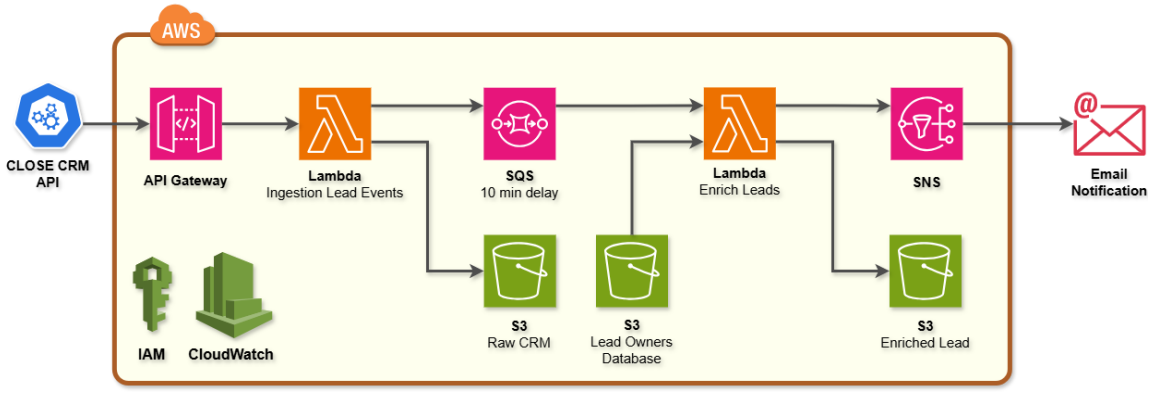
Project 5: Real-Time CRM Lead Processing and Notification System

Overview

This project implements a real-time, event-driven data pipeline to capture newly created leads from Close CRM using webhooks and process them through a fully serverless AWS architecture.

In this porject, the events are posted in HTTP format from Close CRM Webhook to our URL inside AWS, using API Gateway and Lambda function. After getting events, they are stored in S3 bucket in the raw format. After running 10 minutes delay using SQS, those lead events are merged with other lead-owner database, which are available in another S3 bucket, and then after enriching the data, we got some Email notification about the name of leads and name of lead-owners via SNS notificaiton system.



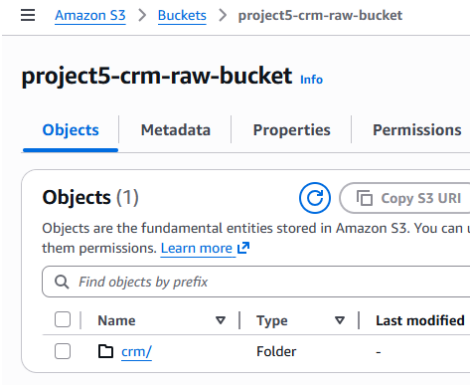
Step 1 – Data Ingestion

1-1- Objective

The objective of this step is to ingest raw lead event files from Close CRM API into an Amazon S3 bucket, named **project5-crm-raw-bucket**, in the JSON format and in the realtime format. To comply with the project requirement that allows me to use Python for data ingestion, I will implement **AWS Lambda** for data ingestion, via **AWS API Gateway**. This approach ensures that the ingestion process is both scalable and aligned with other AWS services.

1-2- Landing Storage (Bronze Layer)

I used Amazon S3 service and create a bucet with name of **s3://project5-crm-raw-bucket/**. After that, I created a folder inside that bucket with the name of **crm** folder for raw CRM lead event files ingestion. This is a sample of these files:
crm_event_lead_68DAVGDg87oy3dsaBrIIi13PGsYYaH3myWKNndeuvkZ.json

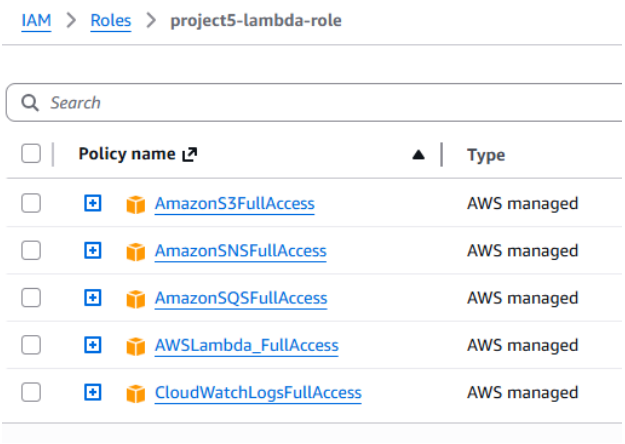


1-3- IAM Role

Before starting data ingestion using Lambda, I configured an IAM role to enable Lambda to securely access the required AWS services during job execution. The IAM role was named **project5-lambda-role** and tagged with **project: 5**, and it was attached to the AWS Lambda functions to ensure secure and authorized access during execution.

The following permissions were configured:

- **AmazonS3FullAccess:** Granted to allow AWS Glue to read from and write to the Bronze, Silver, and Gold S3 buckets.
- **AmazonSNSFullAccess:** Granted to enable Lambda functions to publish notification messages to the SNS topic for email alerts upon successful lead enrichment.
- **AmazonSQSFullAccess:** Granted to allow Lambda functions to send and receive messages from the SQS queue used for delayed and decoupled CRM event processing.
- **AmazonLambda_FullAccess:** Granted to allow creation, execution, and management of Lambda functions involved in webhook ingestion and lead enrichment workflows.
- **CloudWatchLogsFullAccess:** Enabled logging and monitoring of Glue job execution in Amazon CloudWatch.



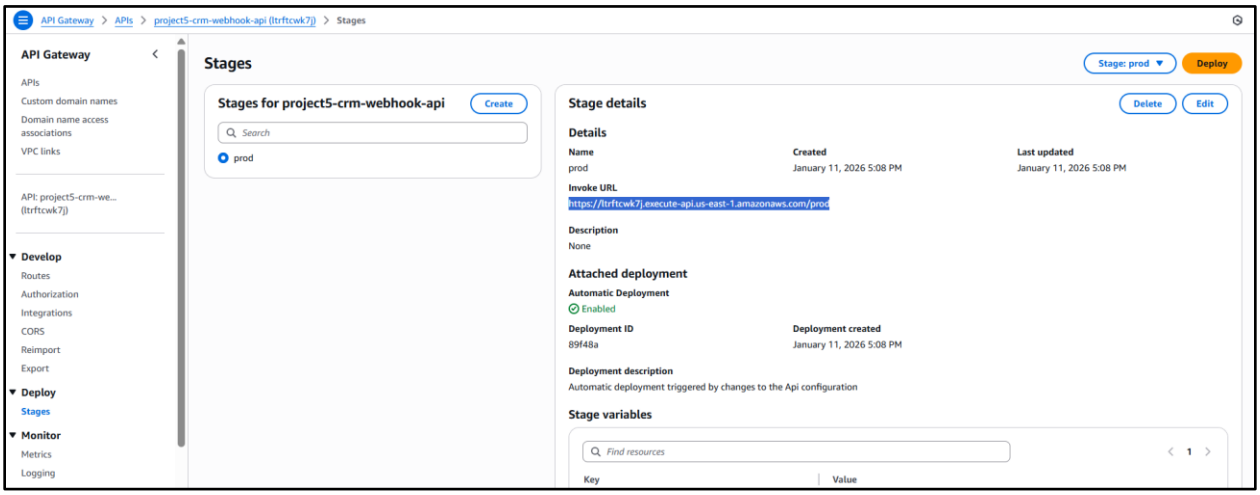
1-4- API Gateway

To get events from Close CRM in the realtime format, I created an API Gateway, named **project5-crm-webhook-api**, to fetch data from CRM Webhook. It will post events to my URL, then I created an Url stage in the AWS using API Gateway.

After creating the API Gateway successfully, I got my URL as:

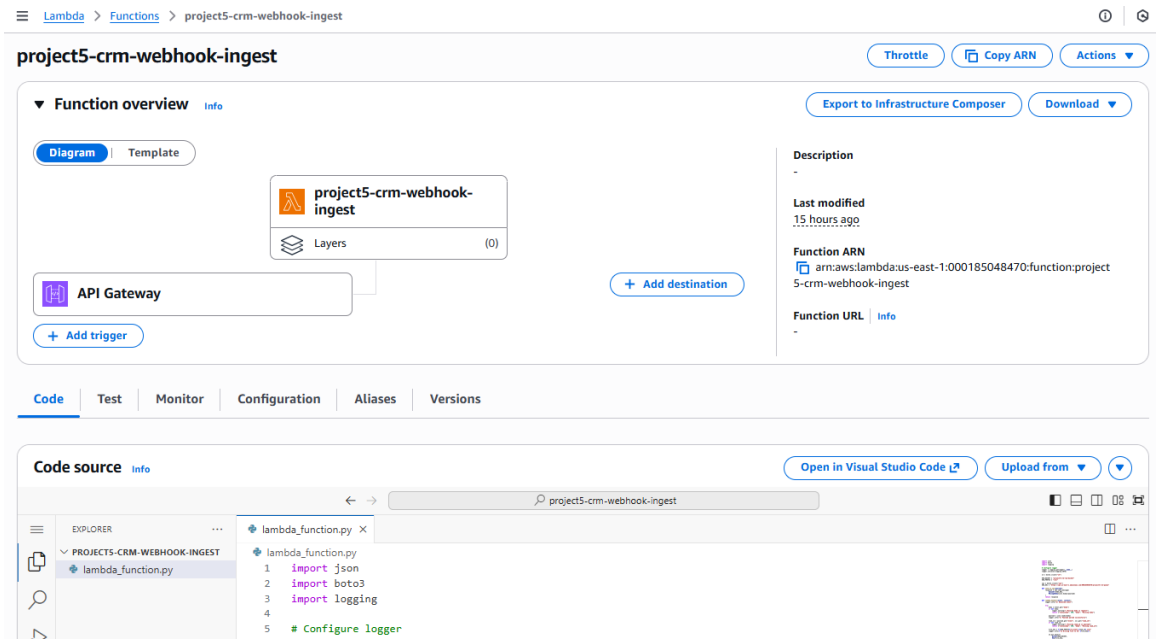
https://ltrftcwk7j.execute-api.us-east-1.amazonaws.com/prod/crm

I used this URL in CLOSE CRM Webhook configuration.



1-5- Lambda function for data ingestion

After securing the connection between CLOSE CRM API and AWS, now I am going to create the first Lambda function, named **project5-crm-webhook-ingest**, to perform raw data ingest from Close CRM Webhook API into S3 Bronze bucket. The script file name in the Python language is **project5-crm-webhook-ingest.py**.



Step 2 – Data Enrighment

2-1- Objective

The objective of this step is creating 10 minutes delay, then merging ingested raw lead event files with other extracted lead-owners' information from Close CRM. I will put the enriched data in another Amazon S3 bucket, named **project5-crm-enriched-bucket**, in the JSON format but after 10 minutes dalay. I will implement **AWS Lambda** for data enrichment, and orchestration the project using **Amazon SNS**.

2-2- Lead-owner Storage (Silver Layer)

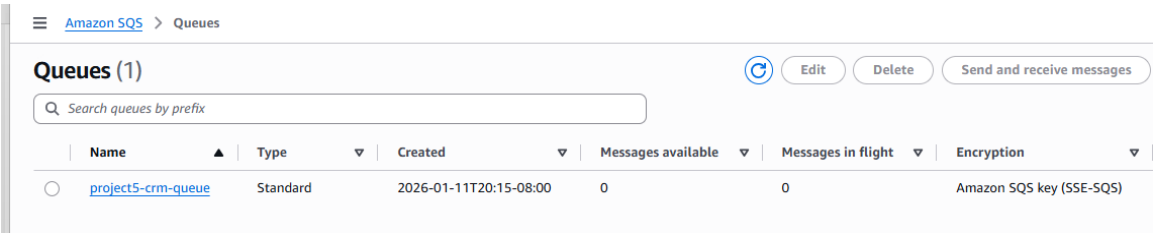
I used Amazon S3 service and create a bucet with name of **s3://project5-crm-enriched-bucket/**. After that, I created a folder inside that bucket with the name of **leads/** folder for enriched CRM lead event pluse lead-owner information files. This is a sample of these files: crm_enriched_lead_68DAVGDg87oy3dsaBrIli13PGsYYaH3myWKNndeuvkZ.json

I use the following public S3 bucket to perform a lookup and assign lead owner details to each newly created lead that was stored in S3 from the CRM webhook event. This is publickly available S3 bucket.

```
bucket_name : dea-lead-owner
file_name = f"{lead_id}.json"
public_url = f"https://{bucket_name}.s3.us-east-1.amazonaws.com/{file_name}"
```

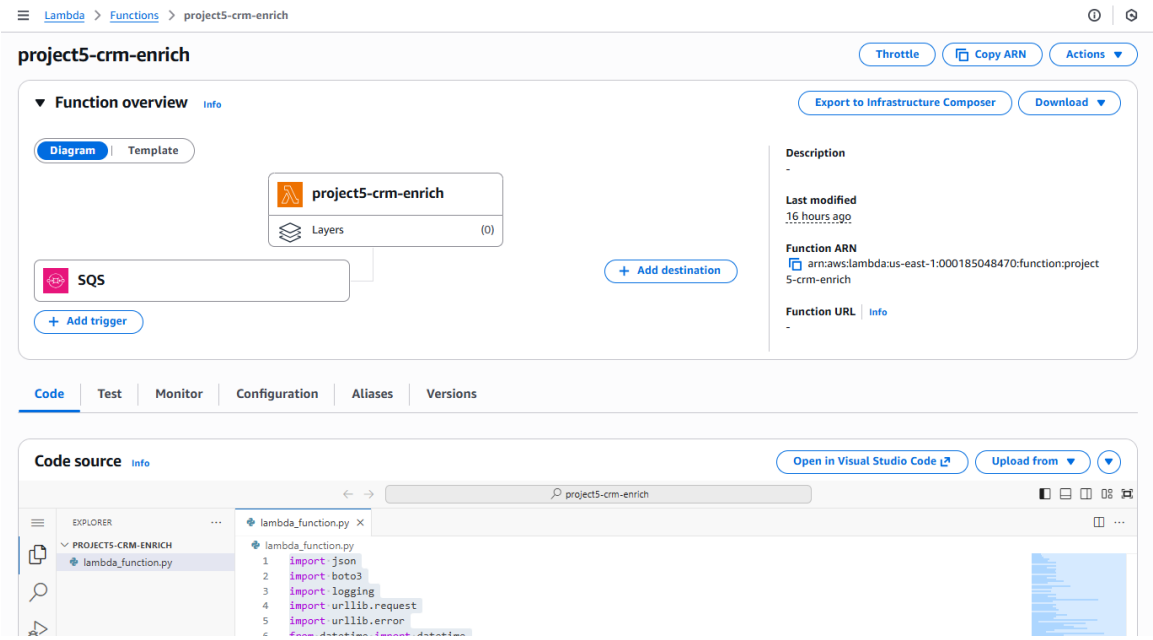
2-3- Amazon SQS

As Close CRM takes some minutes to consider a lead-owner, then we need to create a waiting time. I considered 10 minutes delay after getting the lead event in real time and before looking up the lead-owner's information. To create these 10 minutes delay, I used **Amazon SQS** service with 10 min buffer time. I named this SQS service **project5-crm-queue**.



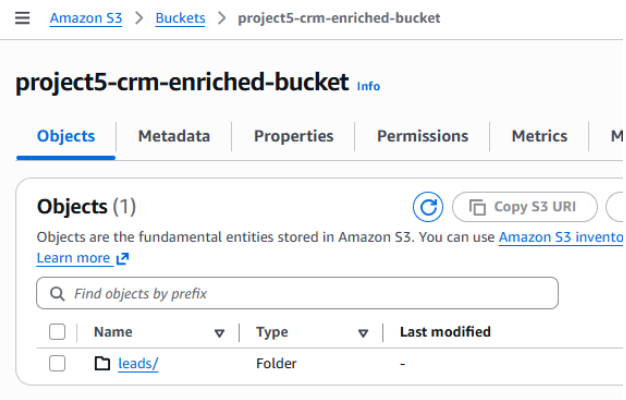
2-4- Lambda function for data enrichment

After getting raw lead data (Silver S3 bucket), waiting for 10 min delay (SQS), then getting lead-owner's informatiokn (Silver S3 bucket), now I am going to create the second Lambda function, named **project5-crm-enrich**, to perform data enrichment and store them in the Gold S3 bucket. The script file name in the Python language is **project5-crm-enrich.py**. This function is triggered by SQS, to make sure of 10 min delay in the processing.



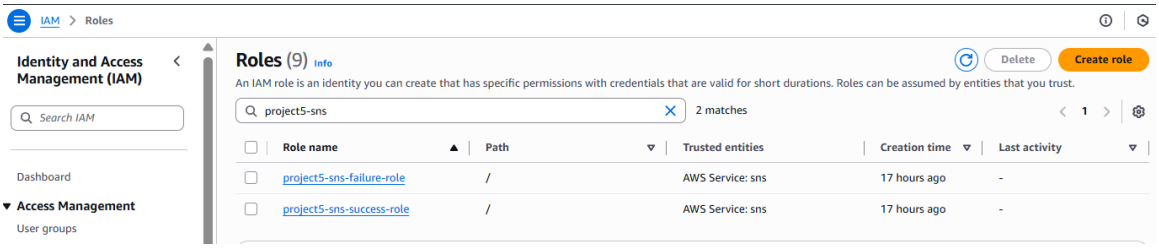
2-5- Enriched Storage (Gold Layer)

I used Amazon S3 service and create a bucet with name of **s3://project5-crm-enriched-bucket/**. After that, I created a folder inside that bucket with the name of **leads/** folder for enriched CRM lead event pluse lead-owner information files. This is a sample of these files: crm_enriched_lead_68DAVGDg87oy3dsaBrIIi13PGsYYaH3myWKNndeuvkZ.json



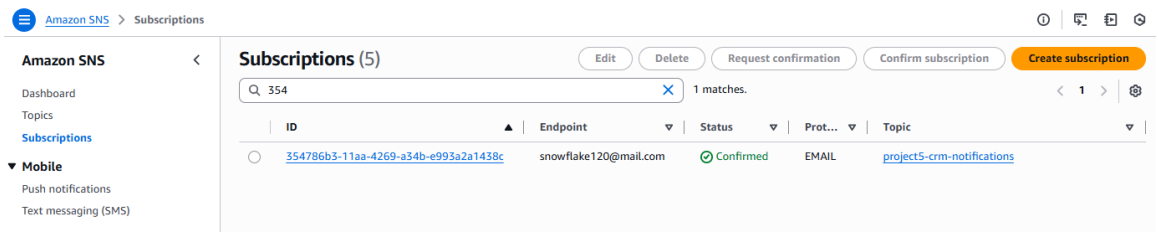
2-6- IAM Role for SNS

While configuring the **Amazon SNS**, I created two IAM roles (as inline policy) for success and failure notifications. These roles give permission to SNS to have access to CloudWatch for logging.



2-7- Amazon SNS

After storing the enriched data in the Gold S3 bucket, then the second lambda function use **Amazon SNS** service to create a notification and send it via email. I created an SNS, named it **project5-crm-notifications**. This SNS, will send the success event notification to my email snowflake120@mail.com.



Below is an example of the enriched lead notification sent via **Amazon SNS** (**Email**):


```
{
"New Lead Alert": " 🔔 ",
"Name": "Kamalpreet kaur",
"Lead ID": "lead_68DAVGDg87oy3dsaBrIIi13PGsYYaH3myWKNndeuvkZ",
"Created Date": "2026-01-12T21:05:46.419000+00:00",
"Label": "Potential",
"Email": "kamalkaur0109@gmail.com",
"Lead Owner": "Ninad Magdum",
"Funnel": "DE ACADEMY Direct VSL"
}
```


Conclusion

In this project, I successfully designed and implemented a **real-time, event-driven CRM lead processing system** using AWS managed services. The solution efficiently captures lead creation events from Close CRM via webhooks, ingests them through API Gateway and AWS Lambda, and stores the raw event data reliably in Amazon S3 for traceability and auditing purposes.

By introducing **Amazon SQS with a controlled 10 min delay**, the architecture decouples ingestion from downstream processing, ensuring scalability, fault tolerance, and protection against traffic spikes. The enrichment layer retrieves additional lead ownership data from a public S3 lookup source, merges it with the original CRM event, and persists the enriched records in a separate S3 bucket, following data lake best practices.

The pipeline also integrates **Amazon SNS for email notifications**, enabling timely and automated alerts to stakeholders whenever a new lead is created and enriched. These notifications include key business-relevant fields such as lead name, ID, creation date, status, owner, and funnel, ensuring actionable insights are delivered in near real time.

This end-to-end implementation showcases practical experience with AWS services including **Lambda, API Gateway, S3, SQS, SNS, IAM, and CloudWatch**, and reflects real-world patterns commonly used in production-grade data platforms.