MLOps scope and deliverables

# Specifications

* Develop three training pipelines that can be triggered to automate the training process for DocAI CDC, DocAI CDE and custom object detection model.
* The training pipeline will be run on Vertex AI pipelines as a custom training job.
* The training pipeline will be composed of multiple components like Preprocessing, Training, Evaluation, etc. which can be reused.
* The training pipeline will update the experiment details and results from the training on Big Query for tracking and lineage identification.
* The user will trigger the training pipeline using Cloud Composer (DAG). The user will update the configuration via environment variables and trigger the DAG (either through the Airflow UI in composer or via command line)
* The deployment of the trained models (CDC, CDE or Custom OD model) post training will not be taken up by the pipeline.

# Services that will be used

* Cloud Composer
* Vertex AI pipelines
* Artifact registry
* Big Query
* Cloud storage
* Document AI (Custom document classification and extraction processors)

# MLops overview

# 

# 

# Details and features of the training pipelines and other components

## CDC/CDE Training pipeline:

This pipeline will be responsible for performing the training on DocAI

**Assumptions:**

1. Before triggering the pipeline the user will first create a processor, ingest the data and annotate the data.
2. Cannot cover lineage tracking of code as DocAI is a product.
3. Training data should have a minimum resolution of 250x250 and minimum DPI of 90
4. The training data and test data uploaded my the User keeps the data within the limit that DocAI supports (as of 24/04/2022 the test set size should be less than 400 images)

**Risks:**

1. There is no python SDK available for calling DocAI processors. We are relying on the beta APIs provided by the Google team. If the APIs change even slightly the training pipeline will not work.
2. Doc AI CDC/CDE doesn’t seem very stable at this point. Some of the issues we have seen are
   1. The training fails without any reason.
   2. Training fails when more documents are provided for training.
   3. The deployment also fails without any proper error message (even through UI). Once the training is triggered from the DocAI UI, the training cannot be stopped for CDC/CDE processors.
3. Due to these issues the MLops training pipeline will be limited to work and cannot override the actions DocAI takes in case of occurrence of the issues. If this is caused during training then the training pipeline will exit midway. If DocAI returns a proper message we will add the same to the logs else not.
4. Vertex AI logs are not visible sometimes (not even logs written by the vertex AI service itself). This will impact understanding the reason for success/failure of a job or debugging of the errors in the pipeline.

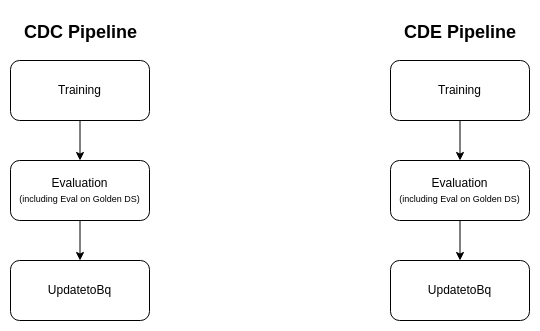
**Components:**

1. Training
   1. This will be used to start training of cdc/cde based model based on task type
   2. Input parameters :
      1. processor id
      2. project id
      3. processor location
      4. status\_retry\_secs
      5. Processor\_version\_name (TBC)
   3. Output:
      1. version id [Artifact]
2. Evaluation (specific to CDC and CDE task)
   1. This component will be used to evaluate cde/cdc models
   2. Input parameters :
      1. processor id
      2. project id
      3. location
      4. task\_type
      5. Golden dataset gcs\_uri
      6. processor\_version id(model) [Artifact]
   3. Output:
      1. evaluation\_gcs\_path
      2. metrics [Artifact-Metric]
3. Update to big query
   1. This component will be used to update bigquery tables with the outputs.
   2. Input parameters :
      1. job\_id
      2. model\_display\_name
      3. processor\_id
      4. processor\_version\_id
      5. processor\_type
      6. model\_version\_name
      7. train\_size
      8. test\_size
      9. doc\_type
      10. tablename\_model
      11. tablename\_class
      12. metrics model (overall) [Artifact-Metric]
      13. metrics classwise [Artifact-Metric]
   3. Output:
      1. N/A

Note:

* Orange Color indicate parameters from previous components

**Pipeline Overview:**

****

**Features:**

1. ML Metadata Artifacts will be used wherever applicable in the training pipeline to help lineage tracking
2. The components are built in such a way that they can shared and reused just by using the component specification file (component yaml file)
3. Components will run based on task type .
4. Only one evaluation and UpdateBQ component will run at a time depending on the task type.
5. All the input parameters along with evaluation metrics on validation set and golden dataset evaluation metrics will be logged into Bigquery
6. Will cover Lineage tracking for Dataset and model.
7. Error handling and logging

**Error Codes from DocAI for Error handling on Vertex AI:**

[To be provided by Tommy and Gopala]

| **Error scenario** | **Error code** | **Error Message (user defined)** | **Error Message (DocAI)** |
| --- | --- | --- | --- |
| Project Id is incorrect/invalid | 403 | The project\_id is incorrect | Permission denied on resource project ibank-development |
| Project ID is blank | 400 | The project is blank | Invalid resource field value in the request |
| Processor ID is incorrect | 404 | The Processor Id is incorrect | Processor with id '60b530fb3aa7e181qw' not found |
| Location is incorrect |  | The Location is incorrect | JSONDecodeError |
| Training/Deploying when 5 processor are already in training | 429 | Exceeded the count of 5 processors for training | Resource has been exhausted(Check Quota) |
| Deploying the processor version which is failed in training | 400 | The processor version has failed in the training | Processor Version State cannot be changed to UNDEPLOYING since it is in FAILED |
| Deploying the processor when it is in deploying state | 400 | The processor version is already in deploying state | Processor Version State cannot be changed to UNDEPLOYING since it is in DEPLOYING |
| Undeploying the processor version which is failed in training | 400 | The processor version has failed in the training | Processor Version State cannot be changed to UNDEPLOYING since it is in FAILED |
| Undeploying the processor when it is in deploying state | 400 | The processor version is already in deploying state | Processor Version State cannot be changed to UNDEPLOYING since it is in DEPLOYING |
| File not found | 400 | File not found or not available | FILE\_NOT\_FOUND |
| Training Failed | 400 | Error occurred while training the model | ERROR\_WHILE\_TRAINING |
| Model could not deployed | 400 | Error occurred while deploying the model | ERROR\_WHILE\_DEPLOYING |

## Custom OD Model Training pipeline:

This pipeline will be responsible for performing the training of the Custom Object detection Model.

**Assumptions:**

1. Before triggering the pipeline the user should generate annotations in pascal voc format xml for each image in the dataset which will contain the bounding box information of every label.
2. Deployment of the model to vertex AI endpoints will be manual after evaluating the accuracy metrics.
3. The user will provide the config file and pretrained weights by uploading it to GCS bucket and providing uri for the same as pipeline params

**Risks:**

1. Vertex AI logs are not visible sometimes (not even logs written by the vertex AI service itself). This will impact understanding the reason for success/failure of a job or debugging of the errors in the pipeline.

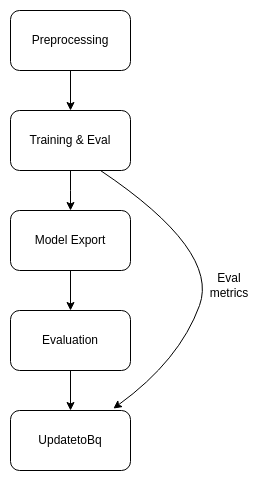
**Components:.**

1. Preprocessing
   1. This component will generate tfrecords from train and validation datasets.
   2. Input parameters :
      1. gcs uri of training dataset
      2. gcs uri of validation dataset
      3. gcs uri of labelmap.pbtxt
   3. Output:
      1. Train dataset [Artifact-Dataset]
      2. Validation dataset [Artifact-Dataset]
2. Training and eval:
   1. This component will train and evaluate the model on train and validation set respectively
   2. Input parameters :
      1. Train dataset [Artifact-Dataset]
      2. Validation dataset [Artifact-Dataset]
      3. Training Configuration (which includes Hyper parameters)
         1. gcs uri of labelmap.pbtxt
   3. Output:
      1. Best checkpoint uri
      2. Eval metrics (mAP) [Artifact-Metrics]
      3. Loss metric
3. Model Export and Conversion:
   1. This component will export the latest checkpoint to “.pb” format
   2. Input parameters
      1. gcs uri of latest trained checkpoints
      2. gcs uri of model pipeline.config
      3. Input type
      4. Input size
   3. Output parameters
      1. Converted model [Artifact-Model]
4. Evaluation
   1. This component will evaluate the model on golden data set and calculate the accuracy metrics
   2. Input parameters
      1. Converted model [Artifact-Model]
      2. Prediction threshold
      3. One sign golden data path
      4. Two sign golden data path
      5. Three sign golden data path
   3. Output parameters
      1. Accuracy metrics [Artifact-Metrics]
5. Bigquery Update
6. This component will update the accuracy metrics of the trained model to Bigquery
7. Input parameters :
   1. job\_id
   2. model\_version\_name
   3. model\_type
   4. train\_size
   5. test\_size
   6. tablename\_model
   7. tablename\_class
   8. metrics model (overall) [Artifact-Metrics]
   9. metrics classwise [Artifact-Metrics]
8. Output:
   1. N/A

Note:

* Orange Color indicate parameters from previous components

**Pipeline overview:**



**Features**:

1. ML Metadata Artifacts will be used wherever applicable in the training pipeline to help lineage tracking
2. The components are built in such a way that they can shared and reused just by using the component specification file (component yaml file)
3. Error handling and logging
4. All the input parameters along with evaluation metrics on validation set and golden dataset evaluation metrics will be logged into Bigquery
5. Will cover Lineage tracking for Code, Dataset and Model.

## Cloud Composer

Cloud composer will be used to trigger the training job.

Assumptions:

1. The user will compulsorily set the required environment variables
2. Three dags will be created.
   1. First dag will be able to trigger CDC Training job
   2. Second will trigger CDE Training job
   3. Third dag will be able to trigger the custom OD model training job
3. Required permissions will be given to install the necessary packages
4. Required accesses to run DAG scripts will be provided.

Features:

1. The user will be able to trigger the job through command line/script (in case access is required those accesses would be provided by the stakeholders)
2. Trigger the job from UI
3. The user also will be able to override the default parameter
4. It has a standard preprocessing module that will filter the bad quality images for custom OD model training. This module can be reused for other pipelines as well.

## Big Query

Big Query will be used to store all the experiment details.

Assumptions:

1. To store nested data we will “Record” format (which will be queryable)

Features:

1. Bigquery can be used for lineage tracking.
2. Will provide Queries to compare performance of the model experiments with different models and model versions.

Schema for Big Query:

|  | **Table name : training\_model\_metrics** | | | |
| --- | --- | --- | --- | --- |
| **Column Name** | **Datatype** | **Description** | **Default** | **Sample/Dummy value** |
| job\_id | String | Id of the Training pipeline job on Vertex AI | NULL | 2560006666846208000 |
| model\_name | string | DocAI Processor Name/ Vertex AI Model name | NULL | documentclassification\_cdc\_v1  aadhaar\_cde\_v1  pan\_cde\_v1  voterid\_cde\_v1 |
| model\_id | string | DocAI ProcessorID / VertexAI Model ID |  | ea39eb6b41766946 |
| processor\_  version\_id | string | Version ID of the trained model |  | 471961d7300dc533 |
| version\_name | string | version name of the trained model | NULL | v1 |
| is\_deployed | bool | Status whether the model is deployed or not | False | True/False |
| pipeline\_name | String | type of the model ( Dag Info / Pipeline Name) | NULL | docai\_cdc\_cde  custom\_object\_detection |
| model\_region | string | Region of the model hosting | NULL | asia-south1 |
| **ref\_code\_file\_version** | string | version of the code (commit id) | NULL | v1 |
| train\_test\_split | int | split of the train and test data | NULL | 70 |
| train\_test\_size | Record | train and test dataset size | NULL | {"train": "700", "test": "300"} |
| configuration | Record | JSON with hyperparameter and threshold values | NULL | Add dummy value |
| gcs\_ref\_training\_data | string | GCS reference path of the training data | NULL | <bucket\_name>/<path> |
| gcs\_path\_model | string | GCS path of the exported vertex AI custom model file | NULL | gs://<bucket\_name>/<path> |
| gcs\_path\_evaluation\_csv | string | GCS path of the raw predictions from the model with confidence scores for each image in golden dataset | NULL | gs://<bucket\_name>/<path> |
| trained\_on | timestamp | timestamp of when the model was trained | current timestamp | 2022-01-25 20:00:00 |
| **deployed\_on**  **(to be discussed)** | timestamp | Record deployed on timestamp | current timestamp | 2020-01-25 20:00:00 |
| precision | RECORD | JSON data with all the metrics of the model | NULL | {"test" " { "Precision" : "0.9", "recall" : "0.88", f1\_score: 0.89, ...},  "golden" " { "Precision" : "0.9", "recall" : "0.88", f1\_score: 0.89, ...}  } |
| recall |
| f1\_score |
| accuracy |
| map (this comes from OD) |

|  | **DB name : icici\_docai\_bq** | | | |
| --- | --- | --- | --- | --- |
|  | **Table name : training\_class\_metrics** | | | |
| **Column Name** | **Datatype** | **Description** | **Default** | **Dummy** |
| job\_id | String | Id of the Training pipeline job on Vertex AI | NULL | 2560006666846208000 |
| model\_name | string | DocAI Processor Name/ Vertex AI Model name | NULL | documentclassification\_cdc\_v1  aadhaar\_cde\_v1  pan\_cde\_v1  voterid\_cde\_v1 |
| model\_id | string | DocAI ProcessorID / VertexAI Model ID | NULL | ea39eb6b41766946 |
| processor\_  version\_id | string | Version ID of the trained model | NULL | 471961d7300dc533 |
| version\_name | string | version name of the trained model | NULL | v1 |
| is\_deployed | bool | Status whether the model is deployed or not | NULL | True/False |
| pipeline\_name | String | type of the model ( Dag Info / Pipeline Name) | NULL | docai\_cdc\_cde  custom\_object\_detection |
| model\_region | string | Region of the model hosting | NULL | asia-south1 |
| doc\_  type | string | type of the document | NULL | pan/passport/voter |
| field | string | field of the document | NULL | Name/ADD/DOB/pan number |
| true\_positive | int |  | NULL | 100 |
| true\_negative | int |  | NULL | 100 |
| false\_positive | int |  | NULL | 20 |
| false\_negative | int |  | NULL | 30 |
| precision | RECORD | JSON data with all the metrics of the model | NULL | { "golden" " { "Precision" : "0.9", "recall" : "0.88", f1\_score: 0.89, ...}  } |
| recall |
| f1\_score |
| accuracy |
| roc\_auc |
| map |

### Sample Evaluation CSV

| **Sample Evaluation CSV** | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **GCS Image path** | **Entity 1 Ground Truth** | **Entity 2 Ground Truth** | **Entity 3 Ground Truth** | **Entity 1 Prediction** | **Entity 2 Prediction** | **Entity 3 Prediction** | **Entity 1 Confidence** | **Entity 2 Confidence** | **Entity 3 Confidence** |
| gs://my\_bucket/image1 | random name1 | 3/1/1999 | CPCP1245 | random naame1 | 3/1/1990 | CPCP1245 | 0.9 | 0.65 | 0.9 |
| gs://my\_bucket/image2 | random name2 | 2/5/2000 | CKCH4568 | randm name2 | 2/5/2009 | CKCH4568 | 0.6 | 0.7 | 0.8 |
| gs://my\_bucket/image3 | some name1 | 3/2/2007 | FGHA1234 | som naame1 | 3/2/2007 | FGHA1234 | 0.7 | 0.7 | 0.7 |
| gs://my\_bucket/image4 | some name2 | 8/2/2005 | FHDA5432 | some nname2 | 8/2/2005 | FHDA5432 | 0.75 | 0.75 | 0.9 |

### BQ queries for creation of Tables:

Class Level Metrics Table:

create or replace table icici\_docai\_bq.training\_class\_metrics(

model\_name string,

model\_id string,

processor\_version\_id string,

version\_name string,

is\_deployed boolean,

job\_id String,

pipeline\_name String,

model\_region string,

doc\_type string,

field string,

gcs\_path\_evaluation\_csv string,

true\_positive int64,

true\_negative int64,

false\_positive int64,

false\_negative int64,

score struct < golden struct<

precision string,

recall string,

f1\_score string,

accuracy string,

roc\_auc string,

map string >>);

Model Level Metrics Table:

create or replace table icici\_docai\_bq.training\_model\_metrics(

model\_name string,

model\_id string,

processor\_version\_id string,

version\_name string,

is\_deployed boolean,

job\_id String,

pipeline\_name String,

model\_region string,

ref\_code\_file\_version string,

train\_test\_split int64,

train\_test\_size struct <train string, test string> ,

configuration string,

gcs\_ref\_training\_data string,

gcs\_path\_evaluation\_csv string,

gcs\_path\_model string,

trained\_on timestamp,

deployed\_on timestamp,

score struct <

test struct<

precision string,

recall string,

f1\_score string,

accuracy string,

roc\_auc string,

map string >,

golden struct <

precision string,

recall string,

f1\_score string,

accuracy string,

roc\_auc string,

map string >

>

);

### BQ queries for Comparison of Metrics:

Model Level Metrics:

select distinct a.model\_name better\_model, b.model\_name compared\_with,

a.version\_name better\_version, b.version\_name version\_compared\_with,

a.golden\_dataset.precision golden\_higher\_precision, b.golden\_dataset.precision golden\_lower\_precision,

a.golden\_dataset.recall golden\_higher\_recall, b.golden\_dataset.recall golden\_lower\_recall,

a.golden\_dataset.f1\_score golden\_higher\_f1\_score, b.golden\_dataset.f1\_score golden\_lower\_f1\_score,

a.golden\_dataset.accuracy golden\_higher\_accuracy, b.golden\_dataset.accuracy golden\_lower\_accuracy,

a.golden\_dataset.roc\_auc golden\_higher\_roc\_auc, b.golden\_dataset.roc\_auc golden\_lower\_roc\_auc,

a.golden\_dataset.map golden\_higher\_map, b.golden\_dataset.map golden\_lower\_map ,a.test\_dataset.precision test\_higher\_precision, b.test\_dataset.precision test\_lower\_precision,

a.test\_dataset.recall test\_higher\_recall, b.test\_dataset.recall test\_lower\_recall,

a.test\_dataset.f1\_score test\_higher\_f1\_score, b.test\_dataset.f1\_score test\_lower\_f1\_score,

a.test\_dataset.accuracy test\_higher\_accuracy, b.test\_dataset.accuracy test\_lower\_accuracy,

a.test\_dataset.roc\_auc test\_higher\_roc\_auc, b.test\_dataset.roc\_auc test\_lower\_roc\_auc,

a.test\_dataset.map test\_higher\_map, b.test\_dataset.map test\_lower\_map

from

(select \* except(score) ,score.golden as golden\_dataset,score.test as test\_dataset from `icici\_docai\_bq.training\_model\_metrics` ) a

join

(select \* except(score) ,score.golden as golden\_dataset,score.test as test\_dataset from `icici\_docai\_bq.training\_model\_metrics` ) b

on

a.golden\_dataset.precision >b.golden\_dataset.precision and

a.golden\_dataset.recall >b.golden\_dataset.recall and

a.golden\_dataset.f1\_score >b.golden\_dataset.f1\_score and

a.golden\_dataset.accuracy >b.golden\_dataset.accuracy and

a.golden\_dataset.roc\_auc >b.golden\_dataset.roc\_auc and

a.golden\_dataset.map >b.golden\_dataset.map and

a.test\_dataset.precision >b.test\_dataset.precision and

a.test\_dataset.recall >b.test\_dataset.recall and

a.test\_dataset.f1\_score >b.test\_dataset.f1\_score and

a.test\_dataset.accuracy >b.test\_dataset.accuracy and

a.test\_dataset.roc\_auc >b.test\_dataset.roc\_auc and

a.test\_dataset.map >b.test\_dataset.map;

Class Level Metrics:

select distinct a.model\_name better\_model, b.model\_name compared\_with,

a.version\_name better\_version, b.version\_name version\_compared\_with,

a.golden\_dataset.precision higher\_precision, b.golden\_dataset.precision lower\_precision,

a.golden\_dataset.recall higher\_recall, b.golden\_dataset.recall lower\_recall,

a.golden\_dataset.f1\_score higher\_f1\_score, b.golden\_dataset.f1\_score lower\_f1\_score,

a.golden\_dataset.accuracy higher\_accuracy, b.golden\_dataset.accuracy lower\_accuracy,

a.golden\_dataset.roc\_auc higher\_roc\_auc, b.golden\_dataset.roc\_auc lower\_roc\_auc,

a.golden\_dataset.map higher\_map, b.golden\_dataset.map lower\_map

from

(select \* except(score) ,score.golden as golden\_dataset from `icici\_docai\_bq.training\_class\_metrics` ) a

join

(select \* except(score) ,score.golden as golden\_dataset from `icici\_docai\_bq.training\_class\_metrics` ) b

on

a.golden\_dataset.precision >b.golden\_dataset.precision and

a.golden\_dataset.recall >b.golden\_dataset.recall and

a.golden\_dataset.f1\_score >b.golden\_dataset.f1\_score and

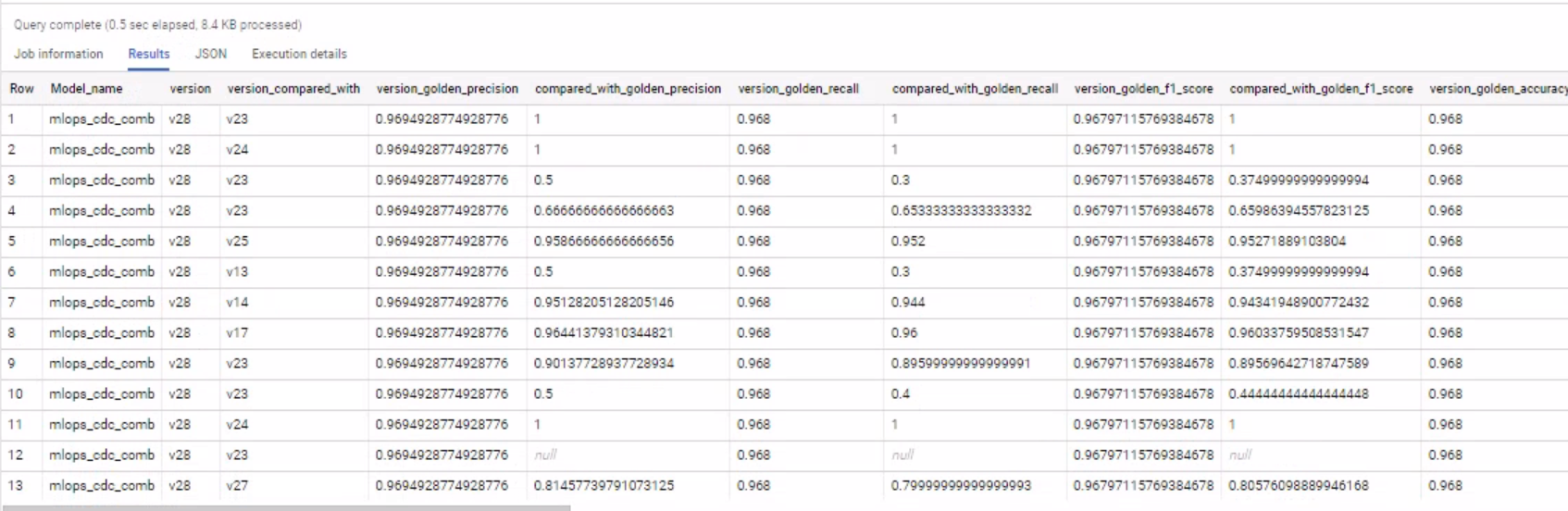
a.golden\_dataset.accuracy >b.golden\_dataset.accuracy and

a.golden\_dataset.roc\_auc >b.golden\_dataset.roc\_auc and

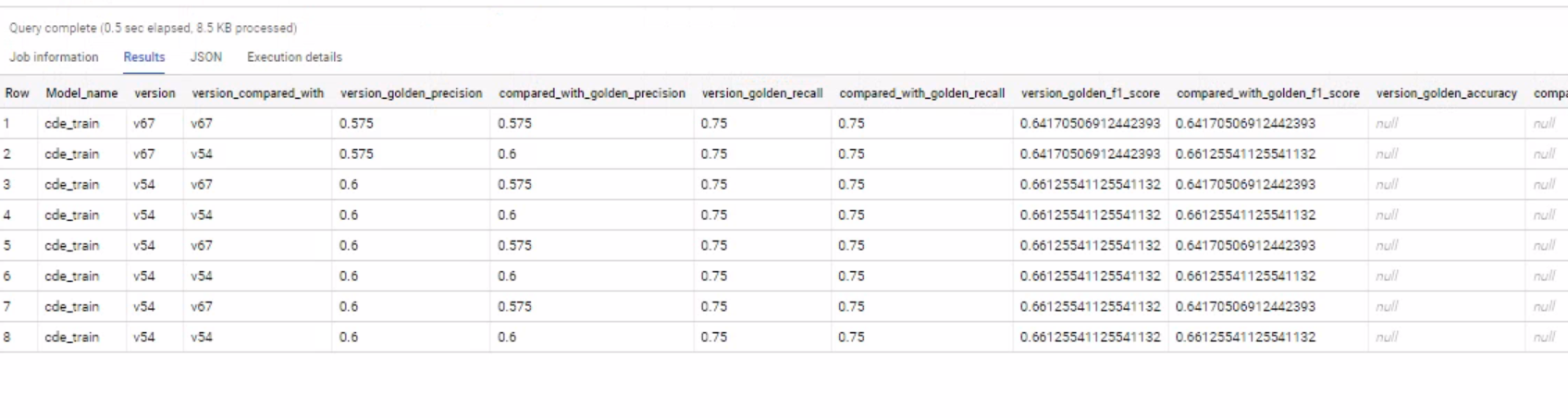
a.golden\_dataset.map >b.golden\_dataset.map;

Snapshots of BQ for Model Comparisons:

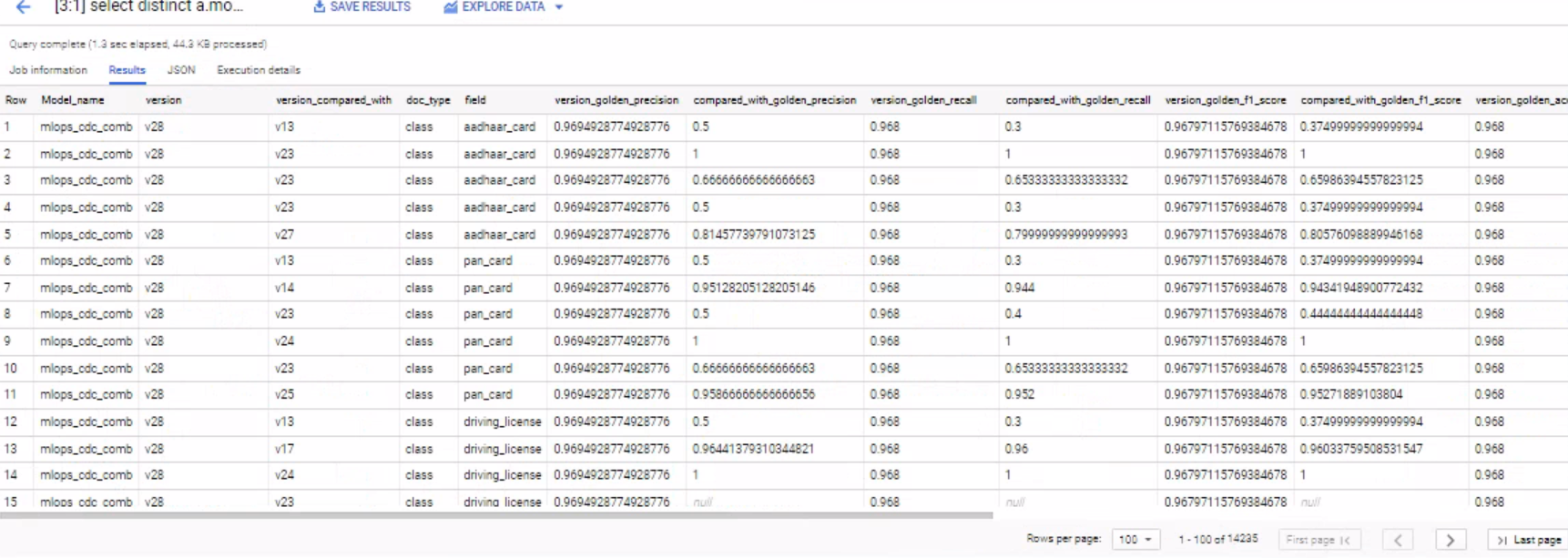
**For Model Level Comparison:**



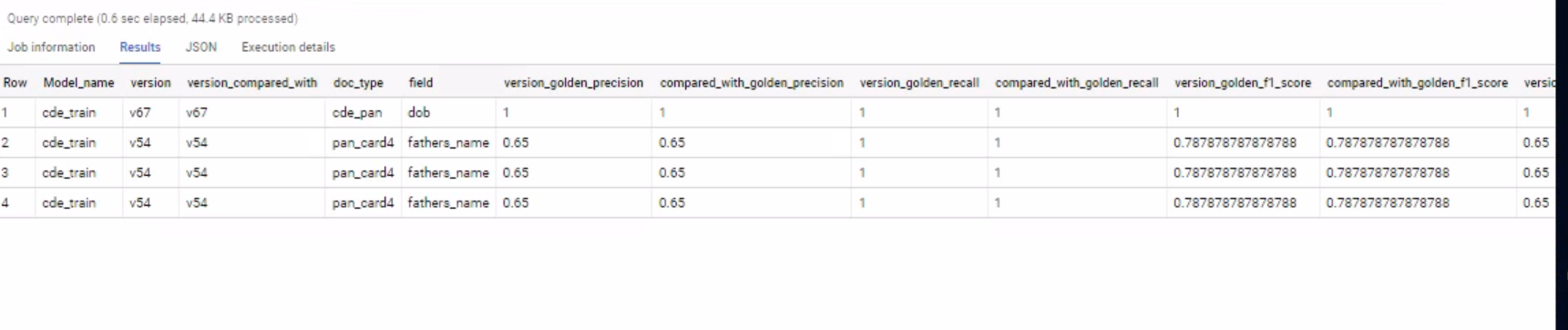
Cde\_train



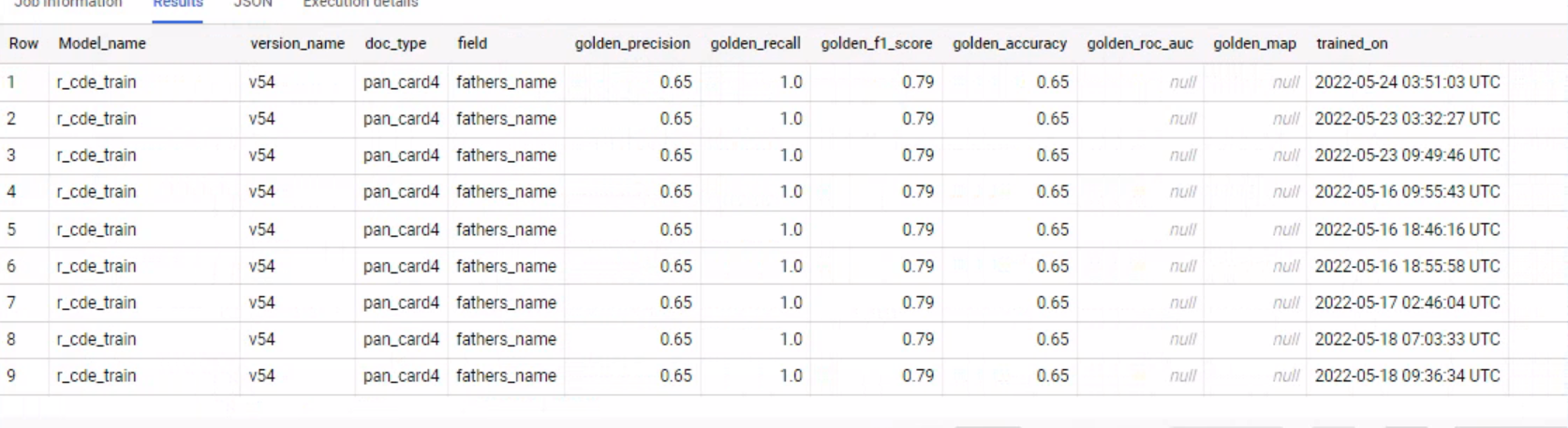
**For Class Level Comparison:**



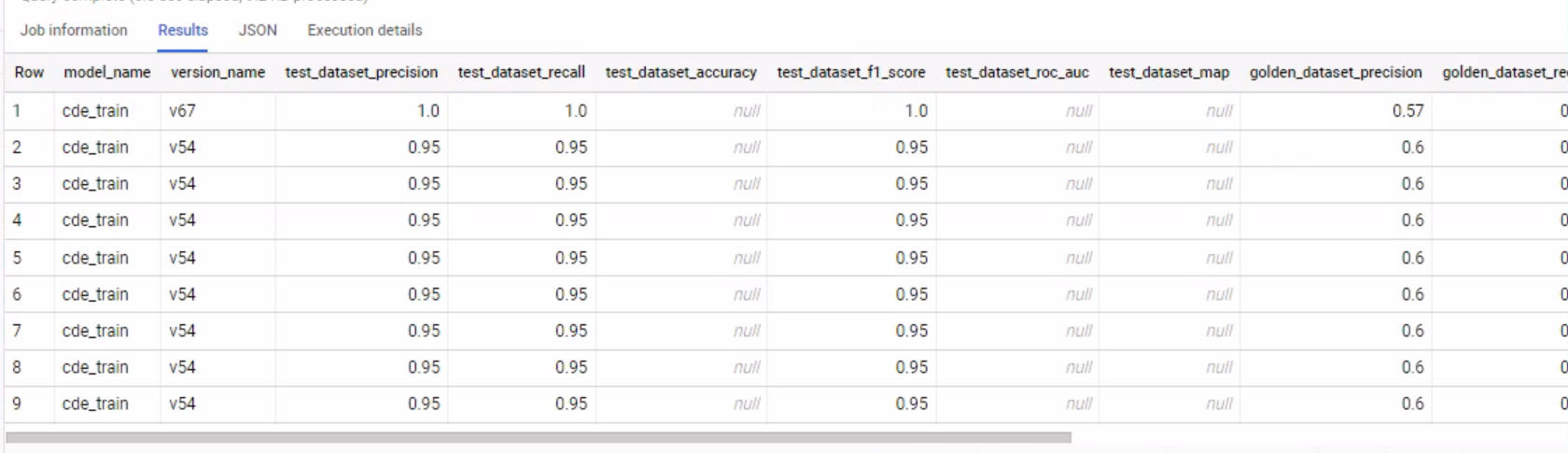
Cde\_train:

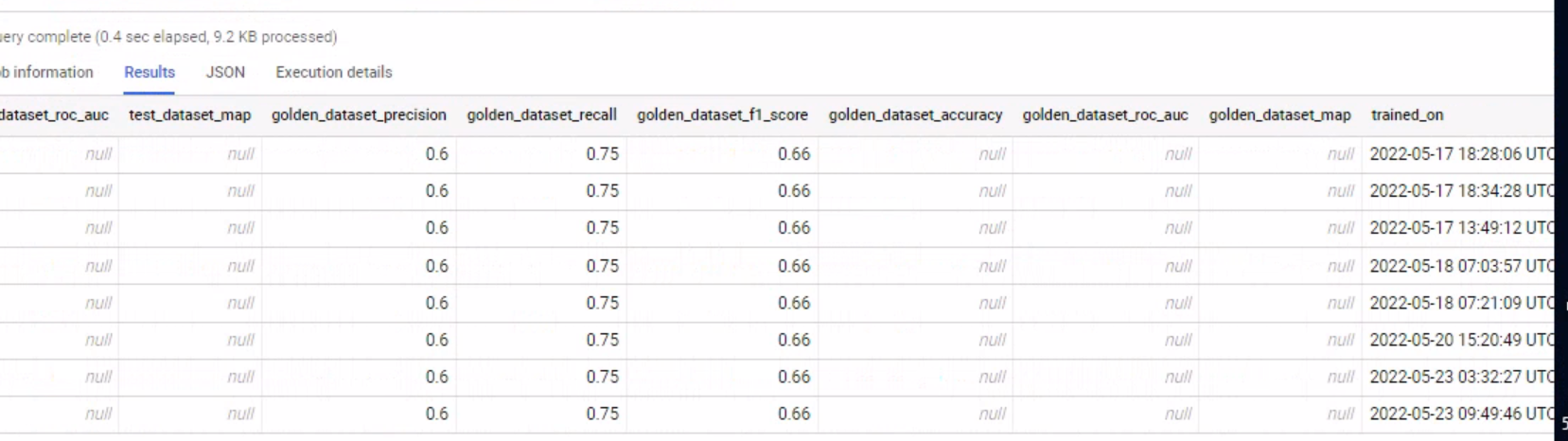


New snapshots for class level:



For Model level:





# Model Deployment

**Input Mode**

**docai**

**vertexai ----> endpoint\_mode = new or existing**

## **Deploy Model**

### Document AI Model deployment

**input\_mode** = ‘docai’

**Input Parameter**

1. Project ID
2. Location
3. Processor ID
4. Version ID

**Steps**

1. Deploy the new version
2. Making a deployed new version as default and routing the 100% traffic.
3. Undeploying the previous version

Steps: 1 – **done**

1. Deploy new version

1.1 using bash script deploying the new version

1.1.1 passing input parameters like project and processor and version id

1.1.2 generating the gcloud auth token

1.1.3 fetch the deployed processor version

1.1.4 deploying new the processor version

1.1.4.1 initial state of deployment “state = UNDEPLOYED”

1.1.4.2 than into the while loop “state= RUNNING”, given the runtime 300 min

For while loop

1.1.4.3 until reaches the “state=Deployed” if not Deployed back to the previous version deployed state

1.1.4.4 while loop ends

Step: 2 - **done**

1. Make it as default

2.1 From step 1 deploying the new version set as a default version.



Step: 3 – **done**

1. Undeploy previous version

**3.1 Undeploy the existing default version**

### Vertex AI ( gcloud )

Method - gcloud ( bash)

Reference link :- <https://cloud.google.com/sdk/gcloud/reference/ai>

Model with version and endpoints deployments

**Scenario 1**

**Deploy model to existing endpoint**

**input\_mode** = ‘vertexai’

**endpoint\_mode = ‘existing’**

**Input Parameter**

1. Model Name
2. GCS path of the model
3. Existing endpoint ID
4. Project ID
5. Location / Region
6. Base Container Image path (optional -- already set to default if no input given)

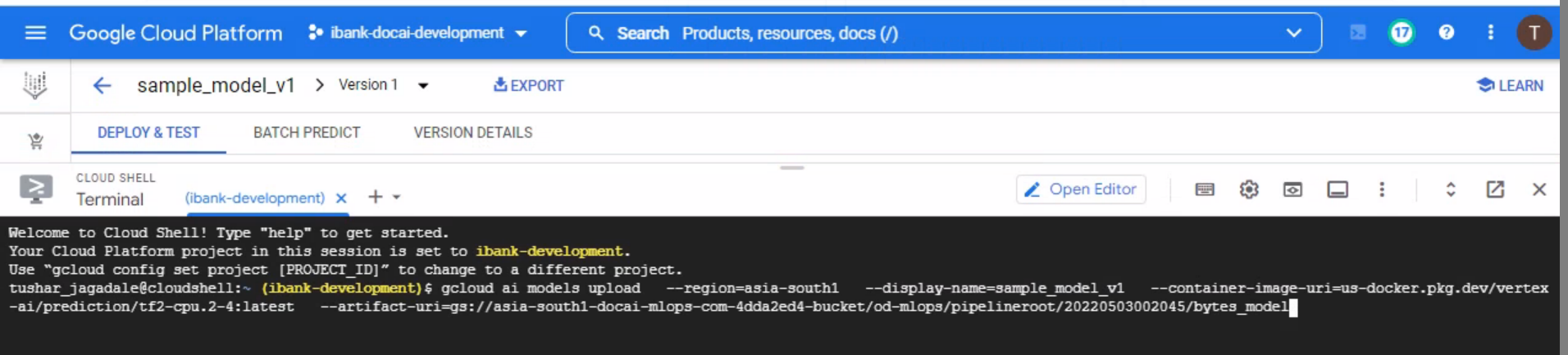
**Steps**

1. Upload model into model registry
2. Get the model ID of the uploaded model
3. List down deployed model IDs from the existing endpoint id
4. Deploy New model to existing endpoint
5. Undeploy previous models

**Step -1 - done**

1. Import model using gcloud command

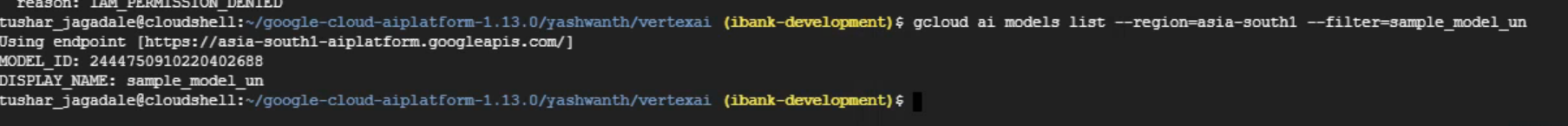
**Gcloud ai model upload –region=() –display-name=(model-name) –container-image-uri= (tensorflow, 2,4) –artifacts-uri= (url template in gcs storage )**



**Step-2 - done**

1. Get the model ID of the uploaded model

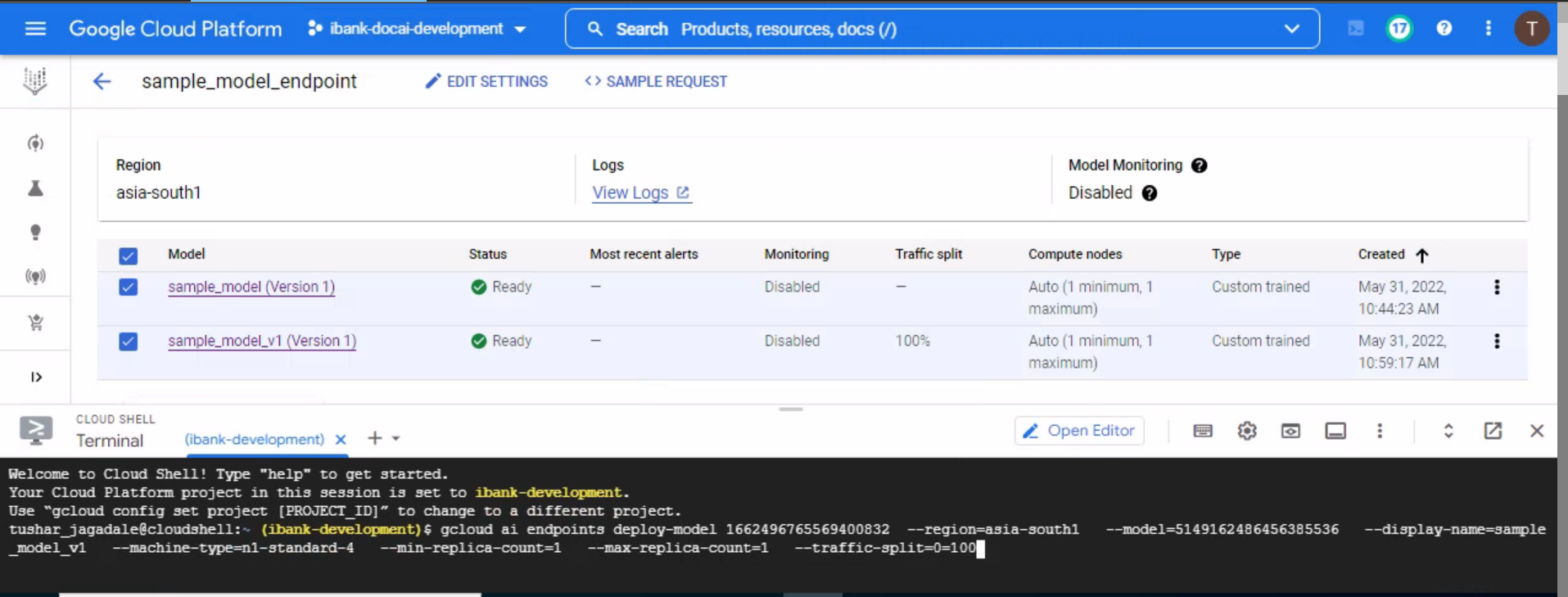
**gcloud ai models list –region= –filter= model name**



**Step -3 -done**

1. New model deploy to existing endpoint

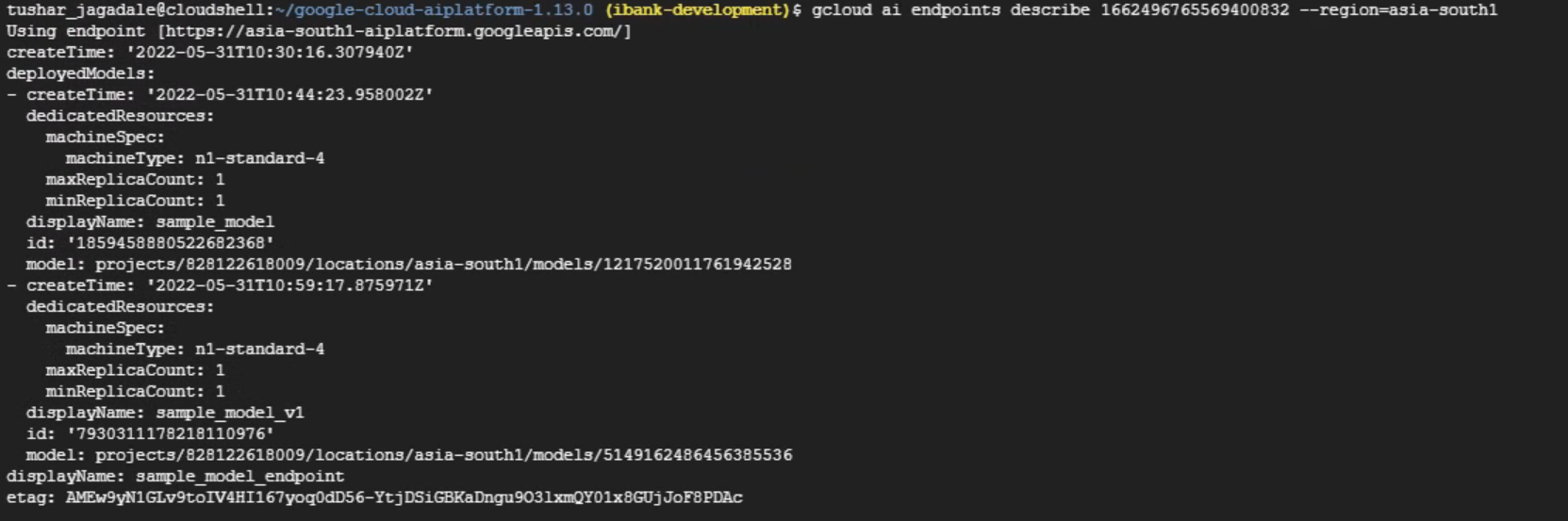
**gcloud ai endpoints deploy-model (endpoint id ) —region=asia-south1 –model=(model id) –display-name=(model name) –machine-type= (n1-standard-4) –min-replica-count=1 –max-replica-count=1 –traffic-split=0=100**



**Step-4 -done**

1. List down deployed model IDs from the existing endpoint id

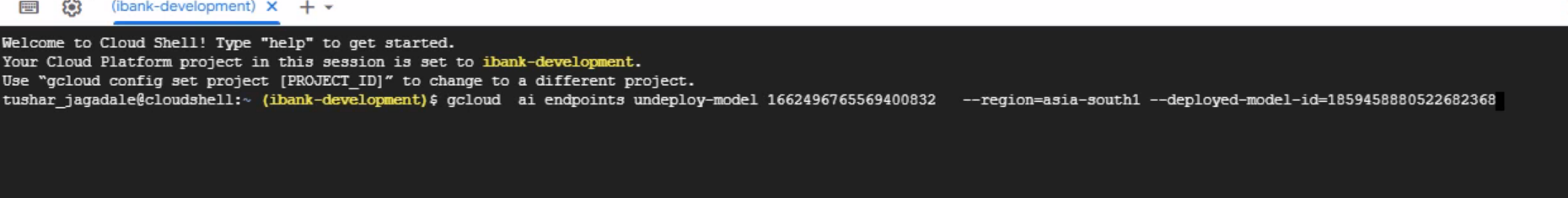
**gcloud ai endpoints describe endpoint.id --region=**



**Step-5-done**

1. Undeploy previous model

**gcloud ai endpoints undeploy-model (endpoint -id ) –region=() –deployed-model-id=()**



**Scenario 2**

**Deploy model to new endpoint**

input\_mode = ‘vertexai’

**endpoint\_mode = ‘new’**

**Input Parameter**

1. Model Name
2. GCS path of the model
3. Project ID
4. Endpoint Name
5. Location / Region
6. Base Container Image path (optional -- already set to default if no input given)
7. Job\_id

-------------------------------------------------------------------------------------------------------------------------------

**Steps**

1. Upload model into model registry
2. Get the model ID of the uploaded model
3. Create new endpoint using Endpoint Name
4. Get the Endpoint ID by using endpoint name
5. Deploy New model to this new endpoint

**Step-1 - done**

1. Upload model into model registry

Import model using gcloud command

**gcloud ai model upload –region=() –display-name=(model-name) –container-image-uri= (tensorflow, 2,4) –artifacts-uri= (url template in gcs storage )**

**Step-2 - done**

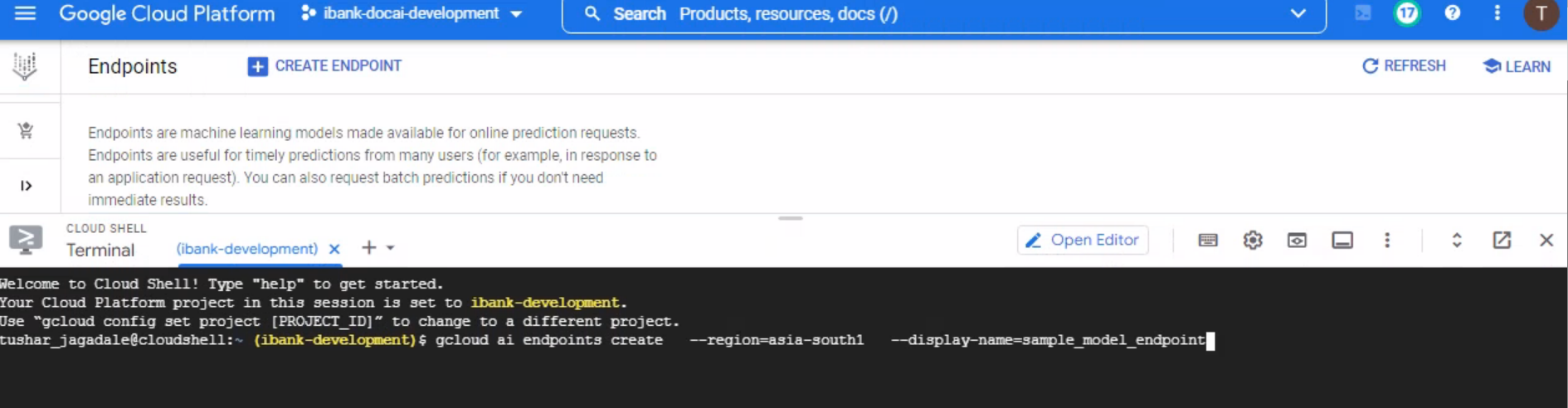
1. Get the model ID of the uploaded model

**gcloud ai models list –region= –filter= model name**

**Step-3 -done**

1. Create new endpoint using Endpoint Name

**gcloud ai endpoints create –region=() –display-name=(endpoint-name)**



**Step-4-done**

1. Get the endpoint id using endpoint name

**gcloud ai endpoints list –region= –filter=display-name= endpoint name**

**Step-5-done**

1. Deploy New model to this new endpoint

**gcloud ai endpoints deploy-model (endpoint id ) —region=asia-south1 –model=(model id) –display-name=(model name) –machine-type= (n1-standard-4) –min-replica-count=1 –max-replica-count=1 –traffic-split=0=100**

-------------------------------------------------------------------------------------------------------------------------------

## 

## 

## 

## 

## **Update BigQuery table**

1. Update for newly deployed model (Query using Job\_id)
   1. **is\_deployed** = True
   2. model\_id (Vertex AI endpoint ID)
   3. processor\_version\_id (Vertex AI Model ID -- Model Registry)
   4. **deployed\_on**

For document ai

Updating to the biq query table based on model\_id and version id

is\_deployed= true for new deployed version

Is\_deployed = false for undeployed version

## **Update BigQuery table**

1. Update **is\_deployed** column to True for newly deployed model
2. Update **is\_deployed** column to False for undeployed previous version model

## **Update Orchestrator YAML**

1. Update model\_id and processor\_version\_id in Yaml of orchestrator deployment.

