Build an End-to-End Data Capture Pipeline using Document AI

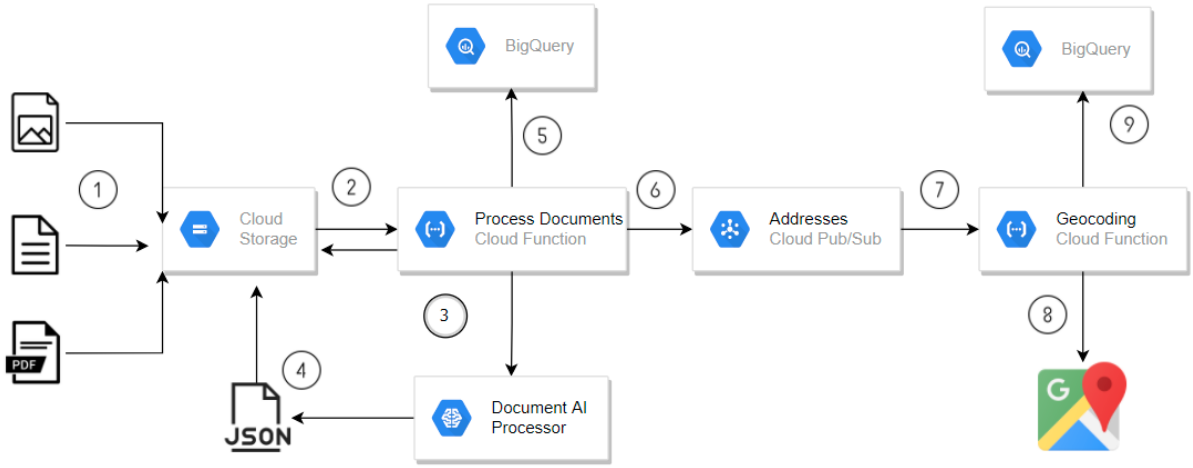
Overview

The Document AI API is a document understanding solution that takes unstructured data, such as documents and emails, and makes the data easier to understand, analyze, and consume.

In this lab, you'll build a document processing pipeline to automatically analyze documents uploaded to Cloud Storage. The pipeline uses a Cloud Run function with a Document AI form processor to extract data and store it in BigQuery. If the form includes address fields, the address data is sent to a Pub/Sub topic. This triggers a second Cloud Run function, which uses the Geocoding API to add coordinates and writes the results to BigQuery.

This simple pipeline uses a general form processor to detect basic form data, like labeled address fields. For more complex documents, Document AI offers specialized parsers (beyond the scope of this lab) that extract detailed information even without explicit labels. For instance, the Invoice parser can identify address and supplier details from an unlabeled invoice by understanding common invoice layouts.

The overall architecture that you will create looks like the following:



1. Upload forms with address data to Cloud Storage.

2. The upload triggers a Cloud Run function call to process the forms.
3. Document AI called from Cloud Run function.
4. Document AI JSON data saved back to Cloud Storage.
5. Form Data written to BigQuery by Cloud Run function.
6. Cloud Run function sends addresses to a Pub/Sub topic.
7. Pub/Sub message triggers Cloud Run function for GeoCode processing.
8. Geocoding API called from Cloud Run function.
9. Geocoding data written to BigQuery by Cloud Run function.

Objectives

In this lab, you learn how to:

- Enable the Document AI API.
- Deploy Cloud Run functions that use the Document AI, BigQuery, Cloud Storage, and Pub/Sub APIs.

You'll configure a Cloud Run function to:

- Trigger when documents are uploaded to Cloud Storage.
- Use the Document AI client library for Python.
- Trigger when a Pub/Sub message is created.

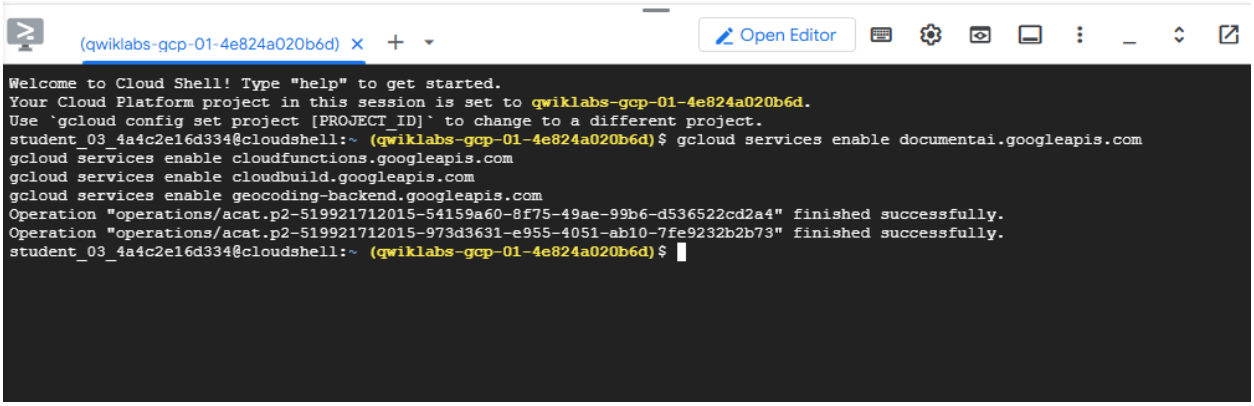
Task 1. Enable APIs and create an API key

You must enable the APIs for Document AI, Cloud Run functions, Cloud Build, and Geocoding for this lab, then create the API key that is required by the Geocoding Cloud Run function.

1. Click **Activate Cloud Shell** at the top of the Google Cloud console.
2. In Cloud Shell, enter the following commands to enable the APIs required by the lab:

```
gcloud services enable documentai.googleapis.com

gcloud services enable cloudfunctions.googleapis.com

gcloud services enable cloudbuild.googleapis.com
```

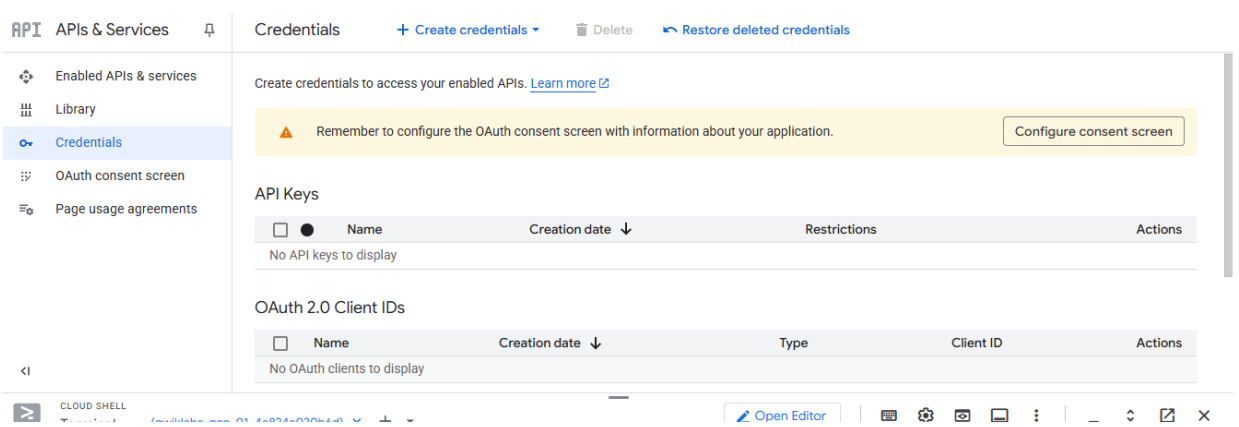
gcloud services enable geocoding-backend.googleapis.com

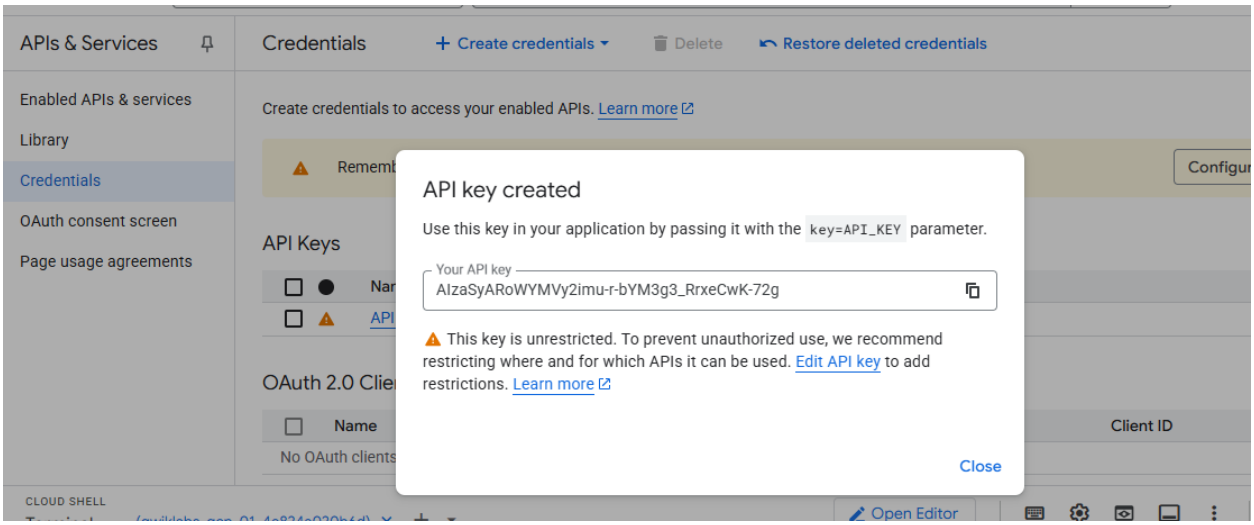


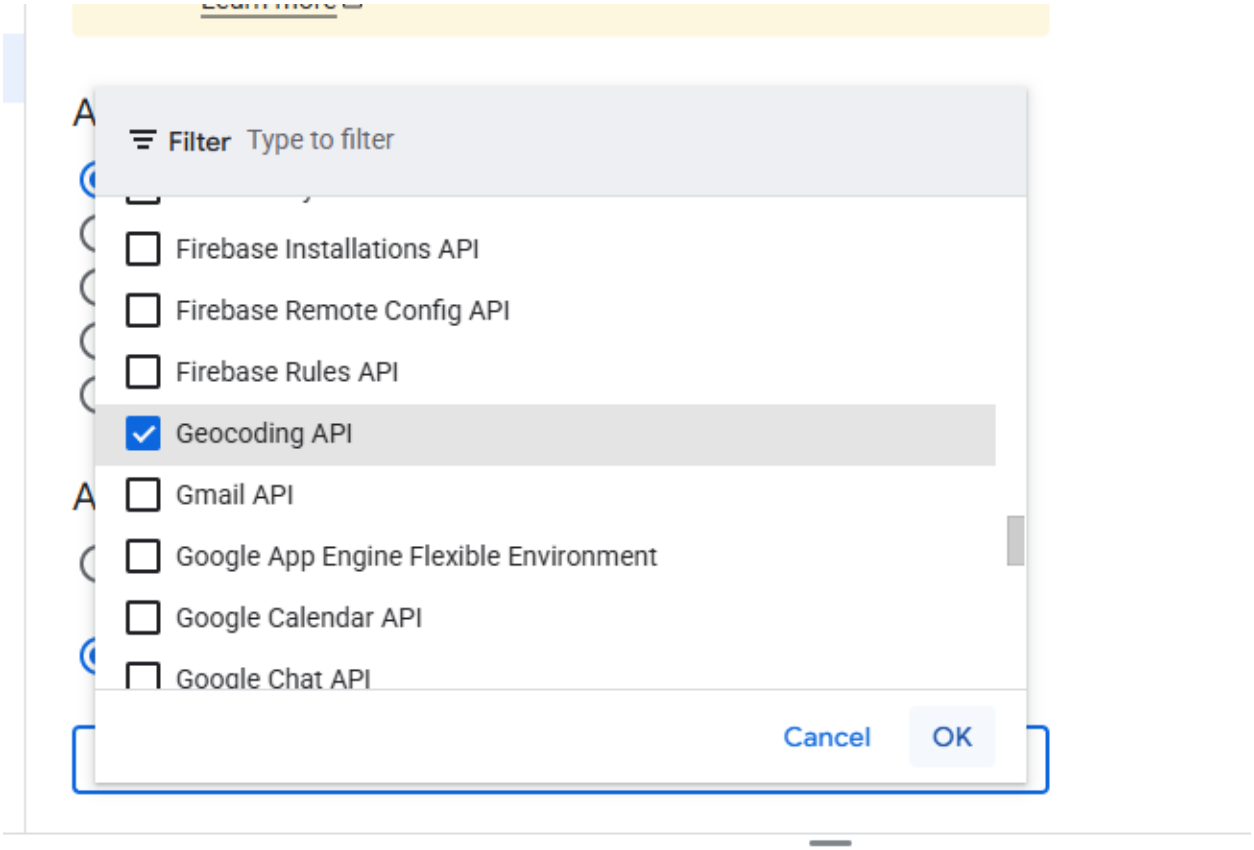
3. In the console, in the **Navigation menu** (☰), click **APIs & services > Credentials**.
4. Select **Create credentials**, then select **API key** from the dropdown menu.

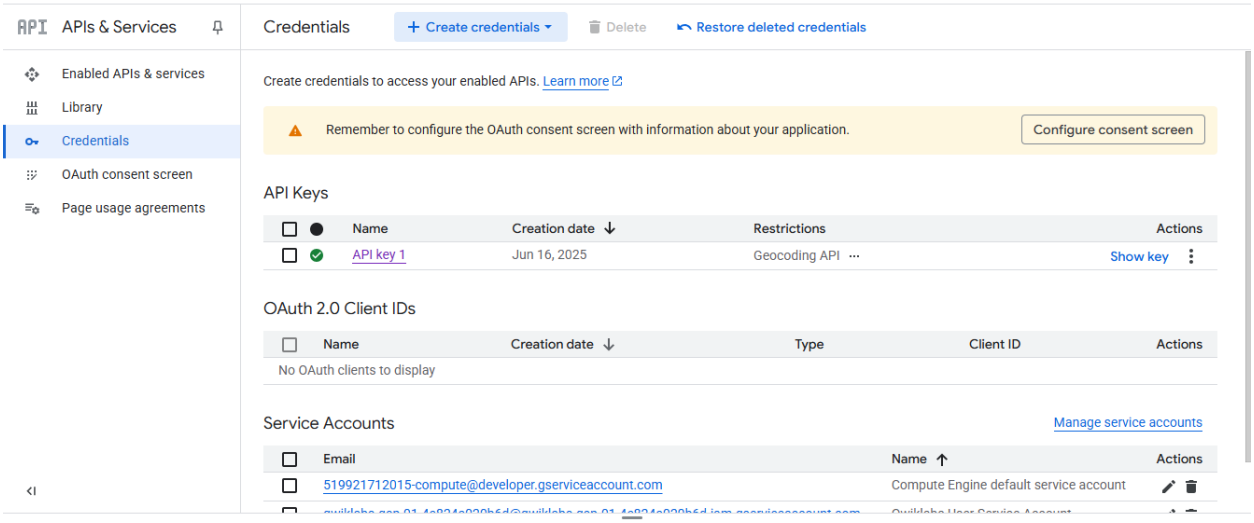
The API key created dialog box displays your newly created key. An API key is a long string containing upper and lower case letters, numbers, and dashes. For example, a4db08b757294ea94c08f2df493465a1.

5. Click three dots under **Actions** then click **Edit API key** in the dialog box.
6. Select **Restrict key** in the **API restrictions** section to add API restrictions for your new API key.
7. Click in the filter box and type **Geocoding API**.
8. Select **Geocoding API** and click **OK**.
9. Click the **Save** button.




APIs & Services
Credentials
+ Create credentials
Delete
Restore deleted credentials
Enabled APIs & services
Library
Credentials
OAuth consent screen
Page usage agreements
Create credentials to access your enabled APIs. Learn more
API Keys
OAuth 2.0 Clie
Name
Client ID
No OAuth clients
API key created
Use this key in your application by passing it with the key=API_KEY parameter.
Your API key
AIzaSyARoWYMVy2imu-r-bYM3g3_RrxeCwK-72g
This key is unrestricted. To prevent unauthorized use, we recommend restricting where and for which APIs it can be used. Edit API key to add restrictions. Learn more
Close
CLOUD SHELL
Open Editor


Filter
Type to filter
Firebase Installations API
Firebase Remote Config API
Firebase Rules API
Geocoding API
Gmail API
Google App Engine Flexible Environment
Google Calendar API
Google Chat API
Cancel
OK

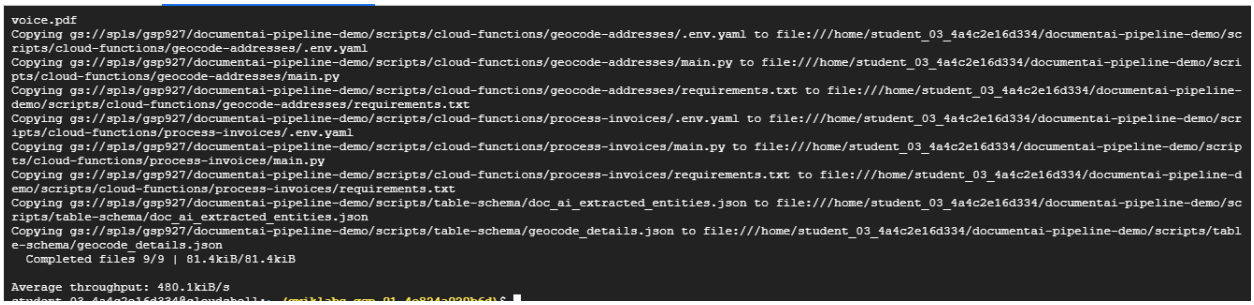


Task 2. Download the lab source code

In this task, you copy the source files into your Cloud Shell. These files include the source code for the Cloud Run functions and the schemas for the BigQuery tables that you will create in the lab.

1. In Cloud Shell, enter the following command to download the source code for this lab:

```
mkdir ./documentai-pipeline-demo

gcloud storage cp -r \

 gs://spls/gsp927/documentai-pipeline-demo/* \

 ~/documentai-pipeline-demo/
```

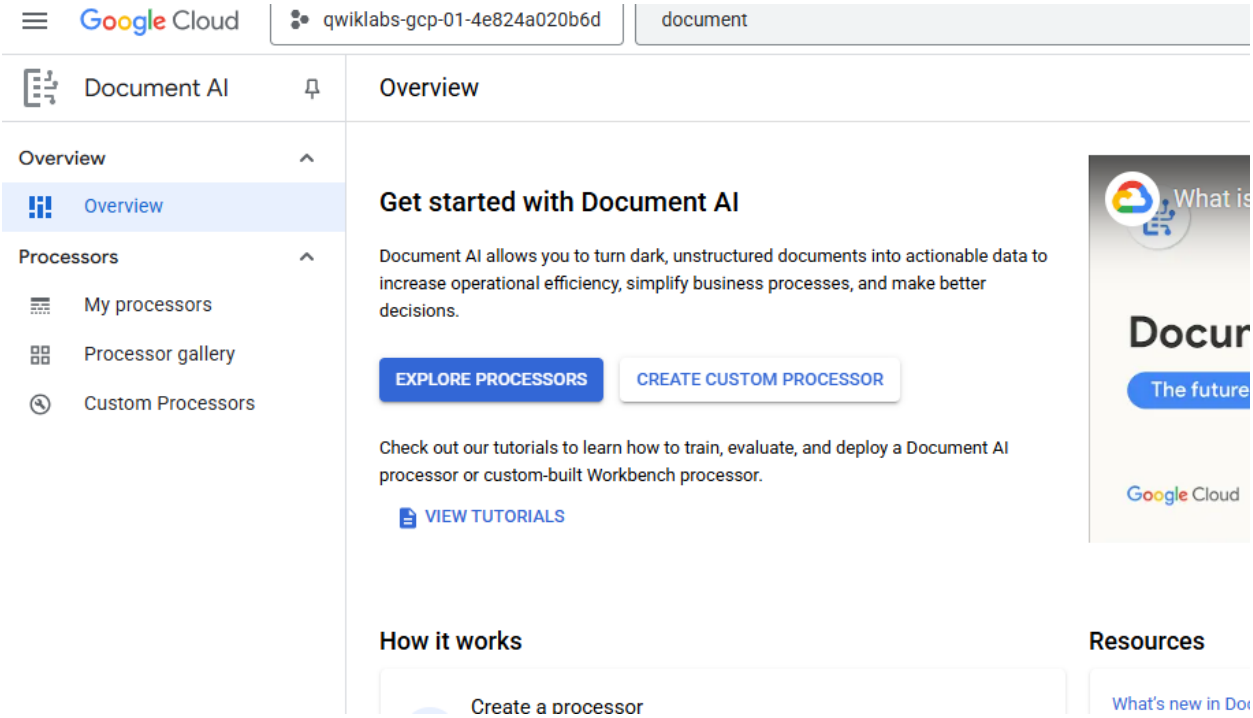


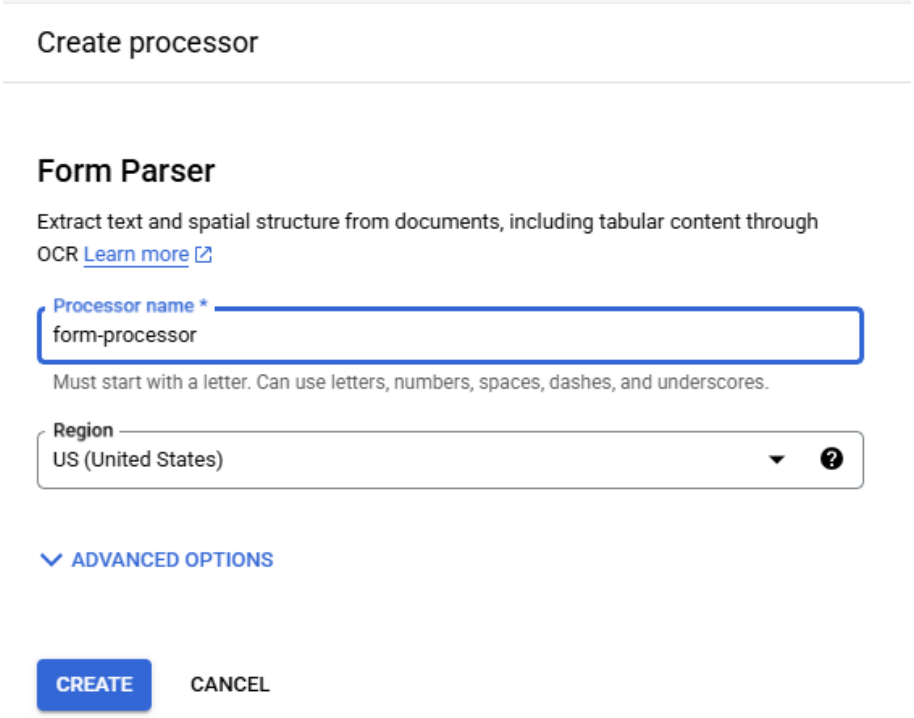
Task 3. Create a form processor

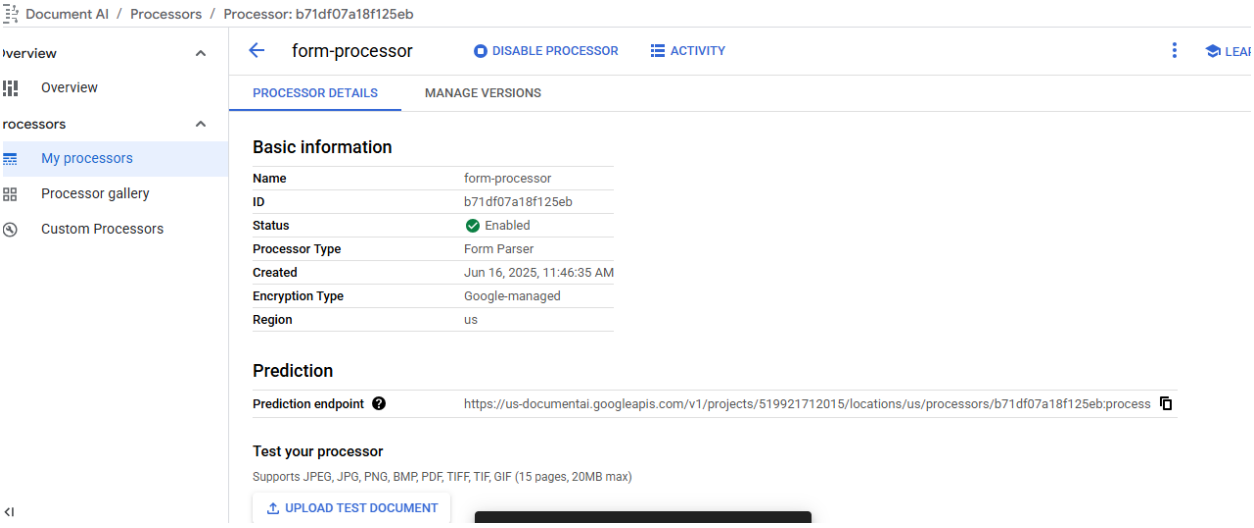
Create an instance of the generic form processor to use in the Document AI Platform using the Document AI **Form Parser** specialized parser. The generic form processor will process any type of document and extract all the text content it can identify in the document. It is not limited to printed text, it can handle handwritten text and text in any orientation,

supports a number of languages, and understands how form data elements are related to each other so that you can extract key:value pairs for form fields that have text labels.

1. In the Google Cloud Console, in the search bar, type Document AI and click the product page result.
2. Click **Explore Processors** and click **Create Processor** for Form Parser.
3. Specify the processor name as **form-processor** and select the region **US (United States)** from the list.
4. Click **Create** to create your processor.







Task 4. Create Cloud Storage buckets and a BigQuery dataset

In this section, you will prepare your environment by creating the Google Cloud resources that are required for your document processing pipeline.

Create input, output, and archive Cloud Storage buckets

Create input, output, and archive Cloud Storage buckets for your document processing pipeline.

1. In Cloud Shell, enter the following command to create the Cloud Storage buckets for the lab:

```
export PROJECT_ID=$(gcloud config get-value core/project)

export BUCKET_LOCATION="REGION"

gsutil mb -c standard -l ${BUCKET_LOCATION} -b on \

  gs://${PROJECT_ID}-input-invoices

gsutil mb -c standard -l ${BUCKET_LOCATION} -b on \

  gs://${PROJECT_ID}-output-invoices

gsutil mb -c standard -l ${BUCKET_LOCATION} -b on \

  gs://${PROJECT_ID}-archived-invoices
```

```
student_03_4a4c2e16d334@cloudshell:~ (qwiklabs-gcp-01-4e824a020b6d)$ export PROJECT_ID=$(gcloud config get-value core/project)
export BUCKET_LOCATION="us-east4"
gsutil mb -c standard -l ${BUCKET_LOCATION} -b on \
  gs://${PROJECT_ID}-input-invoices
gsutil mb -c standard -l ${BUCKET_LOCATION} -b on \
  gs://${PROJECT_ID}-output-invoices
gsutil mb -c standard -l ${BUCKET_LOCATION} -b on \
  gs://${PROJECT_ID}-archived-invoices
Your active configuration is: [cloudshell-26986]
Creating gs://qwiklabs-gcp-01-4e824a020b6d-input-invoices/...
Creating gs://qwiklabs-gcp-01-4e824a020b6d-output-invoices/...
Creating gs://qwiklabs-gcp-01-4e824a020b6d-archived-invoices/...
student_03_4a4c2e16d334@cloudshell:~ (qwiklabs-gcp-01-4e824a020b6d)$
```

Create a BigQuery dataset and tables

Create a BigQuery dataset and the three output tables required for your data processing pipeline.

1. In Cloud Shell, enter the following command to create the BigQuery tables for the lab:

```
bq --location="US" mk -d

--description "Form Parser Results"
```

${PROJECT_ID}:invoice_parser_results cd ~/documentai-pipeline-demo/scripts/table-schema/ bq mk --table

invoice_parser_results.doc_ai_extracted_entities

doc_ai_extracted_entities.json bq mk --table

invoice_parser_results.geocode_details

geocode_details.json

```
student_03_4a4c2e16d334@cloudshell:~ (qwiklabs-gcp-01-4e824a020b6d)$ bq --location="US" mk  -d \
    --description "Form Parser Results" \
    ${PROJECT_ID}:invoice_parser_results
cd ~/documentai-pipeline-demo/scripts/table-schema/
bq mk --table \
  invoice_parser_results.doc_ai_extracted_entities \
  doc_ai_extracted_entities.json
bq mk --table \
  invoice_parser_results.geocode_details \
  geocode_details.json
Dataset 'qwiklabs-gcp-01-4e824a020b6d:invoice_parser_results' successfully created.
Table 'qwiklabs-gcp-01-4e824a020b6d:invoice_parser_results.doc_ai_extracted_entities' successfully created.
Table 'qwiklabs-gcp-01-4e824a020b6d:invoice_parser_results.geocode_details' successfully created.
student_03_4a4c2e16d334@cloudshell:~/documentai-pipeline-demo/scripts/table-schema (qwiklabs-gcp-01-4e824a020b6d)$
```

Create a Pub/Sub topic

Initialize the Pub/Sub topic used to trigger the Geocoding API data enrichment operations in the processing pipeline.

1. In Cloud Shell, enter the following command to create the Pub/Sub topics for the lab:

export GEO_CODE_REQUEST_PUBSUB_TOPIC=geocode_request gcloud pubsub topics

create ${GEO_CODE_REQUEST_PUBSUB_TOPIC}

```
student_03_4a4c2e16d334@cloudshell:~/documentai-pipeline-demo/scripts/table-schema (qwiklabs-gcp-01-4e824a020b6d)$ export GEO_CODE_REQUEST_PUBSUB_TOPIC=geocode_request
gcloud pubsub topics \
  create ${GEO_CODE_REQUEST_PUBSUB_TOPIC}
Created topic [projects/qwiklabs-gcp-01-4e824a020b6d/topics/geocode_request].
student_03_4a4c2e16d334@cloudshell:~/documentai-pipeline-demo/scripts/table-schema (qwiklabs-gcp-01-4e824a020b6d)$
```

Task 5. Create Cloud Run functions

Create the two Cloud Run functions that your data processing pipeline uses to process invoices uploaded to Cloud Storage. These functions use the Document AI API to extract form data from the raw documents, then use the GeoCode API to retrieve geolocation data about the address information extracted from the documents.

You can examine the source code for the two Cloud Run functions using the Code Editor or any other editor of your choice. The Cloud Run functions are stored in the following folders in Cloud Shell:

- Process Invoices - scripts/cloud-functions/process-invoices
- Geocode Addresses - scripts/cloud-functions/geocode-addresses

The main Cloud Run function, process-invoices, is triggered when files are uploaded to the input files storage bucket you created earlier.

The function folder scripts/cloud-functions/process-invoices contains the two files that are used to create the process-invoices Cloud Run function.

The requirements.txt file specifies the Python libraries required by the function. This includes the Document AI client library as well as the other Google Cloud libraries required by the Python code to read the files from Cloud Storage, save data to BigQuery, and write messages to Pub/Sub that will trigger the remaining functions in the solution pipeline.

The main.py Python file contains the Cloud Run function code that creates the Document-AI, BigQuery, and Pub/Sub API clients and the following internal functions to process the documents:

- write_to_bq - Writes dictionary object to the BigQuery table. Note you must ensure the schema is valid before calling this function.
- get_text - Maps form name and value text anchors to the scanned text in the document. This allows the function to identify specific forms elements, such as the Supplier name and Address, and extract the relevant value. A specialized Document AI processor provides that contextual information directly in the entities property.
- process_invoice - Uses the asynchronous Document-AI client API to read and process files from Cloud Storage as follows:
    - Creates an asynchronous request to process the file(s) that triggered the Cloud Run function call.
    - Processes form data to extract invoice fields, storing only specific fields in a dictionary that are part of the predefined schema.
    - Publishes Pub/Sub messages to trigger the Geocoding Cloud Run function using address form data extracted from the document.
    - Writes form data to a BigQuery table.
    - Deletes intermediate (output) files asynchronous Document AI API call.
    - Copies input files to the archive bucket.

- Deletes processed input files.

The process_invoices Cloud Run function only processes form data that has been detected with the following form field names:

- input_file_name
- address
- supplier
- invoice_number
- purchase_order
- date
- due_date
- subtotal
- tax
- total

The other Cloud Run function, geocode-addresses, is triggered when a new message arrives on a Pub/Sub topic and it extracts its parameter data from the Pub/Sub message.

Create the Cloud Run function to process documents uploaded to Cloud Storage

Create a Cloud Run function that uses a Document AI form processor to parse form documents that have been uploaded to a Cloud Storage bucket.

1. Run the command to get the email address of the project's Cloud Storage service agent:

gcloud storage service-agent --project=$PROJECT_ID

```
student_03_4a4c2e16d334@cloudshell:~/documentai-pipeline-demo/scripts/table-schema (qwiklabs-gcp-01-4e824a020b6d)$ gcloud storage service-agent --project=$PROJECT_ID
service-519921712015@gs-project-accounts.iam.gserviceaccount.com
student_03_4a4c2e16d334@cloudshell:~/documentai-pipeline-demo/scripts/table-schema (qwiklabs-gcp-01-4e824a020b6d)$
```

2. Run the below command to allow the required permissions to the Cloud Storage service account:

PROJECT_NUMBER=$(gcloud projects describe $PROJECT_ID --format="value(projectNumber)")

gcloud iam service-accounts create "service-$PROJECT_NUMBER"

--display-name "Cloud Storage Service Account" || true

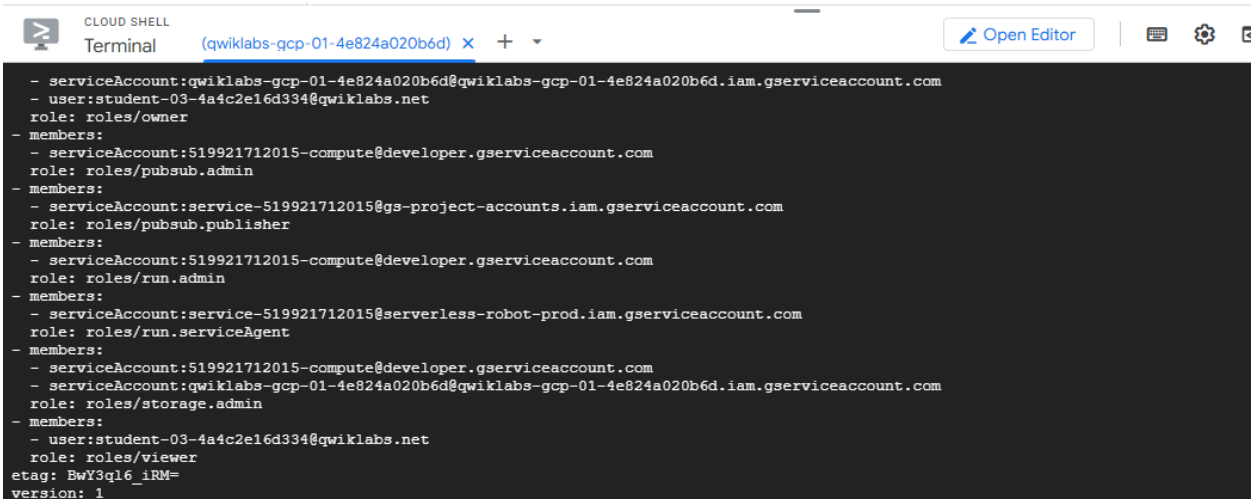
gcloud projects add-iam-policy-binding $PROJECT_ID

--member="serviceAccount:service-$PROJECT_NUMBER@gs-project-accounts.iam.gserviceaccount.com"

--role="roles/pubsub.publisher" gcloud projects add-iam-policy-binding $PROJECT_ID

--member="serviceAccount:service-$PROJECT_NUMBER@gs-project-accounts.iam.gserviceaccount.com"

--role="roles/iam.serviceAccountTokenCreator"



3. Create the Invoice Processor Cloud Run function:

cd ~/documentai-pipeline-demo/scripts export CLOUD_FUNCTION_LOCATION="REGION"

gcloud functions deploy process-invoices

--no-gen2

--region=${CLOUD_FUNCTION_LOCATION}

--entry-point=process_invoice

--runtime=python39

--source=cloud-functions/process-invoices

--timeout=400

--env-vars-file=cloud-functions/process-invoices/.env.yaml

--trigger-resource=gs://${PROJECT_ID}-input-invoices

--trigger-event=google.storage.object.finalize

```
student_03_4a4c2e16d334@cloudshell:~/documentai-pipeline-demo/scripts/table-schema (qwiklabs-gcp-01-4e824a020b6d)$   cd ~/documentai-pipeline-demo/scripts
  export CLOUD_FUNCTION_LOCATION="us-east4"

  gcloud functions deploy process-invoices \
  --no-gen2 \
  --region=${CLOUD_FUNCTION_LOCATION} \
  --entry-point=process_invoice \
  --runtime=python39 \
  --source=cloud-functions/process-invoices \
  --timeout=400 \
  --env-vars-file=cloud-functions/process-invoices/.env.yaml \
  --trigger-resource=gs://${PROJECT_ID}-input-invoices \
  --trigger-event=google.storage.object.finalize
```

Create the Cloud Run function to lookup geocode data from an address

Create the Cloud Run function that accepts address data from a Pub/Sub message and uses the Geocoding API to precisely locate the address.

1. Create the Geocoding Cloud Run function:

cd ~/documentai-pipeline-demo/scripts gcloud functions deploy geocode-addresses

--no-gen2

--region=${CLOUD_FUNCTION_LOCATION}

--entry-point=process_address

--runtime=python39

--source=cloud-functions/geocode-addresses

--timeout=60

--env-vars-file=cloud-functions/geocode-addresses/.env.yaml

--trigger-topic=${GEO_CODE_REQUEST_PUBSUB_TOPIC}

```
entryPoint: process_invoice
environmentVariables:
  GCS_OUTPUT_URI_PREFIX: processed
  GEOCODE_REQUEST_TOPICNAME: geocode_request
  PARSER_LOCATION: YourParserlocation_goeshere
  PROCESSOR_ID: YourProcessorID_goeshere
  TIMEOUT: '300'
eventTrigger:
  eventType: google.storage.object.finalize
  failurePolicy: {}
  resource: projects/_/buckets/qwiklabs-gcp-01-4e824a020b6d-input-invoices
  service: storage.googleapis.com
ingressSettings: ALLOW_ALL
labels:
  deployment-tool: cli-gcloud
maxInstances: 5
name: projects/qwiklabs-gcp-01-4e824a020b6d/locations/us-east4/functions/process-invoices
runtime: python39
serviceAccountEmail: qwiklabs-gcp-01-4e824a020b6d@appspot.gserviceaccount.com
sourceUploadUrl: https://storage.googleapis.com/uploads-774872888311.us-east4.cloudfunctions.appspot.com/21058e56-8519-4824-9d93-da7ece142138.zip
status: ACTIVE
timeout: 400s
updateTime: '2025-06-16T06:21:54.275073956Z'
versionId: '1'
```

Task 6. Edit environment variables for Cloud Run functions

In this task, you finalize the configuration of the Cloud Run functions by editing the environment variables for each function to reflect your lab specific parameters via the Cloud Console.

Edit environment variables for the process-invoices Cloud Run function

Set the Cloud Run function environment variables for the **process-invoices** function.

1. In the Cloud Console, in the search bar, type Cloud Run functions and click the product page result.

It will redirect to the **Cloud Run** console, click on **Go to Cloud Run functions 1st gen** to see the deployed functions process-invoices and geocode-addresses.

2. Click the Cloud Run function **process-invoices** to open its management page.
3. Click **Edit**.
4. Click **Runtime, build, connections and security settings** to expand that section.
5. Under **Runtime environment variables**, add the **GCP_PROJECT** variable and the value to match your Project ID.
6. Under **Runtime environment variables**, update the **PROCESSOR_ID** value to match the Invoice processor ID you created earlier.
7. Under **Runtime environment variables**, update the **PARSER_LOCATION** value to match the region of the Invoice processor you created earlier. This will be us or eu. This parameter **must** be lowercase.
8. Click **Next** and select **.env.yaml** and then update the PROCESSOR_ID, PARSER_LOCATION, and GCP_PROJECT values again for your invoice processor.

Cloud Run functions | Functions (1st Gen) | CREATE FUNCTION | REFRESH | LEARN

This page only shows legacy 1st gen functions. We have integrated Cloud Run functions into Cloud Run UI. Starting August 2025, creating legacy 1st gen functions will only be possible using the gcloud CLI, API or Terraform.

Please go to Cloud Run to manage all new function deployments.

LEARN MORE | GO TO CLOUD RUN

Filter | Filter functions

Name	Last deployed	Region	Recommendation	Trigger	Runtime	Memory allocated	Executed function	Actions
geocode-addresses	Jun 16, 2025, 11:53:55 AM	us-east4		Topic: geocode_request	Python 3.9	256 MB	process_address	
process-invoices	Jun 16, 2025, 11:51:54 AM	us-east4		Bucket: qwiklabs-gcp-01-4e824a020b6d-input-invoices	Python 3.9	256 MB	process_invoice	

Edit function

Runtime environment variables

Name	Value
GCP_PROJECT	qwiklabs-gcp-01-4e824a020b6d
GCS_OUTPUT_URI_PREFIX	processed
GEOCODE_REQUEST_TOPICNAME	geocode_request
PARSER_LOCATION	us
PROCESSOR_ID	b71df07a18f125eb
TIMEOUT	300

+ ADD VARIABLE

NEXT | CANCEL

Cloud Run functions / Functions (1st Gen) / Details: process-invoices / Edit

Edit function | LEARN

Configuration — 2 Code

Runtime: Python 3.9

Entry point *: process_invoice

TEST FUNCTION

Source code: Inline Editor

.env.yaml

requirements.txt

main.py

Press Alt+F1 for Accessibility Options.

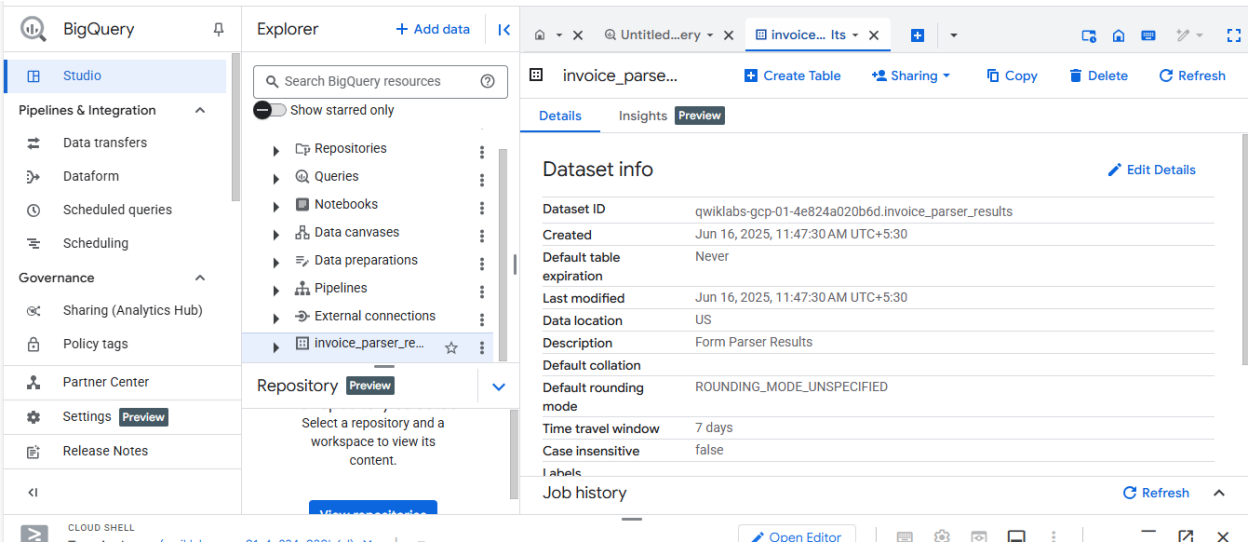
```
import base64
import re
import os
import json
from datetime import datetime
from google.cloud import bigquery
from google.cloud import documentai_v1beta3 as documentai
from google.cloud import storage
from google.cloud import pubsub_v1

#Reading environment variables
gcs_output_uri_prefix = os.environ.get('GCS_OUTPUT_URI_PREFIX')
project_id = os.environ.get('GCP_PROJECT')
location = os.environ.get('PARSER_LOCATION')
processor_id = os.environ.get('PROCESSOR_ID')
geocode_request_topicname = os.environ.get('GEOCODE_REQUEST_TOPICNAME')
```

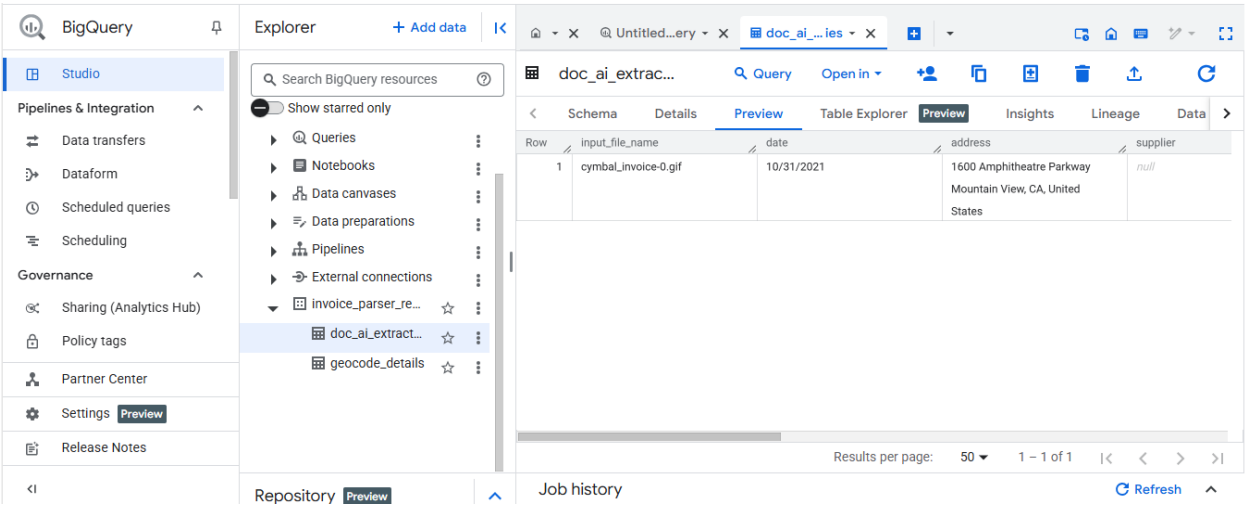
PREVIOUS | DEPLOY | CANCEL

Edit environment variables for the geocode-addresses Cloud Run function

Set the Cloud Run function environment variables for the GeoCode data enrichment function.

1. Click the Cloud Run function **geocode-addresses** to open its management page.
2. Click **Edit**.
3. Click **Runtime, build, connections and security settings** to expand that section.
4. Under **Runtime environment variables**, update the **API_key** value to match to the API Key value created in Task 1.
5. Click **Next** and select **.env.yaml** and then update the API_key value to match the API Key value you set in the previous step.
6. Click **Deploy**.





Task 7. Test and validate the end-to-end solution

Upload test data to Cloud Storage and monitor the progress of the pipeline as the documents are processed and the extracted data is enhanced.

1. In Cloud Shell, enter the following command to upload sample forms to the Cloud Storage bucket that will trigger the process-invoices Cloud Run function:

export PROJECT_ID=$(gcloud config get-value core/project)

gsutil cp gs://spls/gsp927/documentai-pipeline-demo/sample-files/* gs://${PROJECT_ID}-input-invoices/

2. In the Cloud Console, in the search bar, type Cloud Run functions and click the product page result.
3. Click the Cloud Run function **process-invoices** to open its management page.
4. Click **Logs**.

Watch the events until you see a final event indicating that the function execution finished with a LoadJob. If errors are reported double check that the parameters set in the .env.yaml file in the previous section are correct. In particular make sure the Processor ID, location, and Project ID are valid. The event list does not automatically refresh.

At the end of the processing, your BigQuery tables will be populated with the Document AI extracted entities as well as enriched data provided by the Geocoding API if the Document AI Processor has detected address data in the uploaded document.

5. In the Cloud Console, on the **Navigation menu** (☰), click **BigQuery**.

6. Expand your Project ID in the Explorer.
7. Expand **invoice_parser_results**.
8. Select **doc_ai_extracted_entities** and click **Preview**. You will see the form information extracted from the invoices by the invoice processor. You can see that address information and the supplier name has been detected.
9. Select **geocode_details** and click **Preview**. You will see the formatted address, latitude, and longitude for each invoice that has been processed that contained address data that Document AI was able to extract.