

Project Title: Professional Text Enhancer using Amazon Bedrock and Claude

Overview:

This project is a serverless AI powered text rewriting tool that takes the user content and returns a professional version using Claude model through Amazon bedrock. The solution can be integrated into any productivity or enterprise writing assistant.

Architecture:

1. Postman – Send a POST request with raw text
2. AWS Lambda – Python code – event driven - API gateway
3. AWS Claude model to rewrite the input text
4. S3 – Stores input and output logs in JSON format for auditing
5. IAM – Grant permissions to invoke Bedrock and write to S3

Result:

1. Accepts Plain English sentences
2. Sends to Claude to re-write it professionally
3. Returns the response handling errors gracefully and missing input
4. Original content and re-written content with timestamp to S3

Additional Notes

1. Use region us-east-1
2. Understand the Quotas Amazon Bedrock

Step 1

1. Go to IAM Roles
2. Create a Role
3. Add the Following Permissions
 - a. AmazonBedRockFullAccess
 - b. AmazonS3FullAccess
 - c. CloudWatchLogsFullAccess
4. Provide a Name – LambdaBedRockTextEnhancer
5. Click on Create Role

Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.

Maximum 64 characters. Use alphanumeric and "-", "@", "." characters.

Description
Add a short explanation for this role.

Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: ", @, /, [(), :%, "*", "=", {}, ^, ~, |, \, &, #, \$, %, ", ', <, >,

```

1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "allow",
6       "Action": [
7         "sts:AssumeRole"
8       ],
9       "Principal": {
10        "Service": [
11          "lambda.amazonaws.com"
12        ]
13      }
14    ]
15  }
16 }
```

Step 1: Select trusted entities

Trust policy

```

1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "allow",
6       "Action": [
7         "sts:AssumeRole"
8       ],
9       "Principal": {
10        "Service": [
11          "lambda.amazonaws.com"
12        ]
13      }
14    ]
15  }
16 }
```

Step 2: Add permissions

Permissions policy summary

Policy name ▼	Type	Attached as
AmazonBedrockFullAccess	AWS managed	Permissions policy
AmazonS3FullAccess	AWS managed	Permissions policy
CloudWatchAgentFullAccess	AWS managed	Permissions policy

Step 2

1. Create a bucket
2. Select General Purpose
3. Provide a name
4. Click on Create Bucket

General configuration

AWS Region

US East (N. Virginia) us-east-1

Bucket type

General purpose

Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

Directory

Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name

professional-text-logs

Bucket names must be 3 to 63 characters and unique within the global namespace. Bucket names must also begin and end with a letter or number. Valid characters are a-z, 0-9, periods (.), and hyphens (-). [Learn More](#)

Copy settings from existing bucket - optional

Only the bucket settings in the following configuration are copied.

Choose bucket

Format: s3://bucket/prefix

Object Ownership

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

ACLs disabled (recommended)

All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

ACLs enabled

Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Object Ownership

Bucket owner enforced

Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

☒ Block all public access

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

☐ Block public access to buckets and objects granted through new access control lists (ACLs)

S3 will ignore public permissions assigned to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

☐ Block public access to buckets and objects granted through any access control lists (ACLs)

S3 will ignore all ACLs that grant public access to buckets and objects.

☐ Block public access to buckets and objects granted through new public bucket or access point policies

S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

☐ Block public and cross-account access to buckets and objects through any public bucket or access point policies

S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

Bucket Versioning

☒ Enable

☐ Disable

Tags - optional (0)

You can use bucket tags to track storage costs and organize buckets. [Learn more](#)

No tags associated with this bucket.

Add tag

Default encryption

Server-side encryption is automatically applied to new objects stored in this bucket.

Encryption type

☒ Server-side encryption with Amazon S3 managed keys (SSE-S3)

☐ Server-side encryption with AWS Key Management Service keys (SSE-KMS)

☐ Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)

Secure your objects with two separate layers of encryption. For details on pricing, see DSSE-KMS pricing on the Storage tab of the [Amazon S3 pricing page](#).

Bucket Key

Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. [Learn more](#)

☐ Disable

☒ Enable

► Advanced settings

After creating the bucket, you can upload files and folders to the bucket, and configure additional bucket settings.

Cancel

Create bucket

Step 3

1. Create a lambda function
2. Select Author From Scratch
3. Provide a Function Name – Call it text-enhancer
4. Select runtime as Python
5. Select the IAM role
6. Click on Create Function
7. Click on the Function
8. Navigate to Configuration
9. Click on Environment Variables
10. Add the Key LOG_BUCKET
11. Add the value S3 Bucket Name
12. The lambda function will automatically resolve to the S3 bucket time during runtime
13. Deploy the code

Create function [info](#)

Choose one of the following options to create your function.

☒ **Author from scratch**
Start with a simple Hello World example.

☐ **Use a blueprint**
Build a Lambda application from sample code and configuration presets for common use cases.

☐ **Container image**
Select a container image to deploy for your function.

Basic information

Function name
text_enhancer

Runtime [info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.13

Architecture [info](#)
Choose the instruction set architecture you want for your function code.
☐ arm64
☒ x86_64

Permissions [info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

Change default execution role

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

☐ Create a new role with basic Lambda permissions

☒ Use an existing role

☐ Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
LambdaDefaultRoleTextEnhancerRole

[View the LambdaDefaultRoleTextEnhancerRole role](#) on the IAM console.

Additional Configurations
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

Enable code signing [info](#)
☐ Use code signing configurations to ensure that the code has been signed by an approved source and has not been altered since signing.

Enable encryption with an AWS KMS customer managed key [info](#)
☐ By default, Lambda encrypts the zip file archive using an AWS managed key.

Enable function URL [info](#)
☐ Use function URLs to assign HTTP(S) endpoints to your Lambda functions.

Enable tags [info](#)
☐ A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources, track your AWS costs, and enforce attribute-based access control.

Enable VPC [info](#)
☐ Connect your function to a VPC to access private resources during invocations.

[Cancel](#) [Create Function](#)

Edit environment variables

Environment variables
You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#)

Key	Value	
LOG_BUCKET	professional-text-logs	<button>Remove</button>

Add environment variable

► Encryption configuration

Cancel Save

Step 4

1. Go to API Gateway
2. Click on HTTP API
3. Click on Create
4. Set Route as Post
5. Ensure the API name is consistent with the project to avoid confusion will creating routes or configuration
6. Click on Add Integrations
7. Select the region
8. Select the Lambda Function
9. Click on Next
10. Select Method as POST
11. Click on Integration Target – Select from Drop Down – The lambda function name
12. Provide the resource path as /enhance
13. Click on Next
14. Deploy the API

Configure API

API details
API name
An HTTP API must have a name. The name is a non-unique value you use to identify and organize your APIs. To programmatically refer to this API, use the API ID that API Gateway generates for you.

IP address type [Info](#)
Select the type of IP addresses that can invoke the default endpoint for your API. You don't need to redeploy your API for the update to take effect.
☒ IPv4
Includes only IPv4 addresses.
☐ Dualstack
Includes IPv4 and IPv6 addresses.

Integrations (1) [Info](#)
Specify the backend services that your API will communicate with. These are called integrations. For a Lambda integration, API Gateway invokes the Lambda function and responds with the response from the function. For an HTTP integration, API Gateway sends the request to the URL that you specify and returns the response from the URL.

Remove

AWS Region **Lambda function** **Version** [Learn more](#)

Add integration

Cancel Review and create Next

Configure routes - optional

Configure routes [info](#)

API Gateway uses routes to expose integrations to consumers of your API. Routes for HTTP APIs consist of two parts: an HTTP method and a resource path (e.g., GET /pets). You can define specific HTTP methods for your integration (GET, POST, PUT, PATCH, HEAD, OPTIONS, and DELETE) or use the ANY method to match all methods that you haven't defined on a given resource.

Method

POST

Resource path

/enhance

→

Integration target

text_enhancer

Remove

Add route

Cancel Review and create Previous Next

Configure stages [info](#)

Stages are independently configurable environments that your API can be deployed to. You must deploy to a stage for API configuration changes to take effect, unless that stage is configured to autodeploy. By default, all HTTP APIs created through the console have a default stage named \$default. All changes that you make to your API are autodeployed to that stage. You can add stages that represent environments such as development or production.

Stage name

prod

Auto-deploy

Remove

Add stage

Cancel Previous Next

Review and create

API name and integrations

API name

enhance

IP address type

IPv4

Integrations

• text_enhancer (Lambda)

Edit

Routes

Routes

• POST /enhance → text_enhancer (Lambda)

Edit

Stages

Stages

• prod (Auto-deploy: disabled)

Edit

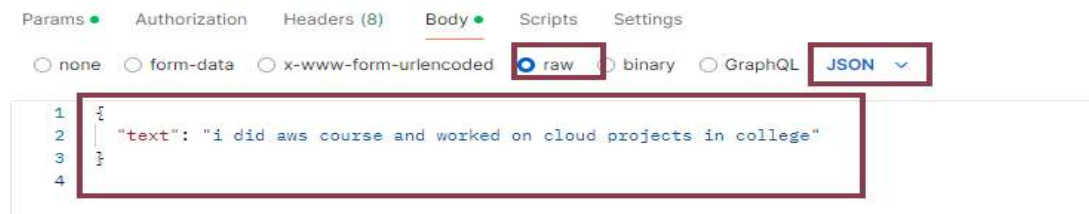
Cancel Previous Create

Step 5

1. Copy the Invoke URL from the Deploy > Stage
2. Open postman
3. This is POC and will test using POSTMAN
4. Go to the section Body
5. Select Raw
6. Select JSON
7. Enter the content in the body as shown
8. You should get a response
9. If you get an internal server error, then ensure the Amazon Bedrock model is enabled
10. Note down the model ID and that ID will be used in the lambda function code



Sample JSON OUTPUT



Enhanced Text from Amazon Bedrock

https://lurt9s7n26.execute-api.us-east-1.amazonaws.com/prod/enhance

POSThttps://lurt9s7n26.execute-api.us-east-1.amazonaws.com/prod/enhance

ParamsAuthorizationHeaders (8)BodyScriptsSettings

noneform-datax-www-form-urlencodedrawbinaryGraphQLJSON

1{
2 "text": "i want to finish the work on time and get appreciated"
3}
4

BodyCookiesHeaders (5)Test Results

JSONPreviewVisualize

1{
2 "original": "i want to finish the work on time and get appreciated"
3 "enhanced": "Here is a rewrite of that text in more professional English:\n\nI aim to complete the work by the deadline and deliver excellent results that merit recognition.\n"
4}

S3 – Log

504f9ead-2b38-48c3-acdf-a372d67dfe1.json X

504f9ead-2b38-48c3-acdf-a372d67dfe1.json > ...

1 {"timestamp": "2025-05-28 05:18:33.825448", "input": "i did aws course and worked on cloud projects in college", "output": "Here is one way to rewrite the text in more professional English:\n\nDuring college, I completed coursework in Amazon Web Services (AWS) and worked on cloud computing projects.\""}

504f9ead-2b38-48c3-acdf-a372d67dfe1.json

text-enhance.py x

C:\> Users > svrar > OneDrive > 1.Study_Plan > 6.Intellipat > Lab_Assignment > 8.Other_Labs > Own_Projects > Project_10 > text-enhance.py

```
1 import boto3
2 import json
3 import uuid
4 import datetime
5 import os
6
7 bedrock = boto3.client("bedrock-runtime", region_name="us-east-1") # Use correct region
8 s3 = boto3.client("s3")
9 BUCKET = "professional-text-logs" # Replace with your actual bucket name
```

```
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33 body=json.dumps(payload),
34     modelId="anthropic.claude-v2:1",
35     accept="application/json",
36     contentType="application/json"
37 )
38
39 response_body = json.loads(response['body'].read())
40 enhanced_text = response_body.get("completion", "").strip()
41
42 log_entry = f
```