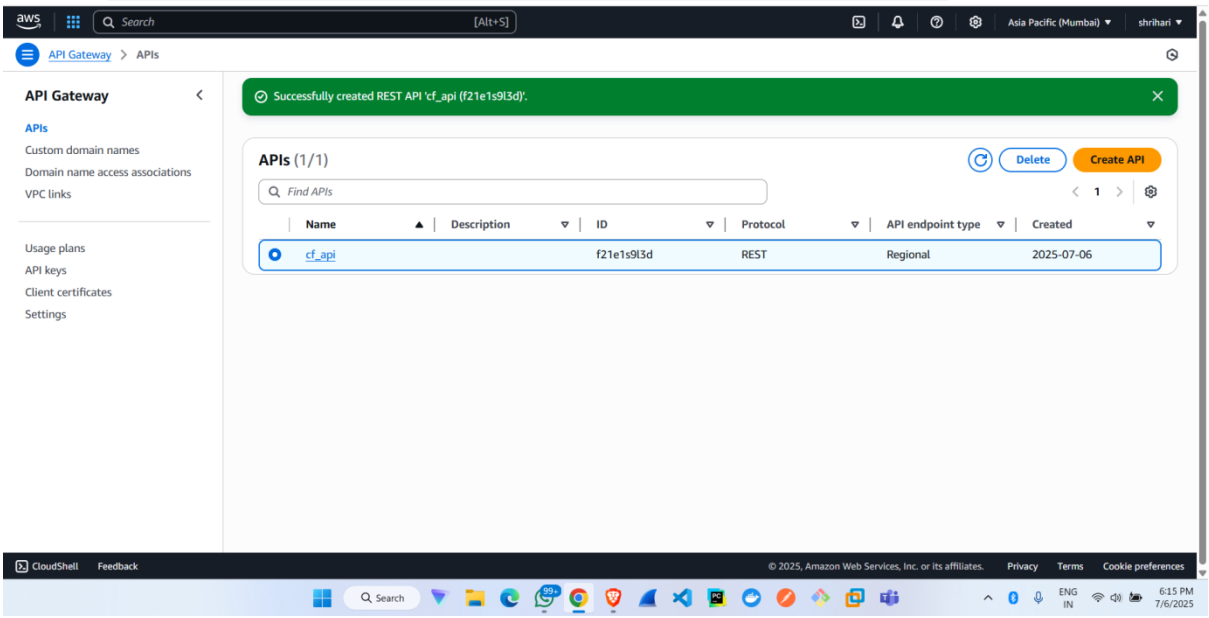
4.1 Test Scenarios

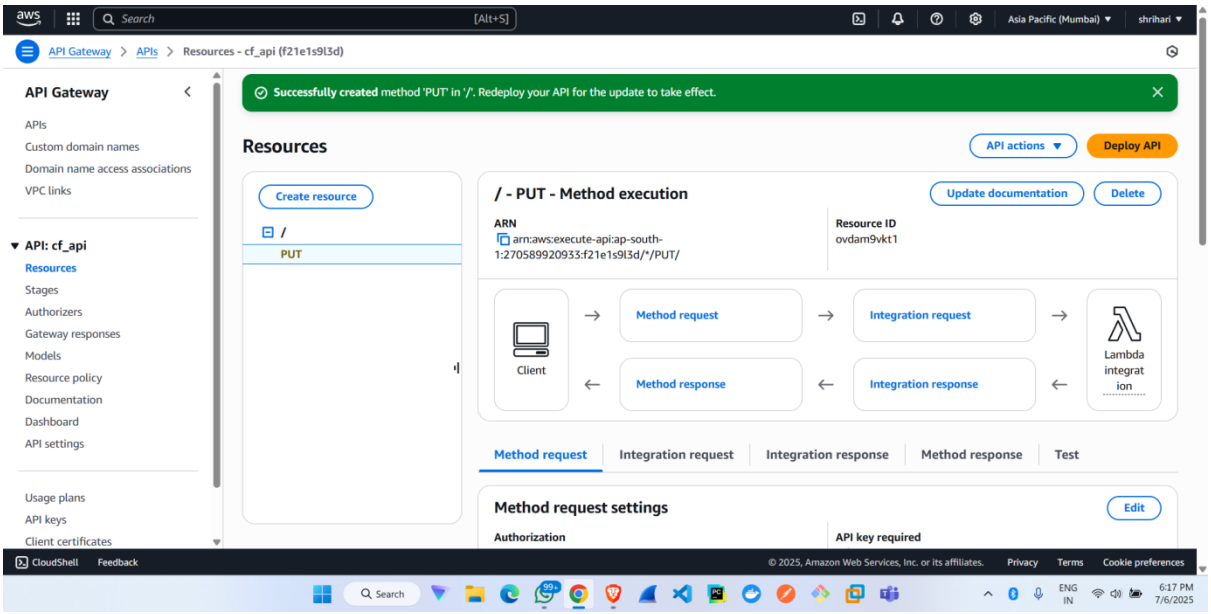
Test Case	Input	Expected Outcome
Valid data	proper JSON with name/email/msg	200 OK, record in DynamoDB
Missing required field	omit email	400 Bad Request, descriptive error
Malformed JSON	invalid JSON syntax	400 Bad Request, "Invalid JSON" response
Very large message	10 KB input	200 OK, no Lambda timeout
Invalid email format	not-an-email	400 Bad Request, "Invalid email"
DynamoDB failure simulation	simulate put_item error	500 Internal Error and appropriate log entry

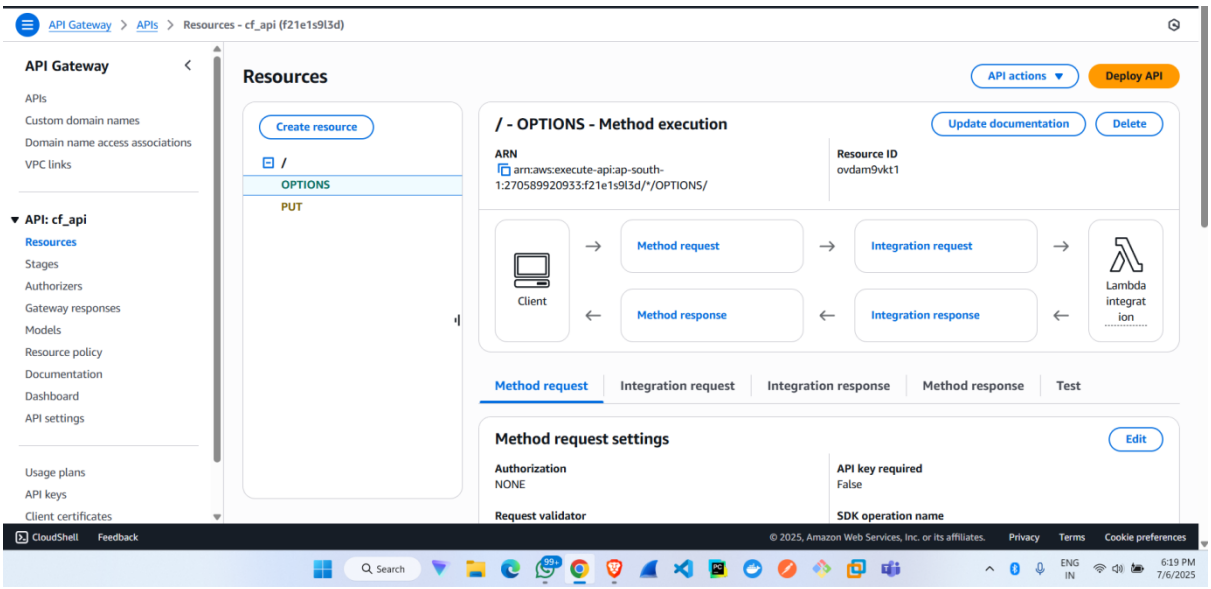
4.2 Tools Used

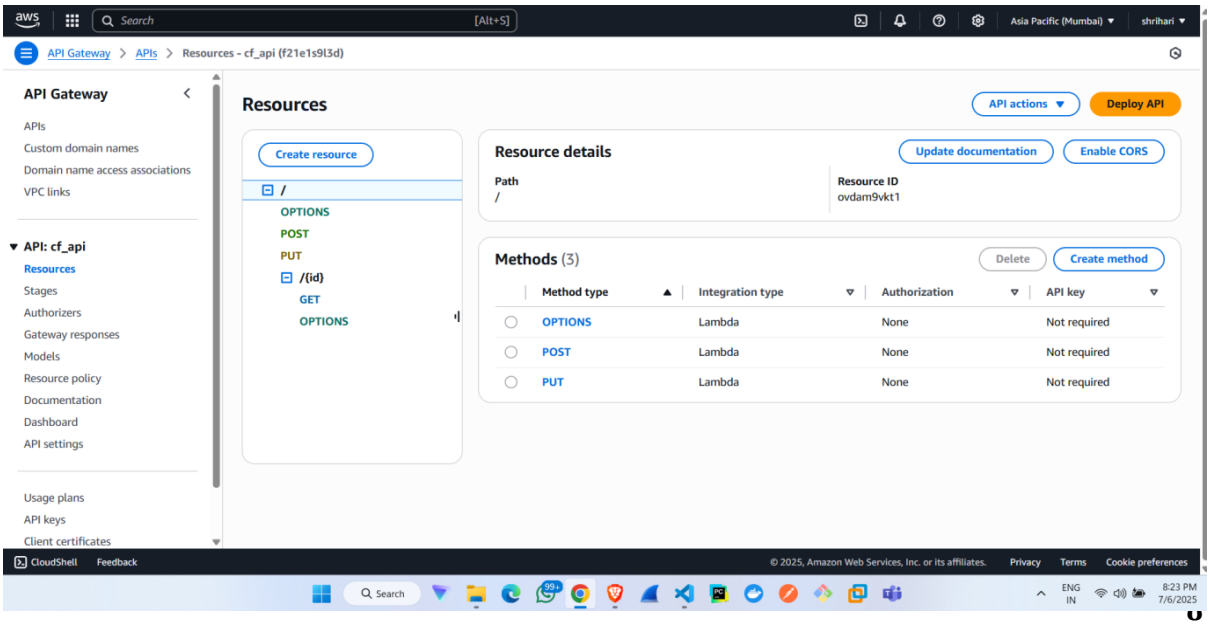
- **AWS SAM CLI** (optional local invocation)
- **Cloud Watch Logs**: to verify correct logging and troubleshoot issues

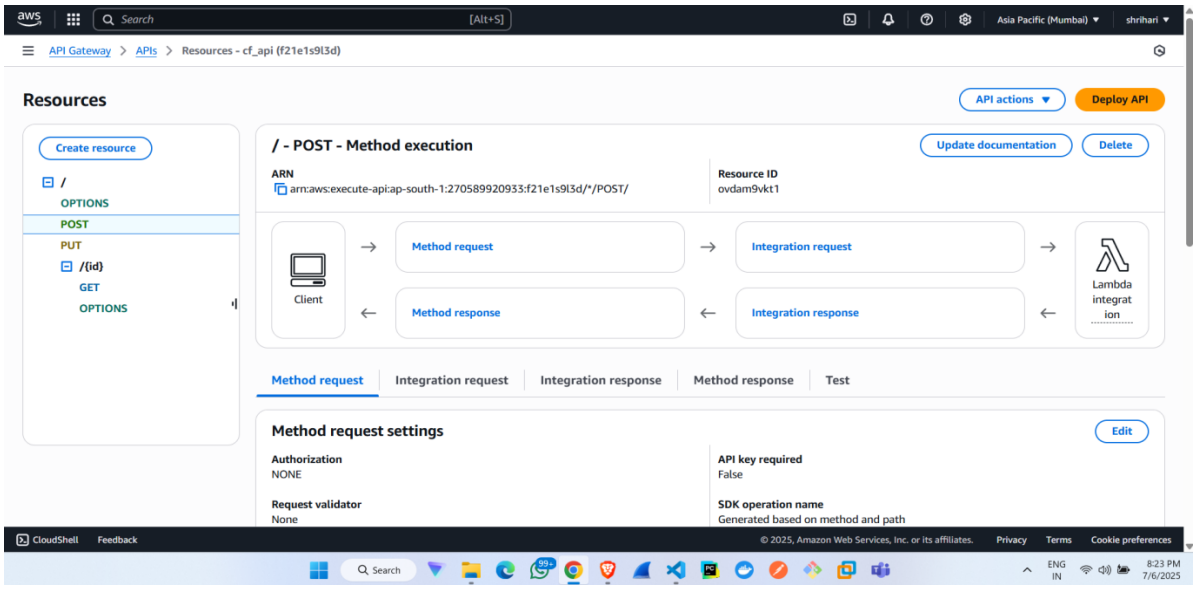
4.3 Snapshots

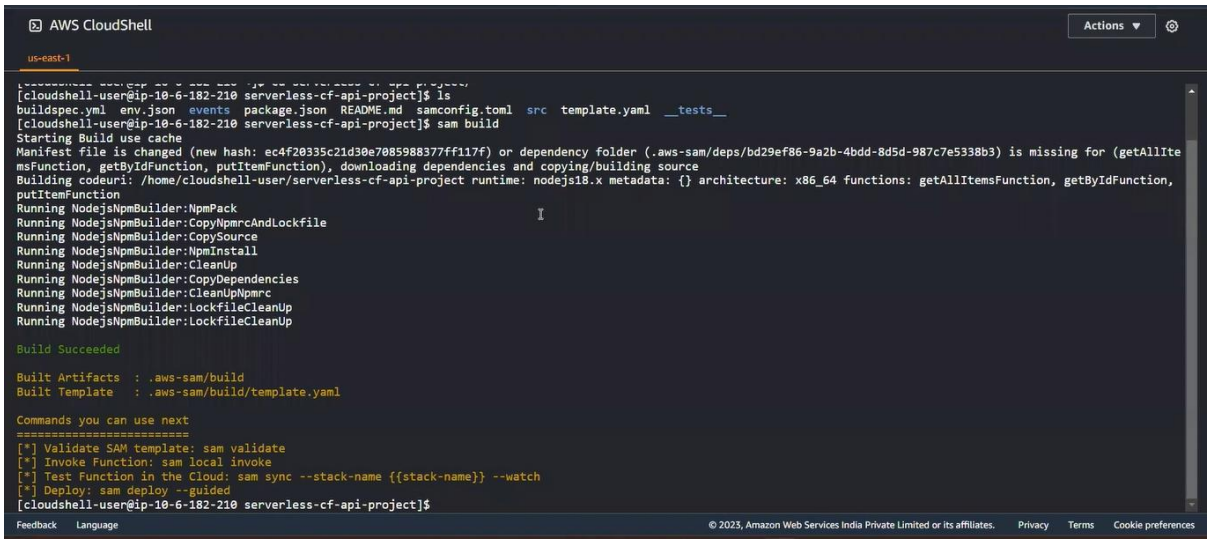


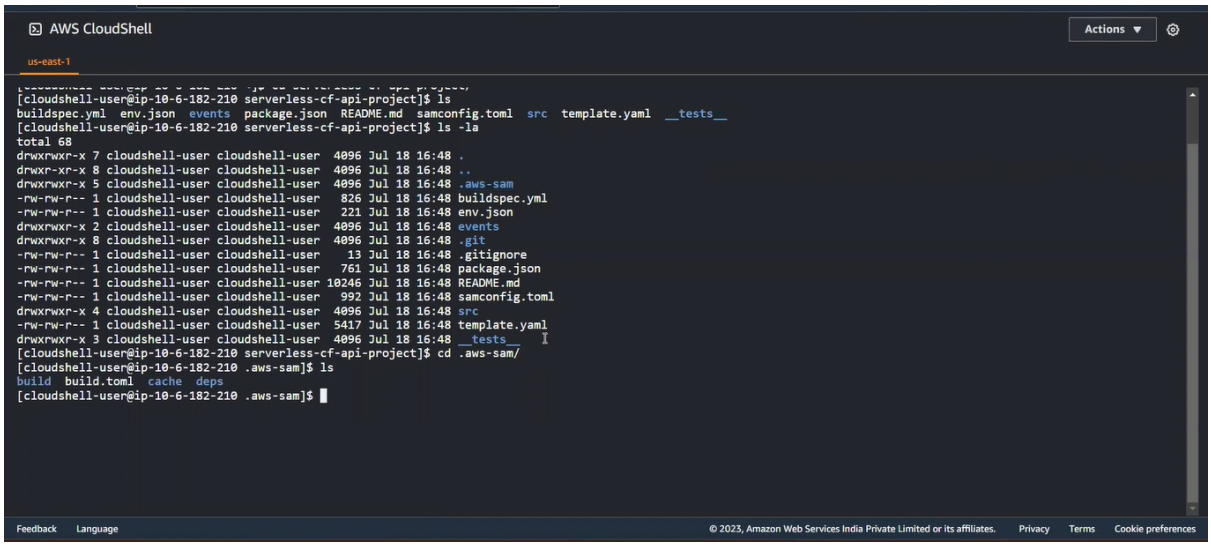

API Gateway > APIs > Resources - cf_api (f21e1s9l3d)
Successfully created method 'PUT' in '/'. Redeploy your API for the update to take effect.
Resources
API actions
Deploy API
Create resource
/
PUT
/ - PUT - Method execution
Update documentation
Delete
ARN
arn:aws:execute-api:ap-south-1:270589920933:f21e1s9l3d/*/PUT/
Resource ID
ovdam9vkt1
Client
Method request
Integration request
Method response
Integration response
Lambda integration
Method request
Integration request
Integration response
Method response
Test
Method request settings
Edit
Authorization
API key required


API Gateway > APIs > Resources - cf_api (f21e1s9l3d)
Resources
API actions
Deploy API
Create resource
/
OPTIONS
PUT
/ - OPTIONS - Method execution
Update documentation
Delete
ARN
arn:aws:execute-api:ap-south-1:270589920933:f21e1s9l3d/*/OPTIONS/
Resource ID
ovdam9vkt1
Client
Method request
Integration request
Method response
Integration response
Lambda integration
Method request
Integration request
Integration response
Method response
Test
Method request settings
Edit
Authorization
NONE
API key required
False
Request validator
SDK operation name


API Gateway > APIs > Resources - cf_api (f21e1s9l3d)
Resources
API actions
Deploy API
Create resource
/
OPTIONS
POST
PUT
/{id}
GET
OPTIONS
Resource details
Update documentation
Enable CORS
Path
/
Resource ID
ovdam9vkt1
Methods (3)
Delete
Create method
Method type
Integration type
Authorization
API key
OPTIONS Lambda None Not required
POST Lambda None Not required
PUT Lambda None Not required


API Gateway > APIs > Resources - cf_api (f21e1s9l3d)
Resources
API actions
Deploy API
Create resource
/
OPTIONS
POST
PUT
/{id}
GET
OPTIONS
/ - POST - Method execution
Update documentation
Delete
ARN
arn:aws:execute-api:ap-south-1:270589920933:f21e1s9l3d/*/POST/
Resource ID
ovdam9vkt1
Client
Method request
Integration request
Method response
Integration response
Lambda integration
Method request
Integration request
Integration response
Method response
Test
Method request settings
Edit
Authorization
NONE
API key required
False
Request validator
None
SDK operation name
Generated based on method and path


AWS CloudShell
Actions
us-east-1
[cloudshell-user@ip-10-6-182-210 serverless-cf-api-project]$ ls
buildspec.yml env.json events package.json README.md samconfig.toml src template.yaml __tests__
[cloudshell-user@ip-10-6-182-210 serverless-cf-api-project]$ sam build
Starting Build use cache
Manifest file is changed (new hash: ec4f20335c21d30e7085988377ff117f) or dependency folder (.aws-sam/deps/bd29ef86-9a2b-4bdd-8d5d-987c7e5338b3) is missing for (getAllItemsFunction, getByIdFunction, putItemFunction), downloading dependencies and copying/building source
Building codeuri: /home/cloudshell-user/serverless-cf-api-project runtime: nodejs18.x metadata: {} architecture: x86_64 functions: getAllItemsFunction, getByIdFunction, putItemFunction
Running NodejsNpmBuilder:NpmPack
Running NodejsNpmBuilder:CopyNpmrcAndLockfile
Running NodejsNpmBuilder:CopySource
Running NodejsNpmBuilder:NpmInstall
Running NodejsNpmBuilder:CleanUp
Running NodejsNpmBuilder:CopyDependencies
Running NodejsNpmBuilder:CleanUpNpmrc
Running NodejsNpmBuilder:LockfileCleanUp
Running NodejsNpmBuilder:LockfileCleanUp
Build Succeeded
Built Artifacts : .aws-sam/build
Built Template : .aws-sam/build/template.yaml
Commands you can use next
[*] Validate SAM template: sam validate
[*] Invoke Function: sam local invoke
[*] Test Function in the Cloud: sam sync --stack-name {{stack-name}} --watch
[*] Deploy: sam deploy --guided
[cloudshell-user@ip-10-6-182-210 serverless-cf-api-project]$
Feedback
Language
© 2023, Amazon Web Services India Private Limited or its affiliates.
Privacy
Terms
Cookie preferences


AWS CloudShell
Actions
us-east-1
[cloudshell-user@ip-10-6-182-210 serverless-cf-api-project]$ ls
buildspec.yml env.json events package.json README.md samconfig.toml src template.yaml __tests__
[cloudshell-user@ip-10-6-182-210 serverless-cf-api-project]$ ls -la
total 68
drwxrwxr-x 7 cloudshell-user cloudshell-user 4096 Jul 18 16:48 .
drwxr-xr-x 8 cloudshell-user cloudshell-user 4096 Jul 18 16:48 ..
drwxrwxr-x 5 cloudshell-user cloudshell-user 4096 Jul 18 16:48 .aws-sam
-rw-rw-r-- 1 cloudshell-user cloudshell-user 826 Jul 18 16:48 buildspec.yml
-rw-rw-r-- 1 cloudshell-user cloudshell-user 221 Jul 18 16:48 env.json
drwxrwxr-x 2 cloudshell-user cloudshell-user 4096 Jul 18 16:48 events
drwxrwxr-x 8 cloudshell-user cloudshell-user 4096 Jul 18 16:48 .git
-rw-rw-r-- 1 cloudshell-user cloudshell-user 13 Jul 18 16:48 .gitignore
-rw-rw-r-- 1 cloudshell-user cloudshell-user 761 Jul 18 16:48 package.json
-rw-rw-r-- 1 cloudshell-user cloudshell-user 10246 Jul 18 16:48 README.md
-rw-rw-r-- 1 cloudshell-user cloudshell-user 992 Jul 18 16:48 samconfig.toml
drwxrwxr-x 4 cloudshell-user cloudshell-user 4096 Jul 18 16:48 src
-rw-rw-r-- 1 cloudshell-user cloudshell-user 5417 Jul 18 16:48 template.yaml
drwxrwxr-x 3 cloudshell-user cloudshell-user 4096 Jul 18 16:48 __tests__
[cloudshell-user@ip-10-6-182-210 serverless-cf-api-project]$ cd .aws-sam/
[cloudshell-user@ip-10-6-182-210 .aws-sam]$ ls
build build.toml cache deps
[cloudshell-user@ip-10-6-182-210 .aws-sam]$
Feedback
Language
© 2023, Amazon Web Services India Private Limited or its affiliates.
Privacy
Terms
Cookie preferences

5. Deployment

5.1 CI/CD with GitHub Actions

A GitHub Actions workflow is configured to automate deployment every time code is pushed to the main branch.

Workflow file: .github/workflows/deploy.yml

Steps:

1. Checkout code
2. Set up Python 3.11 environment
3. Run basic lint/tests (optional)
4. Zip lambda_function.py and dependencies
5. Configure AWS CLI using encrypted secrets
6. Deploy/update Lambda and API Gateway via AWS lambda update-function-code or Cloud Formation

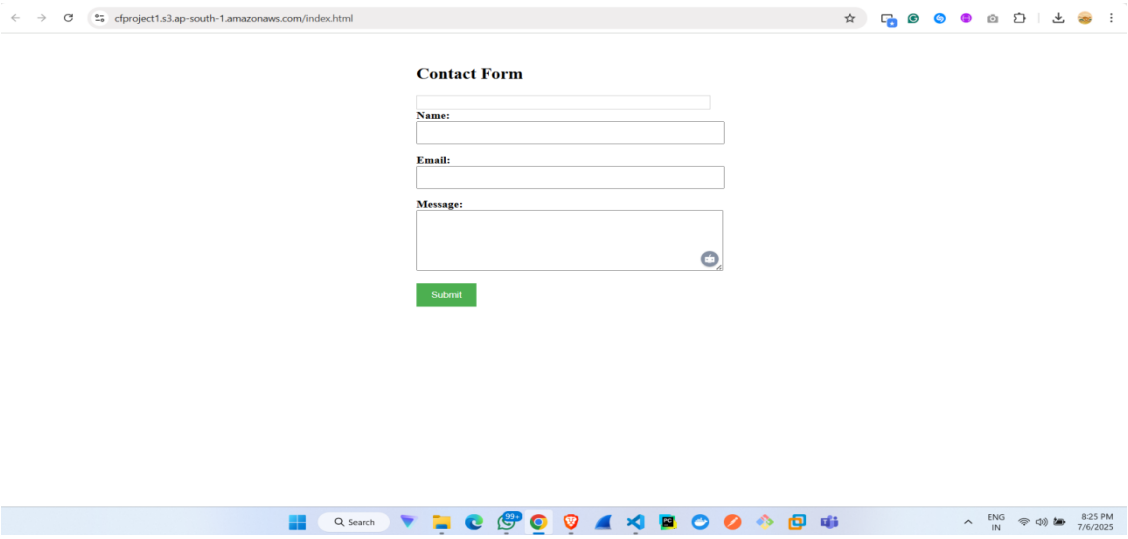
5.2 Secrets Used:

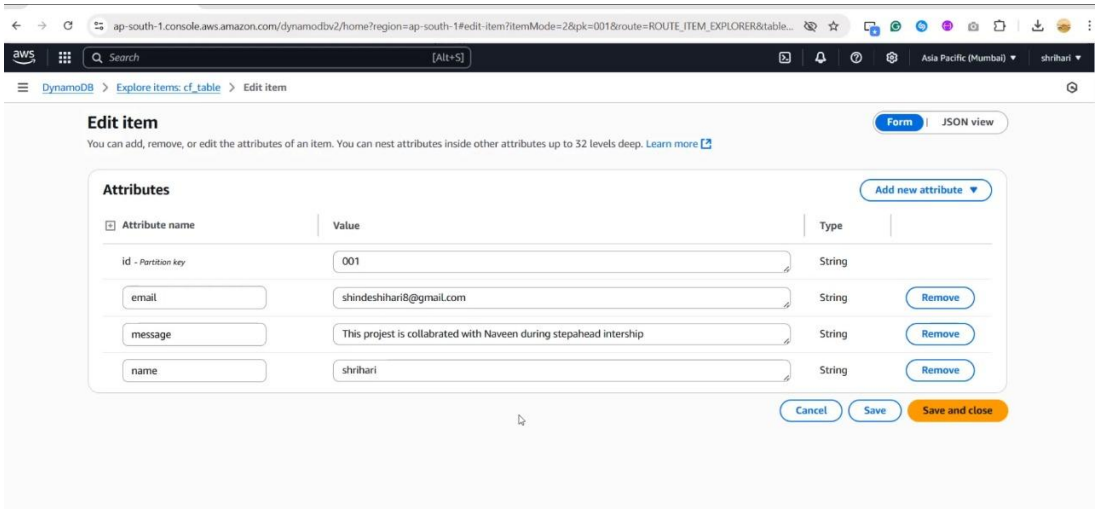
- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY
- AWS_REGION

5.3 Benefits:

- Ensures deployment consistency
- Avoids manual errors during deployment
- Encourages fast iteration and testing

5.4 Snapshots:





6. Outcome & Learning's

6.1 Key Learning's

- **AWS Lambda**: Gained hands-on experience with writing and deploying serverless functions
- **API Gateway**: Learned to create REST endpoints and integrate with Lambda
- **DynamoDB**: Understood NoSQL design and provisioning tables
- **IAM**: Configuring least privilege roles for secure operation
- **CI/CD with GitHub Actions**: Setup and automated AWS deployment workflow
- **Cloud Watch**: Real-time error tracking and performance logging

6.2 Challenges & Solutions

- **IAM Role Errors**: Fixed by refining role policy to include only dynamodb:PutItem and logs: CreateLogStream, logs:PutLogEvents.
- **Mismatch Between Postman and Lambda**: Addressed by normalizing Content-Type handling in Lambda.
- **Cloud Watch Log Overheads**: Optimized by batching logs and buffer handling in Lambda.

6.3 Future Enhancements

- Add **GET** endpoint to retrieve past feedback
- Build a frontend-hosted form via S3 + Cloud Front
- Use **SES** for email confirmation to users
- Configure **DynamoDB TTL** to expire feedback after a set period
- Add **Unit Tests** for Lambda logic (using .pytest)

7. Conclusion

The **Serverless Feedback Form** project provided deep insight into designing scalable, cost-efficient applications using AWS. It enabled us to focus on **business logic** without worrying about infrastructure provisioning or backend servers. From **RESTful API design** to **CI/CD automation**; this mini project covered key DevOps and cloud engineering principles.

We successfully built, tested, and deployed a production-ready serverless application and gained confidence in **cloud-native development workflows**. This experience serves as a stepping stone for more advanced full-stack and micro-services projects in the AWS ecosystem.

8. References

- **API Gateway:** https://youtu.be/c3J5uvdfSfE?si=cs0AhfickUPYTzCZ
- **DynamoDB:** https://youtu.be/2k2GINpO308?si=sNSQ2gR-ix4nQTbV
- **AWS Lambda Docs:** https://docs.aws.amazon.com/lambda/
- **GitHub Actions Docs:** https://docs.github.com/en/actions

Curious? Click here to explore the GitHub repo and dive into the code!

****