00 - Environment Setup

This is the notebook that sets up the GCP project for the creation of a pipeline

In [ ]:
```
project = !gcloud config get-value project
PROJECT_ID = project[0]
PROJECT_ID
```

You can change the region to your own liking

environment variables:

In [ ]:
```
REGION = 'europe-west4'
```

packages:

In [ ]:
```
from google.cloud import storage

import pandas as pd
from sklearn import datasets
```

clients:

In [ ]:
```
gcs = storage.Client(project = PROJECT_ID)
```

In [ ]:
```
BUCKET = PROJECT_ID
```

Create Storage Bucket

Check to see if bucket already exist and create if missing:

- GCS Python Client

In [ ]:
```
if not gcs.lookup_bucket(BUCKET):
    bucketDef = gcs.bucket(BUCKET)
    bucket = gcs.create_bucket(bucketDef, project=PROJECT_ID, location=REGION)
    print(f'Created Bucket: {gcs.lookup_bucket(BUCKET).name}')
else:
    bucketDef = gcs.bucket(BUCKET)
    print(f'Bucket already exist: {bucketDef.name}')
```

In [ ]:
```
print(f'Review the storage bucket in the console here:\nhttps://console.cloud.google.com/storage/browser/{PROJECT_ID};tab=objects&project={PROJECT_ID}')
```

Service Account & Permissions

This notebook instance is running as a service account in GCP. This service account will also be used to run other services in Vertex AI like training jobs and pipelines. The service account will need permission to interact with object in Cloud Storage which requires the role (roles/storage.objectAdmin).

Get the current service account:

In [ ]:
```
SERVICE_ACCOUNT = !gcloud config list --format='value(core.account)'
SERVICE_ACCOUNT = SERVICE_ACCOUNT[0]
SERVICE_ACCOUNT
```

Enable the Cloud Resource Manager API:

In [ ]:
```
!gcloud services enable cloudresourcemanager.googleapis.com
```

List the service accounts current roles:

In [ ]:
```
!gcloud projects get-iam-policy $PROJECT_ID --filter="bindings.members:$SERVICE_ACCOUNT" --format='table(bindings.role)' --flatten="bindings[].members"
```

If the resulting list is missing `roles/storage.objectAdmin` or another role that contains this permission, like the basic role `roles/owner`, then it will need to be added for the service account. Use these instructions to complete this:

In [ ]:
```
print(f'Go To IAM in the Google Cloud Console:\nhttps://console.cloud.google.com/iam-admin/iam?orgonly=true&project={PROJECT_ID}&supportedpurview=organizationId')
```

From the console link above, or by going to https:/console.cloud.google.com and navigating to "IAM & Admin > IAM":

- Locate the row for the service account listed above: `<project number>-compute@developer.gserviceaccount.com`
- Under the `inheritance` column click the pencil icon to edit roles
- In the fly over menu, under `Assign roles` select `Add Another Role`
- Click the `Select a role` box and type `Storage Object Admin`, then select `Storage Object Admin`
- Click Save
- Rerun the list of services below and verify the role has been added:

In [ ]:
```
!gcloud projects get-iam-policy $PROJECT_ID --filter="bindings.members:$SERVICE_ACCOUNT" --format='table(bindings.role)' --flatten="bindings[].members"
```

01 - Preparing pre-existing data

Use Storage to load and prepare data for machine learning:

Source Data

Overview

We'll be making a single class image classification so we will need data which is related. Luckily one is available on kaggle.

The Data

- The data can be researched further at this Kaggle link.

Preparation of the Data

We will have to manually create a mapping on the google cloud bucket with the following:

- /uploads/data/rock
- /uploads/data/paper
- /uploads/data/scissors

Now manually upload the data without any additional folders to the respected folder on the bucket. We will take care of splitting the dataset into training, testing, evaluation later. Make sure there are only scissors in scissors folder and same goes for rock and paper!

Different Source Data (Optional)

You can change the data used here to your own use case.

Note that this is currently set up to be a single class image classification. With the use of AutoML we are limited to a few dataset types: https://cloud.google.com/vertex-ai/docs/datasets/overview

Every dataset has a different way of setting itself up. Make sure to do the proper searching before changing this code to your own use-case.

Simularly you are also limited to a few types of model training: https://cloud.google.com/vertex-ai/docs/training-overview Make sure to carefully read the most up to date documentation as anything on the GCP Platform is subject to frequent changes

---

Setup

inputs:

In [ ]:
```
project = !gcloud config get-value project
PROJECT_ID = project[0]
PROJECT_ID
```

packages:

In [ ]:
```
import pandas as pd
import os
import json
from gcloud import storage
```

clients:

In [ ]:
```
client = storage.Client(project = PROJECT_ID)
bucket = client.get_bucket(PROJECT_ID)
```

parameters:

In [ ]:
```
BUCKET = PROJECT_ID
```

Load data for preparation (This will work if you have the proper file structure set up)

In [ ]:
```
# Get adresses of files from GCS to CVS files
!gcloud storage ls --recursive gs://$PROJECT_ID/uploads/data/rock/** > rock.csv
!gcloud storage ls --recursive gs://$PROJECT_ID/uploads/data/paper/** > paper.csv
!gcloud storage ls --recursive gs://$PROJECT_ID/uploads/data/scissors/** > scissors.csv
```

convert item into dataframe

In [ ]:
```
# Create DataFrames from CSV files
df_rock = pd.read_csv('rock.csv', header=None)
df_paper = pd.read_csv('paper.csv', header=None)
df_scissors = pd.read_csv('scissors.csv', header=None)
```

add labels so the dataset knows which is what

In [ ]:
```
# Add labels
df_rock['label'] = 'rock'
df_paper['label'] = 'paper'
df_scissors['label'] = 'scissors'
```

combine into one

In [ ]:
```
# Merge all 3 DataFrames
df_full = pd.concat([df_rock, df_paper, df_scissors])
#df_full
```

Create CSV input file and upload it to the given path

In [ ]:
```
# Save input file locally
df_full.to_csv('input_file.csv', index=None, header=None)

# Export input file to Cloud Storage
blob = bucket.blob('rps/input_file.csv')
blob.upload_from_filename('input_file.csv')
```

02 - Creating the pipeline

We will be making a pipeline with the use of Kubeflow Pipelines. This is very commonly used for the creation of pipelines in GCP. We will be taking care of each step being:

- Make a new dataset
- Train the model on the dataset
- Make an endpoint to deploy the model to
- Deploy the model to the endpoint

Setup

inputs:

In [ ]:
```
project = !gcloud config get-value project
PROJECT_ID = project[0]
PROJECT_ID
```

In [ ]:
```
SERVICE_ACCOUNT = !gcloud config list --format='value(core.account)'
SERVICE_ACCOUNT = SERVICE_ACCOUNT[0]
SERVICE_ACCOUNT

BUCKET_URI = f"gs://{PROJECT_ID}"
PIPELINE_ROOT = f"gs://{PROJECT_ID}/rps/pipeline_root"
VERSION = "1"
```

packages:

In [ ]:
```
import google.cloud.aiplatform as aip
import os
from typing import Any, Dict, List
import kfp
from kfp.v2 import compiler
```

aiplatform client:

In [ ]:
```
aip.init(project=PROJECT_ID, location=REGION, staging_bucket=PIPELINE_ROOT)
```

defining the Kubeflow Pipeline which has all the listed steps mentioned above:

- Make a new dataset
- Train the model on the dataset
- Make an endpoint to deploy the model to
- Deploy the model to the endpoint

In [ ]:
```
@kfp.dsl.pipeline(name=f"rps-automl-image-training-pipeline-v{VERSION}")
def pipeline(project: str = PROJECT_ID, region: str = REGION, BUCKETURI: str = BUCKET_URI, version: str = VERSION):
    from google_cloud_pipeline_components import aiplatform as gcc_aip
    from google_cloud_pipeline_components.v1.endpoint import (EndpointCreateOp,
                                                              ModelDeployOp)

    ds_op = gcc_aip.ImageDatasetCreateOp(
        project=project,
        display_name=f"rps-dataset-v{version}",
        location=region,
        gcs_source=f"{BUCKET_URI}/rps/input_file.csv",
        import_schema_uri=aip.schema.dataset.ioformat.image.single_label_classification,
    )

    training_job_run_op = gcc_aip.AutoMLImageTrainingJobRunOp(
        project=project,
        display_name=f"train-automl-rps-v{version}",
        prediction_type="classification",
        location=region,
        model_type="CLOUD",
        dataset=ds_op.outputs["dataset"],
        model_display_name=f"automl-rps-v{version}",
        training_fraction_split=0.7,
        validation_fraction_split=0.15,
        test_fraction_split=0.15,
        budget_milli_node_hours=8000,
    )

    endpoint_op = EndpointCreateOp(
        project=project,
        location=region,
        display_name=f"endpoint-automl-rps-v{version}",
    )

    ModelDeployOp(
        model=training_job_run_op.outputs["model"],
        endpoint=endpoint_op.outputs["endpoint"],
        automatic_resources_min_replica_count=1,
        automatic_resources_max_replica_count=1,
    )
```

Compile the just defined pipeline into a json format which will appear locally

In [ ]:
```
compiler.Compiler().compile(
    pipeline_func=pipeline,
    package_path="rps_single_image_classification_pipeline.json"
)
```

Running the pipeline

In [ ]:
```
job = aip.PipelineJob(
    display_name='rps-pipeline',
    template_path="rps_single_image_classification_pipeline.json",
    pipeline_root=PIPELINE_ROOT,
    enable_caching=False,
)
```

In [ ]:
```
job.run()
```

In [ ]:
```
!rm rps_single_image_classification_pipeline.json
```

Cleaning up (Optional)

To clean up all Google Cloud resources used in this project, you can delete the Google Cloud project.

Otherwise, you can delete the individual resources you created here.

Get resources from the pipeline to clean up

Function to get details of a task

In [ ]:
```
def get_task_detail(task_details: List[Dict[str, Any]], task_name: str) -> List[Dict[str, Any]]:
    for task_detail in task_details:
        if task_detail.task_name == task_name:
            return task_detail
```

In [ ]:
```
pipeline_task_details = (
    job.gca_resource.job_detail.task_details
)  # fetch pipeline task details


# fetch endpoint from pipeline and delete the endpoint
endpoint_task = get_task_detail(pipeline_task_details, f"endpoint-automl-rps-v{VERSION}")
endpoint_resourceName = (
    endpoint_task.outputs["endpoint"].artifacts[0].metadata["resourceName"]
)
endpoint = aip.Endpoint(endpoint_resourceName)
# undeploy model from endpoint
endpoint.undeploy_all()
endpoint.delete()

# fetch model from pipeline and delete the model
model_task = get_task_detail(pipeline_task_details, f"automl-rps-v{VERSION}")
model_resourceName = model_task.outputs["model"].artifacts[0].metadata["resourceName"]
model = aip.Model(model_resourceName)
model.delete()


# fetch dataset from pipeline and delete the dataset
dataset_task = get_task_detail(pipeline_task_details, f"rps-dataset-v{VERSION}")
dataset_resourceName = (
    dataset_task.outputs["dataset"].artifacts[0].metadata["resourceName"]
)
dataset = aip.ImageDataset(dataset_resourceName)
dataset.delete()
```

Everything is deleted except for the Google Cloud Bucket. You can empty the whole bucket using the following cell:

In [ ]:
```
!gsutil rm -r $BUCKET_URI
```

Google Functions (Optional)

With addition to this I have made an endpoint which would update the input_file.csv file with each image upload. To launch this you will have to go Google Cloud Functions.

There enable the API and make a new function.

specs:

- 1st gen
- function name (your preference)
- region (your preference)
- httptrigger http
- You can choose how you want to do autenthication.
- Enable HTTPS
- ** Hit next **
- Select Python 3.9
- Entrypoint (your preference but make sure it matches the function inside)

Now in the main.py insert the following code:

```
from google.cloud import storage
import uuid
import pandas as pd
import io

def hello_world(request):
    if request.method=='POST':
        """Process the uploaded file and upload it to Google Cloud Storage."""
        uploaded_file = request.files.get('image')

        #Check if file is in request
        if not uploaded_file:
            return 'No file uploaded.', 400

        #Parse type
        type_of_image = request.form.get('type')
        if type_of_image not in ['rock', 'paper', 'scissors']:
            return 'Invalid type', 400

        BUCKET_NAME = BUCKETNAME
        extension = (uploaded_file.filename.split('.'))[-1:][0]

        if extension != "png":
            return 'Invalid image type', 400

        # Create a Cloud Storage client.
        gcs = storage.Client()

        # Get the bucket that the file will be uploaded to.
        bucket = gcs.get_bucket(BUCKET_NAME)

        new_file_name = f"{type_of_image}-{str(uuid.uuid4())}"

        # Create a new blob and upload the file's content.
        #print(f"uploads/data/{type_of_image}/" + new_file_name)
        blob = bucket.blob(f"uploads/data/{type_of_image}/" + new_file_name + f".{extension}")

        blob.upload_from_string(
            uploaded_file.read(),
            content_type=uploaded_file.content_type
        )

        #Manipulate existing csv with new data paths for dataset
        dataset_import_csv = 'rps/input_file.csv'
        blob_csv = bucket.blob(dataset_import_csv)
        csv_string = blob_csv.download_as_string().decode("utf-8")
        new_data_row = f"gs://{BUCKET_NAME}/uploads/data/{type_of_image}/{new_file_name}.{extension},{type_of_image}\n"

        print(csv_string)
        print(type(csv_string))
        csv_string += new_data_row
        blob_csv.upload_from_string(
            csv_string,
            'text/csv'
        )

        # The public URL can be used to directly access the uploaded file via HTTP.
        return f"Uploaded {type_of_image}"
    else:
        return "Allowed methods: [POST]", 400
```

make sure to adjust the `BUCKET_NAME = BUCKET NAME` to whatever your bucket name is. in this case it will be the name of your project you are running this in.

and in the requirements.txt make sure to not forget the following:

```
# Function dependencies, for example:
# package>=version
google-cloud-storage==2.7.0
uuid
pandas
```

Once this is done. You can deploy the functions. Once deployed you can use the endpoint to add new data to the input.csv through a post.

Using endpoint in Python code (Optional)

To make it easy, go to your created endpoint where your model is currently launched.

When in the endpoint you'll see a button to see a sample request. Make sure to choose the python tab.

You can follow the instructions there.

Incase you need additional directions to use your endpoint in python:

Go to: https://github.com/googleapis/python-aiplatform/blob/main/samples/snippets/prediction_service/predict_image_classification_sample.py

Copy this into your python file. You will also need to copy the 3rd step code and paste that into your python file You will need credentials to use the endpoint. To create this follow the following: IAM & Admin > Service Accounts > Keys > Add Key (JSON). Make sure you put the json file in the project.

Now `import os` into your project and setup the environment variables to use the function. `os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = 'PATH TO JSON CREDENTIALS'`

Make sure the picture you will test with is 300x300