Introduction

This document discusses the techniques for implementing and automating continuous integrations(CI), continuous delivery(CD), and continuous training(CT) for machine learning systems. This document is for Data Scientists and ML Engineers who want to apply Devops principles to ML Systems, including integration, releasing, deployment and infrastructure management.

The main goal of Devops is to shorten the development lifecycles, increasing deployment velocity and release models on production systems. To achieve this benefits, you introduce two concepts

- Continuous integration(CI)
- Continuous Deployment(CD)

Scenario

Consider you are a retailer, selling products on Amazon. You have built a ML workflow predict next 1 year worth of orders for different products. Now, you want to apply the same workflow for other scenarios where instead of predicting the orders of the products, you want to predict the sales of the product based on the historical data without ground up work.

You can use the same workflow model and add few more required functionalities and deploy the ML models on scale. This document guides you to deploy the ML workflow on different projects.

In this tutorial, we want to have 2 different GCP projects. One is for the shared repository where we store all our model code in **Project A,** and to **deploy the ML workflow on Project B,** and run the actual ML workflow.

NOTE
Here for the demonstration, we have used **Cloud Source Repository** as a version control system to store all our model code for our ML workflow. GCP **Cloud Build Triggers** also supports the external Connections(Github, BitBucket).

Download and Installation

This document comes with the required source code, datasets to run and experiment with the pipeline. The ML workflow deployments are made on kubeflow which is a machine learning toolkit for kubernetes.

To know more about kubeflow on GCP - Document Link

The main goal of this document is to move the ML workflow with minimal steps from project to other projects in GCP. The **Build files** to move the ML workflow can be seen in the package.

To download the package,

- Go to the deployed VM instance, SSH into the system
- Change the default username
    - In the upper-right corner of the SSH window, click the settings icon
    - Select **Change Linux Username**.
    - Type **ubuntu**
    - Click **Change**
- Type `ls` command to list all directories. You'll be able to see the **demand-ml** package. For CICD pipeline, required folders are **dml-cicd, dml-mlops.zip**, **MLOps-Getting Started Guide**
- Set the above package as a current working directory
- To copy the files from VM instance to local machine follow these steps - Document Link

Few Changes to Make In the Code

Once you have downloaded the folder to your local machine, cd to that folder. We need to make few changes in the script to make work. Extract the file **dml-mlops.zip**

- Go to the file **build/build.sh**, change the value of the PROJECT_ID(Project B). Ie.., replace the PROJECT_ID value with your current project id
- Go to the file **deploy_kubeflow/start_ui.sh**, change the value PROJECT_Id. Ie.., replace the PROJECT_ID value with your current project id(Project B)
- Go to the file **cloudbuild_run.yaml** file, change the SUBSTITUTION VALUE "**_PROD_BUCKET_NAME**" to "your naming convenient" For example - "kfp-pipeline-bucket"

- Go to the cloud shell, download the source code. Change the directory to **dml-mlops,** execute **build/build.sh** file. If your local system has docker setup execute above command locally, (or) go to cloud shell, download the packages, make require changes and execute the build.sh command

NOTE - This build will take upto 15 - 30mins because it has to prepare the environment to execute the scripts. Once the build is completed, make sure you can see the docker image in the GCP container registry

Code Structure Explanation

CI/CD with kubeflow pipelines and Cloud Build

Cloud Build setup executes builds with the permissions granted to the cloud build service account tied to that project. You need to grant the service account to the **Kubernetes Engine Developer** role to allow running Kubeflow pipeline.

1. Select **Cloud build** from the GCP console menu
2. Select **Settings** from the right-hand-side menu
3. Enable Kubernetes Engine Developer Role
4. Enable container registry admin role if not already enabled

Repository Structure

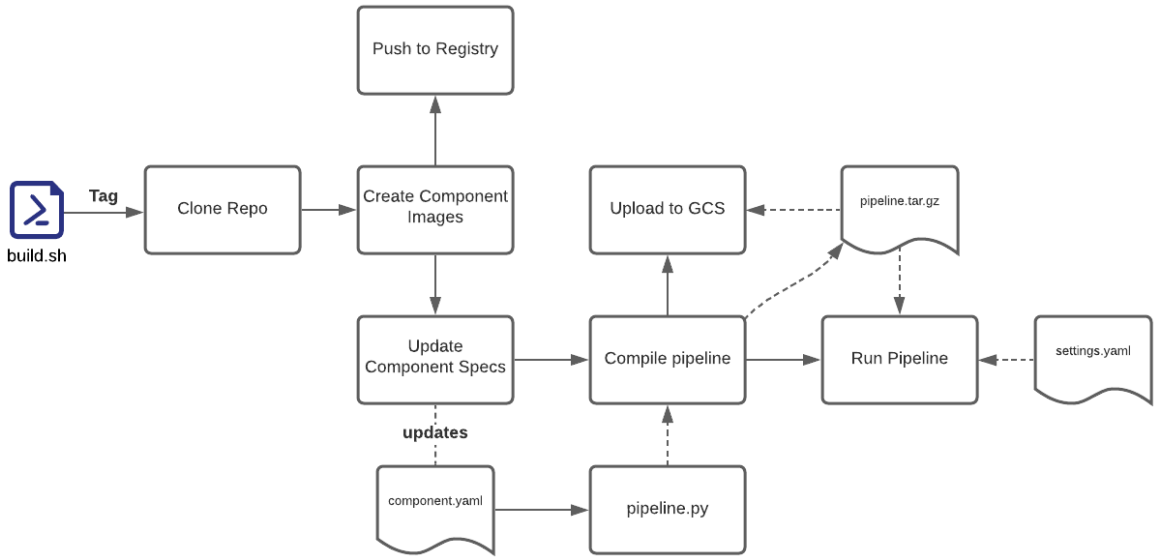
1. **Components -** this directory includes the artifacts required for each component in the pipeline. Each component has its own directory. In the code package there are four components: **aijob, fetchdata, preprocess, statstest**

   Each component directory includes the following files
   - **DockerFile** - This file defines the container image to run the component code
   - **[component-name].py** - This is the main file for the component
   - **Component.yaml** - This defines the component specs for the pipeline. The specs include components **inputs, outputs** and **image.**
   - **Build.sh** - This script responsible to create a docker image incase to test the component locally

2. **Pipeline -** The pipeline directory includes the following file
    - **Workflow.py -** This includes the definition of the kubeflow pipelines. More specifically:
        i. Component Ops are created from the **component.yaml** file
        ii. A pipeline function is created, with **@dsl_pipeline** decorator, where the component ops are executed and chained together to compose the workflow of the pipeline
    - **settings.yaml** - This includes the parameter values for the pipeline to run with
    - **helper.py -** This implements the following functionality
        i. **update_component_spec() -** This change the image property for all of the **component.yaml** with respect to the docker container image repository and image tag. This is executed when new container images are created, as part of the build process, and new tags are generated
        ii. **read_settings() -** This reads all the parameters values from the **settings.yaml** file, to invoke the pipeline run with
        iii. **run_pipeline() -** This creates, experiment and run a given pipeline package
3. **Build -** This build directory includes the following files
    - **Dockerfile -** This is a definition for the **kfp-util** container image required to deploy and run the kubeflow pipeline
        i. It is based on **gcr.io/cloud-builders/kubectl** image, which has the **gcloud sdk** installed and sets the **kubeconfig**
        ii. It installs **Python 3.6.x**
        iii. It installs the **kubeflow pipeline SDK** and other python required packages
    - **Cloudbuild.yaml -** This is the cloud build YAML file that explains the various workflow steps
    - **build.sh -** This is the entry point to execute the build

Cloud Build Steps

The following diagram shows the build steps:



1. **Clone Repository -** When executing the build.sh script with the necessary substitutions, the first step is to clone the code repository to the cloud build environment
2. **Build Component Images** - Build docker container images for all the four components
3. **Push images to Registry** - This is performed automatically when you define the **image section** in the **cloudbuild.yaml** file
4. **Update Component Specs -** This executes the **update_component_specs** method in **helper.py** modules. This method uses the created and tagged image URLs and update them to **component.yaml** file
5. **Compile Pipeline -** This executes **dsl-compile** to the compile the pipeline defined in workflow.py to generate **pipeline.tar.gz** package
6. **Upload to GCS -** This uploads the **pipeline.tar.gz** to the GCS

Create Bucket and upload a files in Project A

Before further steps, we need to create a GCS bucket in the project A and place all the required datasets and files to work with the ML workflow. To create a empty bucket in the project, please follow the steps mentioned in the link - Create A Storage Bucket

Once the storage bucket is successfully created, upload the files to the bucket. To upload the files to the bucket, follow these steps

1. Assuming current you are in the demand-ml directory, change the directory to **dml-cicd**
2. Upload the files to the bucket - Uploading files to Bucket
3. Once you uploaded the storage directory must look like this
    - dml-cicd
        |- datasets
            |- restaurant_dataset.csv
        |- module_testing
            |- tc2_config.json

NOTE - Using gsutil command if the file directory changes, you can manually upload the files to the bucket itself (or) leave it as it is. But, later in steps, when you are mentioning URI to kubeflow URI make sure that correct URI is mentioned in the path

Steps to follow to create Triggers

The goal of the guide is to provide steps for packaging the application from **Project A** to **Project B.**

Let's make a notation for better understanding :

- Project A - Source or developer Environment where the pipeline code are stored in a repository
- Project B - Target or production Environment where the repository needs to be transferred to run the ML workflow

The guide helps you to achieve the following

1. Push the files to a cloud source repository in Project A
2. Packaging the files from Project A to Project B

Before proceeding further the developer must have the following access in the project in order to achieve the goal. However if the user is the admin of the project he has access to all the required tools by default.

- Cloud build admin
- Source repository admin
- Kubernetes Engine admin
- Storage admin role

- Bigquery data Editor access
- Bigquery Job creation access
- iam.resourcemanager.projects.setIamPolicy (to give the permissions to cloud build service accounts)

The process mentioned below is to be followed one after the other to achieve the packaging the file between two projects.

Changes to be done in yaml files

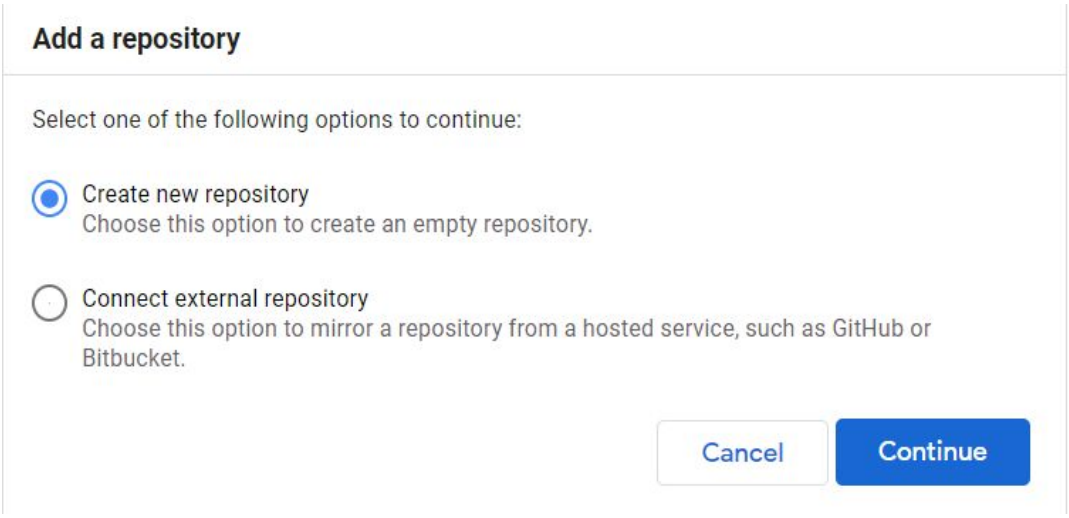
- There are certain changes that need to be done to ***cloudbuild.yaml*** and ***cloudbuild_run.yaml*** according to the environment before pushing it to the repository.
- **Changes in cloudbuild.yaml**
  - Update the _PROJECT_ID(PROJECT_A), _REPO_NAME(ie.., **dml-mlops**) at the beginning of the file
  - **Update the user name and email in the line 14 of the file**.
  - All the changes that made in this fame should be with respect to Project A
- **Changes in cloudbuild_run.yaml**
  - Update the _PROJECT_ID(PROJECT_B), _REPO_NAME, _BUCKET_NAME(ie.., **dml-cicd**. This bucket name should be same as GCS bucket created in Project A), dir(need to be same as _REPO_NAME)
  - All the changes that made in this fame should be with respect to Project B

Creating empty repositories in both projects

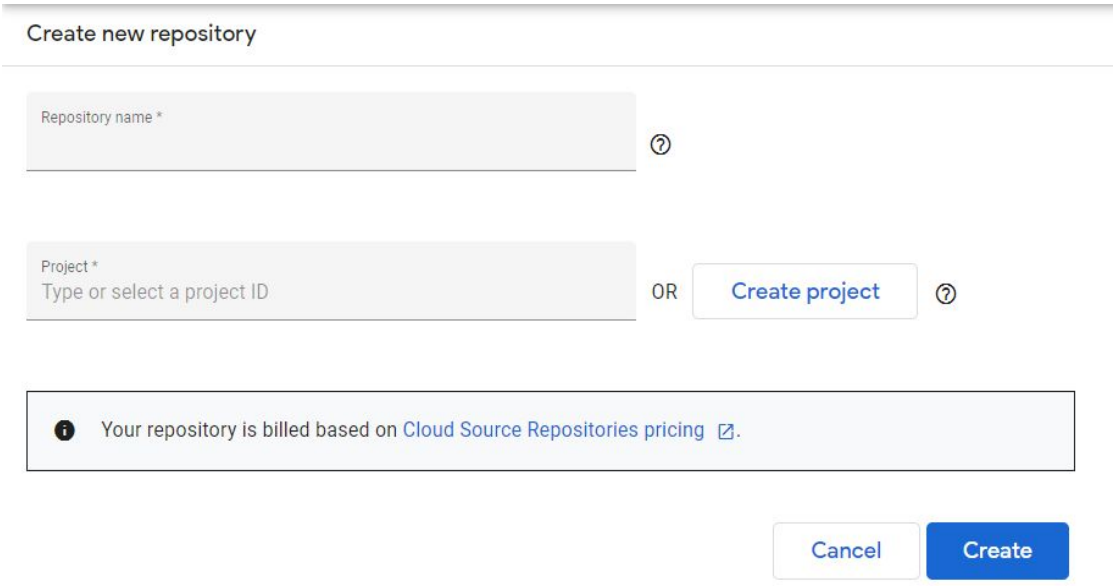
- To create an empty repository click on the **navigation bar** present on the left side of the **console**.
- click on the **Source Repositories** under **Tools** in the **navigation bar**.
- In the **Cloud Source Repositories** click on the **Add Repository** on the right side of the **console**.



- Select **create new repository** and click on **continue**

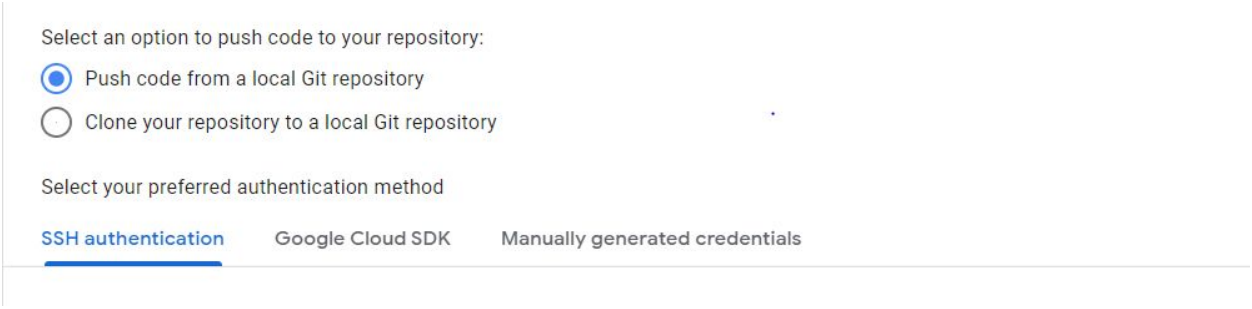


- Enter the name of the repository under **Repository Name** and **project id**. Click on **create**.



Pushing the files to Cloud Source Repository(Project A) from Local Machine

- Once the empty repository is created in Project A, the below window is shown.



- Select **Push code from a local Git repository** to push files to the repository and one of the preferred authentication methods as shown above. Then follow the steps in the authentication method to push the files to the repository.

  [OR]

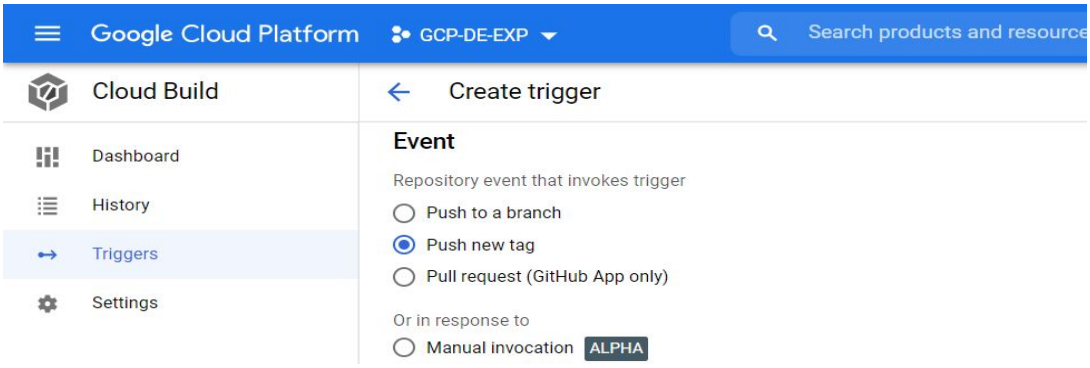
- Refer to the cloud source repository to push the files.
- To confirm that files are successfully cloned to the cloud repository, click on the repository in which cloning is done and files will be present in the respective repository.

Packaging the files from Project A to Project B

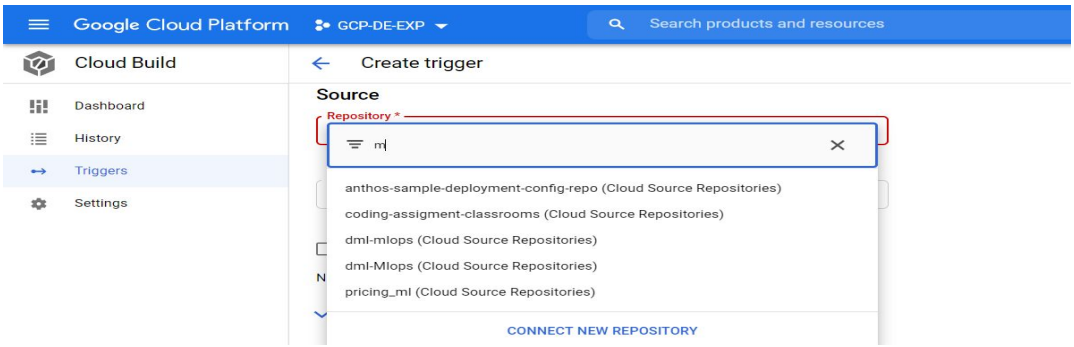
- In order to package the files successfully, there are certain manual steps to be performed in both the projects.
- The packaging is done by creating the triggers in both Project A and Project B. These triggers can be created using **Cloud Build**.

Steps to create trigger in Project A (where files are presently residing)

- Click on **Create Trigger** in **cloud Build** and there are some fields that need to be filled manually.
- **Name** : The name of the trigger. It should be unique in the project.
- **Description** : Any description about the trigger which explains the purpose of the trigger.
- **Event** : select " **Push new tag**"



- **Source** : It shows the drop drow list of all the repositories that one has access to. Required repository needs to be selected.



- **Tag** : The naming of this tag can be in any format. For example, if you want to trigger the build steps only when you push the source code with tag **bt-vx.x.** You can fill the tag section with following **bt-v*.***. So, whenever you push the source code with the above tag expression the build will be triggered which start copying the source code from project A to Project B
- **File type** : Cloud Build Configuration File

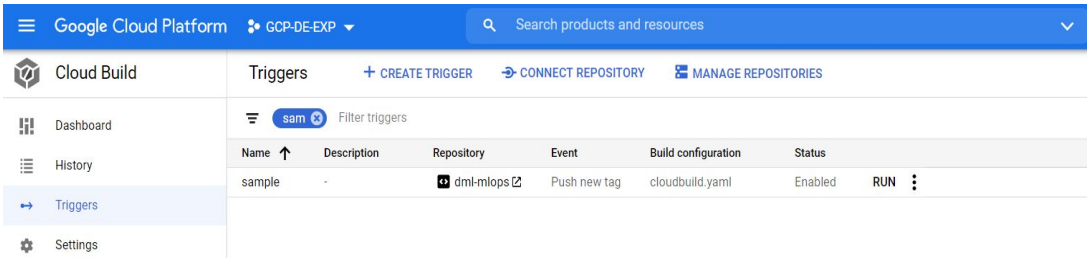


- **File Location** : select the file that ends with " **Cloudbuild.yaml**"

- The above yaml file (**Cloudbuild.yaml**) needs to be updated with the Project_id and Repo_name in line **1,** name and email in line **14**.
- In the advanced section add two variables (**_PROJECT_ID**, **_REPO_NAME**) along with the values(both **_PROJECT_ID** and **_REPO_NAME** Should be of Project B) .



- Click on the create button. If the trigger is successfully created it displays the trigger name in the Page. If the trigger name is not displayed then one of the above steps is not done.



- The above figure shows the sample trigger created successfully by following the steps mentioned.
- Before clicking on run, the Cloud Build Service Account should have writer access for the target repository in Cloud Source Repositories and the tag should be present in the remote location to run the trigger.
- Steps to create and add tag to remote location using Git:
  - If the repository(Cloud Source repository) is already cloned to local, then navigate to that respective directory.
  - If the repository is not cloned, then clone it using this command

```
gcloud source repos clone dml-mlops --project=<project_id>
```

  - Once the repository is successfully cloned, check if any tag is created using this command

```
git tag
```

- If the tag is not created create the tag using this command(Ignore if it is already created)

```
git tag <tag-name>
```

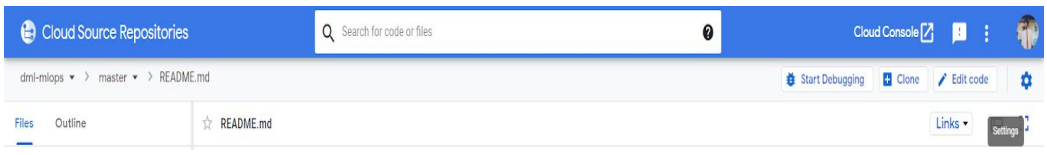
- Once the tag is created successfully, push the tag to remote repository using this command

```
git push --tags
```

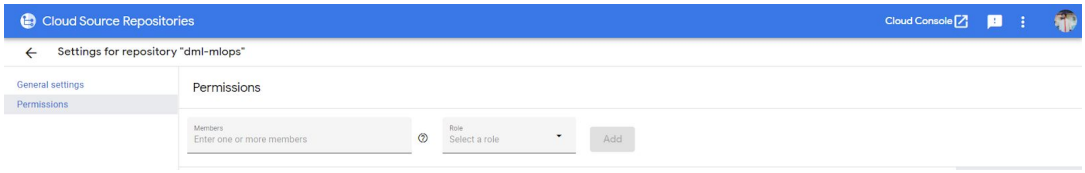
- To check if the tag is pushed to remote location, use this command

```
git ls -remote
```

- To copy the cloud Build Service Account of Project A and paste it in Cloud Source Repository of Project B, these are the steps to follow :
  - Click on **IAM & Admin** on navigation bar that is present on right side of the **Google Cloud Platform**
  - Select **IAM**
  - There will be a set of service accounts listed on the page.
  - The cloud build service account looks in the following format specified below **'[set of numbers] @cloudbuild.gserviceaccount.com'**
  - Copy the **service account** of the above format from Project A.
  - Navigate to the **Cloud Source Repositories** on Project B and click on the Target repository.
  - On the right side of the page, click on the **settings**.



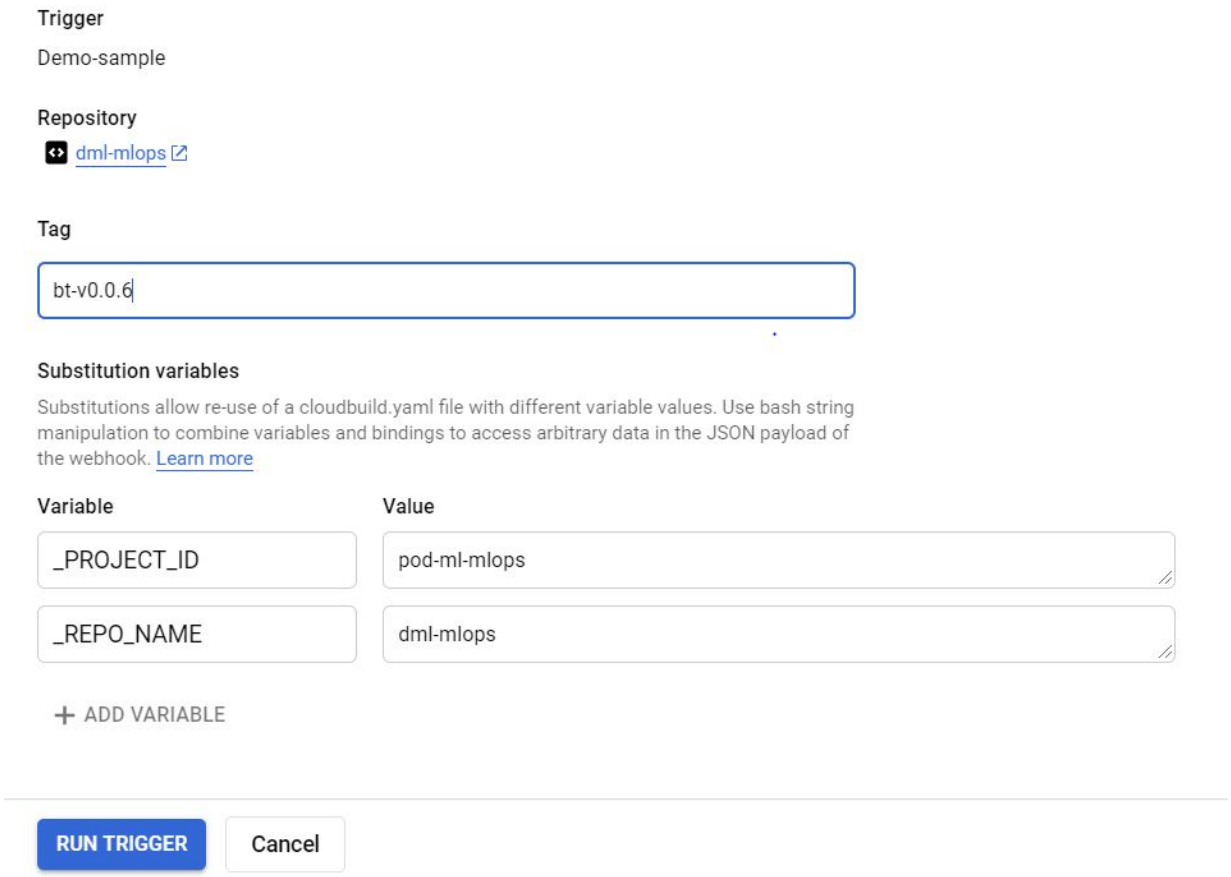
  - Once we click on **Permissions**, there will be two empty fields named **Members** and **Role.**



- Paste the **cloud Build Service Account** of Project A in the Member Field.
- Click on the **Role**, Select **Source Repository** and then click on **Source Repository Writer**.



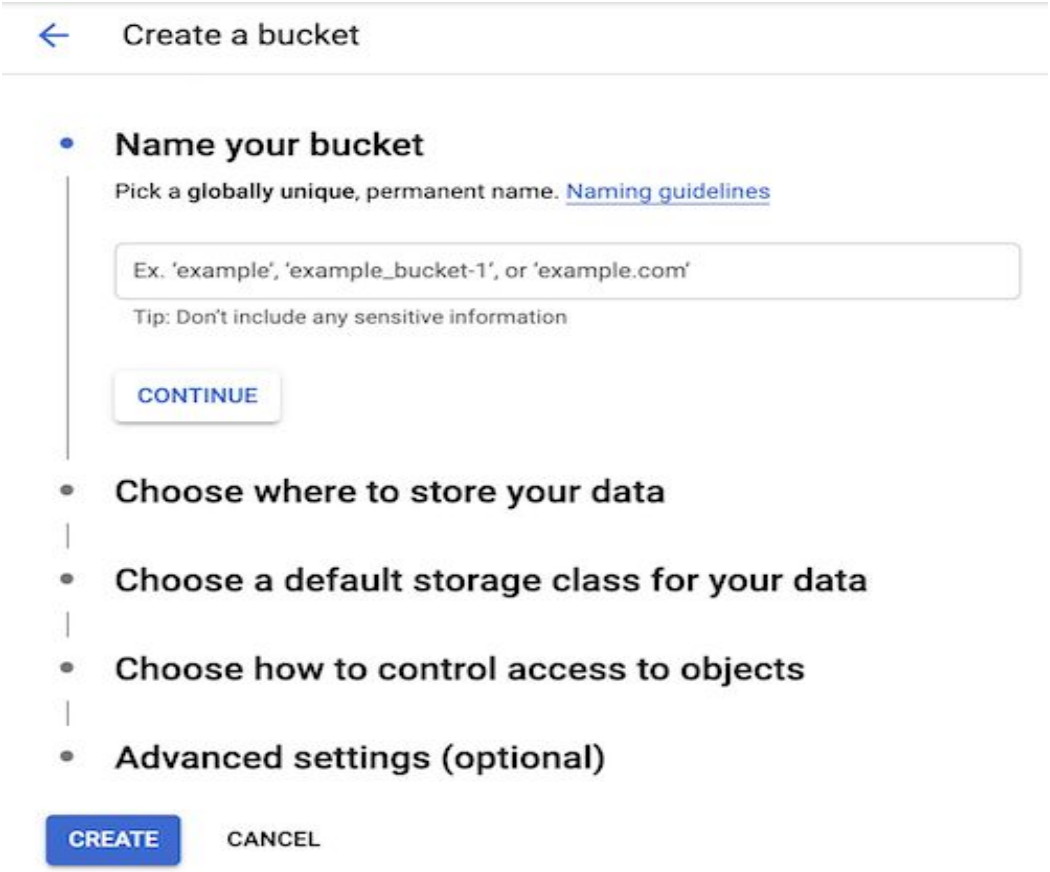
- Click on the Add button next to **Role** Field.
- Once the above steps are completed, click on **run**. It pops the window where the tag is to be entered to run the trigger.



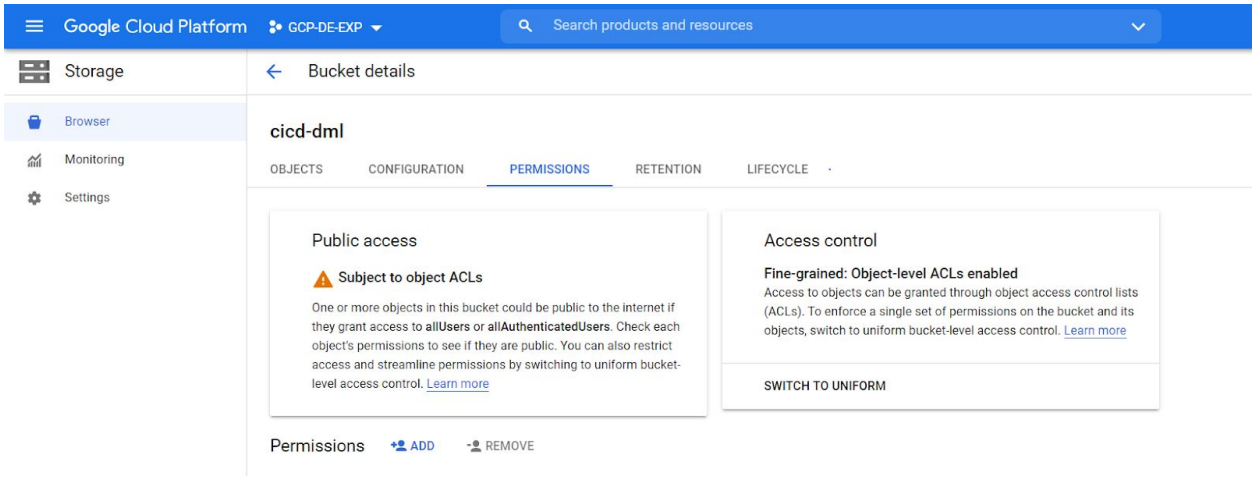
- Enter the **tag** that is pushed using git previously and click on **Run Trigger** to run the trigger.

Steps to create the Google Cloud Storage Bucket to upload the dataset on <u>Project B</u>

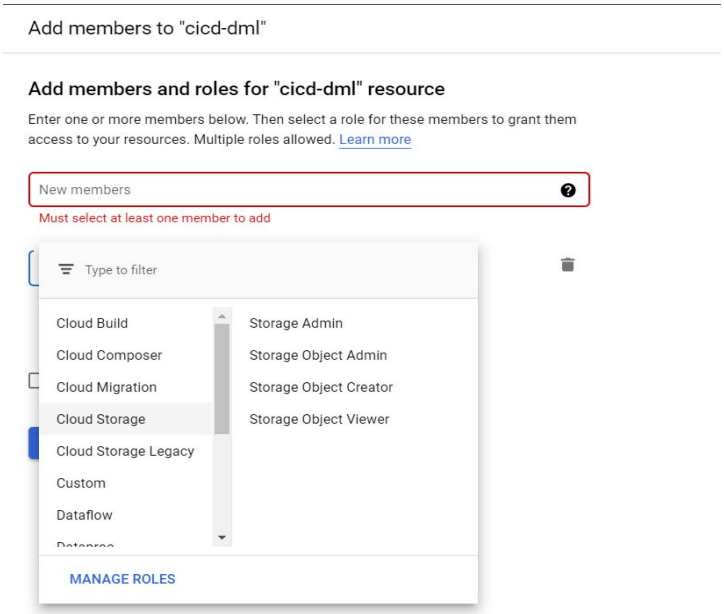
- Open the Cloud Storage browser in the Google Cloud Console.
- Click **Create bucket** to open the bucket creation form.



- Enter your bucket information and click **Continue** to complete each step:
  - Enter a unique **Name** for your bucket. - Bucket name should be “dml-cicd”
  - Choose **Region** for **Location type** and **us-east1 (South Carolina)** for **Location**.
  - Choose **Standard** for **default storage class**.
  - Choose **Uniform** for **Access control**.
- Click **Create**
- Once the **GCS Bucket** is created create two folders named **datasets** and **module_testing.**
- Inorder to copy the files present in the **GCS bucket**, The **cloud Build Service Account** (**[set of numbers] @cloudbuild.gserviceaccount.com**) of Project B should have **storage admin** access to this **Bucket** in project A.
  - Navigate to **permissions** in the **GCS Bucket** and click on the **add** as shown in below figure



- A window is popped to enter the **member** and **role** for **GCS Bucket** once the **add** button is clicked.

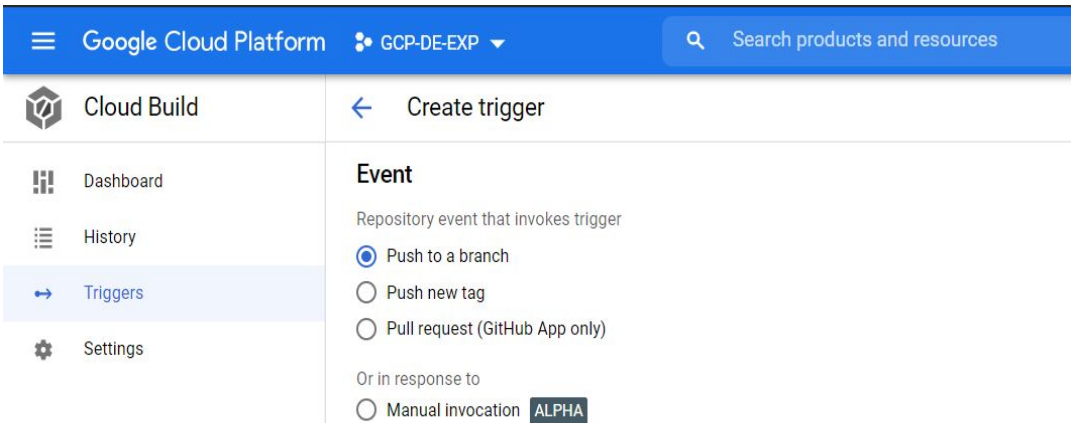


- Enter the **Cloud Build Service Account** of Project B in the **member** field and select **Storage Admin** under **Cloud Storage**.Then click on **add** to add the member.

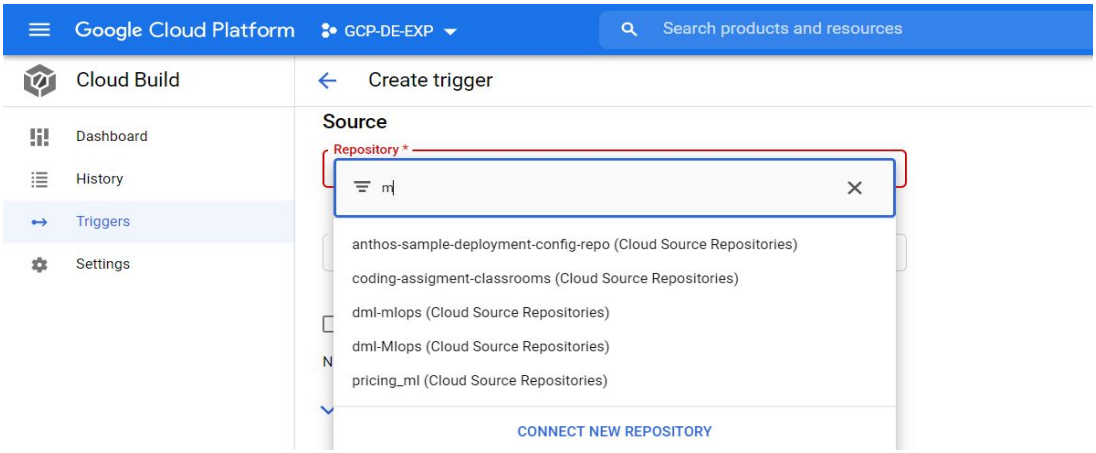
Steps to create trigger in Project B (where the files are transferred)

- Click on **Create Trigger** in **cloud Build** and there are some fields that need to be filled manually.

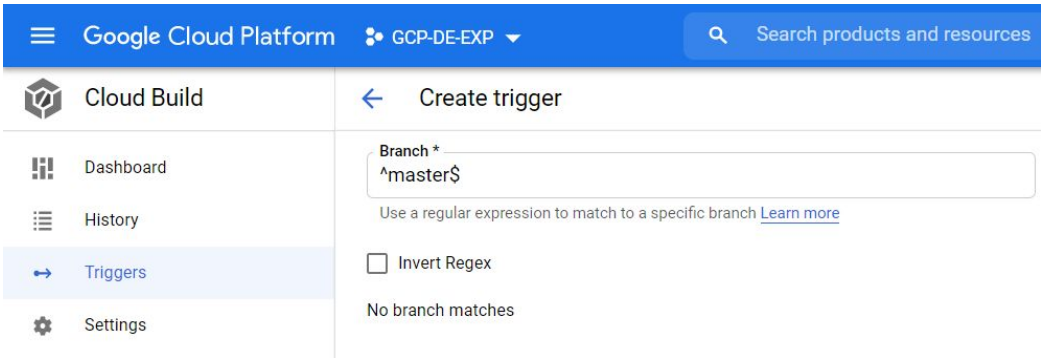
- **Name** : The name of the trigger. It should be unique in the project.
- **Description** : Any description about the trigger which explains the purpose of the trigger.
- **Event** : select " **Push to a branch**"



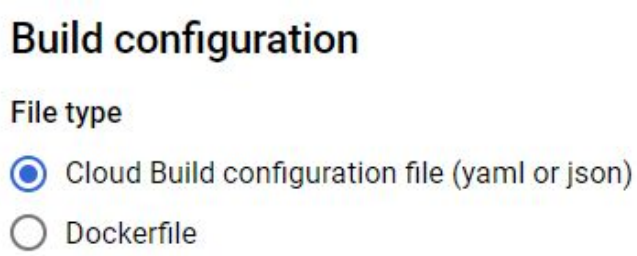
- **Source** : It shows the drop drow list of all the repositories that one has access to. Required repositorie need to be selected.



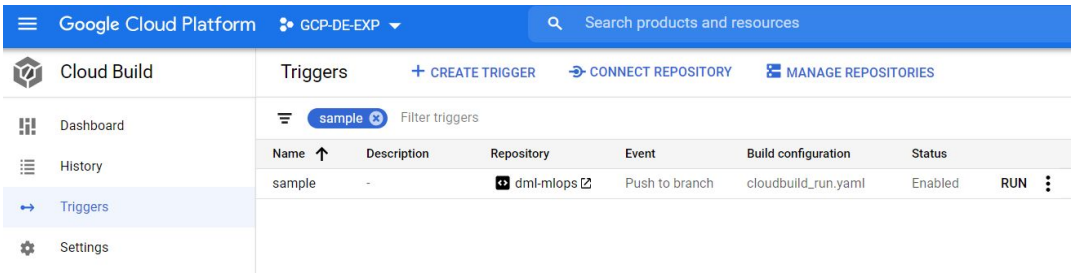
- **Branch** : '**^master$**'



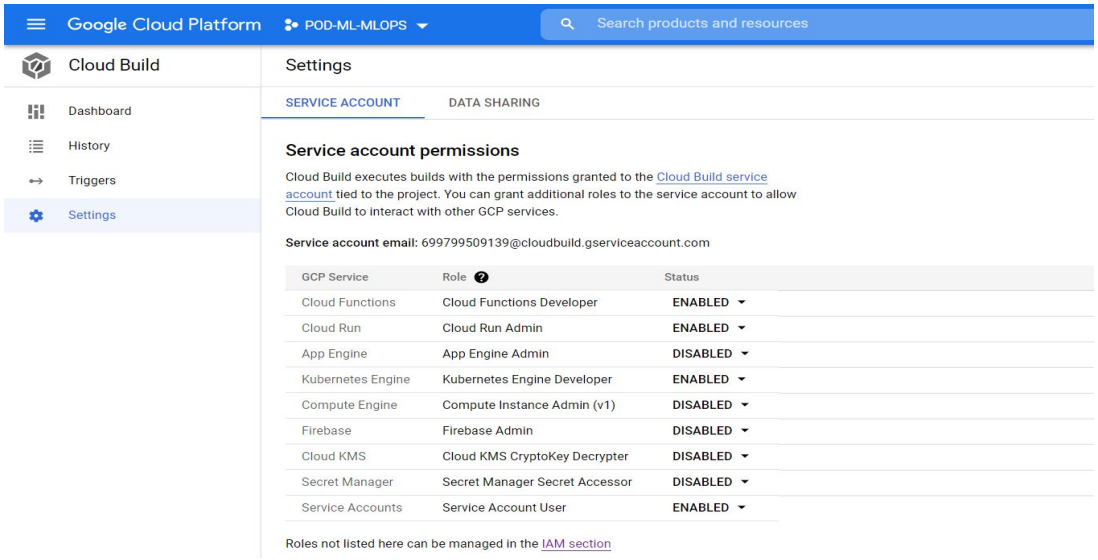
- **File type** : Cloud Build Configuration File



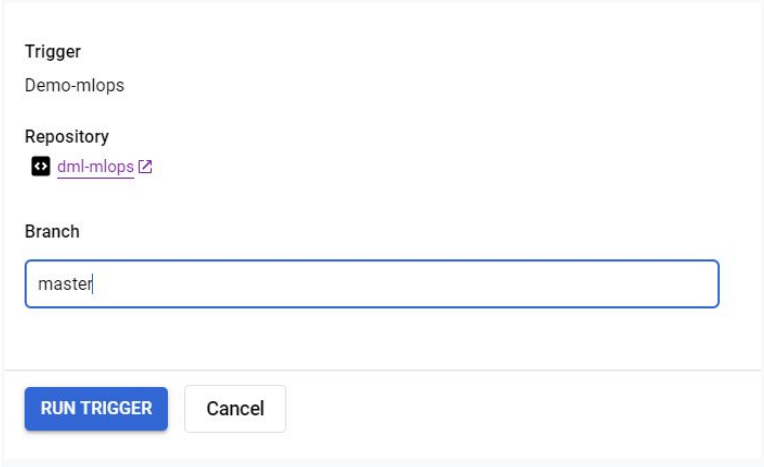
- **File Location** : select the file that ends with " **Cloudbuild_run.yaml**"
- Click on the create button. If the trigger is successfully created it displays the trigger name in the Page.If the trigger name is not displayed then one of the above steps is not done.



- The above figure shows the trigger that is successfully created following the steps mentioned above.
- Before running the trigger there are a set of services that need to be enabled for the trigger. Those services are shown below.



- The **cloud Build Service account** of Project B should have the following **roles** to run the trigger successfully.
    - BigQuery Data Editor
    - BigQuery Job User
    - Cloud Build Service Account
    - Cloud Functions Developer
    - Kubernetes Engine Developer
    - Service Account User
- Once the above steps are completed, click on run. This pops window as shown below



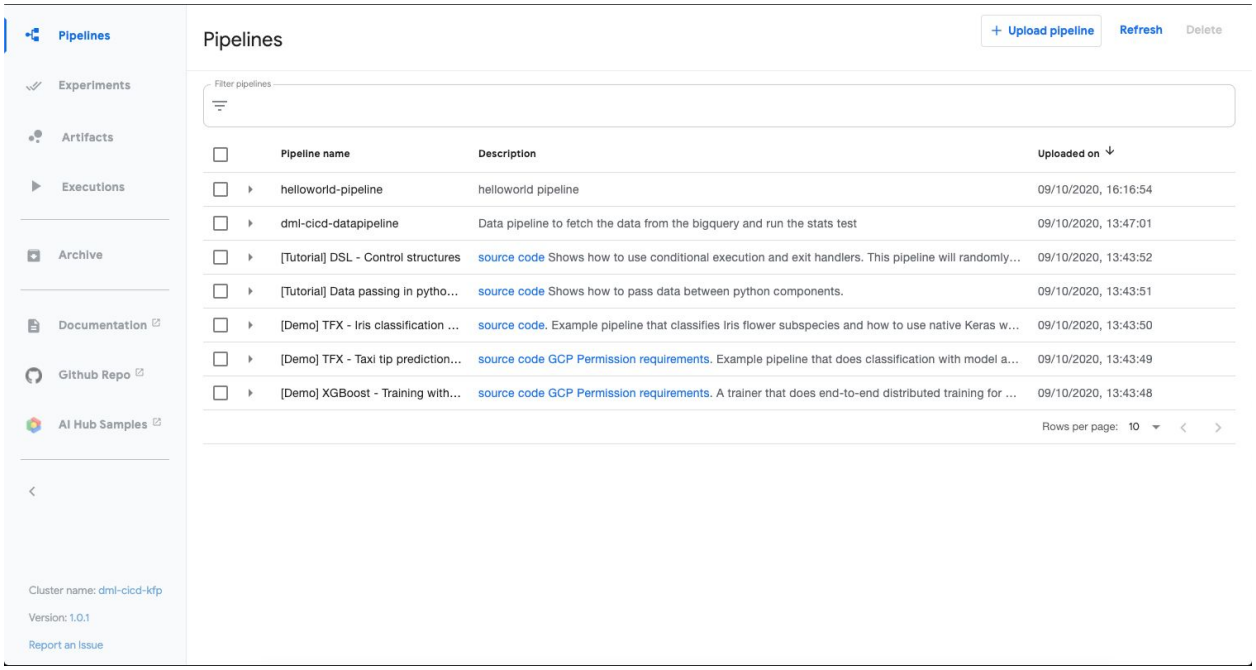
- Click on the **Run Trigger** to successfully run the trigger.

NOTE - Before running the ML workflow, make sure that dataset and Tables have been created in bigquery. Navigate to bigquery and check if the dataset, and table is created

Steps to run the ML workflow

Once you see the **link.txt** file in the bucket URI **gs://<bucket_name>/build/pipeline/latest/** open the file and you see the link with format like this https://xxxxxxxxx-dot-us-central2.pipelines.googleusercontent.com/#/pipelines copy the link and paste it in the browser.

The browser should open the kubeflow dashboard

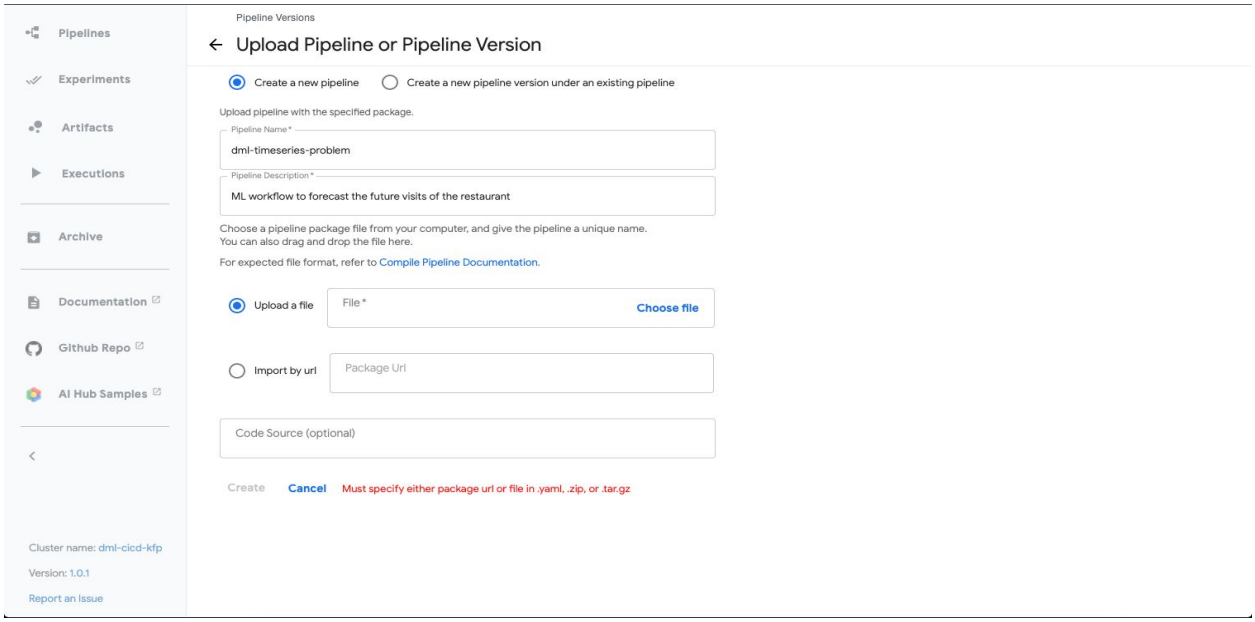


If you see the above page, then it means, successfully kubeflow is installed to the kubernetes cluster and we are ready to upload our custom ML workflow packages and run the pipeline

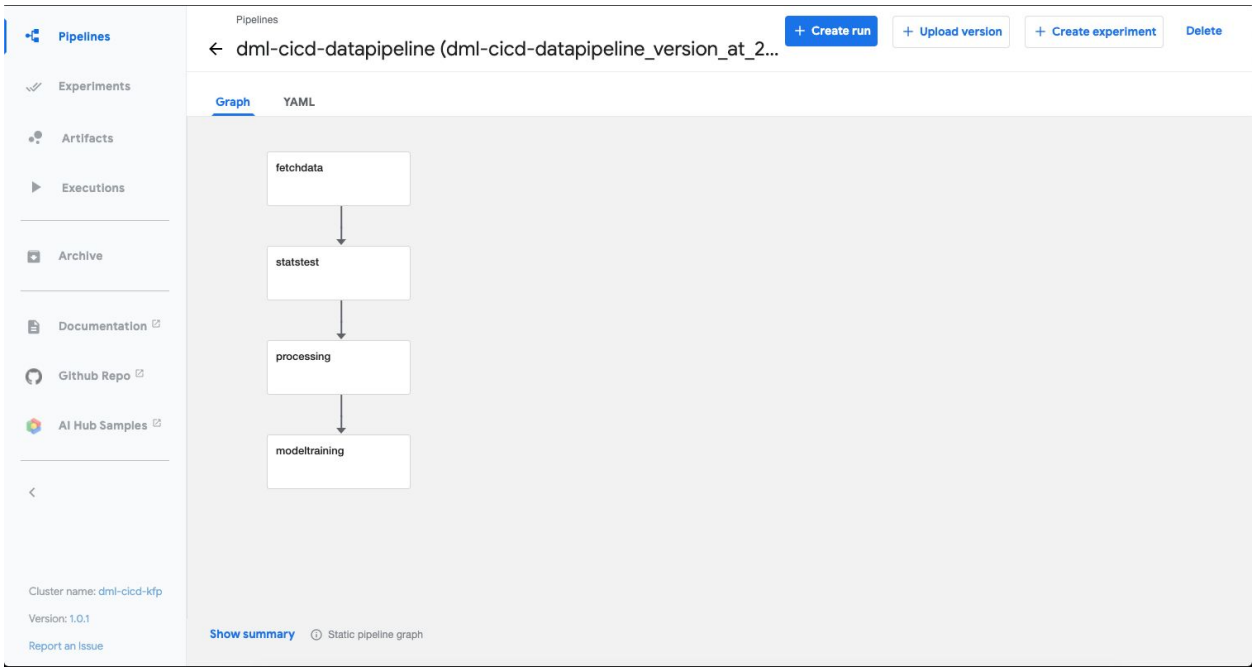
NOTE - Before running the pipeline, make sure the table is created in Bigquery

Follow the steps to run a pipeline in kubeflow

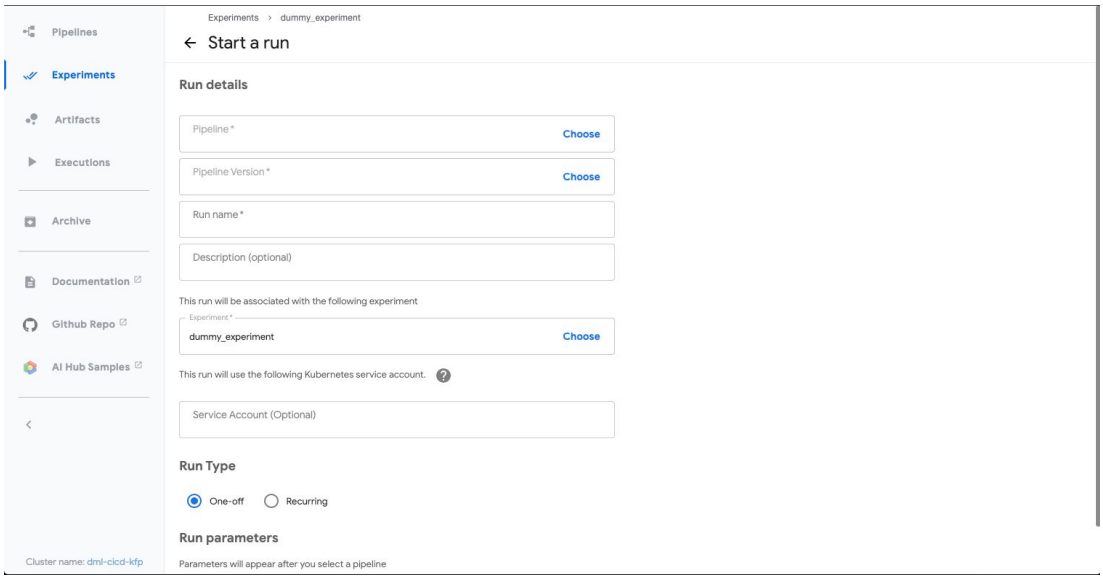
1. First step is to upload the **pipeline.tar.gz** package. You can find the package in GCS bucket path **gs://<bucket_name>/build/pipelines/latest/pipeline.tar.gz**. Download this file locally, and upload the pipeline (or) you can specify GCS URI for that package



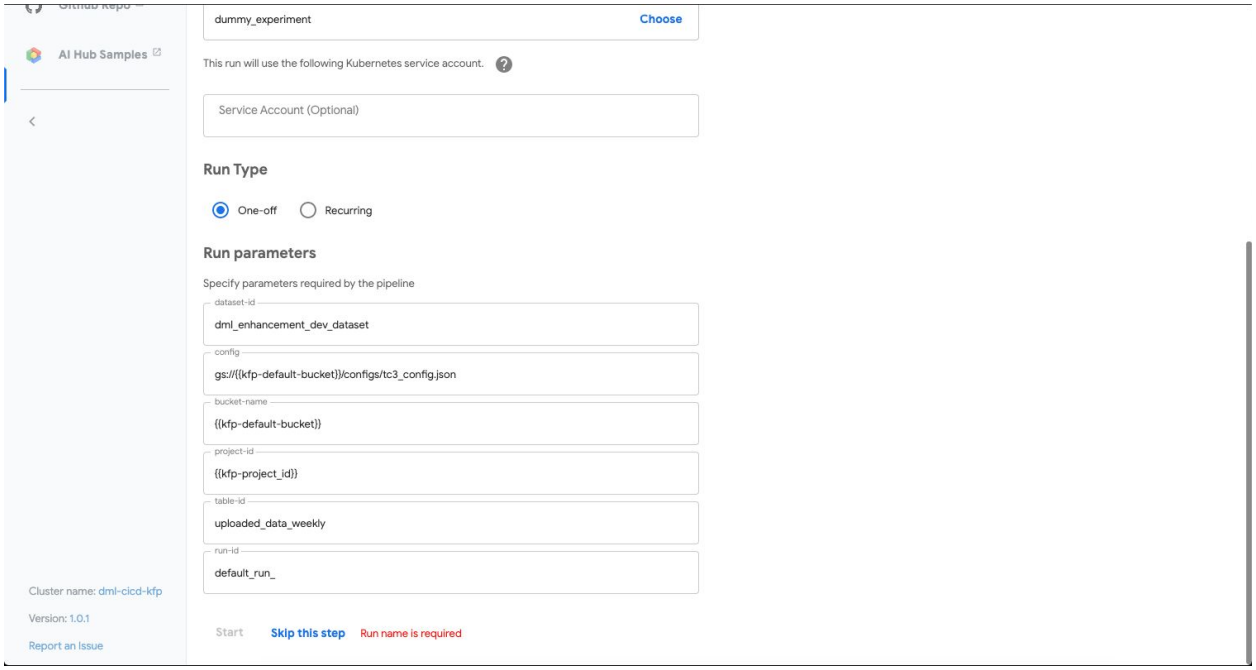
2. Click **Create**
3. If the above step is successful you can go to the pipelines page and see the created pipeline. Click the created pipeline, the graph should be shown like this



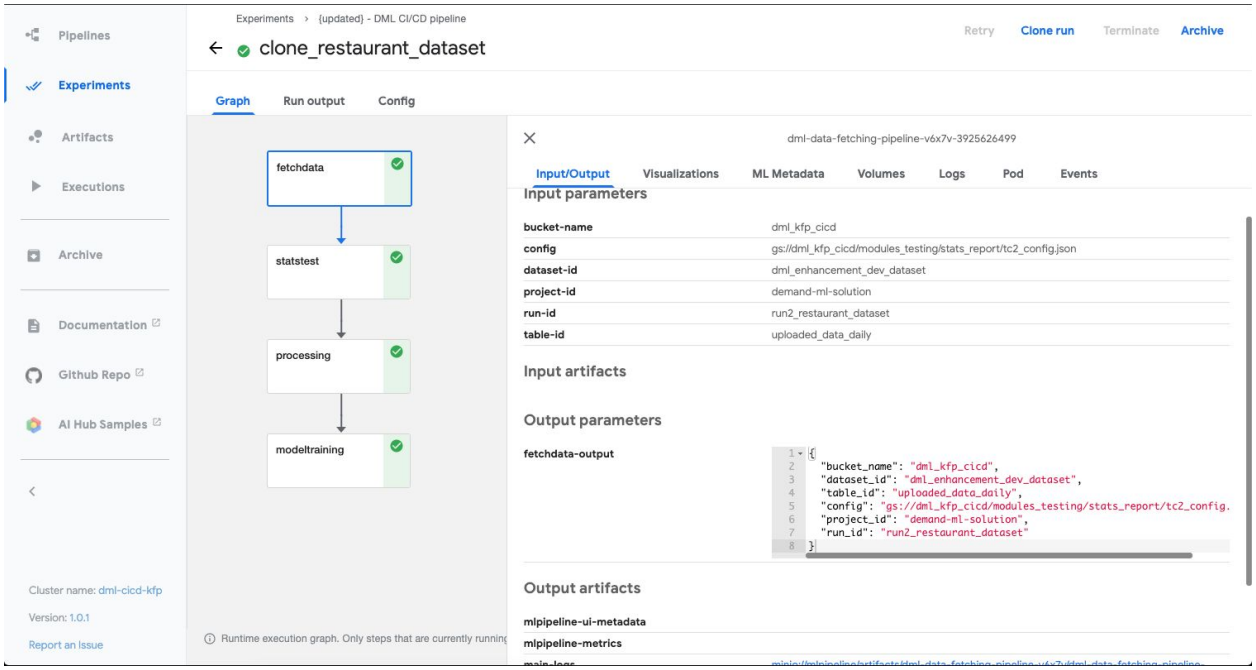
4. The pipeline is successfully created, now we are ready to run the pipeline and experiment with it
5. Go to **Experiments**
    a. click **create** experiment
    b. Fill details to the placeholder asked
    c. Click **Next**
6. Start a **New Run**



    a. Choose the **pipeline**
    b. Select the pipeline version, if there are many versions to the same pipeline
    c. Give **Run name**
    d. Give Description(optional)
    e. Select Run-Type **one-off**
    f. Based on the pipeline selected, the run parameters placeholder will be appear
        i. Give dataset name - The **name** should be same as what we have defined in the substitutions **_DATASET_NAME** in **cloudbuild_run.yaml**
        ii. For config, give the GCS URI path for the object **tc2_config.json** The file path should be **gs://<bucket_name>/module_testing/tc2_config.json**
        iii. Give **bucket_name**
        iv. Give **Project Id**
        v. Give Table Id - The **name** should be same as what we have defined in the substitutions **_TABLE_NAME** in **cloudbuild_run.yaml** file
        vi. Give **run_id -** name it with your convenience
        vii. Click **Start**



7. The pipeline should start
8. Once the pipeline is completed you can see the graph like this



9. On the last component, the kubeflow submits AI jobs for model training. So, go to the **AI platform > Jobs**. You can see the jobs getting submitted

10. The model forecasts can be see in the file path **gs://<bucket_name>/kfp_pipeline/ai_predictions/<run_id>/*.csv.** The statistical reports can be seen in the bucket path **gs://<bucket_name>/kfp_pipeline/reports/*.json.** This JSON file indicates the characteristics of the trend