# Medical Image De-Identification (MIDI) Standard Operation Procedure

Update: 11/27/2024

Author: Ben Kopchick

This article serves as a Standard Operating Procedure (SOP) for running the Medical Image De-Identification (MIDI) Pipeline hosted in the CBIIT managed Google Cloud Platform (GCP) environment. The audience of this SOP are members with access to the MIDI pipeline and approved to run the pipeline.

## Requirement/Prerequisites

* The current GCP account the pipeline is setup in is *nih-nci-cbiit-midi-dev2*
  + To access the MIDI pipeline hosted on CBIIT’s GCP platform, users must have access to GCP and the [nih\_nci\_cbiit\_imaging\_informatics\_midi\_dev\_folder\_admin@nih.gov](mailto:nih_nci_cbiit_imaging_informatics_midi_dev_folder_admin@nih.gov) distribution list.
  + For access, a request can be made through a [Google Cloud Service Request](https://service.cancer.gov/ncisp?id=nci_sc_cat_item&sys_id=1d7996f61b818910f360a681f54bcb31).
* User will require the following IAM permissions either through the user list or a service account:
  + Storage Admin
  + Cloud Functions Invoker
  + Cloud Run Invoker
  + Vertex AI Service Agent
* The required APIs that need activated:
  + Cloud Run Functions
  + Healthcare API
  + Bigquery

## Getting Started – Account Setup

If you are an NCI employee looking to deploy this within the NCI Cloud Environment, please submit a consultation request using the [Google Cloud Service Request](https://service.cancer.gov/ncisp?id=nci_sc_cat_item&sys_id=1d7996f61b818910f360a681f54bcb31). The Cloud Engineering team will assist you in setting up the environment with the listed above permissions and required APIs.

If you are using a personal account, please proceed to [Cloud Computing Services | Google Cloud](https://cloud.google.com/?hl=en) to setup a free account. If you are new to GCP, we highly recommend you take some tutorials to familiarize yourself with GCP console and terminology. In your personal account you will have administrator privileges and will be able to proceed to activating the necessary APIs. Otherwise, you will need the proper IAM permissions listed in the Requirements section. To activate APIs, click the hamburger button in the top left and select “View All Products”. Then find “Cloud Run functions” in the Serverless section and select it. It is also recommended to pin the product for easy access later. Then select the “Enable” button to enable the product. Proceed to do this with the “Healthcare API” and “Bigquery”.

The final step is to copy the pipeline functions in to Cloud Run.

1. In “Cloud Run functions” select “Create Function”. Give function a name such as *De-identification*.
2. Select the Region you would like it to run in. For NCI, it should use us-east4 (Northern Virginia).
3. Trigger will be HTTPS, though other triggers can be explored.
4. In “Runtime”, select the necessary settings for Memory and CPU. We recommend with 1GB memory and 1 CPU to start with. This can be edited in the future as needed.
5. Set “Timeout” to the maximum setting of 3600 seconds.
   1. This determines how long a job can run in Cloud Functions. Since the process can take a long time, we maximize this. In addition, this determines how you will need to split up your jobs depending on number of files you are de-identifying.
6. Concurrency should be 1
   1. This determines how many requests can run on the same instance. To run in parallel, we want this to be 1 so that new instances are spun up instead of attempting to run on the same instance.
7. “Autoscaling” we recommend to have a minimum of 0 and a maximum of 100.
   1. This allows 100 instances to run at the same time for this function.
8. “Runtime Service Account” should be the accounts service account that has the minimum permissions to run the pipelines and the user should also have permission to run as the service account.
   1. More details about service accounts can be found here: <https://cloud.google.com/iam/docs/service-account-overview.>

Click next to go to Code. Select a version of Python, anything Python 3.8 or greater will work. Copy over *main.py, requirements.txt,* and *script.py*. In *main.py* make the necessary edits to “project” and “location”. Other variables are described below. The *requirements.txt* and *script.py* do not need to be edited. Please proceed below to run the pipeline.

## Procedure

The pipeline is run through two GCP Cloud Run Functions. See below for more information.

1. *De-identification*
2. *Post-processing*

The functions take the process through linear steps of:

1. Storage
2. Heatlhcare DICOM Dataset/Datastore
3. Healthcare De-Identification
4. De-identified Healthcare DICOM Dataset/Datastore and FHIR store
5. De-identified Storage
6. Post-processing

### Steps

#### Bucket Creation

1. Go to **Cloud Storage** in the GCP console
2. DICOM data needs to be uploaded into a bucket. At NCI, created buckets should be in a *single region* in *us-east4*.
   1. Data can be uploaded by dragging and dropping or through an API command.
   2. For purposes of this SOP, we will use the bucket *midi-storage-input*
3. DICOM output bucket needs to be created.
   1. For purposes of this SOP, we will use the bucket *midi-storage-output*
4. FHIR output bucket needs to be created.
   1. For purposes of this SOP, we will use the bucket *fhir\_output*

#### Cloud Run Function 1: De-Identification using Healthcare API

1. Go to **Cloud Run functions** in the GCP console
2. Select *de-identification-v1*
3. Go to the **Source** tab
4. In the main.py file check that all variables are correct

Graphical user interface, text, application

Description automatically generated

* 1. project: Name of the GCP project
  2. location: Location of your buckets and Healthcare stores
     1. location should be ‘us-east4’
  3. source\_dataset: Name of Healthcare datastore you want created to upload raw data
     1. *Cannot be already existing*
  4. destination\_dataset: Name of Healthcare datastore de-identified data will be exported to
     1. *Cannot already be existing*
  5. source\_bucket: Bucket created in Bucket Creation 2 where raw data was uploaded
     1. The ‘/\*\*’ at end of bucket takes all files in bucket for de-identification
     2. You can add a path to file or path to folder in bucket to not take all files
        1. Ex. source\_bucket = ‘midi-storage-input/patient1/\*\*’
  6. destination\_bucket: Bucket created in Bucket Creation 3
  7. fhir\_destination\_bucket: Bucket you want fhir output to be exported to
  8. fhir\_prefix: Folder the fhir output will be placed in in the fhir\_destination\_bucket
  9. fhir\_bucket: Ignore
  10. fhirname: Healthcare FHIR store created when script is run
      1. *Cannot already be existing*
  11. mapping\_output: Folder inside fhir\_destination\_bucket where UID mapping file will reside

1. Variables can be edited by selecting **EDIT**
2. After variables are correct, select **SAVE AND REDEPLOY**
   1. This will take a couple of minutes
3. To run the pipeline open the Cloud Shell or a terminal in an AI Notebook
4. Run the following curl command:

*curl -m 3610 -X POST https://us-east4-nih-nci-cbiit-midi-dev2.cloudfunctions.net/de-identification-v1 \*

*-H "Authorization: bearer $(gcloud auth print-identity-token)" \*

*-H "Content-Type: application/json" \*

*-d '{*

*"name": "Hello World"*

*}'*

1. This can be found in the **Testing** tab under CLI test command.

Graphical user interface, text, application, email

Description automatically generated

1. You can follow the progress of the script under the **Logs** tab

Graphical user interface, text, application, email

Description automatically generated

1. After successful completion, check the destination buckets for the final outputs
2. Files can be checked by navigating to a Jupyter Notebook environment.
   1. **VertexAI** > **Workbench** > **User-Managed Notebooks**

Graphical user interface, application

Description automatically generated

* 1. In the *analysis-midi-phase4* notebook, we have a demo.ipynb notebook that showcases a quick way to look at some of the images using pydicom

Graphical user interface

Description automatically generated

#### Cloud Run Function 2: Post Processing Script

1. Go to **Cloud Run functions** in the GCP console
2. Select *post-processing-v1*
3. Go to the **Source** tab
4. In the main.py file check that all variables are correct

Graphical user interface, text, application

Description automatically generated

* 1. bucketname: Bucket where your results are from *de-identification-v1 run*
  2. folder\_in: Folder in bucket where files are located from *de-identification-v1 run*
  3. folder\_out: Name of folder you want postprocessing results to end up in

1. Variables can be edited by selecting **EDIT**
2. After variables are correct, select **SAVE AND REDEPLOY**
   1. This will take a couple of minutes
3. To run the pipeline open the Cloud Shell or a terminal in an AI Notebook
4. Run the following curl command:

*curl -m 3610 -X POST https://us-east4-nih-nci-cbiit-midi-dev2.cloudfunctions.net/post-processing-v1 -H "Authorization: bearer $(gcloud auth print-identity-token)" -H "Content-Type: application/json" -d '{*

*"name": "Hello World", "filestart": "0", "fileend": "1"*

*}'*

1. Edit the filestart and fileend variables to match with which files you want to use. This can be used to parallelize the runs. For example: If you have 500 files, you can run in one terminal *"filestart": "0", "fileend": "249"* and in another terminal *"filestart": "250", "fileend": "500"* to complete the task twice as fast.
2. filestart: 0 – Integer that determines which file to start with
3. fileend: 1 – Integer that determines which file to end with
4. You can follow the progress of the script under the **Logs** tab
5. Check files in bucketname/folder\_out to make sure all files are present.

## Code Deep Dive

The code is split into 3 python scipts: *main.py*, *script.py*, and *requirements.txt*.

#### main.py

A picture containing text

Description automatically generated

* This code is where all the variables for bucket names and datasets are defined and are found near the top after the package imports.
* The code then follows through a step-by-step process to create the necessary datasets and dicom/fhir stores. It then imports data from the defined source bucket. The deidentify\_dataset() function then runs the de-identification script that calls the Healthcare API function.
* Data is then exported to the de-identification bucket and fhir output bucket.
* A UID mapping is created and put in the fhir output bucket

#### script.py

Text

Description automatically generated

script.py contains all the functions that are called by main.py.

The main part of the script is the deidentify\_dataset function. This contains the Healthcare API call that is configured to this project. It is important to note the structure of this request under the dicomTagConfig options:

‘dicomTagConfig’ : {

“actions” : [

{“queries”: [

‘List of DICOM tags’, ‘Ex. 00091008’, ‘DT’

],

“action”: {}

}

]

}

The red is what is customized. Actions can include: keepTag, removeTag, resetTag, cleanTextTag, cleanImageTag, recurseTag, and regenUidTag. In addition, we specify a profile type of *DEIDENTIFY\_TAG\_CONTENTS*’ and that we are collecting the metadata of changed files through the fhir store by activating the *operationMetadata* option. More details about all these actions can be found here: <https://cloud.google.com/healthcare-api/docs/how-tos/dicom-deidentify-dicomtagconfig>

Table

Description automatically generated