Optical Character Recognition (OCR) with Document AI (Python) | Google Cloud Skills Boost

Qwiklabs : 26-33 minutes

GSP1138

Overview

Document AI is a document understanding solution that takes unstructured data (e.g. documents, emails, invoices, forms, etc.) and makes the data easier to understand, analyze, and consume. The API provides structure through content classification, entity extraction, advanced searching, and more.

In this lab, you will perform Optical Character Recognition (OCR) of PDF documents using Document AI and Python. You will explore how to make both Online (Synchronous) and Batch (Asynchronous) process requests.

We will utilize a PDF file of the classic novel "Winnie the Pooh" by A.A. Milne, which has recently become part of the Public Domain in the United States. This file was scanned and digitized by Google Books.

Objectives

In this lab, you will learn how to perform the following tasks:

- Enable the Document AI API
- Authenticate API requests
- Install the client library for Python
- Use the online and batch processing APIs
- Parse text from a PDF file

Setup and requirements

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google

Cloud for the duration of the lab.

To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).

Note: Use an Incognito or private browser window to run this lab. This prevents any conflicts between your personal account and the Student account, which may cause extra charges incurred to your personal account.

- Time to complete the lab---remember, once you start, you cannot pause a lab.

Note: If you already have your own personal Google Cloud account or project, do not use it for this lab to avoid extra charges to your account.

How to start your lab and sign in to the Google Cloud Console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is the **Lab Details** panel with the following:

    - The **Open Google Console** button
    - Time remaining
    - The temporary credentials that you must use for this lab
    - Other information, if needed, to step through this lab

2. Click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.

    Tip: Arrange the tabs in separate windows, side-by-side.

    Note: If you see the **Choose an account** dialog, click **Use Another Account**.

3. If necessary, copy the **Username** from the **Lab Details** panel and paste it into the **Sign in** dialog. Click **Next**.

4. Copy the **Password** from the **Lab Details** panel and paste it into the **Welcome** dialog. Click **Next**.

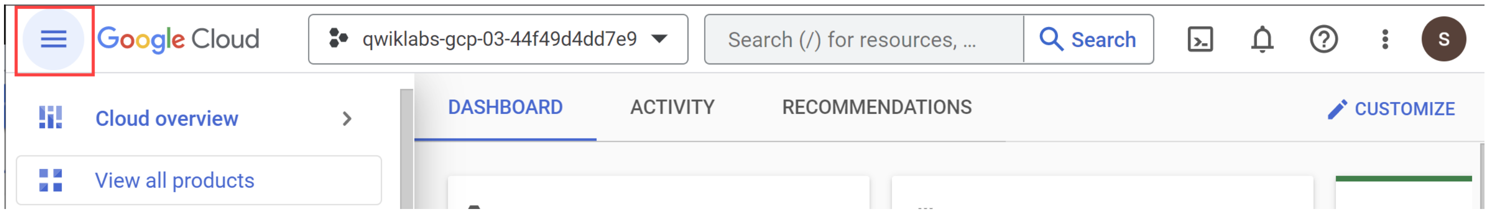
    Important: You must use the credentials from the left panel. Do not use your Google Cloud Skills Boost credentials. **Note:** Using your own Google Cloud account for this lab may incur extra charges.

5. Click through the subsequent pages:

    - Accept the terms and conditions.
    - Do not add recovery options or two-factor authentication (because this is a temporary account).
    - Do not sign up for free trials.

After a few moments, the Cloud Console opens in this tab.

Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.



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.

When you are connected, you are already authenticated, and the project is set to your **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 YOUR_PROJECT_ID

`gcloud` is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

2. (Optional) You can list the active account name with this command:

gcloud auth list

3. Click **Authorize**.

4. Your output should now look like this:

Output:

ACTIVE: * ACCOUNT: student-01-xxxxxxxxxxxx@qwiklabs.net 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>

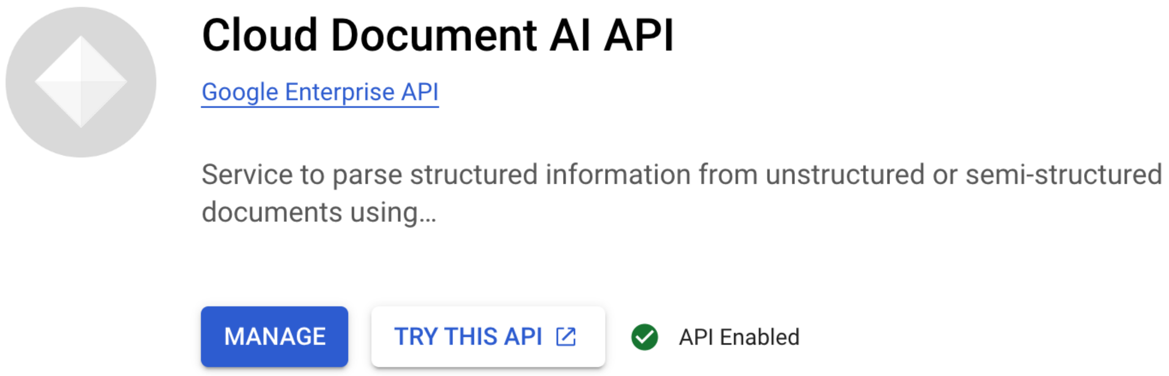
Example output:

[core] project = qwiklabs-gcp-44776a13dea667a6 **Note:** For full documentation of `gcloud`, in Google Cloud, refer to the gcloud CLI overview guide.

Task 1. Enable the Document AI API

Before you can begin using Document AI, you must enable the API.

1. Using the Search Bar at the top of the console, search for "Document AI API", then click **Enable** to use the API in your Google Cloud project.


Cloud Document AI API
Google Enterprise API
Service to parse structured information from unstructured or semi-structured documents using...
MANAGE
TRY THIS API
API Enabled

2. Use the search bar to search for "Cloud Storage API", if not already enabled, click **Enable**.

3. Alternatively, the APIs can be enabled using the following `gcloud` commands.

gcloud services enable documentai.googleapis.com gcloud services enable storage.googleapis.com

You should see something like this:

Operation "operations/..." finished successfully.

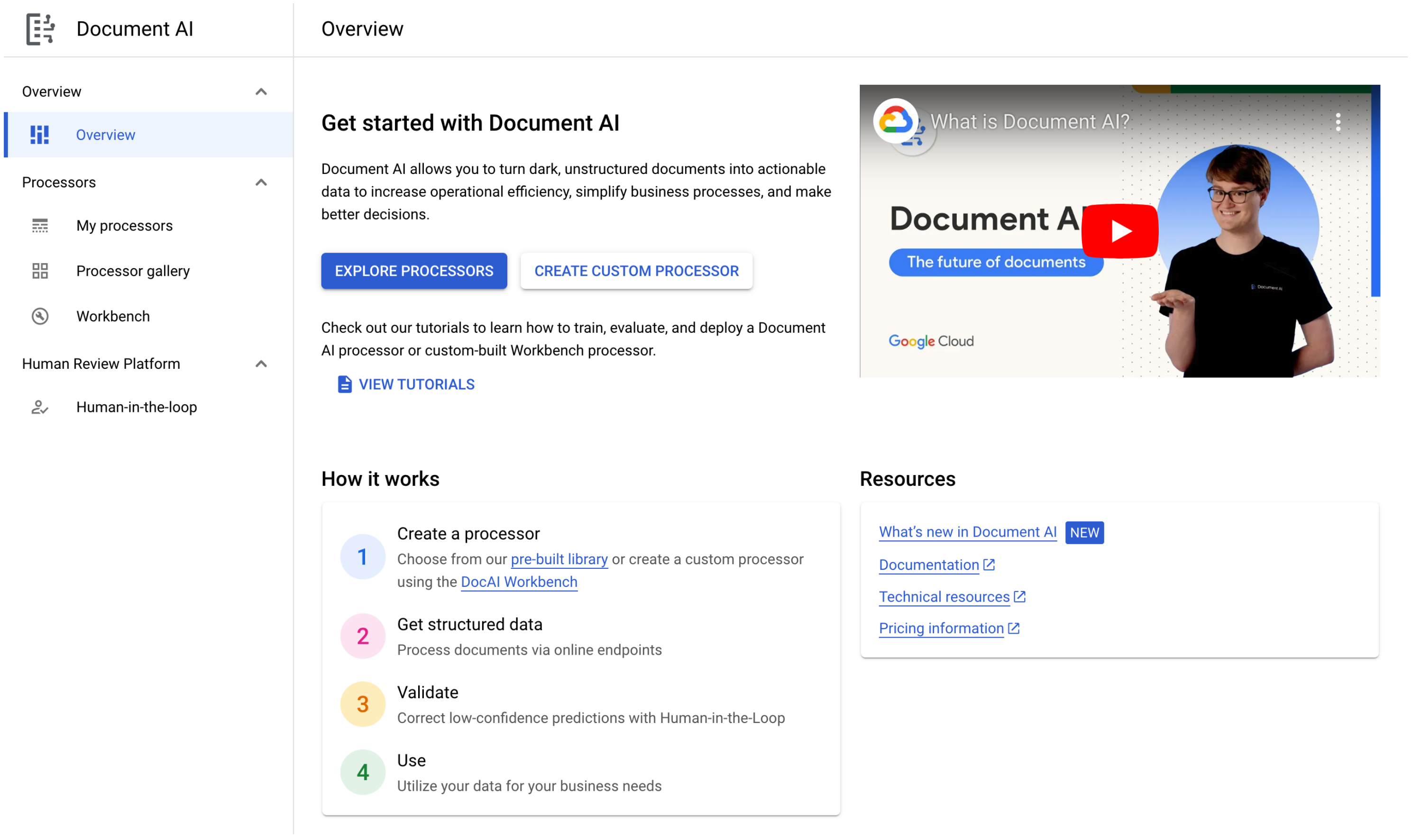
Now, you can use Document AI!

Click **Check my progress** to verify the objective. Enable the Document AI API

Task 2. Create and test a processor

You must first create an instance of the Document OCR processor that will perform the extraction. This can be completed using the Cloud Console or the Processor Management API.

1. From the Navigation Menu, under **Artificial Intelligence**, select **Document AI**.



Document AI

Overview

Overview
Overview
Processors
My processors
Processor gallery
Workbench
Human Review Platform
Human-in-the-loop

Get started with Document AI

Document AI allows you to turn dark, unstructured documents into actionable data to increase operational efficiency, simplify business processes, and make better decisions.

EXPLORE PROCESSORS CREATE CUSTOM PROCESSOR

Check out our tutorials to learn how to train, evaluate, and deploy a Document AI processor or custom-built Workbench processor.

VIEW TUTORIALS

How it works

1 Create a processor
Choose from our pre-built library or create a custom processor using the DocAI Workbench

2 Get structured data
Process documents via online endpoints

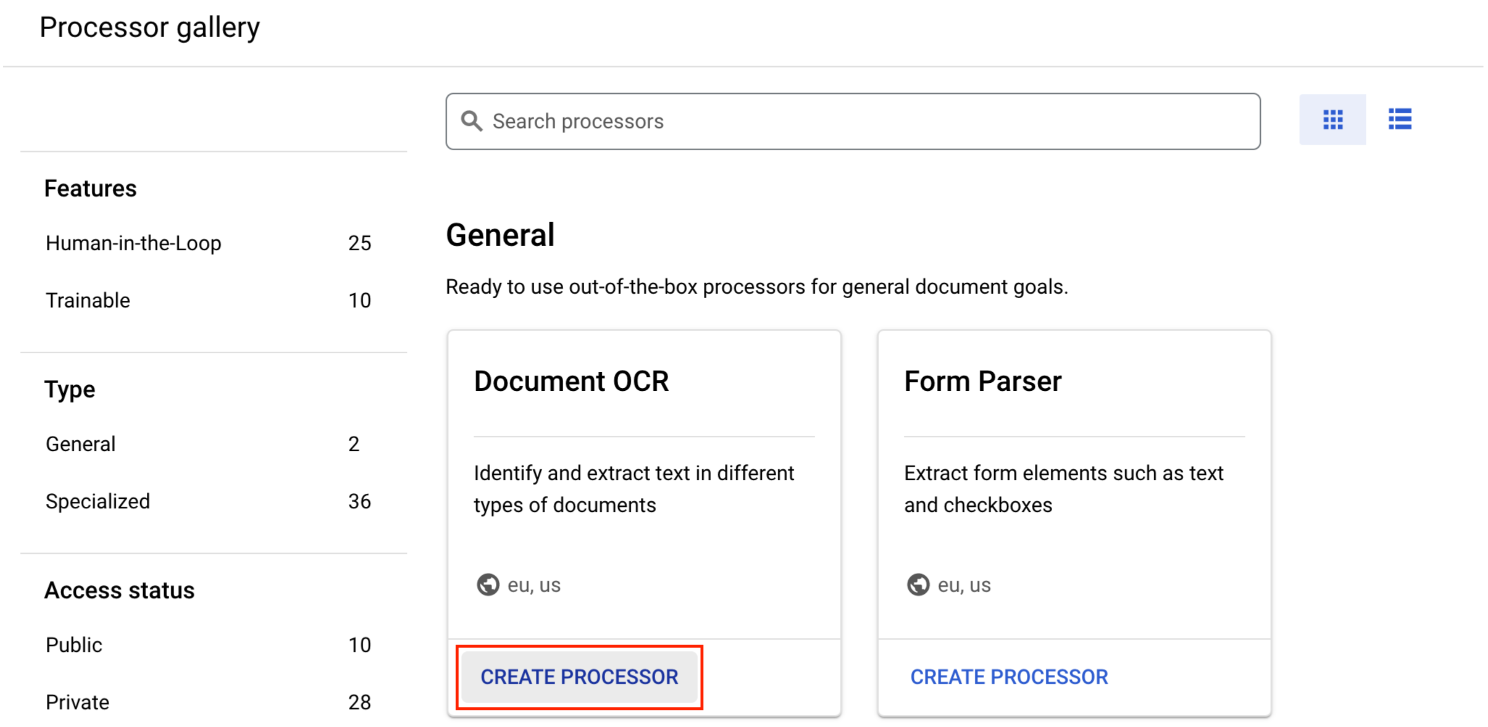
3 Validate
Correct low-confidence predictions with Human-in-the-Loop

4 Use
Utilize your data for your business needs

Resources

What's new in Document AI NEW
Documentation
Technical resources
Pricing information

2. Click **Explore Processors**, and inside **Document OCR**, click **Create Processor**.



Processor gallery

Search processors

Features	
Human-in-the-Loop	25
Trainable	10

Type	
General	2
Specialized	36

Access status	
Public	10
Private	28

General

Ready to use out-of-the-box processors for general document goals.

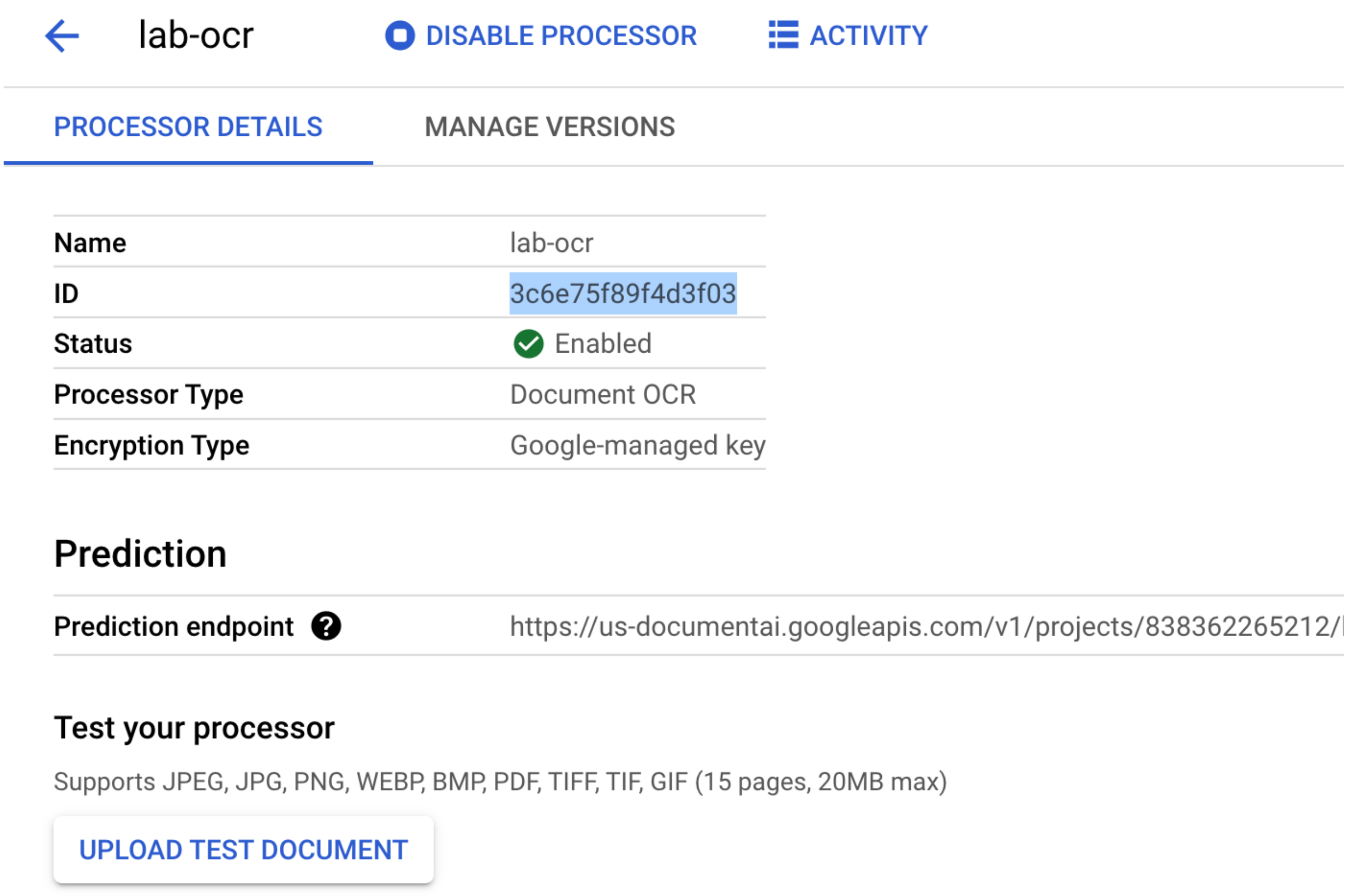
Document OCR
Identify and extract text in different types of documents
eu, us
CREATE PROCESSOR

Form Parser
Extract form elements such as text and checkboxes
eu, us
CREATE PROCESSOR

3. Give it the name `lab-ocr` and select the closest region on the list.

4. Click **Create** to create your processor

5. **Copy** your Processor ID. You must use this in your code later.



Name	lab-ocr
ID	3c6e75f89f4d3f03
Status	Enabled
Processor Type	Document OCR
Encryption Type	Google-managed key

Prediction

Prediction endpoint	https://us-documentai.googleapis.com/v1/projects/838362265212/

Test your processor

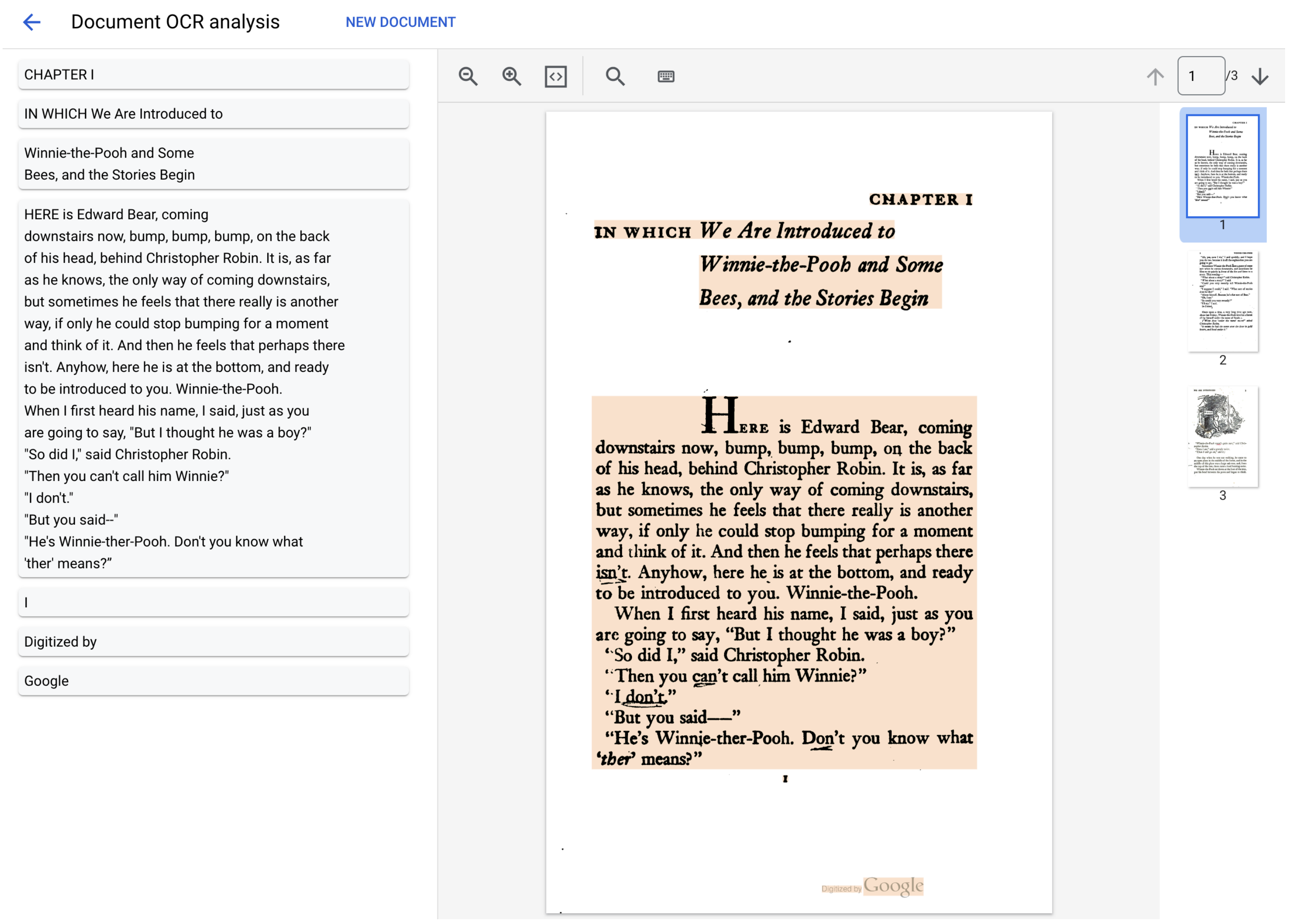
Supports JPEG, JPG, PNG, WEBP, BMP, PDF, TIFF, TIF, GIF (15 pages, 20MB max)

6. Download the PDF file below, which contains the first 3 pages of the novel "Winnie the Pooh" by A.A. Milne.

Now you can test out your processor in the console by uploading a document.

7. Click **Upload Test Document** and select the PDF file you downloaded.

Your output should look this:


Document OCR analysis NEW DOCUMENT

1 /3

CHAPTER I

IN WHICH We Are Introduced to

Winnie-the-Pooh and Some
Bees, and the Stories Begin

HERE is Edward Bear, coming
downstairs now, bump, bump, bump, on the back
of his head, behind Christopher Robin. It is, as far
as he knows, the only way of coming downstairs,
but sometimes he feels that there really is another
way, if only he could stop bumping for a moment
and think of it. And then he feels that perhaps there
isn't. Anyhow, here he is at the bottom, and ready
to be introduced to you. Winnie-the-Pooh.
When I first heard his name, I said, just as you
are going to say, "But I thought he was a boy?"
"So did I," said Christopher Robin.
"Then you can't call him Winnie?"
"I don't."
"But you said--"
"He's Winnie-ther-Pooh. Don't you know what
'ther' means?"

I

Digitized by

Google

Click **Check my progress** to verify the objective. Create and test a processor

Task 3. Authenticate API requests

In order to make requests to the Document AI API, you must use a *Service Account*. A *Service Account* belongs to your project and it is used by the Python Client library to make API requests. Like any other user account, a service account is represented by an email address. In this section, you will use the *Cloud SDK* to create a service account and then create credentials you need to authenticate as the service account.

1. First, open a new Cloud Shell window and set an environment variable with your Project ID by running the following command:

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

2. Next, create a new service account to access the Document AI API by using:

```
gcloud iam service-accounts create my-docai-sa \ --display-name "my-docai-service-account"
```

3. Give your service account permissions to access Document AI, Cloud Storage, and Service Usage by using the following commands:

```
gcloud projects add-iam-policy-binding ${GOOGLE_CLOUD_PROJECT} \ --member="serviceAccount:my-
docai-sa@${GOOGLE_CLOUD_PROJECT}.iam.gserviceaccount.com" \ --role="roles/documentai.admin"
gcloud projects add-iam-policy-binding ${GOOGLE_CLOUD_PROJECT} \ --member="serviceAccount:my-
docai-sa@${GOOGLE_CLOUD_PROJECT}.iam.gserviceaccount.com" \ --role="roles/storage.admin" gcloud
projects add-iam-policy-binding ${GOOGLE_CLOUD_PROJECT} \ --member="serviceAccount:my-docai-
sa@${GOOGLE_CLOUD_PROJECT}.iam.gserviceaccount.com" \ --
role="roles/serviceusage.serviceUsageConsumer"
```

4. Create credentials that your Python code uses to login as your new service account. Create these credentials and save it as a JSON file `~/key.json` by using the following command:

```
gcloud iam service-accounts keys create ~/key.json \ --iam-account my-docai-
sa@${GOOGLE_CLOUD_PROJECT}.iam.gserviceaccount.com
```

5. Finally, set the `GOOGLE_APPLICATION_CREDENTIALS` environment variable, which is used by the library to find your credentials. The environment variable should be set to the full path of the credentials JSON file you created, by using:

```
export GOOGLE_APPLICATION_CREDENTIALS=$(realpath key.json)
```

To read more about this form of authentication, see the guide.

Click **Check my progress** to verify the objective. Authenticate API requests

Task 4. Install the client library

1. In Cloud Shell, run the following commands to install the Python client libraries for Document AI and Cloud Storage.

```
pip3 install --upgrade google-cloud-documentai pip3 install --upgrade google-cloud-storage
```

You should see something like this:

```
... Installing collected packages: google-cloud-documentai Successfully installed google-cloud-documentai-
1.4.1 . . Installing collected packages: google-cloud-storage Successfully installed google-cloud-storage-1.43.0
```

Now, you're ready to use the Document AI API!

Note: If you're setting up your own Python development environment, you can follow these guidelines.

Upload the sample PDF to Cloud Shell

1. From the Cloud Shell toolbar, click the three dots and select **Upload**.

2. Select **File** > **Choose Fils** and select the 3-page PDF file you downloaded earlier.

3. Click **Upload**.

4. **Alternatively**, you can also download the PDF from a public Google Cloud Storage Bucket using `gcloud storage cp`.

```
gcloud storage cp gs://cloud-samples-data/documentai/codelabs/ocr/Winnie_the_Pooh_3_Pages.pdf .
```

Task 5. Make an online processing request

In this step, you'll process the first 3 pages of the novel using the online processing (synchronous) API. This method is best suited for smaller documents that are stored locally. Check out the full processor list for the maximum pages and file size for each processor type.

1. In Cloud Shell, create a file called `online_processing.py` and paste the following code into it:

```
from google.api_core.client_options import ClientOptions from google.cloud import documentai_v1 as
documentai PROJECT_ID = "YOUR_PROJECT_ID" LOCATION = "YOUR_PROJECT_LOCATION" # Format
is 'us' or 'eu' PROCESSOR_ID = "YOUR_PROCESSOR_ID" # Create processor in Cloud Console # The local
file in your current working directory FILE_PATH = "Winnie_the_Pooh_3_Pages.pdf" # Refer to
https://cloud.google.com/document-ai/docs/file-types # for supported file types MIME_TYPE = "application/pdf"
# Instantiates a client docai_client = documentai.DocumentProcessorServiceClient(
client_options=ClientOptions(api_endpoint=f"{LOCATION}-documentai.googleapis.com") ) # The full resource
name of the processor, e.g.: # projects/project-id/locations/location/processor/processor-id # You must create
new processors in the Cloud Console first RESOURCE_NAME = docai_client.processor_path(PROJECT_ID,
LOCATION, PROCESSOR_ID) # Read the file into memory with open(FILE_PATH, "rb") as image:
image_content = image.read() # Load Binary Data into Document AI RawDocument Object raw_document =
documentai.RawDocument(content=image_content, mime_type=MIME_TYPE) # Configure the process
request request = documentai.ProcessRequest(name=RESOURCE_NAME, raw_document=raw_document) #
Use the Document AI client to process the sample form result =
docai_client.process_document(request=request) document_object = result.document print("Document
processing complete.") print(f"Text: {document_object.text}")
```

2. Replace `YOUR_PROJECT_ID`, `YOUR_PROJECT_LOCATION`, `YOUR_PROCESSOR_ID`, and the `FILE_PATH` with appropriate values for your environment.

Note that the `FILE_PATH` is the name of the file you uploaded to Cloud Shell in the previous step. If you didn't rename the file, it should be `Winnie_the_Pooh_3_Pages.pdf` which is the default value and doesn't need to be changed.

3. Run the code, which will extract the text and print it to the console.

```
python3 online_processing.py
```

You should see the following output:

```
Document processing complete. Text: IN WHICH We Are Introduced to CHAPTER I Winnie-the-Pooh and
Some Bees, and the Stories Begin HERE is Edward Bear, coming downstairs now, bump, bump, bump, on the
back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but
```

sometimes he feels that there really is another way, if only he could stop bumping for a moment and think of it. And then he feels that perhaps there isn't. Anyhow, here he is at the bottom, and ready to be introduced to you. Winnie-the-Pooh. When I first heard his name, I said, just as you are going to say, "But I thought he was a boy?" "So did I," said Christopher Robin. "Then you can't call him Winnie?" "I don't." "But you said--" ...

Task 6. Make a batch processing request

Now, suppose that you want to read in the text from the entire novel.

- Online Processing has limits on the number of pages and file size that can be sent and it only allows for one document file per API call.
- Batch Processing allows for processing of larger/multiple files in an asynchronous method.

In this section, you will process the entire "Winnie the Pooh" novel with the Document AI Batch Processing API and output the text into a Google Cloud Storage Bucket.

Batch processing uses Long Running Operations to manage requests in an asynchronous manner, so we have to make the request and retrieve the output in a different manner than online processing. However, the output will be in the same `Document` object format whether using online or batch processing.

This section shows how to provide specific documents for Document AI to process. A later section will show how to process an entire directory of documents.

Upload PDF to Cloud Storage

The `batch_process_documents()` method currently accepts files from Google Cloud Storage. You can reference `documentai_v1.types.BatchProcessRequest` for more information on the object structure.

1. Run the following command to create a Google Cloud Storage Bucket to store the PDF file and upload the PDF file to the bucket:

```
gcloud storage buckets create gs://$GOOGLE_CLOUD_PROJECT gcloud storage cp gs://cloud-samples-
data/documentai/codelabs/ocr/Winnie_the_Pooh.pdf gs://$GOOGLE_CLOUD_PROJECT/
```

Using the batch_process_documents() method

1. Create a file called `batch_processing.py` and paste in the following code:

```
import re from typing import List from google.api_core.client_options import ClientOptions from google.cloud
import documentai_v1 as documentai from google.cloud import storage PROJECT_ID =
"YOUR_PROJECT_ID" LOCATION = "YOUR_PROJECT_LOCATION" # Format is 'us' or 'eu'
PROCESSOR_ID = "YOUR_PROCESSOR_ID" # Create processor in Cloud Console # Format
'gs://input_bucket/directory/file.pdf' GCS_INPUT_URI = "gs://cloud-samples-
data/documentai/codelabs/ocr/Winnie_the_Pooh.pdf" INPUT_MIME_TYPE = "application/pdf" # Format
'gs://output_bucket/directory' GCS_OUTPUT_URI = "YOUR_OUTPUT_BUCKET_URI" # Instantiates a client
docai_client = documentai.DocumentProcessorServiceClient( client_options=ClientOptions(api_endpoint=f"
{LOCATION}-documentai.googleapis.com") ) # The full resource name of the processor, e.g.: # projects/project-
```

```
id/locations/location/processor/processor-id # You must create new processors in the Cloud Console first
RESOURCE_NAME = docai_client.processor_path(PROJECT_ID, LOCATION, PROCESSOR_ID) # Cloud
Storage URI for the Input Document input_document = documentai.GcsDocument( gcs_uri=GCS_INPUT_URI,
mime_type=INPUT_MIME_TYPE ) # Load GCS Input URI into a List of document files input_config =
documentai.BatchDocumentsInputConfig( gcs_documents=documentai.GcsDocuments(documents=
[input_document]) ) # Cloud Storage URI for Output directory gcs_output_config =
documentai.DocumentOutputConfig.GcsOutputConfig( gcs_uri=GCS_OUTPUT_URI ) # Load GCS Output URI
into OutputConfig object output_config =
documentai.DocumentOutputConfig(gcs_output_config=gcs_output_config) # Configure Process Request
request = documentai.BatchProcessRequest( name=RESOURCE_NAME, input_documents=input_config,
document_output_config=output_config, ) # Batch Process returns a Long Running Operation (LRO) operation
= docai_client.batch_process_documents(request) # Continually polls the operation until it is complete. # This
could take some time for larger files # Format:
projects/PROJECT_NUMBER/locations/LOCATION/operations/OPERATION_ID print(f"Waiting for operation
{operation.operation.name} to complete...") operation.result() # NOTE: Can also use callbacks for
asynchronous processing # # def my_callback(future): # result = future.result() # #
operation.add_done_callback(my_callback) print("Document processing complete.") # Once the operation is
complete, # get output document information from operation metadata metadata =
documentai.BatchProcessMetadata(operation.metadata) if metadata.state !=
documentai.BatchProcessMetadata.State.SUCCEEDED: raise ValueError(f"Batch Process Failed:
{metadata.state_message}") documents: List[documentai.Document] = [] # Storage Client to retrieve the output
files from GCS storage_client = storage.Client() # One process per Input Document for process in
metadata.individual_process_statuses: # output_gcs_destination format:
gs://BUCKET/PREFIX/OPERATION_NUMBER/0 # The GCS API requires the bucket name and URI prefix
separately output_bucket, output_prefix = re.match( r"gs://(.*?)/(.*)", process.output_gcs_destination ).groups()
# Get List of Document Objects from the Output Bucket output_blobs = storage_client.list_blobs(output_bucket,
prefix=output_prefix) # DocAI may output multiple JSON files per source file for blob in output_blobs: #
Document AI should only output JSON files to GCS if ".json" not in blob.name: print(f"Skipping non-supported
file type {blob.name}") continue print(f"Fetching {blob.name}") # Download JSON File and Convert to Document
Object document = documentai.Document.from_json( blob.download_as_bytes(), ignore_unknown_fields=True
) documents.append(document) # Print Text from all documents # Truncated at 100 characters for brevity for
document in documents: print(document.text[:100])
```

2. Replace `YOUR_PROJECT_ID`, `YOUR_PROJECT_LOCATION`, `YOUR_PROCESSOR_ID`, `GCS_INPUT_URI` and `GCS_OUTPUT_URI` with the appropriate values for your environment.

    - For `GCS_INPUT_URI`, use the URI of the file you uploaded to your bucket in the previous step, i.e `gs://qwiklabs-gcp-02-47d26g8c5abe/Winnie_the_Pooh.pdf`. (Make sure to replace the bucket name with your own bucket name.)
    - For `GCS_OUTPUT_URI`, use the URI of the bucket you created in the previous step, i.e `gs://qwiklabs-gcp-02-47d26g8c5abe`. (Make sure to replace the bucket name with your own bucket name.)

3. Run the code, and you should see the full novel text extracted and printed in your console.

python3 batch_processing.py **Note:** this may take some time to complete as the file is much larger than the previous example. However, with the Batch Processing API, you will receive an Operation ID which can be used to get the output from Cloud Storage once the task is completed.

Your output should look something like this:

Document processing complete. Fetching 16218185426403815298/0/Winnie_the_Pooh-0.json Fetching 16218185426403815298/0/Winnie_the_Pooh-1.json Fetching 16218185426403815298/0/Winnie_the_Pooh-10.json Fetching 16218185426403815298/0/Winnie_the_Pooh-11.json Fetching 16218185426403815298/0/Winnie_the_Pooh-12.json Fetching 16218185426403815298/0/Winnie_the_Pooh-13.json Fetching 16218185426403815298/0/Winnie_the_Pooh-14.json Fetching 16218185426403815298/0/Winnie_the_Pooh-15.json .. This is a reproduction of a library book that was digitized by Google as part of an ongoing effort t 0 TAM MTAA Digitized by Google Introduction (I$_2$ F YOU happen to have read another book about Christo 84 Eeyore took down his right hoof from his right ear, turned round, and with great difficulty put u 94 ..

Great! You have now successfully extracted text from a PDF file using the Document AI Batch Processing API.

Click **Check my progress** to verify the objective. Make a batch processing request

Task 7. Make a batch processing request for a directory

Sometimes, you may want to process an entire directory of documents, without listing each document individually. The `batch_process_documents()` method supports input of a list of specific documents or a directory path.

In this section, you will learn how to process a full directory of document files. Most of the code is the same as the previous step, the only difference is the GCS URI sent with the `BatchProcessRequest`.

1. Run the following command to copy the sample directory (which contains multiple pages of the novel in separate files) to your Cloud Storage bucket.

gcloud storage cp --recursive gs://cloud-samples-data/documentai/codelabs/ocr/multi-document gs://$GOOGLE_CLOUD_PROJECT/ **Note:** you can ignore any errors with `no attribute 'default_encryption_key'`.

You can read the files directly or copy them into your own Cloud Storage bucket.

1. Create a file called `batch_processing_directory.py` paste in the following code:

import re from typing import List from google.api_core.client_options import ClientOptions from google.cloud import documentai_v1 as documentai from google.cloud import storage PROJECT_ID = "YOUR_PROJECT_ID" LOCATION = "YOUR_PROJECT_LOCATION" # Format is 'us' or 'eu' PROCESSOR_ID = "YOUR_PROCESSOR_ID" # Create processor in Cloud Console # Format 'gs://input_bucket/directory' GCS_INPUT_PREFIX = "gs://cloud-samples-data/documentai/codelabs/ocr/multi-document" # Format 'gs://output_bucket/directory' GCS_OUTPUT_URI = "YOUR_OUTPUT_BUCKET_URI" # Instantiates a client docai_client = documentai.DocumentProcessorServiceClient(

```
client_options=ClientOptions(api_endpoint=f"{LOCATION}-documentai.googleapis.com") ) # The full resource
name of the processor, e.g.: # projects/project-id/locations/location/processor/processor-id # You must create
new processors in the Cloud Console first RESOURCE_NAME = docai_client.processor_path(PROJECT_ID,
LOCATION, PROCESSOR_ID) # Cloud Storage URI for the Input Directory gcs_prefix =
documentai.GcsPrefix(gcs_uri_prefix=GCS_INPUT_PREFIX) # Load GCS Input URI into Batch Input Config
input_config = documentai.BatchDocumentsInputConfig(gcs_prefix=gcs_prefix) # Cloud Storage URI for Output
directory gcs_output_config = documentai.DocumentOutputConfig.GcsOutputConfig(
gcs_uri=GCS_OUTPUT_URI ) # Load GCS Output URI into OutputConfig object output_config =
documentai.DocumentOutputConfig(gcs_output_config=gcs_output_config) # Configure Process Request
request = documentai.BatchProcessRequest( name=RESOURCE_NAME, input_documents=input_config,
document_output_config=output_config, ) # Batch Process returns a Long Running Operation (LRO) operation
= docai_client.batch_process_documents(request) # Continually polls the operation until it is complete. # This
could take some time for larger files # Format:
projects/PROJECT_NUMBER/locations/LOCATION/operations/OPERATION_ID print(f"Waiting for operation
{operation.operation.name} to complete...") operation.result() # NOTE: Can also use callbacks for
asynchronous processing # # def my_callback(future): # result = future.result() # #
operation.add_done_callback(my_callback) print("Document processing complete.") # Once the operation is
complete, # get output document information from operation metadata metadata =
documentai.BatchProcessMetadata(operation.metadata) if metadata.state !=
documentai.BatchProcessMetadata.State.SUCCEEDED: raise ValueError(f"Batch Process Failed:
{metadata.state_message}") documents: List[documentai.Document] = [] # Storage Client to retrieve the output
files from GCS storage_client = storage.Client() # One process per Input Document for process in
metadata.individual_process_statuses: # output_gcs_destination format:
gs://BUCKET/PREFIX/OPERATION_NUMBER/0 # The GCS API requires the bucket name and URI prefix
separately output_bucket, output_prefix = re.match( r"gs://(.*?)/(.*)", process.output_gcs_destination ).groups()
# Get List of Document Objects from the Output Bucket output_blobs = storage_client.list_blobs(output_bucket,
prefix=output_prefix) # DocAI may output multiple JSON files per source file for blob in output_blobs: #
Document AI should only output JSON files to GCS if ".json" not in blob.name: print(f"Skipping non-supported
file type {blob.name}") continue print(f"Fetching {blob.name}") # Download JSON File and Convert to Document
Object document = documentai.Document.from_json( blob.download_as_bytes(), ignore_unknown_fields=True
) documents.append(document) # Print Text from all documents # Truncated at 100 characters for brevity for
document in documents: print(document.text[:100])
```

2. Replace the `PROJECT_ID`, `LOCATION`, `PROCESSOR_ID`, `GCS_INPUT_PREFIX` and `GCS_OUTPUT_URI` with the appropriate values for your environment.

    - For `GCS_INPUT_PREFIX`, use the URI of the **directory** you uploaded to your bucket in the previous section, i.e `gs://qwiklabs-gcp-02-47d26g8c5abe/multi-document`. (Make sure to replace the bucket name with your own bucket name.)
    - For `GCS_OUTPUT_URI`, use the URI of the bucket you created in the previous section, i.e `gs://qwiklabs-gcp-02-47d26g8c5abe`. (Make sure to replace the bucket name with your own bucket name.)

3. Use the following command to run the code, and you should see the extracted text from all of the document files in the Cloud Storage directory.

```
python3 batch_processing_directory.py
```

Your output should look something like this:

```
Document processing complete. Fetching 16354972755137859334/0/Winnie_the_Pooh_Page_0-0.json
Fetching 16354972755137859334/1/Winnie_the_Pooh_Page_1-0.json Fetching
16354972755137859334/2/Winnie_the_Pooh_Page_10-0.json .. Introduction (I₂ F YOU happen to have read
another book about Christopher Robin, you may remember th IN WHICH We Are Introduced to CHAPTER I
Winnie-the-Pooh and Some Bees, and the Stories Begin HERE is ..
```

Great! You've successfully used the Document AI Python Client Library to process a directory of documents using a Document AI Processor, and output the results to Cloud Storage.

Click **Check my progress** to verify the objective. Make a batch processing request for a directory

Congratulations!

Congratulations! In this lab, you learned how to use the Document AI Python Client Library to process a directory of documents using a Document AI Processor, and output the results to Cloud Storage. You also learned how to authenticate API requests using a service account key file, install the Document AI Python Client Library, and use the Online (Synchronous) and Batch (Asynchronous) APIs to process requests.

Next steps/Learn more

Check out the following resources to learn more about Document AI and the Python Client Library:

- The Future of Documents - YouTube Playlist
- Document AI Documentation
- Document AI Python Client Library
- Document AI Samples Repository

Google Cloud training and certification

...helps you make the most of Google Cloud technologies. Our classes include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. Certifications help you validate and prove your skill and expertise in Google Cloud technologies.

Manual Last Updated: August 25, 2023

Lab Last Tested: August 25, 2023

Copyright 2023 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are

associated.