


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

Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

1. Click **Activate Cloud Shell**  at the top of the Google Cloud console.
2. Click through the following windows:
 - Continue through the Cloud Shell information window.
 - Authorize Cloud Shell to use your credentials to make Google Cloud API calls.

When you are connected, you are already authenticated, and the project is set to your **Project_ID**, `PROJECT_ID`. The output contains a line that declares the **Project_ID** for this session:

```
Your Cloud Platform project in this session is set to "PROJECT_ID"
gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.
```

3. (Optional) You can list the active account name with this command:
`gcloud auth list`

4. Click **Authorize**.

Output:

```
ACTIVE: *
ACCOUNT: "ACCOUNT"

To set the active account, run:
```

```
$ gcloud config set account `ACCOUNT`
```

5. (Optional) You can list the project ID with this command:

```
gcloud config list project
```


Output:

```
[core]
project = "PROJECT_ID"
```

Note: For full documentation of `gcloud`, in Google Cloud, refer to [the gcloud CLI overview guide](#).


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
Copied!
```

content_copy

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.

Note: If you cannot find the Geocoding API in the **Restrict key** dropdown list, refresh the page to refresh the list of available APIs.

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://splis/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.

You will configure a Cloud Run function later in this lab with the processor ID and location of this processor so that the Cloud Run function will use this specific processor to process sample invoices.

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="us-central1"
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
```

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
```

You can navigate to BigQuery in the Cloud Console and inspect the schemas for the tables in the `invoice_parser_results` dataset using the BigQuery SQL workspace.

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}
```

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
```

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"
```

Note: If the Cloud Storage service account exists, you can ignore the error.

3. Create the Invoice Processor Cloud Run function:

```
cd ~/documentai-pipeline-demo/scripts
export CLOUD_FUNCTION_LOCATION="us-central1"
```

```
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
```

Note: If the command fails with a permission error, wait a minute and try it again.

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}
```

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`.

Note: If you are not able to see the link for **Go to Cloud Run functions 1st gen**, then refresh the Cloud Run console.

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.

Configuration — 2 Code

Runtime: Python 3.9 Entry point *: process_invoice TEST FUNCTION

Source code: Inline Editor

Press Option+F1 for Accessibility Options.

```
1 PARSER_LOCATION: us
2 PROCESSOR_ID: e3f5582893836300
3 GCS_OUTPUT_URI_PREFIX: processed
4 TIMEOUT: "300"
5 GEOCODE_REQUEST_TOPICNAME: geocode_request
6 GCP_PROJECT: qwiklabs-gcp-01-4a92ee1ebe10
```

9. Click **Deploy**.

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 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=$(gcloud config get-value core/project)
gsutil cp gs://spl/spls/gsp927/documentai-pipeline-demo/sample-files/*
gs://${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**.

You will see events related to the creation of the function and the updates made to configure the environment variables followed by events showing details about the file being processed, and the data detected by Document AI.

Metrics

Details

Source

Variables

Trigger

Permissions

Logs

Testing






Logs


Severity

Default

Filter

Search all fields and values


Severity	Timestamp	Summary
> 	2024-10-16 11:16:13.200 PDT	Default STARTUP TCP probe succeeded after 1 attempt for container "worker" on port 8080.
> 	2024-10-16 11:16:13.285 PDT	Cloud Run ReplaceInternalService process-invoices-00005-gug {@type: type.googleapis.com/google.cloud.audit.AuditLog, methodName:
> 	2024-10-16 11:16:14.949 PDT	Cloud Run ReplaceInternalService process-invoices {@type: type.googleapis.com/google.cloud.audit.AuditLog, methodName: /Internal:
> 	2024-10-16 11:16:15.235 PDT	Cloud Functions UpdateFunction us-west1:process-invoices student-01-aed4815b7636@qwiklabs.net {@type: type.googleapis.com/google.c
> 	2024-10-16 11:16:19.472 PDT	Default STARTUP TCP probe succeeded after 1 attempt for container "worker" on port 8080.
> 	2024-10-16 11:17:28.660 PDT	POST 200 130 B 35.2 s APIs-Google; (+https://developers.goo... https://process-invoices-6szgbm6jxa-uw.a.run.app/?__GCP_CloudEvents
> 	2024-10-16 11:17:28.760 PDT	Printing the contentType: application/pdf
> 	2024-10-16 11:17:58.121 PDT	Output files:
> 	2024-10-16 11:17:58.121 PDT	Fetching from processed/6606720682359711702/0/cymbal_invoice-0.json
> 	2024-10-16 11:17:58.121 PDT	input_filename cymbal_invoice-0.gif
> 	2024-10-16 11:17:59.362 PDT	{ 'input_file_name': 'cymbal_invoice-0.gif', 'due_date': '11/30/2021\n', 'purchase_order': '00002\n', 'invoice_number': '00001\n', 'dat
> 	2024-10-16 11:17:59.362 PDT	Writing to BQ
> 	2024-10-16 11:18:03.751 PDT	LoadJob<project=qwiklabs-gcp-03-c4ef0b0e9eea, location=US, id=e23eae96-0ac3-4da4-9c80-07f61b1b8e01>



No newer entries found matching current filter.

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.

- In the Cloud Console, on the **Navigation menu** () , click **BigQuery**.
- Expand your Project ID in the Explorer.
- Expand **invoice_parser_results**.
- 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.
- 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.