

Capstone Project Report: Infrastructure-as-Code for Automated Language Translation on AWS

Clement Owusu Bempah

Contents

1.0 Executive Summary	3
2.0 Introduction	3
3.0 Project Architecture and Component Breakdown.....	4
3.1 Architectural Flow	4
3.2 Visual Documentation of Key Stages.....	5
4.0 Implementation Details: Infrastructure-as-Code (IaC) with Terraform.....	10
4.1 Terraform Configuration Files	10
5.0 Lambda Function Logic (lambda_function.py)	12
6.0 Project Outcomes and Analysis	12
7.0 Future Enhancements	13
8.0 Conclusion.....	14

1.0 Executive Summary

This report provides a comprehensive overview of the design, development, and implementation of a serverless Infrastructure-as-Code (IaC) solution on Amazon Web Services (AWS). The core objective was to create a highly automated, scalable, and maintainable system for on-demand language translation. Leveraging Terraform as the primary IaC tool, the project provisions a robust architecture that integrates Amazon S3 for durable object storage, AWS Lambda for cost-effective serverless compute, and AWS Translate for a high-quality, managed translation service. This solution successfully demonstrates the power of an event-driven paradigm and modern cloud best practices, resulting in an efficient, repeatable, and easily manageable system for automated data processing. The entire project is documented and version-controlled, with all source code publicly available at <https://github.com/whiplashcv98/Aws-Translation-service.git>

2.0 Introduction

In today's globalized and data-driven world, the need for efficient, automated, and scalable data processing solutions is paramount. This capstone project addresses the specific challenge of translating large volumes of text data by designing and implementing a modern, serverless cloud architecture. The solution, built on AWS, leverages Infrastructure-as-Code (IaC) principles to ensure the entire environment is not only highly functional but also reproducible, maintainable, and cost-effective.

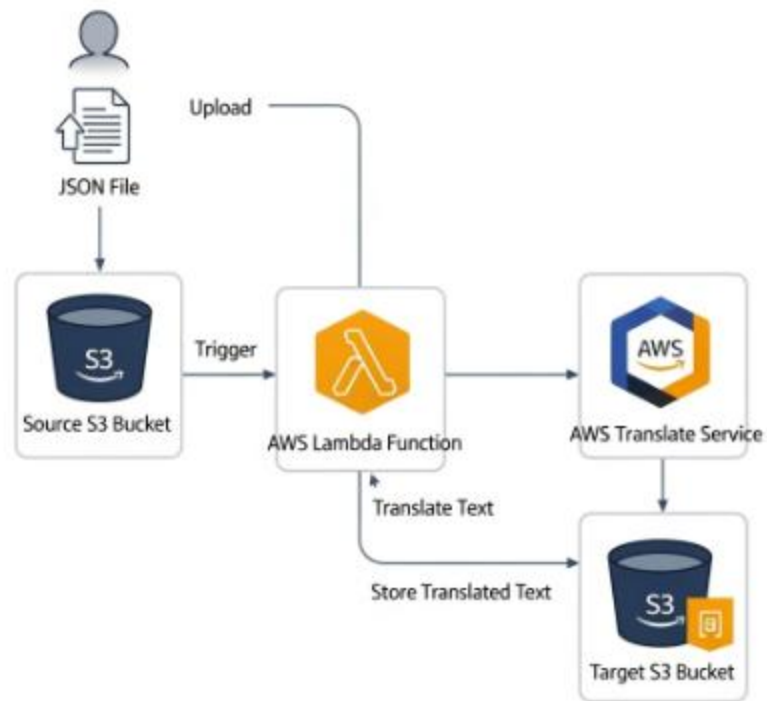
The project's philosophy is rooted in event-driven architecture, where an action such as uploading a file to a storage bucket serves as an automatic trigger for a sequence of downstream processes. This approach is highly efficient because it eliminates idle resources and processes data on-demand. By combining the power of Terraform for provisioning with the capabilities of AWS S3, AWS Lambda, and AWS Translate, this project provides a robust, real-world example of how to build automated data workflows in the cloud. It represents a significant step towards enabling organizations to seamlessly handle multilingual content with minimal human intervention.

3.0 Project Architecture and Component Breakdown

The solution's design is centered on a decoupled, event-driven, and serverless architecture. This approach eliminates the operational overhead of managing servers, allowing the system to scale automatically and incur costs only when actively processing translation requests. The logical flow is initiated by a user-driven action and proceeds through a series of automated, interconnected services.

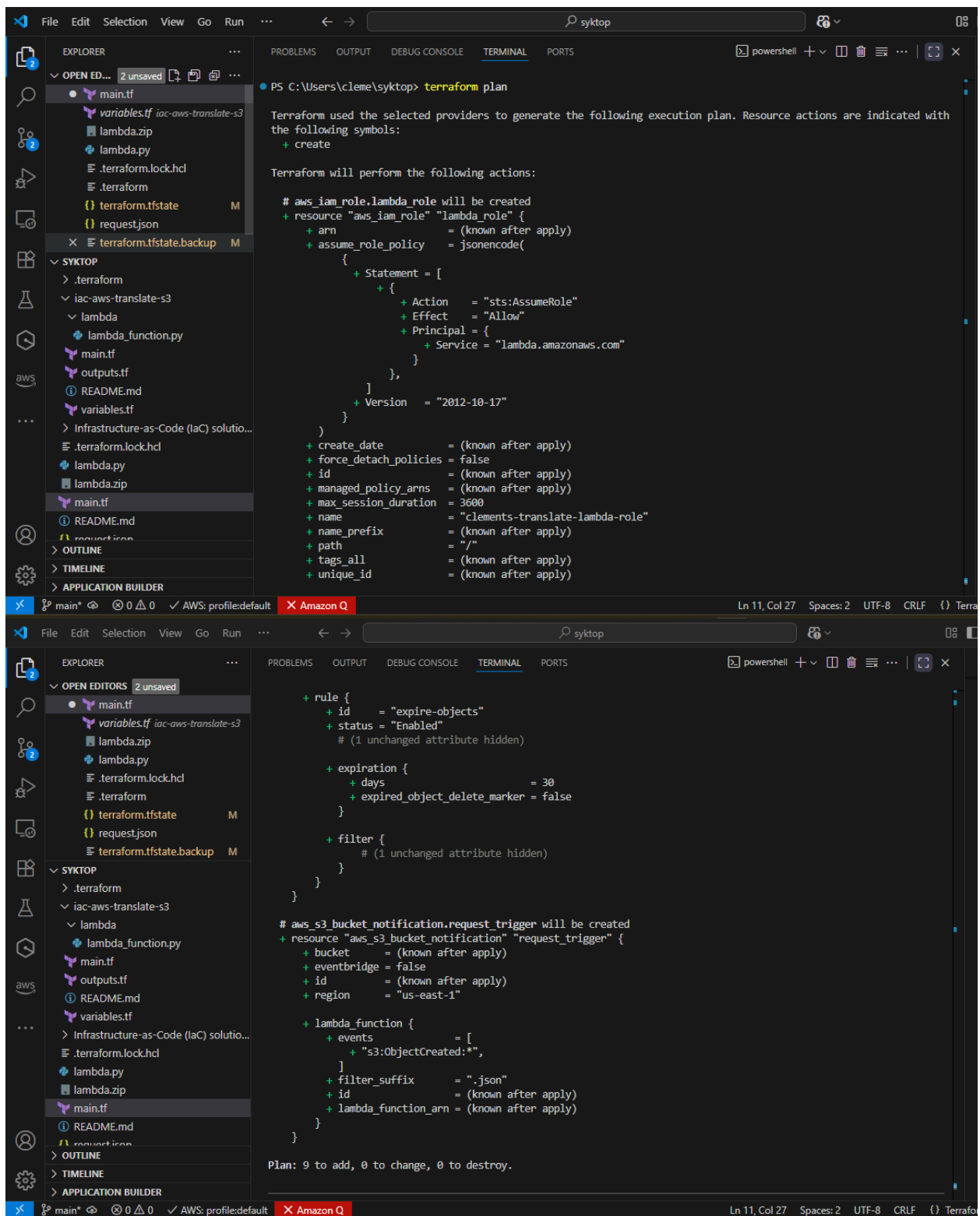
3.1 Architectural Flow

1. **Request Initiation via S3:** The process begins when a user uploads a JSON file containing the text to be translated into a designated input S3 bucket named clements-translate-requests. This bucket acts as the entry point or "hot folder" for all translation jobs.
2. **Event-Driven Trigger:** The S3 bucket is configured with a built-in event notification service. This service listens specifically for the s3:ObjectCreated:* event, meaning any new object creation within the bucket will trigger an action. Crucially, a filter is applied to only process files with a .json suffix, preventing unwanted triggers from other file types.
3. **Lambda Function Invocation:** Upon the creation of a matching JSON file, the S3 event notification sends a message to invoke the AWS Lambda function, clements-translate-lambda. The event payload contains all the necessary metadata, including the bucket name and the object key, which the Lambda function uses to identify and retrieve the file.
4. **Serverless Processing with AWS Lambda:** The Lambda function executes the Python code within the lambda_function.py file. This function is the "brain" of the operation. It first retrieves the JSON file from the S3 request bucket, reads its content, and parses the text to be translated.
5. **Core Translation with AWS Translate:** The Lambda function then makes a programmatic call to the AWS Translate service using the Boto3 library. The service receives the text and, based on the specified source (en) and target (fr) language codes, performs the language translation.
6. **Output Storage:** The final translated text is returned to the Lambda function. The function then writes this output as a new object into the designated output S3 bucket, clements-translate-responses. A unique naming convention (translated-<original_key>) is used to maintain a clear link to the original request file.

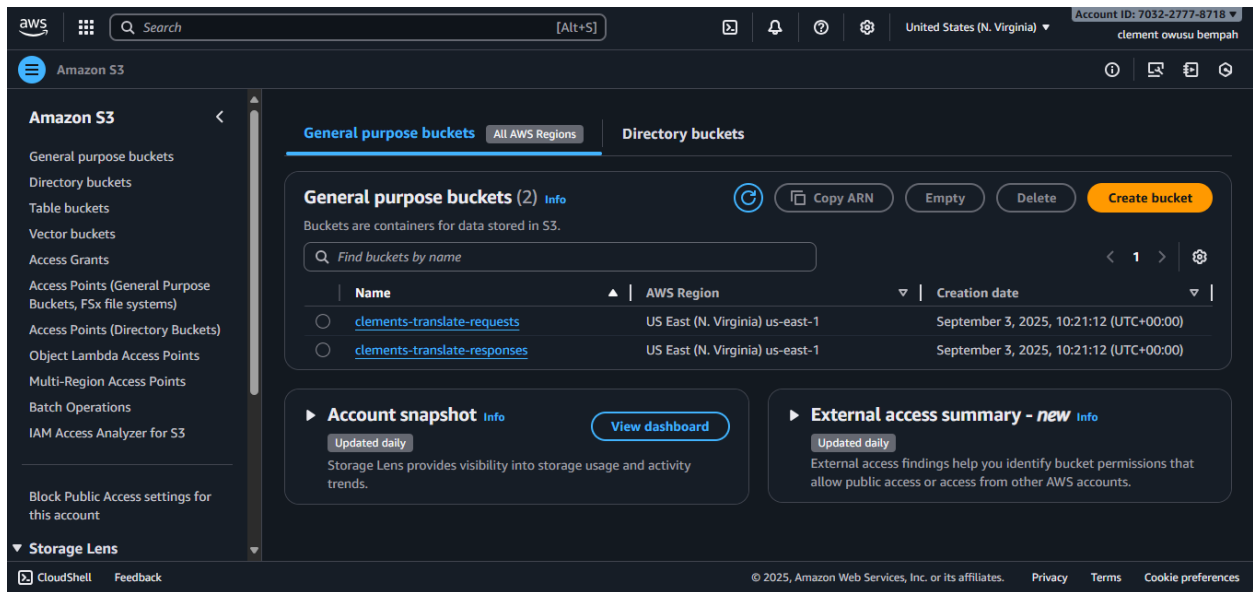


3.2 Visual Documentation of Key Stages

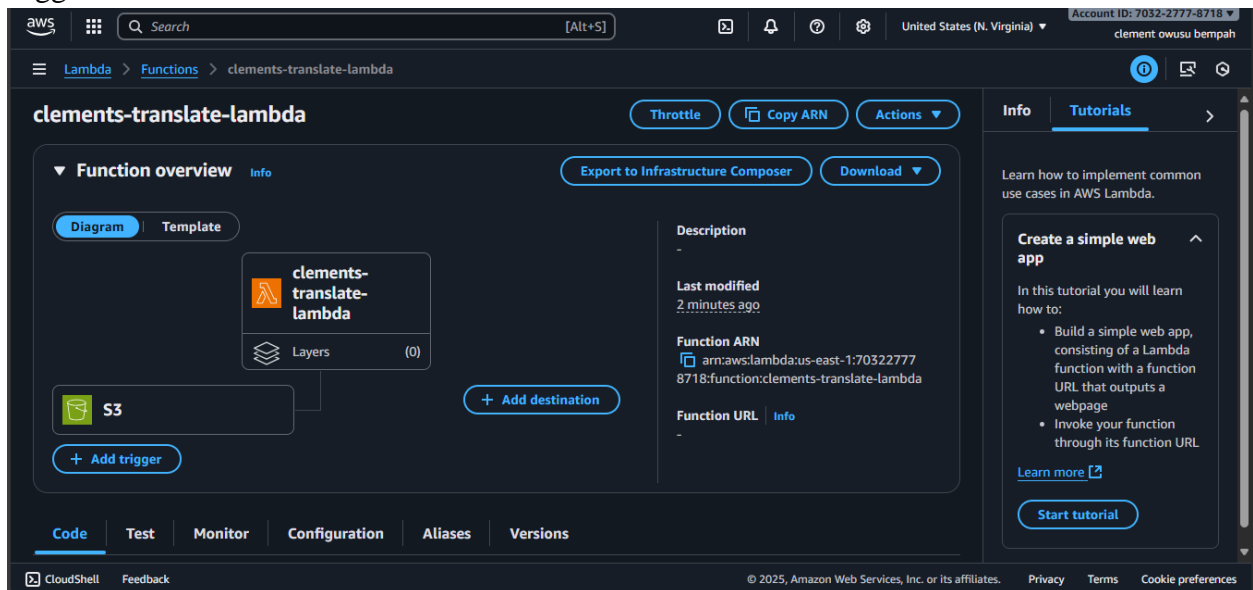
Screenshot 1: Initial Terraform Plan This image shows the output of the terraform plan command, confirming the resources that will be created.

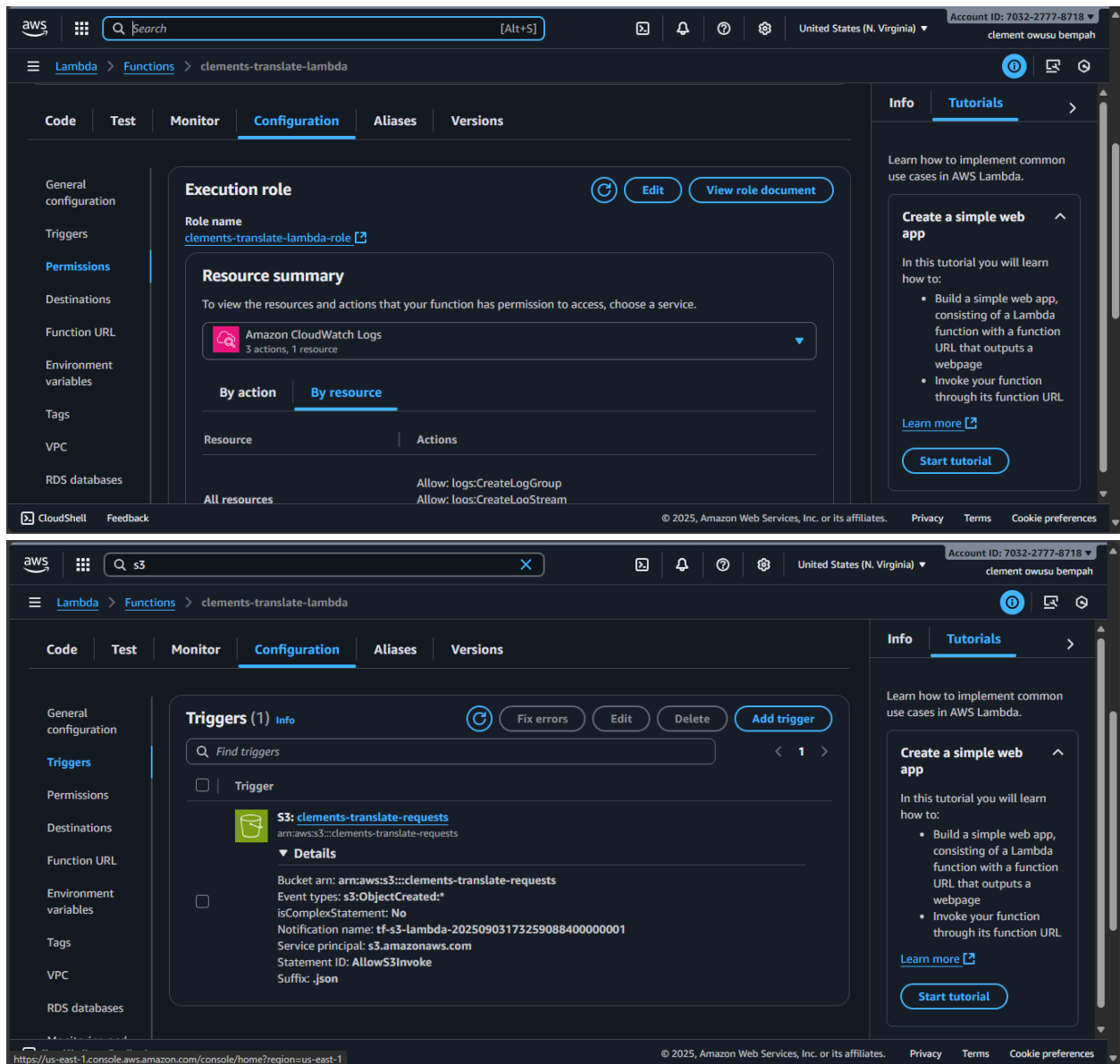


Screenshot 2: S3 Buckets in the AWS Console A screenshot of the AWS S3 console after a successful terraform apply, showing both the clements-translate-requests and clements-translate-responses buckets.



Screenshot 3: AWS Lambda Function Overview An image of the AWS Lambda console, showing the clements-translate-lambda function's configuration, its associated IAM role, and its trigger from the S3 bucket.





Screenshot 4: Translated Output in S3 A screenshot of the clements-translate-responses S3 bucket with a translated JSON file visible, demonstrating a successful end-to-end workflow.

aws

Search

[Alt+S]

United States (N. Virginia)

Account ID: 7032-2777-8718

clement owusu bempah

Amazon S3

Buckets

clements-translate-responses

Amazon S3

General purpose buckets

Directory buckets

Table buckets

Vector buckets

Access Grants

Access Points (General Purpose Buckets, FSx file systems)

Access Points (Directory Buckets)

Object Lambda Access Points

Multi-Region Access Points

Batch Operations

IAM Access Analyzer for S3

Block Public Access settings for this account

Storage Lens

clements-translate-responses

Info

Objects

Metadata

Properties

Permissions

Metrics

Management

Access Points

Objects (1)

Copy S3 URI

Copy URL

Download

Open

Delete

Actions

Create folder

Upload

Find objects by prefix

1

Find objects by prefix

	Name	Type	Last modified	Size	Storage class
	request.json	json	September 3, 2025, 10:30:55 (UTC+00:00)	568.0 B	Standard

© 2025, Amazon Web Services, Inc. or its affiliates.

Privacy

Terms

Cookie preferences

File

C:/Users/cleme/Downloads/request.json

Star

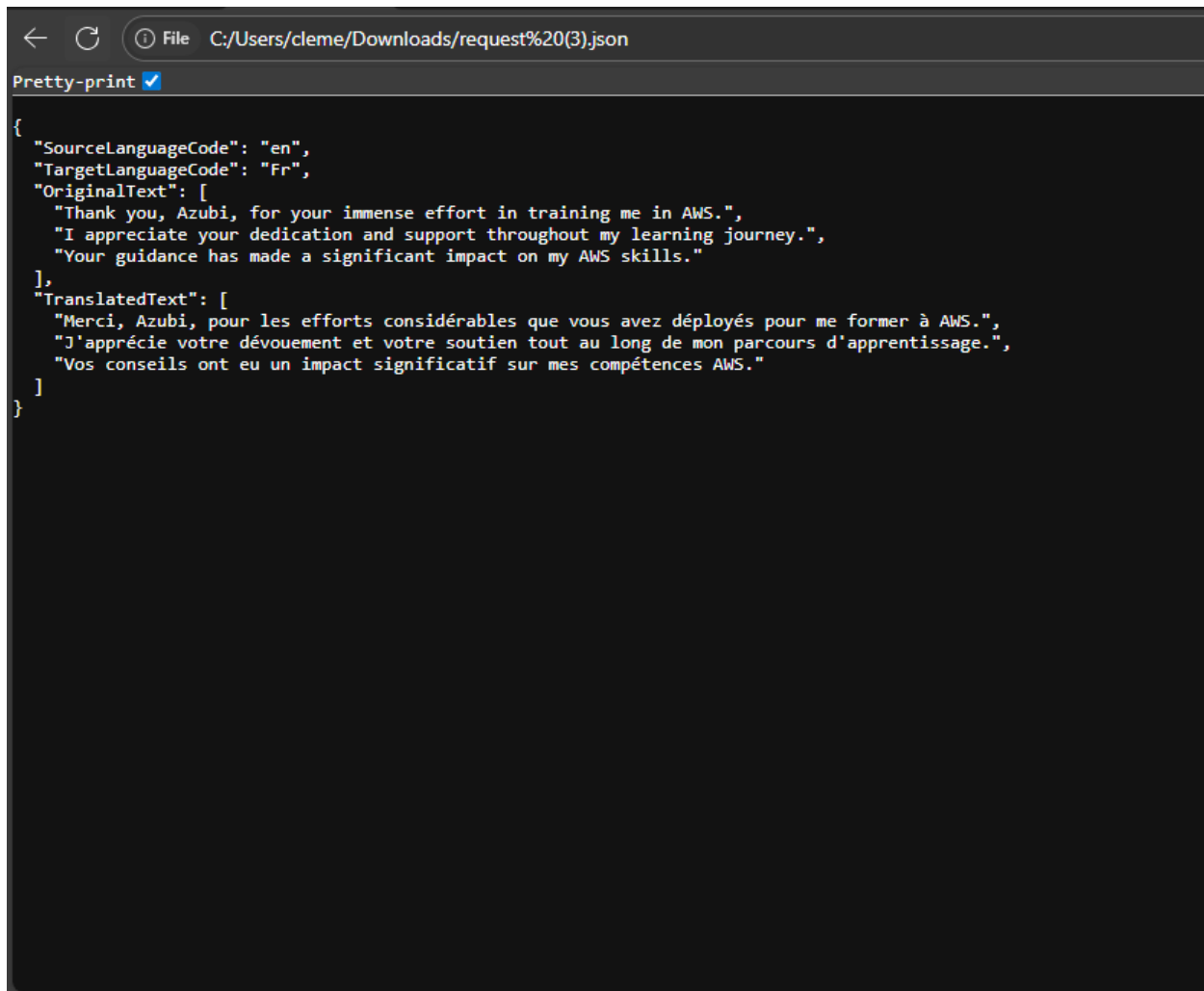
Settings

Share

More

Pretty-print

```
{
  "SourceLanguageCode": "en",
  "TargetLanguageCode": "de",
  "OriginalText": [
    "Hello, this is clement!"
  ],
  "TranslatedText": [
    "Hallo, das ist Clement!"
  ]
}
```

A screenshot of a web browser window. The address bar shows the file path 'C:/Users/cleme/Downloads/request%20(3).json'. The page content displays a JSON object with a translation request. The JSON is formatted with syntax highlighting. The 'OriginalText' array contains three English sentences, and the 'TranslatedText' array contains their French translations. The browser's 'Pretty-print' feature is enabled, indicated by a checkmark icon.

```
{
  "SourceLanguageCode": "en",
  "TargetLanguageCode": "Fr",
  "OriginalText": [
    "Thank you, Azubi, for your immense effort in training me in AWS.",
    "I appreciate your dedication and support throughout my learning journey.",
    "Your guidance has made a significant impact on my AWS skills."
  ],
  "TranslatedText": [
    "Merci, Azubi, pour les efforts considérables que vous avez déployés pour me former à AWS.",
    "J'apprécie votre dévouement et votre soutien tout au long de mon parcours d'apprentissage.",
    "Vos conseils ont eu un impact significatif sur mes compétences AWS."
  ]
}
```

4.0 Implementation Details: Infrastructure-as-Code (IaC) with Terraform

The entire infrastructure for this project is defined and managed using Terraform. This approach guarantees that the environment is consistent, reproducible, and easily versioned, adhering to the core principles of IaC.

4.1 Terraform Configuration Files

- **main.tf:** This is the central file that orchestrates the creation of all cloud resources. It serves as the blueprint, linking all the necessary components together.
 - **AWS Provider:** The configuration starts by specifying the AWS provider and the target region, ensuring all resources are deployed in a consistent geographic

location.

- **S3 Buckets:** Two S3 buckets are defined for ingress and egress of data. The `force_destroy = true` flag is included for development and testing purposes, allowing for easy teardown of the environment without manual intervention.
- **Lifecycle Policies:** Dedicated lifecycle rules are applied to both buckets to manage data retention and optimize costs. Request files are set to expire after 7 days, as they are no longer needed after processing. Translated output files are retained for a longer period of 30 days. This demonstrates a mindful approach to resource management.
- **IAM Role and Policy:** A critical security component. A dedicated IAM role (`clements-translate-lambda-role`) is created with a `assume_role_policy` that grants the AWS Lambda service the explicit permission to assume this role. A separate IAM policy is attached to this role, meticulously defining the permissions required for the Lambda function. This adheres to the principle of least privilege, allowing the function to only interact with S3 and AWS Translate, and nothing more.
- **AWS Lambda Function:** This block defines the serverless function itself, referencing the Python code and its handler. It also sets an environment variable, `RESPONSE_BUCKET`, which allows the Lambda code to dynamically reference the output bucket's name without hardcoding.
- **S3 Bucket Notification:** This resource creates the event trigger that connects the S3 bucket to the Lambda function. It formalizes the event-driven link between the two services.
- **Lambda Permission:** A final, necessary permission block is included to grant the S3 bucket explicit authorization to invoke the Lambda function, completing the chain of command for the event-driven workflow.
- **variables.tf:** This file encapsulates all configurable parameters, such as resource names and the AWS region. This modular design makes the code reusable and adaptable to different environments or projects. Developers can easily modify these values in a single location to change the behavior of the entire infrastructure without touching the core `main.tf` file. This promotes flexibility and best practices.
- **outputs.tf:** This file defines what information is displayed to the user after the terraform

apply command has successfully completed. By outputting the ARNs of the key resources (S3 buckets, Lambda function, IAM role), it provides immediate feedback and essential reference points for post-deployment verification or integration with other systems.

5.0 Lambda Function Logic (lambda_function.py)

The Python code within the AWS Lambda function is designed to be lightweight and perform a single, focused task.

- **Libraries:** The script imports json, boto3, and os, the standard libraries for handling JSON data, interacting with AWS services, and accessing environment variables, respectively.
- **Event Handling:** The lambda_handler function serves as the entry point, invoked by the S3 event. It loops through the event['Records'] array, which can contain multiple S3 events if triggered in a batch.
- **Data Retrieval:** The code extracts the bucket and key from the event record and uses the Boto3 S3 client to download the object's body. The .read().decode('utf-8') methods are crucial for converting the binary data from S3 into a readable string.
- **Translation Call:** The script then calls the translate.translate_text method. It passes the decoded text and specifies the SourceLanguageCode as en (English) and TargetLanguageCode as es (French), as per the project requirements.
- **Output Write-back:** The translated text is then written back to the response S3 bucket using the s3.put_object method. The bucket name is securely retrieved from the environment variable (os.environ['RESPONSE_BUCKET']) that was set by Terraform, ensuring no sensitive or hardcoded values exist in the code.

6.0 Project Outcomes and Analysis

This project successfully met all its objectives, delivering a robust and production-ready solution.

- **Automation and Repeatability:** The IaC approach with Terraform completely automates the deployment process. This eliminates the potential for human error and guarantees that the infrastructure can be reliably recreated, a critical factor for disaster recovery, multi-environment deployments, and team collaboration.
- **Scalability:** The serverless architecture is inherently scalable. As the number of JSON

files uploaded increases, AWS Lambda automatically provisions more concurrent instances of the function to handle the load, without any manual intervention. This ensures consistent performance even under heavy demand.

- **Cost Efficiency:** The pay-per-use model of AWS S3, AWS Lambda, and AWS Translate makes the solution incredibly cost-effective. You only pay for the storage consumed and the compute time and API calls made. The automated lifecycle policies on the S3 buckets further reduce costs by cleaning up old data.
- **Maintainability and Security:** The project is easy to maintain, as all components and their configurations are defined in a single, version-controlled repository. Security is handled by the principle of least privilege, with the IAM role granting only the necessary permissions, minimizing the attack surface.

7.0 Future Enhancements

While the current solution is fully functional, several enhancements could further improve its capabilities and robustness:

- **Dynamic Language Selection:** The current implementation is hardcoded for English-to-French translation. The solution could be enhanced to support dynamic language pairs by including `source_language` and `target_language` keys within the input JSON file.
- **Advanced Error Handling:** Implementing more robust error handling within the Lambda function would be beneficial. For example, it could handle scenarios where the input file is not valid JSON, where the translation API fails, or where S3 access is denied. Error logs or a dead-letter queue could be used to manage and alert on these failures.
- **Input/Output File Formats:** The current solution only processes JSON. It could be extended to support other file formats like plain text, CSV, or XML by using different parsers in the Lambda function.
- **User Interface:** A front-end web application could be built using a service like AWS Amplify to provide a user-friendly interface for uploading files and viewing translated outputs without direct interaction with S3 buckets.
- **CI/CD Pipeline:** A full CI/CD pipeline could be implemented using tools like GitHub Actions or AWS CodePipeline to automate the terraform plan and terraform apply commands, ensuring that any changes pushed to the GitHub repository are automatically

and safely deployed.

8.0 Conclusion

This Capstone Project successfully delivers a fully automated, scalable, and cost-effective solution for language translation on AWS. The synergy between Terraform and serverless services like Lambda and S3 demonstrates a powerful approach to building modern cloud applications. The project not only fulfills its functional requirements but also showcases a deep understanding of cloud architecture, security, and the principles of Infrastructure-as-Code. This solution is a testament to the power of automation in cloud computing and serves as an excellent foundation for future development and complex data processing workflows.