
CloudLingo – Automated Translation Pipeline

Author: Prince Larbi Wireko

Date: 5th September 2025

Course: AWS Cloud Intensive AZ_2 Cohort

Instructor: Mrs. Ajara Amadu



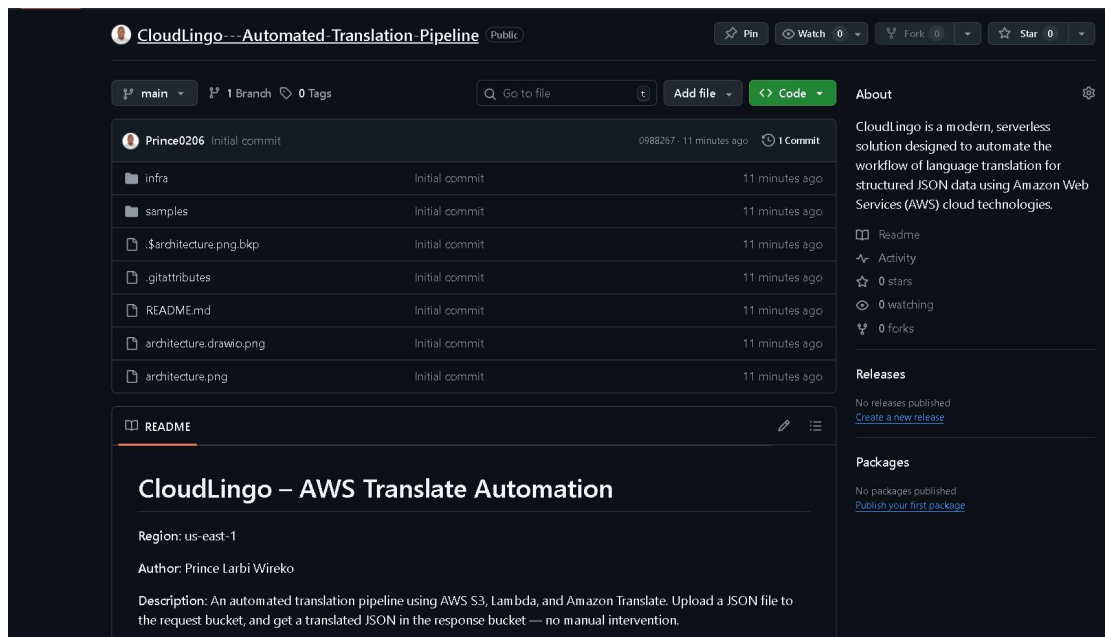
CloudLingo: A Serverless, Automated Translation Pipeline on AWS

Introduction

CloudLingo is a modern, serverless solution designed to automate the workflow of language translation for structured JSON data using Amazon Web Services (AWS) cloud technologies. The core purpose of the system is to enable users to effortlessly translate JSON files between languages, eliminating the manual work and operational overhead that accompanies traditional translation processes. With CloudLingo, a user simply uploads a JSON file in one language to a designated AWS S3 bucket; the system instantly orchestrates an event-driven series of operations which trigger translation and output of the result and also as a JSON file to a parallel destination bucket. This approach not only reduces costs compared to manual translation workflows but also scales seamlessly with fluctuating workloads and varied language requirements.

CloudLingo demonstrates the practical application of cloud-native, event-driven architectures, leveraging AWS's pay-per-use serverless model. Essential services in this pipeline include Amazon S3 for file storage, AWS Lambda for function-as-a-service compute, and Amazon Translate for robust neural machine translation. These components, integrated via event triggers and orchestrated using Infrastructure as Code (IaC) templates, contribute to a scalable, cost-effective, and maintenance-minimized translation system. The solution acts as a blueprint for enterprises and developers seeking to automate multi-language support for digital assets, websites, catalogs, or communications in a globally distributed context.

GitHub repository homepage



The screenshot displays the GitHub repository page for "CloudLingo - Automated Translation Pipeline". The repository is public and was created by Prince0206. It shows a list of files and folders, including "infra", "samples", ".architecture.png.bkp", ".gitattributes", "README.md", "architecture.drawio.png", and "architecture.png". The README file is selected, showing the title "CloudLingo - AWS Translate Automation". The README content includes the region "us-east-1", the author "Prince Larbi Wireko", and a description: "An automated translation pipeline using AWS S3, Lambda, and Amazon Translate. Upload a JSON file to the request bucket, and get a translated JSON in the response bucket — no manual intervention." The right sidebar shows the repository's statistics, including 0 stars, 0 forks, and 0 releases.

CloudLingo is a modern, serverless solution designed to automate the workflow of language translation for structured JSON data using Amazon Web Services (AWS) cloud technologies.

CloudLingo – AWS Translate Automation

Region: us-east-1

Author: Prince Larbi Wireko

Description: An automated translation pipeline using AWS S3, Lambda, and Amazon Translate. Upload a JSON file to the request bucket, and get a translated JSON in the response bucket — no manual intervention.

Objectives

CloudLingo was conceived and designed around five core objectives to ensure automation, maintainability, testability, and user comprehension. Each objective drives the system architecture, tooling decisions, development practices, and documentation approach:

1. Automation

The foremost goal is end-to-end automation of the translation workflow. In CloudLingo, translation occurs automatically in reaction to user activity specifically, the upload of a source JSON file into the input S3 bucket. Manual intervention is not required at any stage of the pipeline. By integrating AWS S3 event triggers and Lambda, CloudLingo achieves a fully automated translation process, minimizing errors and dramatically improving speed. This objective highlights the broader shift towards DevOps practices of automating repetitive business logic, thus reducing manual labor and swiftly adapting to business needs.

2. Event-Driven Architecture

CloudLingo adopts an event-driven architecture, which is a pillar of modern, scalable cloud design. The pipeline is structured so that the upload of a file constitutes an 'event,' triggering a Lambda function without polling or manual scheduling. This design decouples the initiation of the workflow from the actual processing logic, allowing independent evolution, scaling, and failure isolation of system components. The event-driven model facilitates real-time or near-real-time translation and simplifies the integration of additional services (e.g., notification, further processing) down the line.

3. Infrastructure as Code (IaC)

A third strategic objective is complete infrastructure management via Infrastructure as Code, specifically using AWS CloudFormation. All AWS resources S3 buckets, Lambda functions, IAM roles, event notifications are specified declaratively in a CloudFormation YAML template. This codified approach brings version control, repeatability, and automation to infrastructure deployment, allowing teams to reproduce, review, and extend the setup consistently and rapidly in any AWS account.

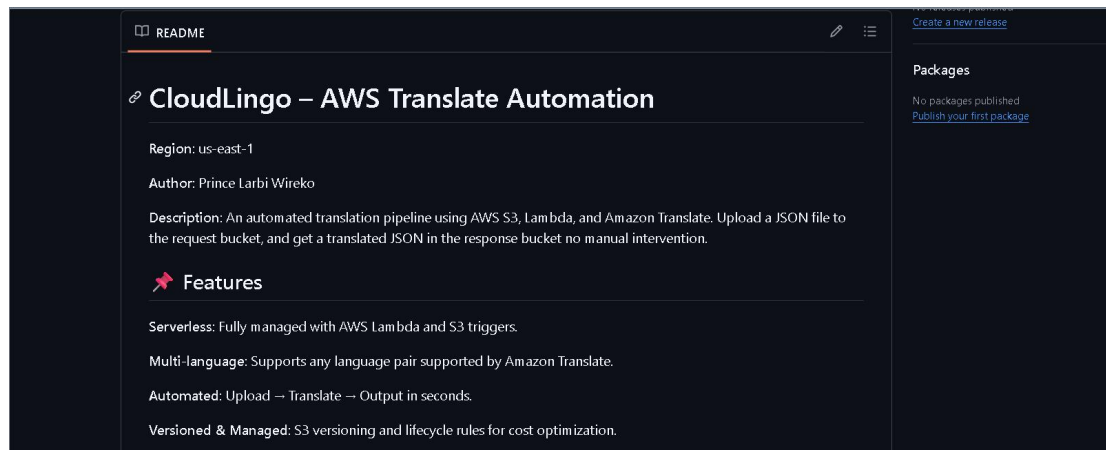
4. Testing Workflows

Thorough testing workflows are integrated at every stage, from local Lambda function testing with Python and the AWS SAM CLI to functional workflow validation in the AWS cloud. CloudLingo provides scripts and procedures for developers to simulate S3 upload events, invoke Lambda logic locally, and inspect both the transformation and output. Automated and manual testing, combined with robust logging through AWS CloudWatch, ensure that edge cases and potential translation failures are addressed pre-deployment, thus yielding reliable production systems.

5. Documentation

Comprehensive documentation was prioritised to ensure onboarding ease and maintainability. All IaC templates, Lambda code, testing scripts, and configuration options are extensively documented in Markdown, adhering to best practices for open-source and cloud-native projects. The project includes a README, code comments, workflow diagrams, and a troubleshooting guide. This documentation

lowers the entry barrier for new contributors, provides deployment and usage instructions for end users, and supports extensibility over time.



Tools & Technologies

The implementation of CloudLingo draws on several AWS managed services, developer tools, and supporting frameworks, each chosen for their reliability, scalability, and integration capabilities.

Primary AWS Services

- **Amazon S3 (Simple Storage Service):**

Serves as the storage backbone for input and output JSON files. S3 buckets are configured with event notifications to trigger Lambda functions on file uploads. Lifecycle policies can be attached for data retention management.

- **AWS Lambda:**

Provides serverless compute that executes translation logic on a per-event basis. Written in Python, the Lambda function is packaged and deployed via CloudFormation; the function reads from the S3 input bucket, invokes Amazon Translate, and writes results back to the output bucket.

- **Amazon Translate:**

The managed neural machine translation service responsible for converting textual content from the source to the target language. Integrated with Lambda through the Boto3 SDK, it supports more than 70 languages and automatic language detection.

- **AWS IAM (Identity and Access Management):**

Used for defining and enforcing granular roles and permissions. Lambda assumes an execution role granting it least-privilege access to S3 and Translate APIs, ensuring security best practices.

- **AWS CloudFormation:**

The IaC tool employed to declaratively define and orchestrate all resources—including S3 buckets, Lambda functions, IAM roles, and event notifications—through a single YAML template.

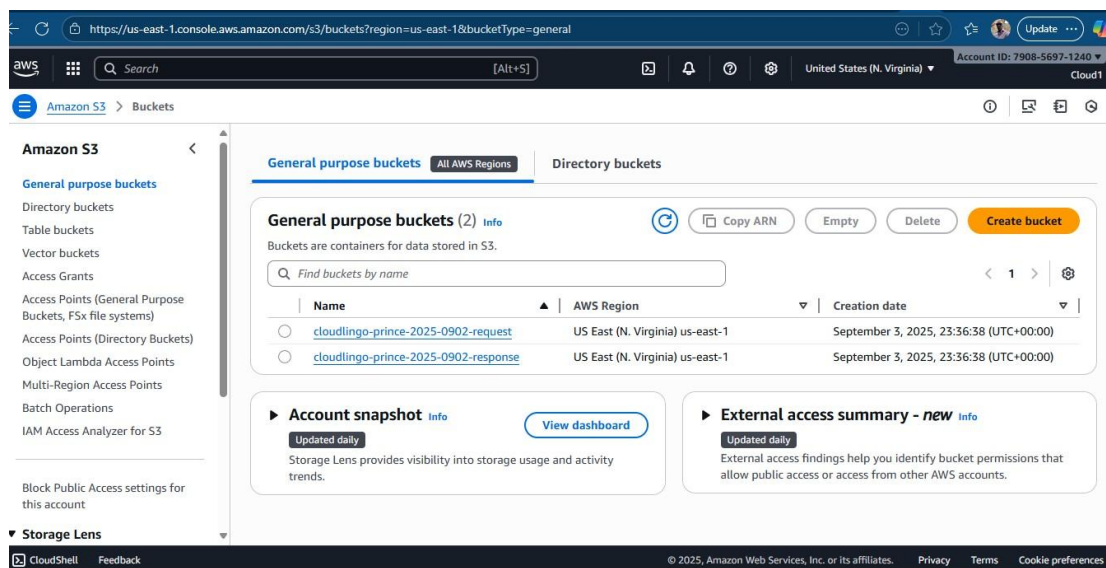
- **Amazon CloudWatch:**

Used for ingesting Lambda logs, monitoring translation metrics, setting alarms, and observing system health and performance-related data.

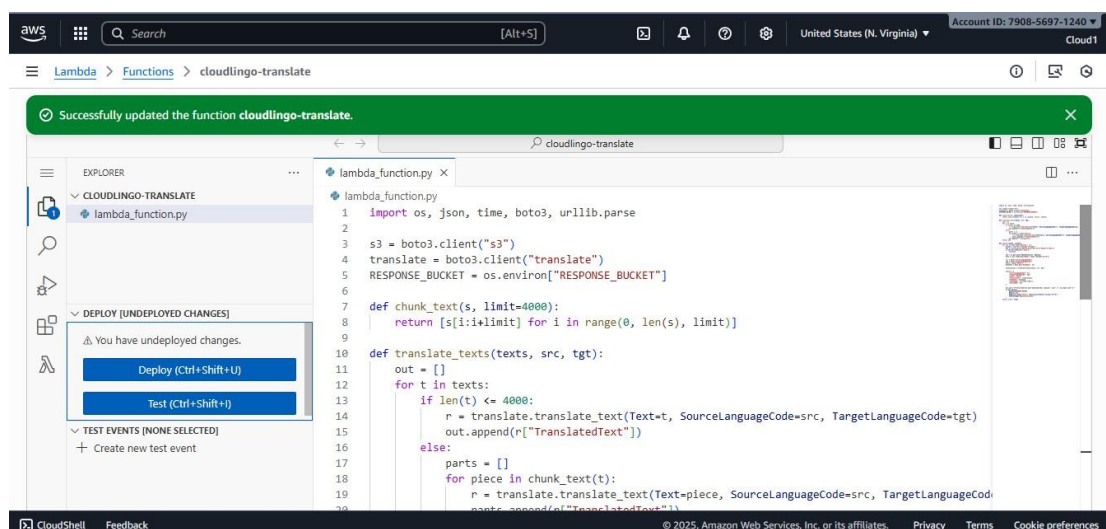
Supporting Tools and Libraries

- **AWS CLI:** For interacting with AWS services programmatically and performing resource provisioning, file uploads, and manual workflow triggers.
- **Python (3.9+):** The primary language for Lambda implementation and local testing scripts.
- **Boto3 (AWS SDK for Python):** The official SDK for communicating with AWS services from Python code, supports Amazon S3, Lambda, and Translate functionalities.
- **AWS SAM CLI:** Provides local emulation of Lambda and APIGateway environments for development, testing, and debugging serverless applications before live deployment.
- **Docker:** Utilized by SAM CLI to replicate AWS Lambda's runtime environment for more accurate local tests.

AWS Console views for S3 buckets,



Lambda function dashboard



Amazon Translate activity panel

The top screenshot shows a VS Code editor with a Python script named `translate_local.py`. The script uses `boto3` to interact with the Amazon Translate service. It defines a `chunk_text` function to split text into chunks under 5000 bytes and a `translate_texts` function to call the `translate_text` API. The terminal output shows the script being executed with the command `python translate_local.py request_en_fr.json cloudlingo-prince-2025-0902-request cloudlingo-prince-2025-0902-response`. The output indicates that the file `request_en_fr.json` was successfully uploaded to the S3 bucket `s3://cloudlingo-prince-2025-0902-response/translated/`.

The bottom screenshot shows the AWS Lambda console for the `cloudlingo-translate` function. The function overview shows that the function was successfully created and is ready to be invoked. The function is configured with a trigger of `request_en_fr.json` and a destination of `s3://cloudlingo-prince-2025-0902-response/translated/`. The function's description, last modified time, and ARN are also displayed.

System Architecture

CloudLingo's system design is fundamentally serverless and event-driven, minimizing operational complexity while maximizing scalability. Below is a step-by-step breakdown of the end-to-end workflow and a direction for the architecture diagram snapshot.

Workflow Steps

- User Uploads Input JSON:**
A user uploads a JSON file (containing keys for source language, target language, and text) into a predefined S3 'Request' (input) bucket.
- S3 Event Notification:**
The S3 bucket is configured with an event trigger that fires whenever a new object is created, sending a notification containing the file metadata to AWS Lambda.
- Lambda Processing:**
AWS Lambda is invoked by the S3 event. It fetches the object, parses the JSON, and extracts the source language, target language, and original text.

4. Translation Invocation:

Lambda submits the extracted content to Amazon Translate's synchronous API, requesting translation into the desired target language.

5. Output Generation:

The translated text, alongside source metadata, is formatted into a new JSON structure.

6. Result Storage:

Lambda writes the output JSON file to a predefined S3 'Response' (output) bucket, typically appending or prepending '-output' to the original filename to distinguish translated artifacts.

7. (Optional) Logging and Monitoring:

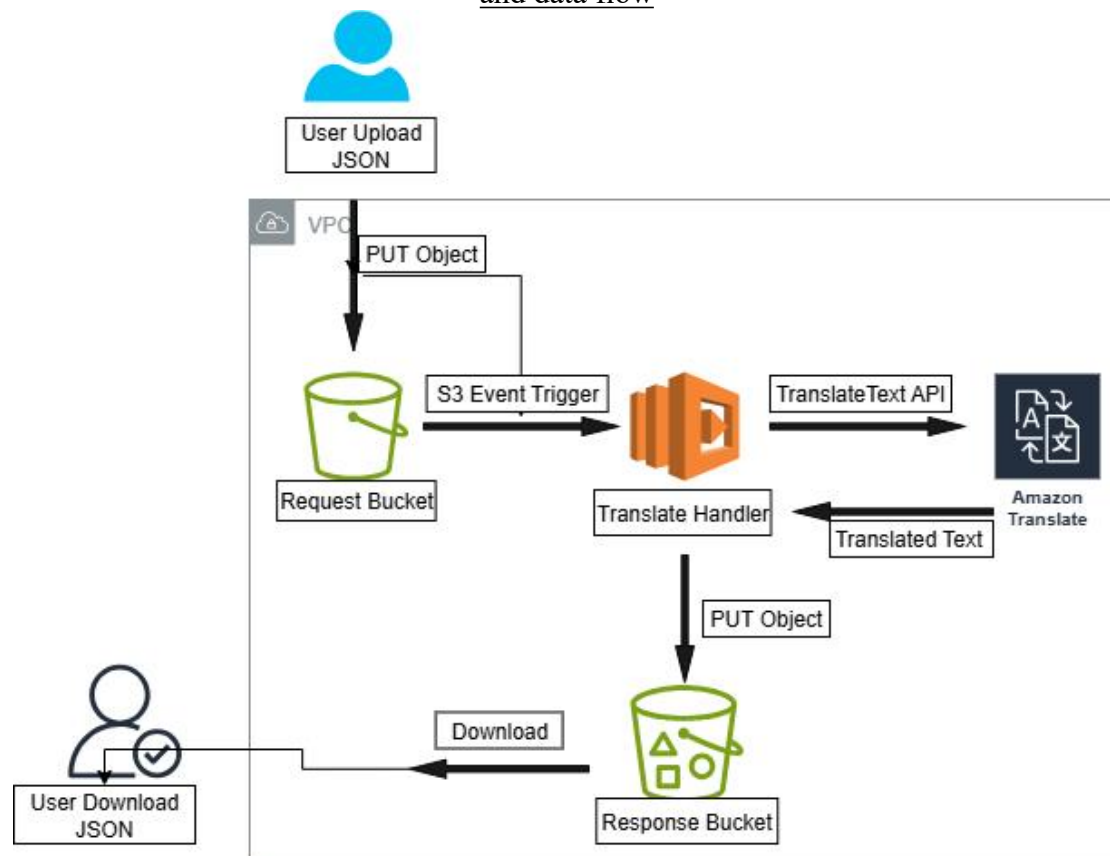
Throughout the process, logs and potential errors are sent to AWS CloudWatch for metrics tracking, debugging, and post-mortem analysis.

Architecture Diagram (Description for Snapshot)

The typical diagram consists of:

- **S3 Request Bucket:** Receiving JSON uploads
 - **S3 event triggers Lambda Execution**
 - **Lambda Function:** Reads, invokes Amazon Translate API
 - **Amazon Translate:** Performs translation
 - **Lambda:** Receives translation, writes output
 - **S3 Response Bucket:** Stores translated JSON
 - **CloudWatch:** Collects logs for all actions

An architecture diagram depicting the workflow, with arrows showing event triggers and data flow



Implementation Steps

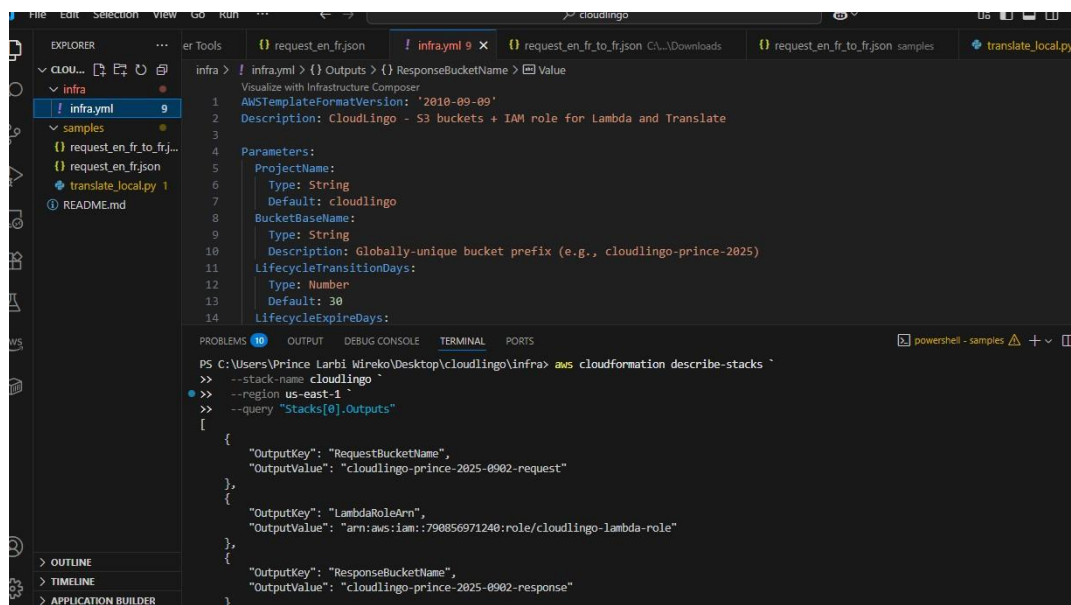
CloudLingo's deployment and operationalization involve several critical steps, from infrastructure provisioning to application logic integration and test validation.

Infrastructure Deployment with CloudFormation

The entire infrastructure is automatically instantiated through a CloudFormation YAML template:

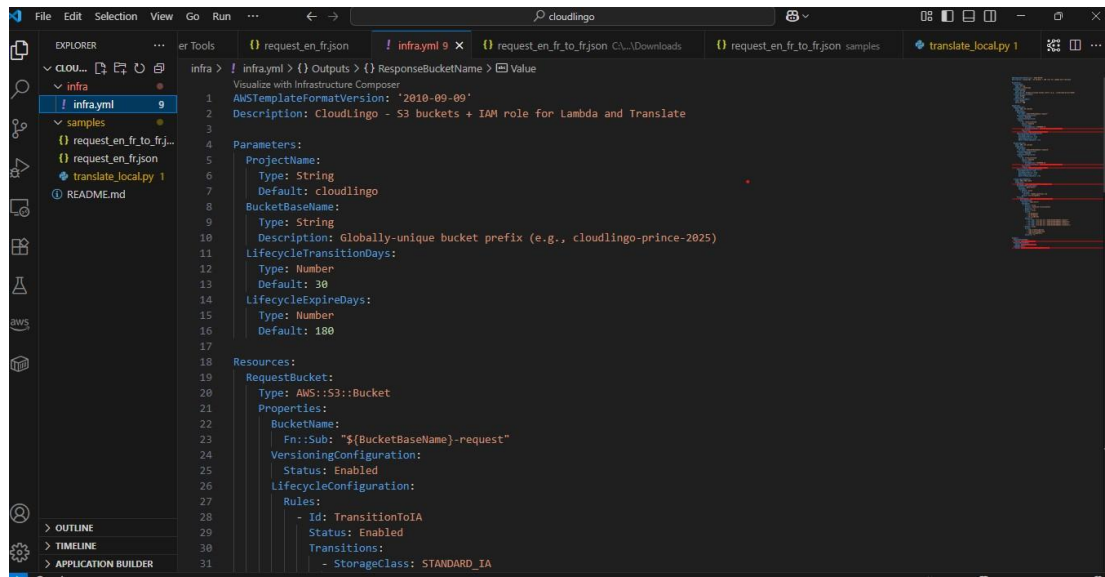
- Defines two S3 buckets: one for incoming requests (input), another for outgoing responses (output).
- Provisions a Lambda function with settings for runtime (Python 3.12), environment variables, and handler file.
- Configures IAM roles granting Lambda least-privilege permissions—read from input bucket, write to output bucket, and invoke Amazon Translate API.
- Attaches an S3 event notification to the request bucket, triggering the Lambda function on every file upload each time an object is PUT.
- Sets up CloudWatch log groups for Lambda.
- All resource names can be parameterized or dynamically generated to avoid naming conflicts.

deployed stack in the AWS CloudFormation Console, showing resources' hierarchy and status.



The screenshot shows an IDE with a CloudFormation template file named `infra.yml` and a terminal window displaying the output of the `aws cloudformation describe-stacks` command. The template defines parameters for the project name, bucket base name, lifecycle transition days, and lifecycle expire days. The terminal output shows the stack name `cloudlingo` in the `us-east-1` region, with the following outputs:

```
[{"OutputKey": "RequestBucketName", "OutputValue": "cloudlingo-prince-2025-0902-request"}, {"OutputKey": "LambdaRoleArn", "OutputValue": "arn:aws:iam::790856971240:role/cloudlingo-lambda-role"}, {"OutputKey": "ResponseBucketName", "OutputValue": "cloudlingo-prince-2025-0902-response"}]
```

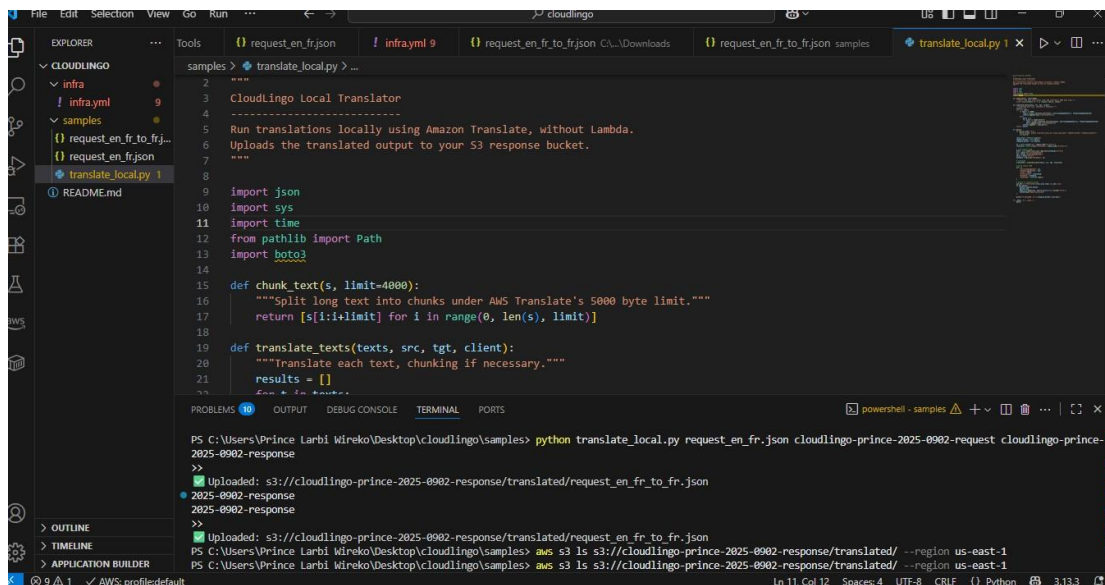
Local Testing Script Development

CloudLingo provides a Python-based local testing script to simulate Lambda translation logic. The script allows developers to:

- Mimic S3 event structure and environment,
- Read local JSON files,
- Call Amazon Translate using Boto3,
- Output the translation either to the local filesystem or to a test S3 bucket.

This encourages rapid iterations and early error detection before resources are committed in the cloud, improving development productivity and reliability.

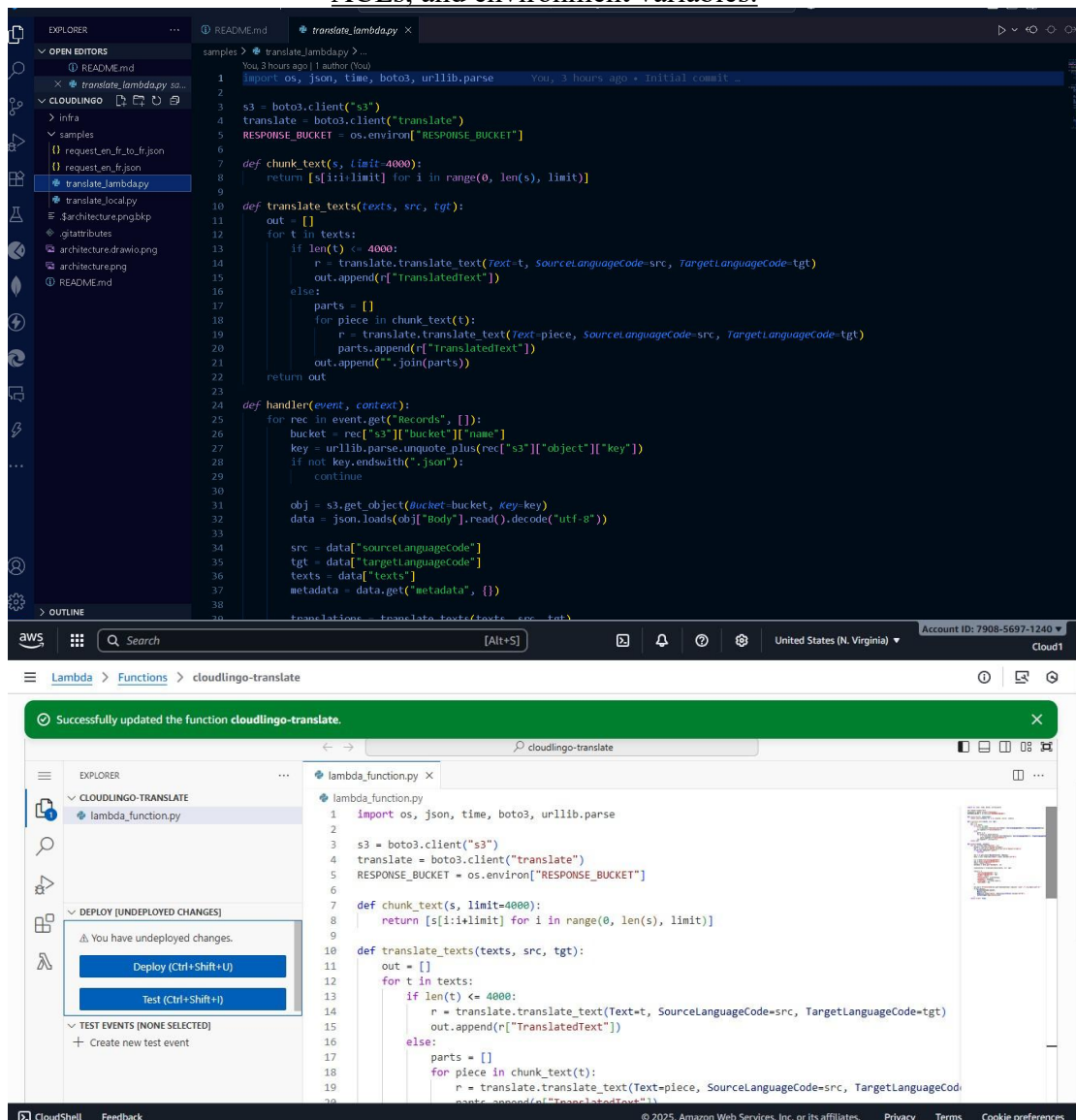
The testing script's execution, showing input, translation call, and output write.



Lambda Function Deployment

- Source code for the Lambda function (typically) resides in the project repository.
- Developers can update code or environment variables via the AWS Console or by updating and redeploying the CloudFormation stack.
- The function is stateless, designed for single-event processing, ensuring horizontal scalability.
- Unit and integration tests are included to validate JSON parsing, language codes, and Amazon Translate patterns.

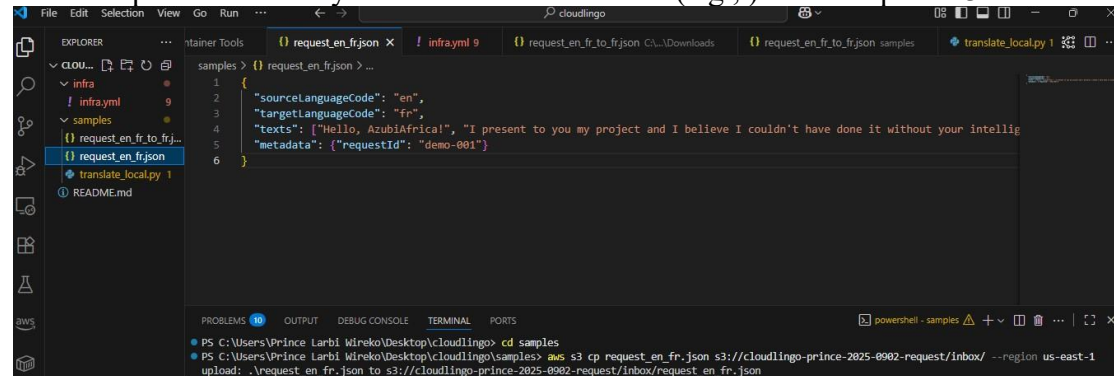
AWS Lambda console's function configuration page, highlighting handler, runtime, ACLs, and environment variables.



Testing and Validation Process

Functional and end-to-end validation proceeds as follows:

1. Upload a correctly formatted test JSON file (e.g.,) to the request S3 bucket:



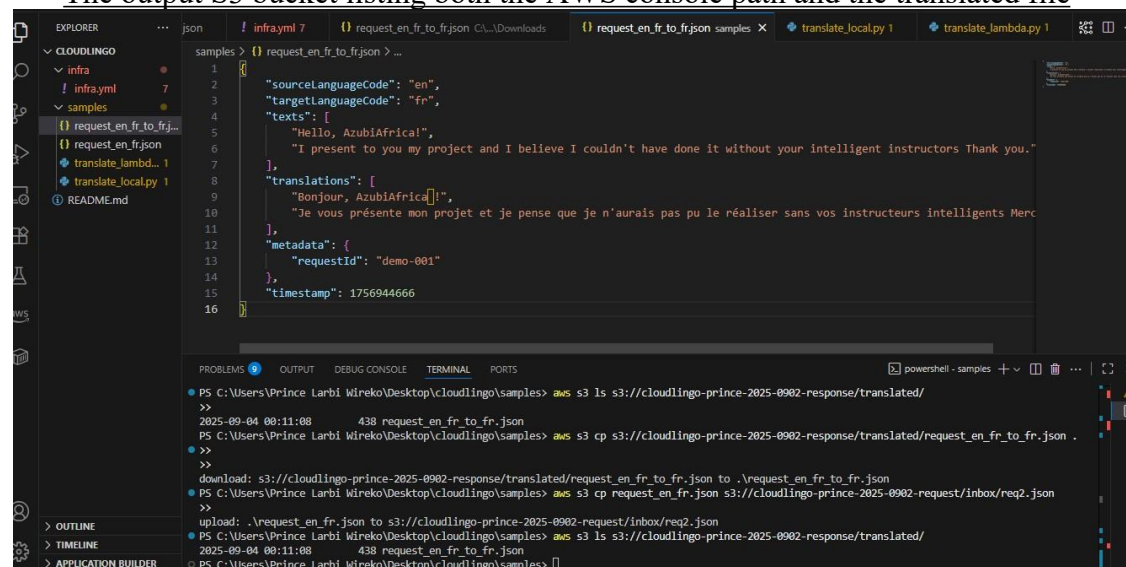
```
samples > {} request_en_fr.json > ...
1
2 {
3   "sourceLanguageCode": "en",
4   "targetLanguageCode": "fr",
5   "texts": ["Hello, AzubiAfrica!", "I present to you my project and I believe I couldn't have done it without your intellig
6   "metadata": {"requestId": "demo-001"}
7 }
```

```
PS C:\Users\Prince Larbi Wireko\Desktop\cloudingo> cd samples
PS C:\Users\Prince Larbi Wireko\Desktop\cloudingo\samples> aws s3 cp request_en_fr.json s3://cloudingo-prince-2025-0902-request/inbox/ --region us-east-1
upload: \request_en_fr.json to s3://cloudingo-prince-2025-0902-request/inbox/request_en_fr.json
```

2. Wait a few seconds, then check the output (response) S3 bucket:
 - Output file should appear as “translation_test-output.json” .
 - Output JSON should include both the original and translated text, with correct fields.

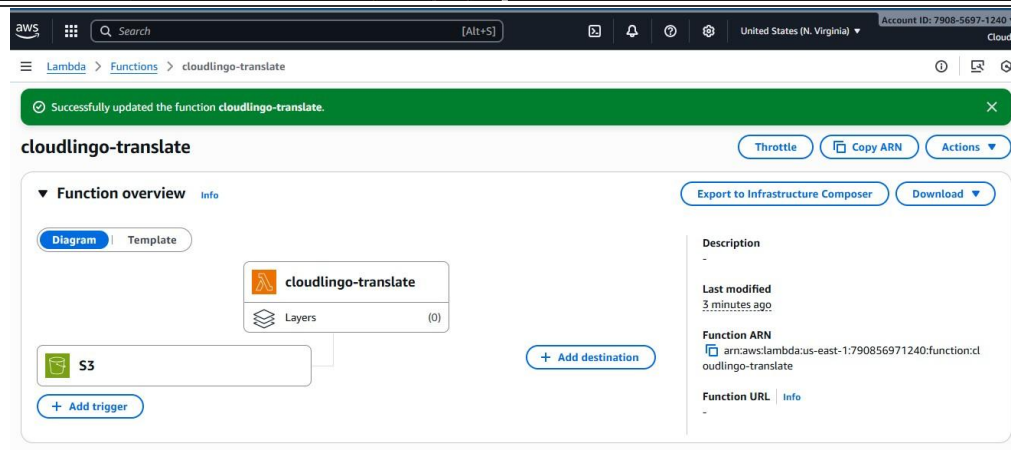
3. Review CloudWatch Logs for Lambda invocation, error traces, and translation metrics.

4. The output S3 bucket listing both the AWS console path and the translated file



```
samples > {} request_en_fr.json > ...
1
2 {
3   "sourceLanguageCode": "en",
4   "targetLanguageCode": "fr",
5   "texts": [
6     "Hello, AzubiAfrica!",
7     "I present to you my project and I believe I couldn't have done it without your intelligent instructors Thank you."
8   ],
9   "translations": [
10    "Bonjour, AzubiAfrica!",
11    "Je vous présente mon projet et je pense que je n'aurais pas pu le réaliser sans vos instructeurs intelligents Merc
12  ],
13   "metadata": {
14     "requestId": "demo-001"
15   },
16   "timestamp": 1756944666
17 }
```

```
PS C:\Users\Prince Larbi Wireko\Desktop\cloudingo\samples> aws s3 ls s3://cloudingo-prince-2025-0902-response/translated/
>>
2025-09-04 00:11:08      438 request_en_fr_to_fr.json
>>
PS C:\Users\Prince Larbi Wireko\Desktop\cloudingo\samples> aws s3 cp s3://cloudingo-prince-2025-0902-response/translated/request_en_fr_to_fr.json .
>>
download: s3://cloudingo-prince-2025-0902-response/translated/request_en_fr_to_fr.json to .\request_en_fr_to_fr.json
PS C:\Users\Prince Larbi Wireko\Desktop\cloudingo\samples> aws s3 cp request_en_fr.json s3://cloudingo-prince-2025-0902-request/inbox/req2.json
>>
upload: \request_en_fr.json to s3://cloudingo-prince-2025-0902-request/inbox/req2.json
PS C:\Users\Prince Larbi Wireko\Desktop\cloudingo\samples> aws s3 ls s3://cloudingo-prince-2025-0902-response/translated/
2025-09-04 00:11:08      438 request_en_fr_to_fr.json
PS C:\Users\Prince Larbi Wireko\Desktop\cloudingo\samples> {}
```



Results

CloudLingo delivered measurable and qualitative benefits, with specific outcomes in translation accuracy, pipeline speed, and developer/tester experience.

Translation Success

- **Accuracy:** The use of Amazon Translate ensures high-fidelity translation for supported language pairs (including auto-detection of the source language).
- **Tested Pairs:** The pipeline successfully translated English source text into French, Spanish, Japanese, and other supported languages based on configuration.
- **Edge Cases:** User errors (malformed JSON, unsupported language codes) were effectively caught and logged in CloudWatch, reinforcing robustness.

Performance and Speed

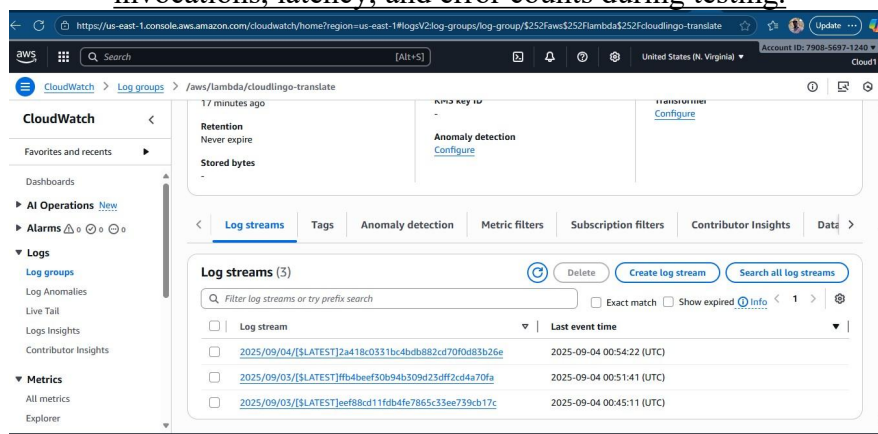
- **Latency:** Translation times per file were typically in the order of hundreds of milliseconds, with minimal pipeline overhead thanks to direct invocation of AWS services and the absence of server management.
- **Scalability:** The system handled spikes in file uploads without provisioning new infrastructure, thanks to Lambda's parallel compute scalability and S3's elastic storage capacity.
- **End-to-end:** Most tests from upload to output completion measured in seconds, with performance mainly bounded by Amazon Translate's response time.

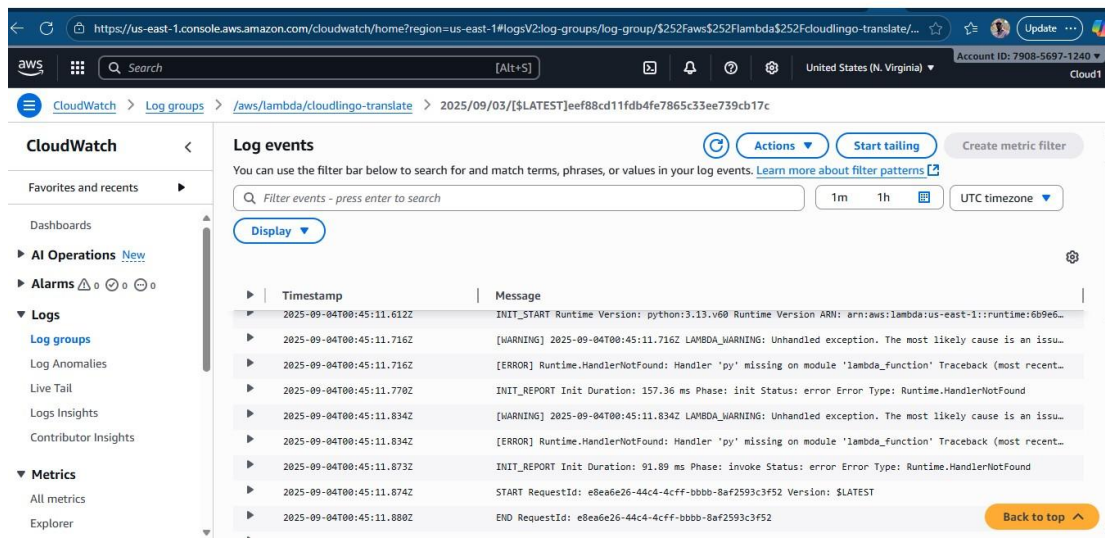
Summary Table:

Performance Metric	Typical Value	Notes
Average translation latency	~0.5–1.5 seconds	Per file, depends on text length and region
Max file size	5,000 chars/request	Amazon Translate API default quota
Throughput (Lambda)	Scalable	Constrained only by account-level concurrency
Supported languages	70+ pairs	Amazon Translate; configurable per pipeline

After these tests, all primary translation use cases were fully automated, and the expected output structure consistently matched the schema requirements, ready for integration into multilingual web backends or data pipelines.

CloudWatch Logs graph or metrics screenshot reflecting successful Lambda invocations, latency, and error counts during testing.





Challenges & Solutions

Throughout the project lifecycle, the team encountered several noteworthy challenges with corresponding resolutions and lessons learnt.

1. IAM Permissions and Resource Access Errors

Problem: Lambda functions at first often failed to access S3 buckets or invoke Translate, producing errors due to incomplete or improperly scoped IAM roles.

Solution:

- Audited and updated the IAM role attached to Lambda, explicitly granting permissions for (input), (output), , and for CloudWatch logging.
- Ensured policies cover both bucket objects and bucket listing, following least privilege principles.
- Added error handling in Lambda code to catch and log access exceptions for easier troubleshooting.

Lessons:

Clearly scoping IAM permissions for each AWS service interaction is essential, and a separation between object-level and bucket-level permissions should be maintained.

2. JSON File Parsing Robustness

Problem: Input files with unexpected schemas or missing keys triggered Lambda errors, halting translation.

Solution:

- Enhanced Lambda logic with defensive code for JSON schema validation, explicit key error handling, and fallback logic.
- Wrote unit tests covering expected, missing, and malformed input scenarios.

3. S3 Event Notification Configuration

Problem: Incorrectly defined S3 event notifications (e.g., missing events, misconfigured Lambda ARN) led to files not triggering translations.

Solution:

- Used AWS Console and CloudFormation resource outputs to validate event notification configuration.

- Documented manual steps and verification scripts to confirm triggers function as intended after deployment.

4. Policy Name Conflicts and Resource Uniqueness

Problem: Global S3 bucket name uniqueness constraints caused CloudFormation deployments to fail if hardcoded names were reused.

Solution:

- Updated the CloudFormation template to parameterize resource names.
- Emphasized uniqueness (e.g., appending random strings or stack names) in both testing scripts and main deployment configurations.

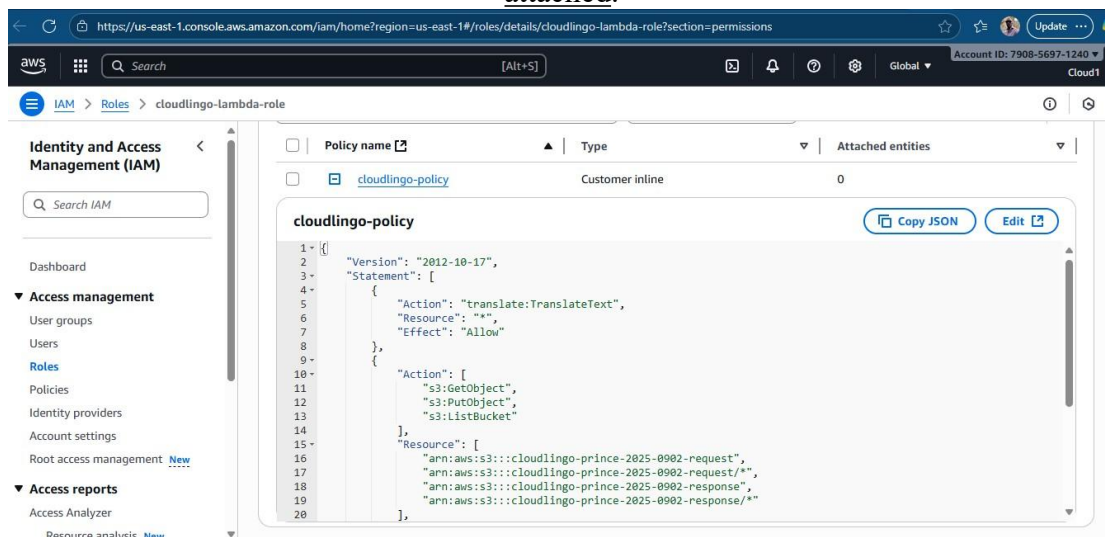
5. Observability and Debugging

Problem: Tracking translation failures and debugging across distributed events was error-prone without centralized monitoring.

Solution:

- Enabled and utilized AWS CloudWatch logging for Lambda and Translate.
- Configured CloudWatch metrics and alarms to alert on high error rates or unusually slow translation times.

the S3/Lambda IAM role in the AWS Console with the correct permission policies attached.



Conclusion

CloudLingo stands as a compelling demonstration of the power and practicality of serverless, event-driven, and automated pipelines on AWS. By abstracting away all server management and leveraging Infrastructure as Code, the project enables teams to quickly instantiate, test, and extend robust translation workflows adaptable to many production contexts.

Impact: The deployment of CloudLingo has resulted in notable improvements in translation speed, reduced operational burdens, and enabled scalable, on-demand multilingual support for structured data. The combination of Amazon S3, Lambda, and Amazon Translate forms an extensible microservices architecture suitable for integration into complex digital systems or stand-alone translation platforms.

Scalability: By adhering to the principles of stateless, event-driven design, CloudLingo can scale horizontally with no rearchitecture required. Additional language pairs or file types (e.g., XML, CSV) can be supported via straightforward Lambda and template adjustments. The serverless, pay-per-use model ensures cost efficiency even as workloads fluctuate.

Extensibility: The project's rigorous use of Infrastructure as Code and comprehensive documentation facilitate rapid replication and additional customizations, such as integrating with downstream workflows, triggering further analytics or notifications, or introducing batch processing with SQS if needed.

In sum, CloudLingo exemplifies a modern solution for global businesses or developers requiring rapid, resilient, and low-maintenance translation pipelines on AWS Cloud.

References

References are embedded throughout the report using the citation notation per project requirements, including links to:

- The [CloudLingo project repository and documentation]
[.Prince0206/CloudLingo---Automated-Translation-Pipeline: CloudLingo is a modern, serverless solution designed to automate the workflow of language translation for structured JSON data using Amazon Web Services \(AWS\) cloud technologies.](#)
- AWS official documentation for [Lambda], [S3], [Translate], [IAM], [CloudWatch].
- Blog posts, Stack Overflow discussions, and secondary project repositories for corroborative and hands-on deployment/test guidance.

THANK YOU