CI/CD for Kubeflow Pipelines on Vertex AI

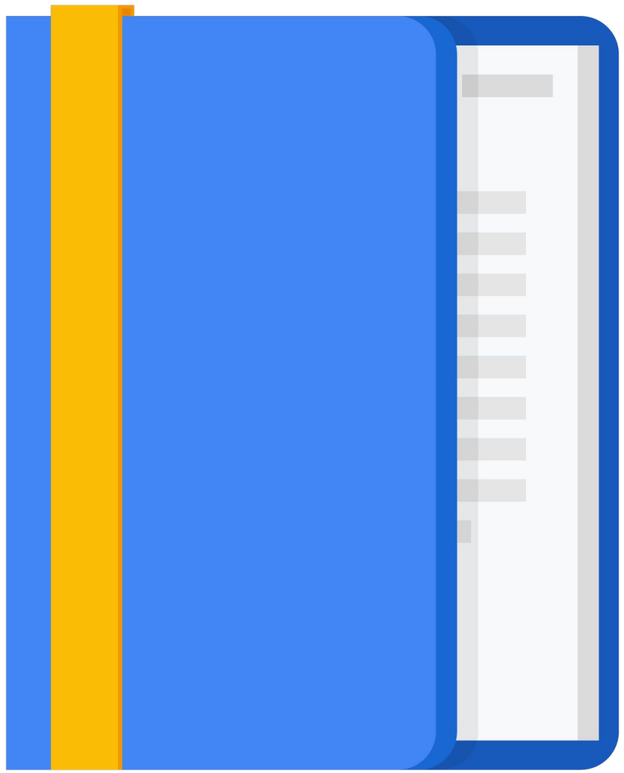
Agenda

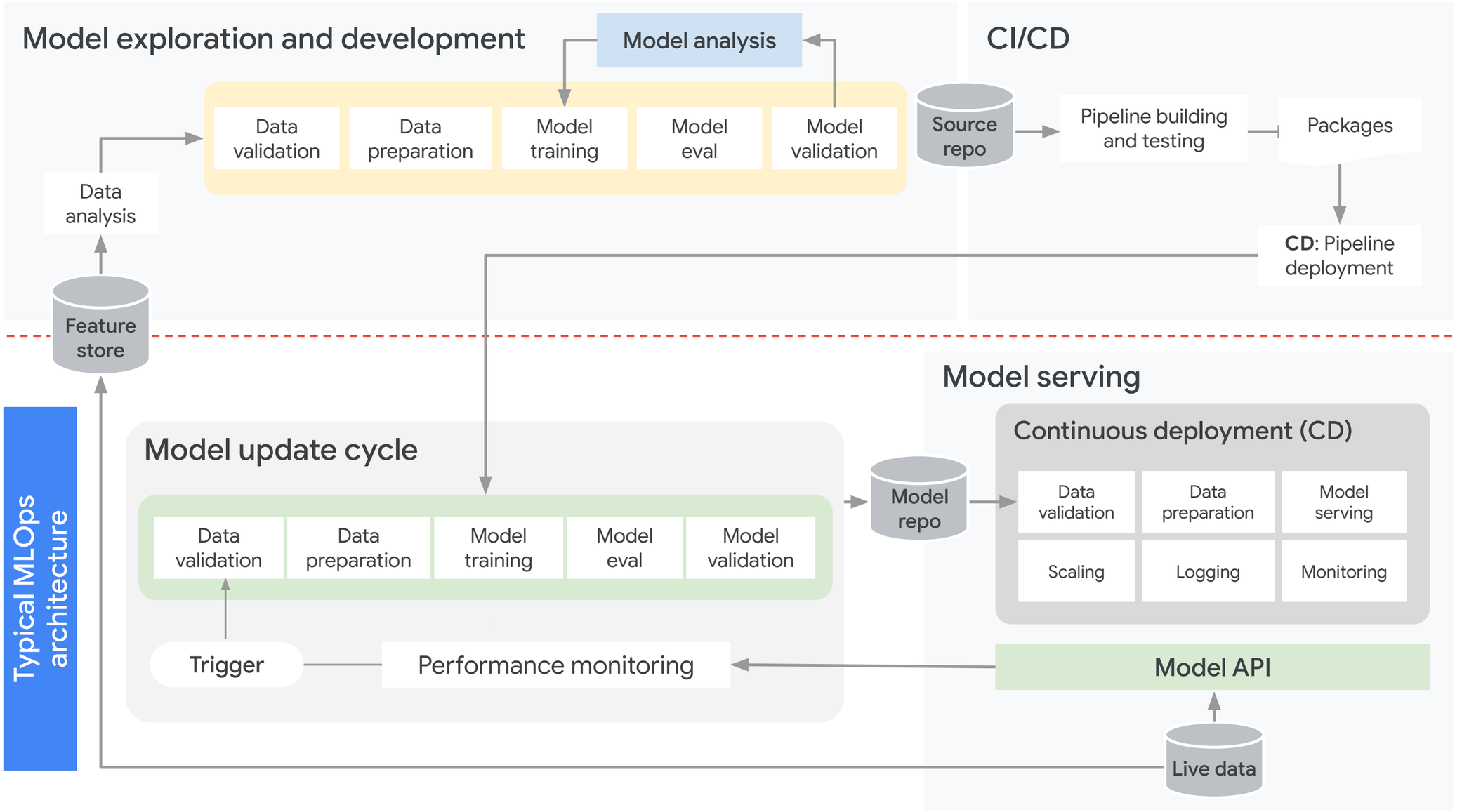
Concept Overview

Cloud Build Builders

Cloud Build Configuration

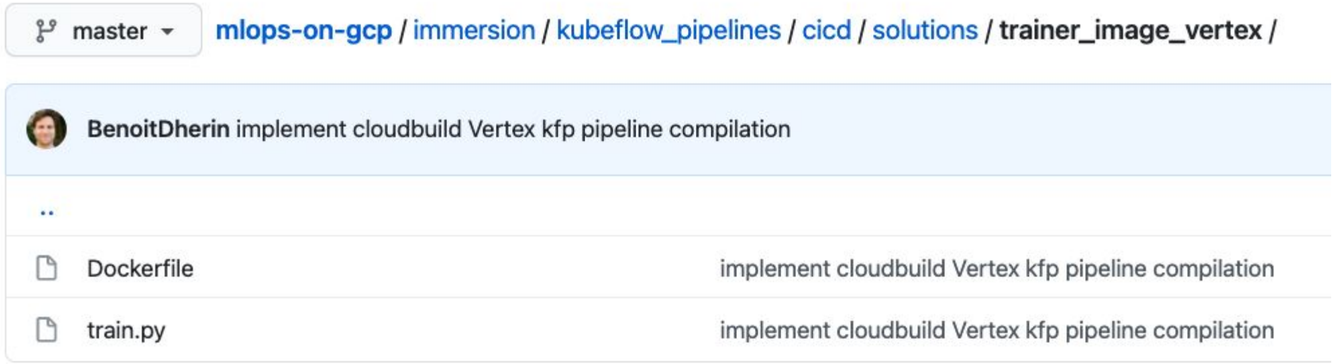
Cloud Build Triggers

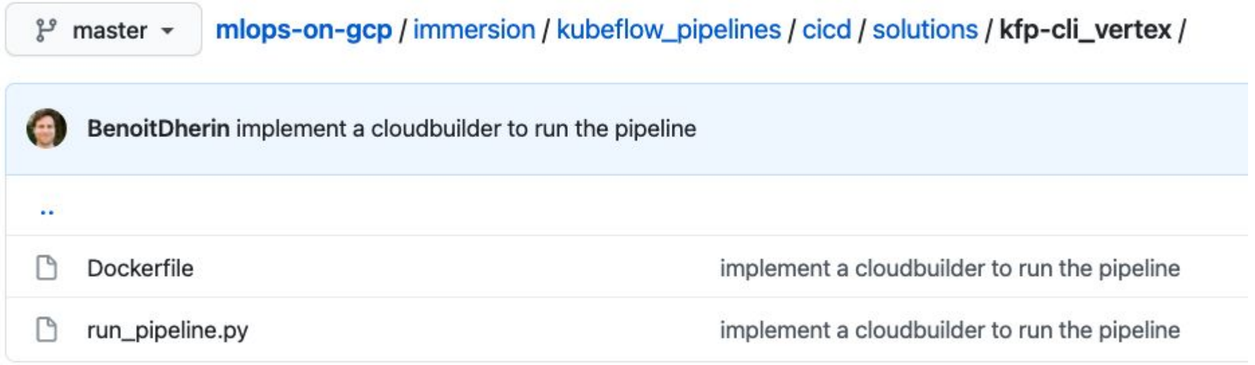



Model exploration and development
Model analysis
Data validation
Data preparation
Model training
Model eval
Model validation
Data analysis
Feature store
CI/CD
Source repo
Pipeline building and testing
Packages
CD: Pipeline deployment
Typical MLOps architecture
Model update cycle
Data validation
Data preparation
Model training
Model eval
Model validation
Trigger
Performance monitoring
Model repo
Model serving
Continuous deployment (CD)
Data validation
Data preparation
Model serving
Scaling
Logging
Monitoring
Model API
Live data

Every container is a self-contained directory in repo

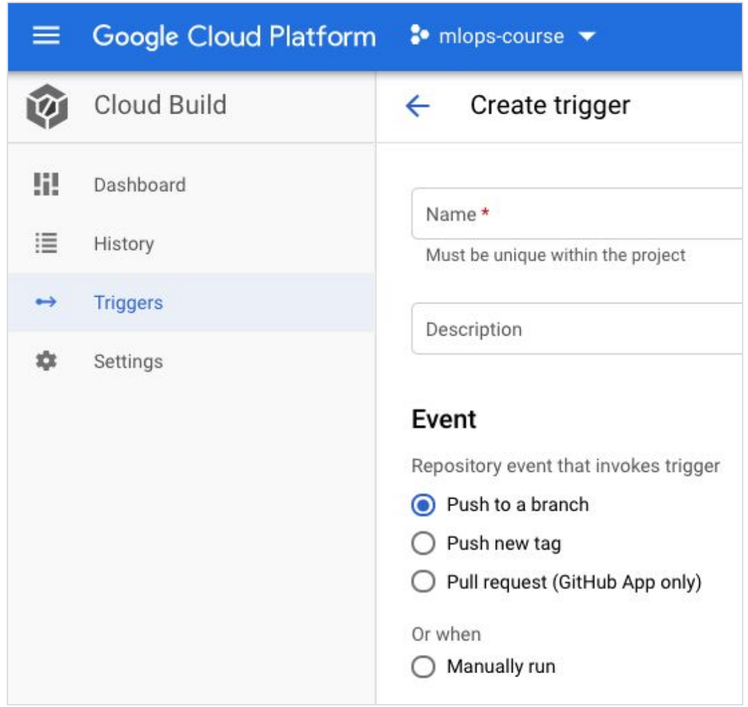
If any of these files is changed, you need to rebuild and push the Docker image.





For CI/CD, use a GitHub trigger to rebuild ML artifacts

Connect the GitHub repository to your Google Cloud account, and then trigger a Cloud Build from a GitHub trigger.
Do this for every container in your ML pipeline.




```yaml
steps:
# Build the trainer image
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/$PROJECT_ID/trainer_image_covertype_vertex:latest', '.']
  dir: trainer_image_vertex


# Compile the pipeline
- name: 'gcr.io/$PROJECT_ID/kfp-cli-vertex'
  args:
  - '-c'
  - |
    dsl-compile-v2 --py pipeline.py --output covertype_kfp_pipeline.json
  env:
  - 'PIPELINE_ROOT=gs://$PROJECT_ID-vertex/pipeline'
  - 'PROJECT_ID=$PROJECT_ID'
  - 'REGION=$_REGION'
  - 'SERVING_CONTAINER_IMAGE_URI=us-docker.pkg.dev/vertex-ai/prediction/sklearn-cpu.0-20:latest'
  - 'TRAINING_CONTAINER_IMAGE_URI=gcr.io/$PROJECT_ID/trainer_image_covertype_vertex:latest'
  - 'TRAINING_FILE_PATH=gs://$PROJECT_ID-vertex/data/training/dataset.csv'
  - 'VALIDATION_FILE_PATH=gs://$PROJECT_ID-vertex/data/validation/dataset.csv'
  dir: pipeline_vertex
```

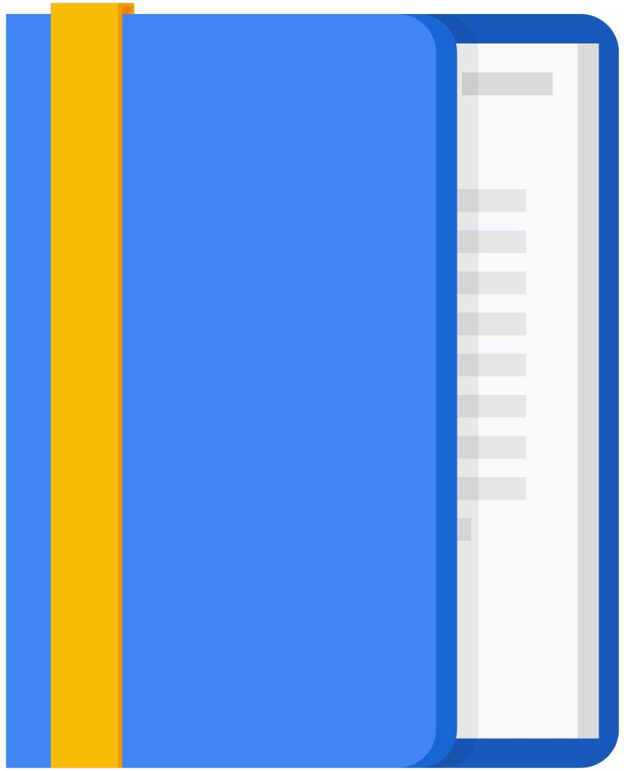
Agenda

Concept Overview

Cloud Build Builders

Cloud Build Configuration

Cloud Build Triggers



What are cloud builders?

Cloud configuration/provisioning actions that are packaged as Docker containers

Typical cloud builder actions:

- Building a Docker image from a Dockerfile
- Pushing a Docker image into a Google Cloud project registry
- Deploying a VM instance on Compute Engine
- Running a Kubeflow pipeline on Vertex Pipelines

What are cloud builders?

Standard

Already packaged config actions

Docker Registry:

`gcr.io/cloud-builders/`

Container Code:
GoogleCloudPlatform/cloud-builders

Custom

config actions packaged by you

Docker Registry:

`gcr.io/<YOUR_PROJECT>/`

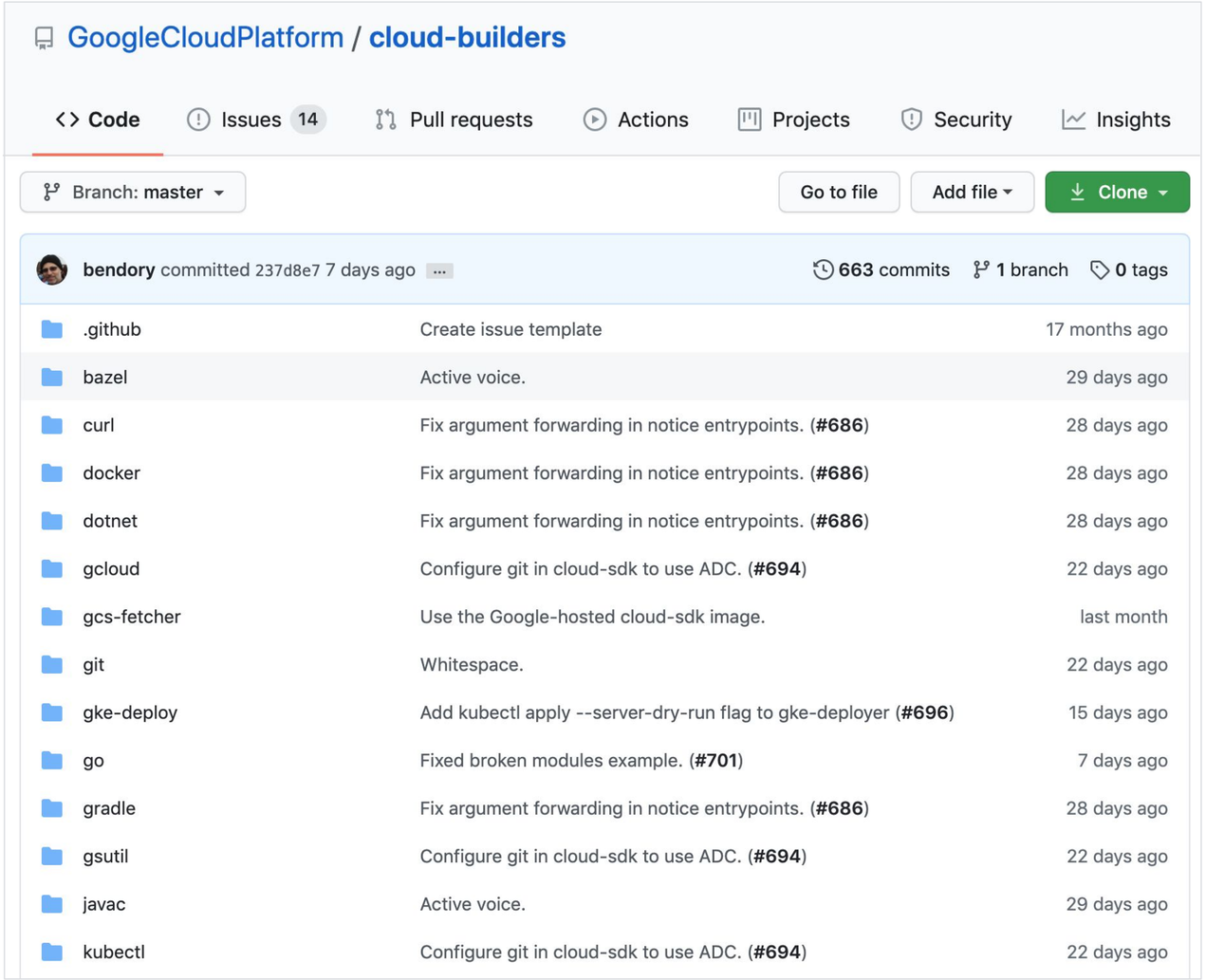
Standard Cloud Builders

Standard cloud builders

Builder	Name	Example
bazel	`gcr.io/cloud-builders/bazel`	bazel example
docker	`gcr.io/cloud-builders/docker`	docker example
git	`gcr.io/cloud-builders/git`	git example
gcloud	`gcr.io/cloud-builders/gcloud`	gcloud example
gke-deploy	`gcr.io/cloud-builders/gke-deploy`	gke-deploy example

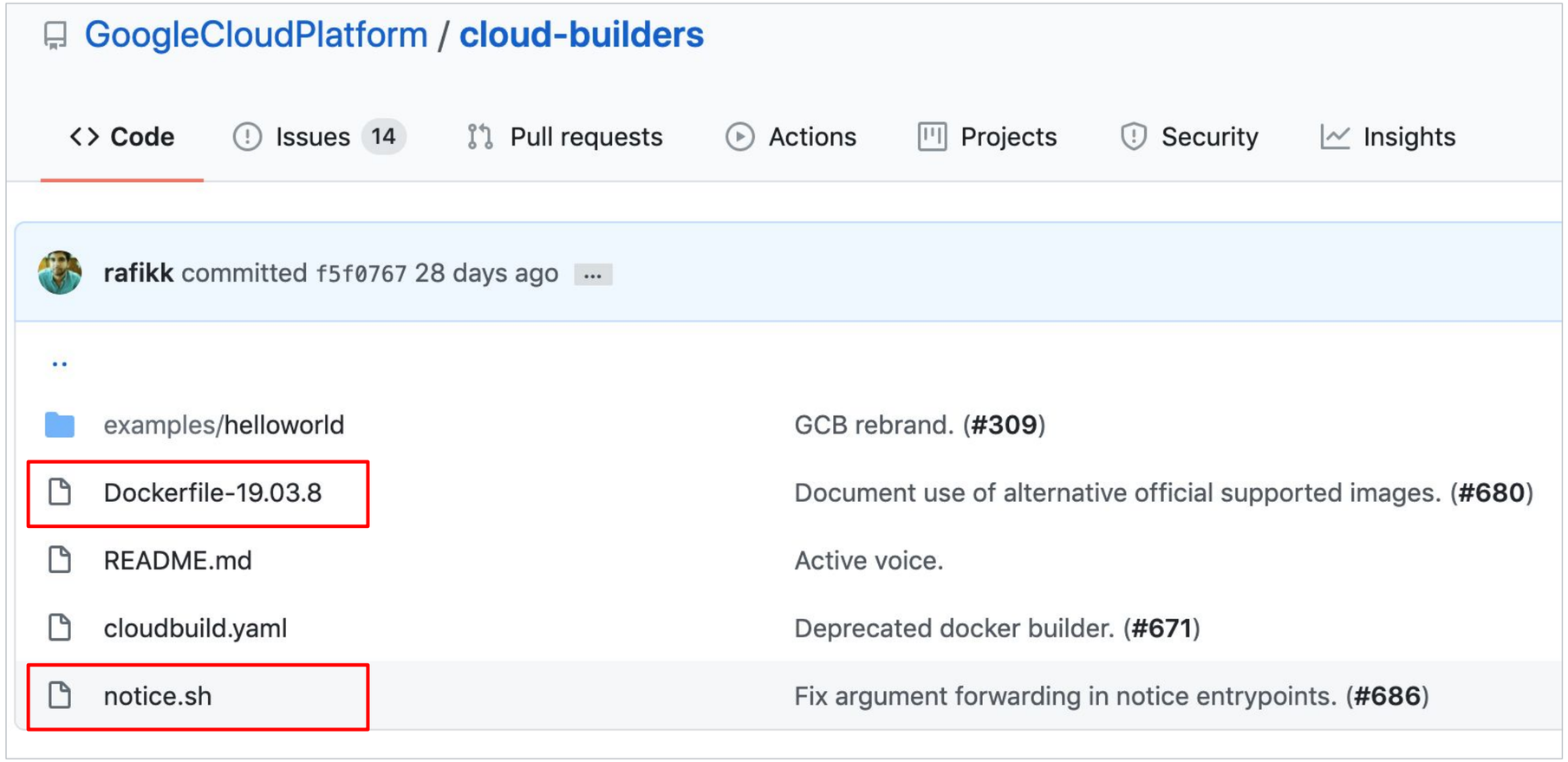
...wrap standard config tools as Docker containers

- `bazel` : runs the bazel tool
- `curl` : runs the curl tool
- `docker` : runs the docker tool
- `dotnet` : run the dotnet tool
- `gcloud` : runs the gcloud tool
- `gcs-fetcher` : efficiently fetches objects from Google Cloud
- `git` : runs the git tool
- `gke-deploy` : deploys an application to a Kubernetes cluster,
- `go` : runs the go tool
- `gradle` : runs the gradle tool
- `gsutil` : runs the gsutil tool
- `javac` : runs the javac tool
- `kubectl` : runs the kubectl tool
- `mvn` : runs the maven tool
- `npm` : runs the npm tool
- `wget` : runs the wget tool
- `yarn` : runs the yarn tool



https://github.com/GoogleCloudPlatform/cloud-builders/tree/master/docker

Example: Docker cloud builder



```
FROM launcher.gcr.io/google/ubuntu16_04

ARG DOCKER_VERSION=5:19.03.8~3-0~ubuntu-xenial

RUN apt-get -y update && \
    apt-get -y install \
        apt-transport-https \
        ca-certificates \
        curl \
        make \
        software-properties-common && \
    curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add - && \
    apt-key fingerprint 0EBFCD88 && \
    add-apt-repository \
       "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
       xenial \
       edge" && \
    apt-get -y update && \
    apt-get -y install docker-ce=${DOCKER_VERSION} docker-ce-cli=${DOCKER_VERSION}

COPY notice.sh /usr/bin

ENTRYPOINT ["/usr/bin/notice.sh"]
```

When ran, `/usr/bin/notice.sh` is executed

Example: Docker cloud builder

notice.sh

```
#!/bin/sh
echo '
                    ***** NOTICE *****

Alternative official `docker` images, including multiple versions across
multiple platforms, are maintained by the Docker Team. For details, please
visit https://hub.docker.com/_/docker.

                  ***** END OF NOTICE *****
'
/usr/bin/docker "$@"
```

When run, the container passes its args to the `docker` command.

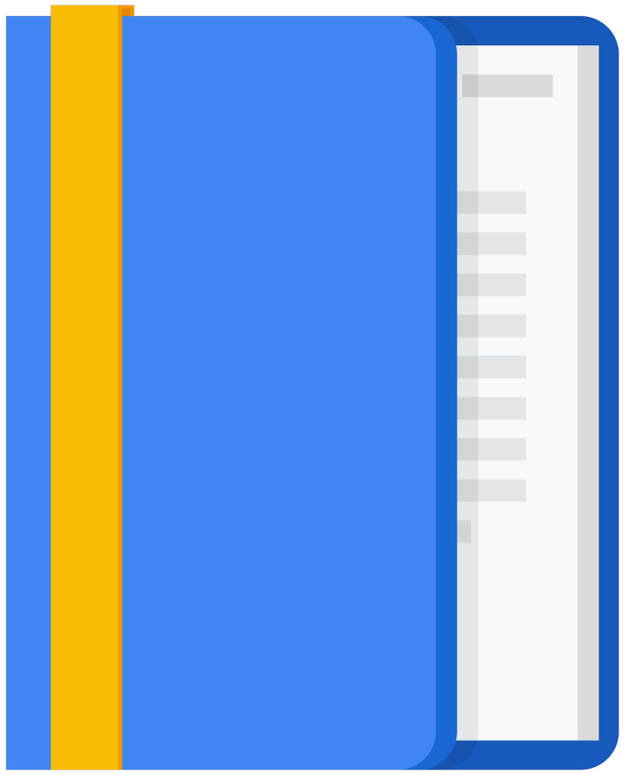
Agenda

Concept Overview

Cloud Build Builders

Cloud Build Configuration

Cloud Build Triggers



Cloud Build configuration file

Describe the cloud builders to be run step by step

cloudbuild.yaml

```
steps:

- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/$PROJECT_ID/trainer_image_covertype_vertex:latest', '.']
  dir: trainer_image_vertex
```

Running Cloud Build

```
gcloud builds submit . --config cloudbuild.yaml --substitutions $SUBSTITUTIONS
```

Command

Cloud Build config file describing the cloud builders to be run

Variable substitutions to be made in the cloudbuild.yaml config file

A simple Cloud Build step

```
steps:

- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/$PROJECT_ID/trainer_image_covertype_vertex:latest', '.']
  dir: trainer_image_vertex
```

The cloud builder container URI to be run

The CWD in the Docker container from which the entrypoint is executed

The arguments to be passed to the container entrypoint

Persistence `dir:`

```
steps:

- name:
  args:
  dir: <rel path>
```

A cloud builder container has its current working directory:

- Set by default to /workspace in the container.
- Shared between steps.

This can be modified by specifying a dir path that:

- Will resolve to /workspace/<path>.
- Will still be shared between steps if the path is relative.

Caution: If the path is absolute, the CWD will not persist between steps.

Substitutions: $_VARIABLE_NAME

```
steps:

- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/$PROJECT_ID/$_TRAINER_IMAGE_NAME:$TAG_NAME', '.']
  dir: $_PIPELINE_FOLDER/$_IMAGE_NAME
```

```
gcloud builds submit . --config cloudbuild.yaml \
   --substitutions ‘_PIPELINE_FOLDER=.,_IMAGE_NAME=trainer_base’
```

Custom cloud builder

```
# Upload the pipeline
- name: 'gcr.io/$PROJECT_ID/kfp-cli-vertex'
  args:
    - '-c'
    - |
        dsl-compile-v2 --py pipeline.py --output covertype_kfp_pipeline.json
  env:
  - 'PIPELINE_ROOT=gs://$PROJECT_ID-vertex/pipeline'
  - 'REGION=$_REGION'
  - etc.

  dir: pipeline_vertex
```

No different from the standard cloud builders (but the registry is your own project registry)

Passing environment variables

```
- name: 'gcr.io/$PROJECT_ID/kfp-cli-vertex'
  args:
    - '-c'
    - 'dsl-compile-v2 --py pipeline.py --output covertype_kfp_pipeline.json
  env:
  - 'PIPELINE_ROOT=gs://$PROJECT_ID-vertex/pipeline'
  - 'PROJECT_ID=$PROJECT_ID'
  - 'REGION=$_REGION'
  - 'SERVING_CONTAINER_IMAGE_URI=<your_serving_container>'
  - 'TRAINING_CONTAINER_IMAGE_URI=<your_training_container>'
  - 'TRAINING_FILE_PATH=gs://$PROJECT_ID-vertex/data/training/dataset.csv'
  - 'VALIDATION_FILE_PATH=gs://$PROJECT_ID-vertex/data/validation/dataset.csv'

  dir: pipeline_vertex
```

This script takes its input from env variables

That's how the env variables are passed to the script

Pushing images to Container Registry

Build the image locally on the build node

```
steps:
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/$PROJECT_ID/trainer_image_covertype_vertex:latest', '.']
  dir: trainer_image_vertex


Images: ['gcr.io/$PROJECT_ID/trainer_image_covertype_vertex:latest']
```

Push the image to the registry

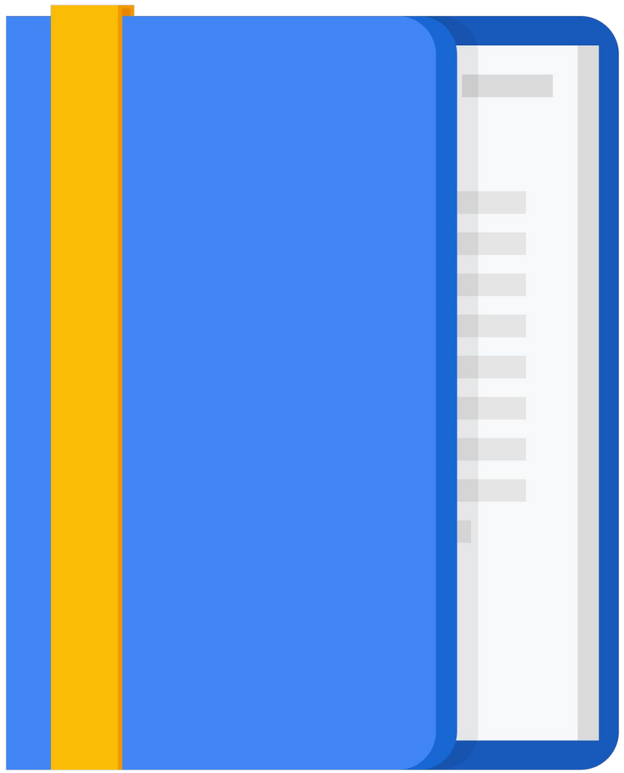
Agenda

Concept Overview

Cloud Build Builders

Cloud Build Configuration

Cloud Build Triggers



Manually executing a Cloud Build

1. Clone the Github repo with your cloudbuild.yaml and ML code on a build node.
    - The build node can be any node with the gcloud sdk properly authenticated:
        - JupyterLab VM
        - Cloud Shell
          (if the containers to build are small)
        - Your laptop
        - A dedicated build VM on Cloud Compute
2. Run gcloud builds submit on the cloudbuild.yaml with the proper substitutions.

Manually executing a Cloud Build

```
[3]: SUBSTITUTIONS= f'_REGION={REGION}'
     SUBSTITUTIONS
```

```
[3]: '_REGION=us-central1'
```

```
[ ]: !gcloud builds submit . --config cloudbuild_vertex.yaml --substitutions {SUBSTITUTIONS}
```

CI/CD: Automated Cloud Build triggers

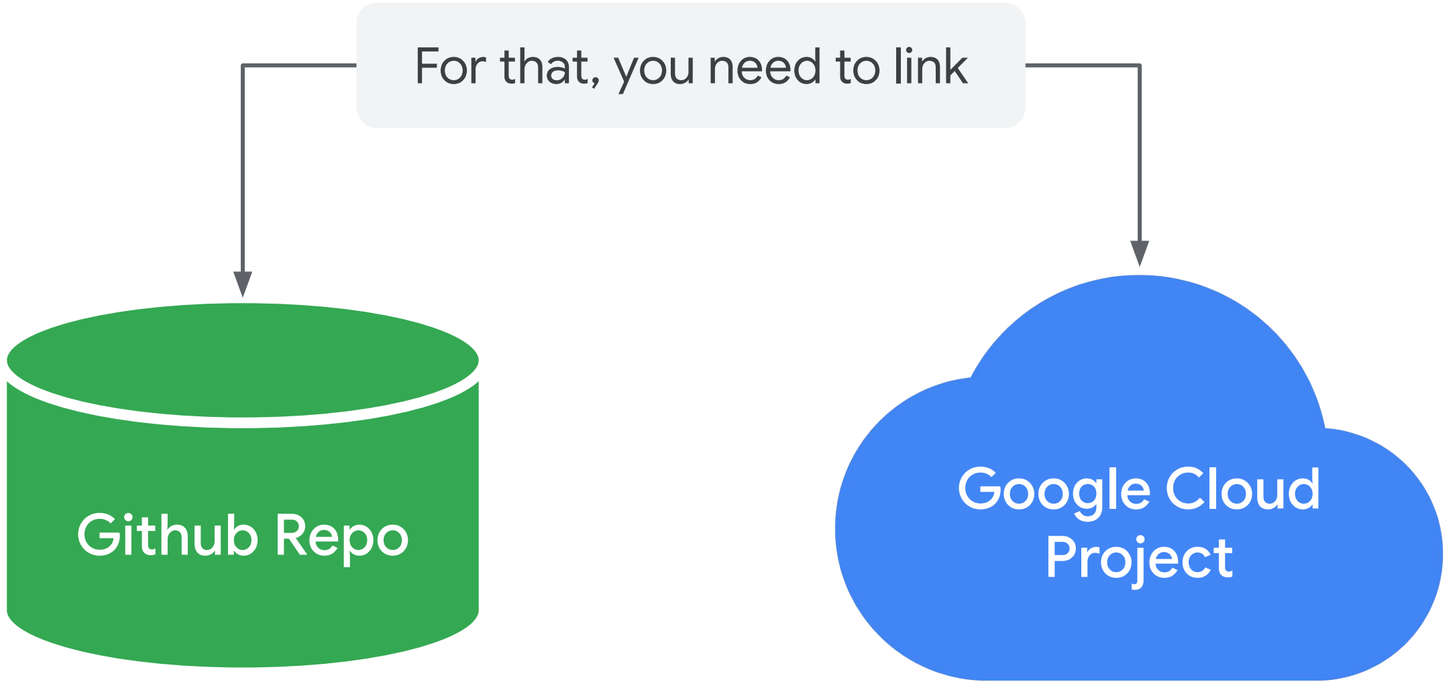
When the ML code changes in Github via:

- Push on a branch
- New tag
- Pull request
- etc.

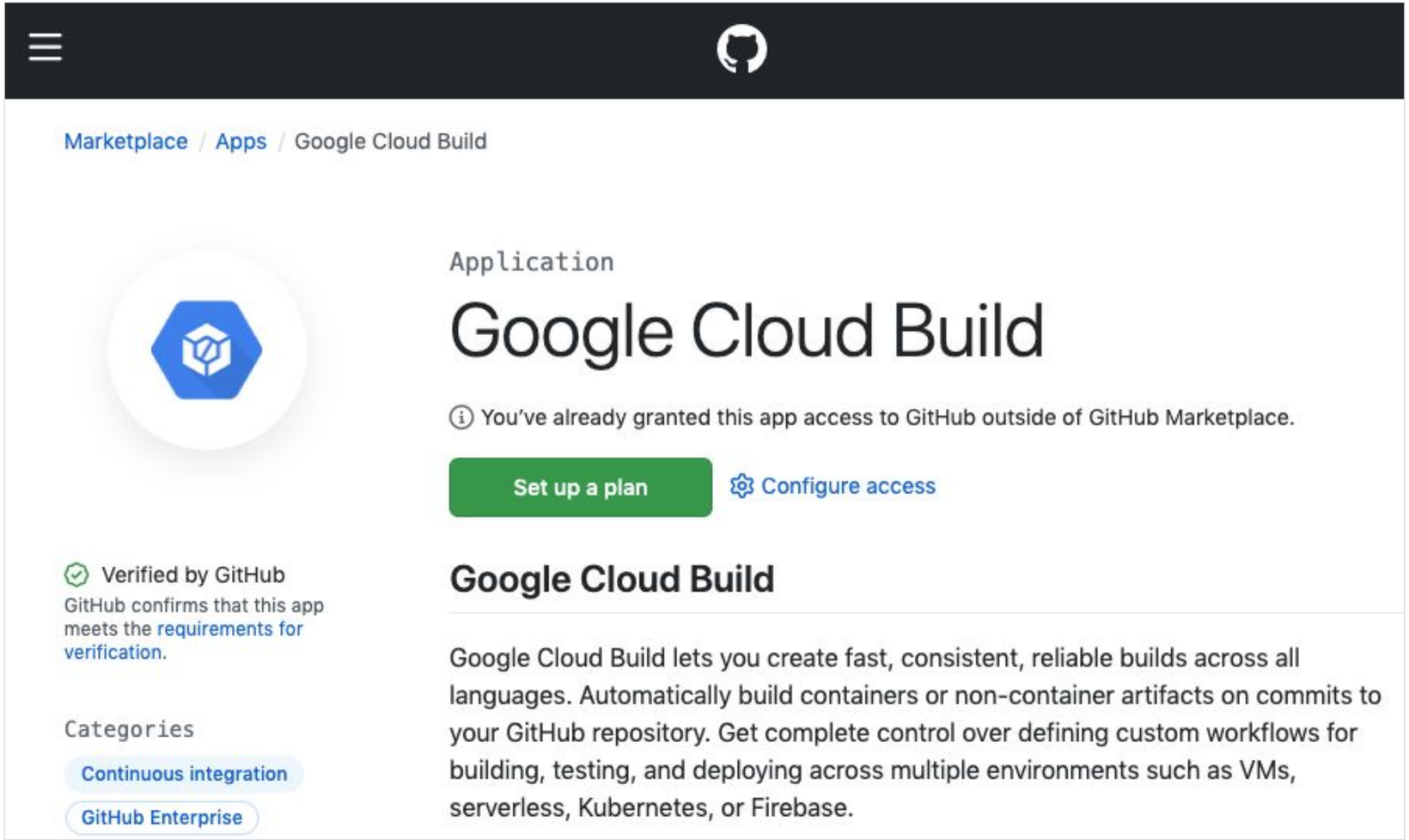
...you want a Cloud Build to be triggered automatically, so that the training artifacts are updated from the new code:

- Training containers
- Kubeflow pipelines
- etc.

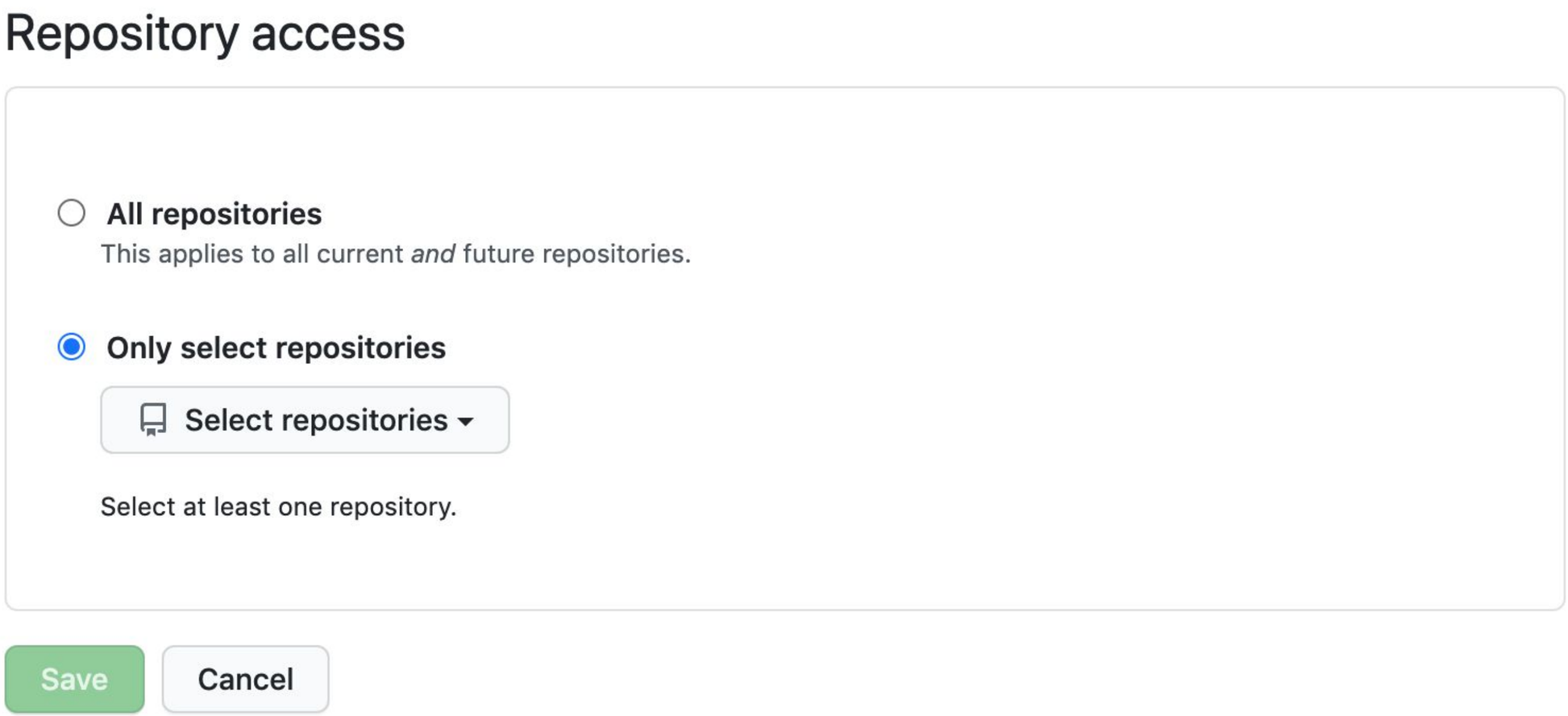
CI/CD: Automated Cloud Build triggers



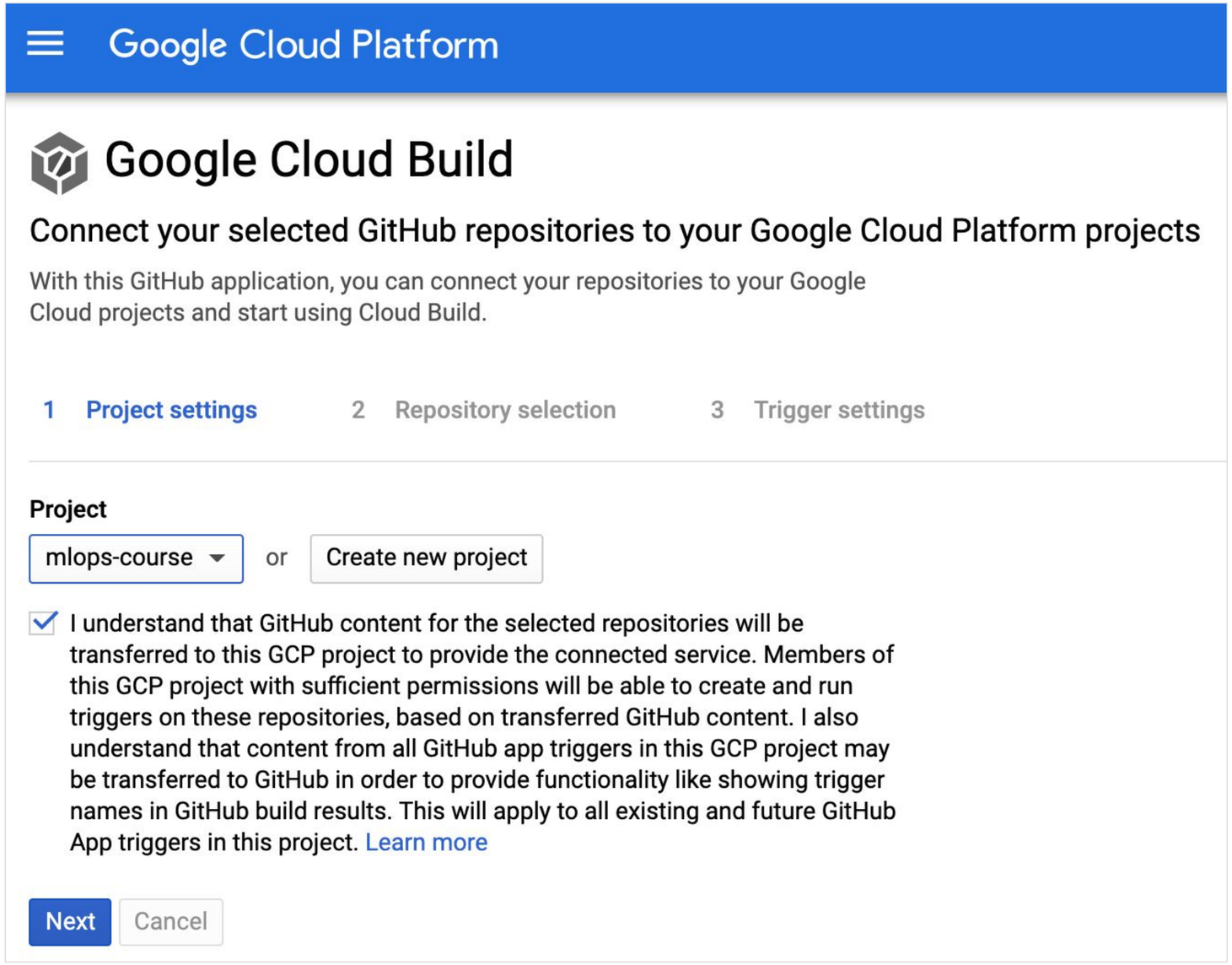
Set up your github repo to work with Cloud Build



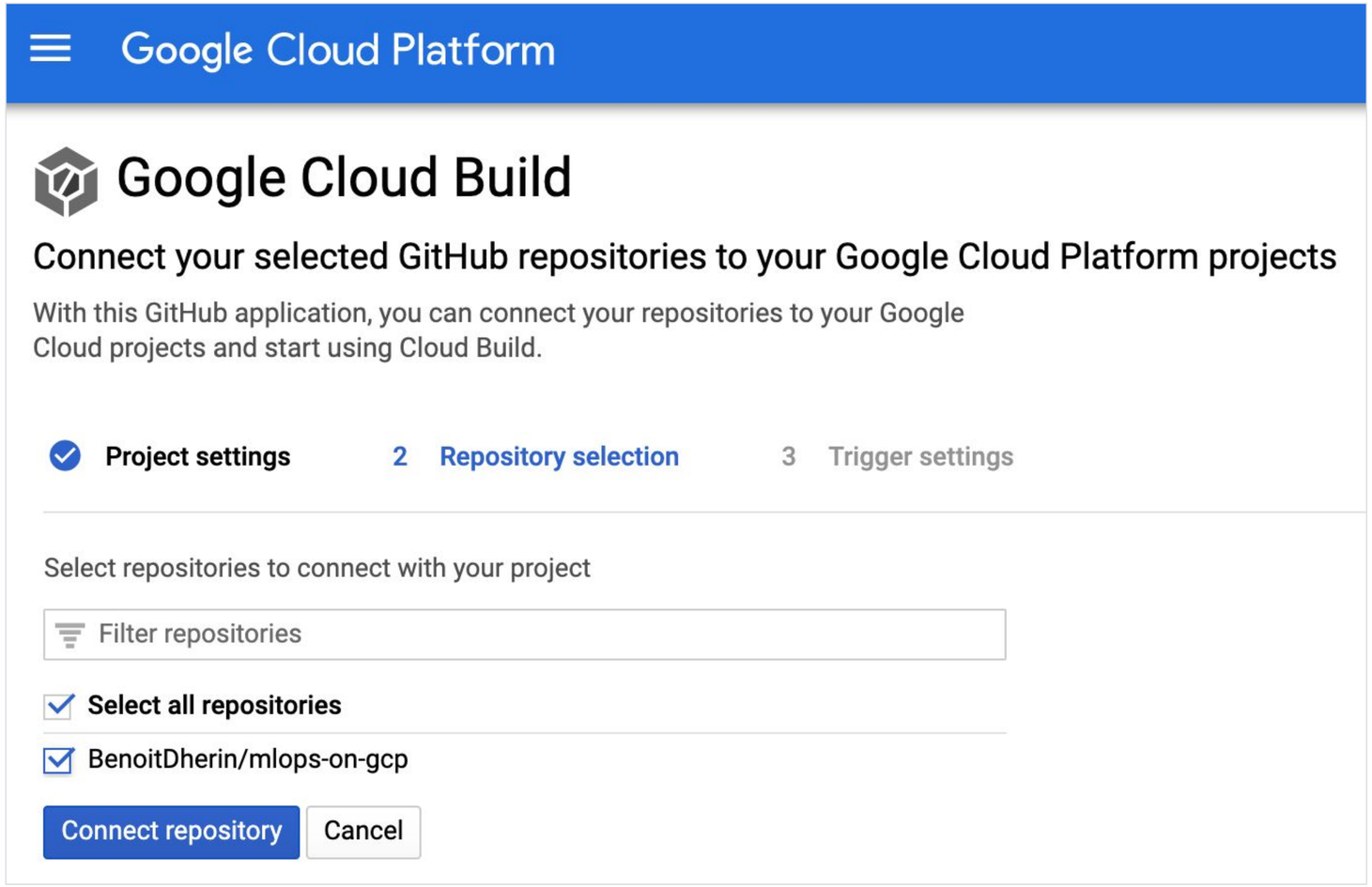
Allow your repo to be accessed by Cloud Build



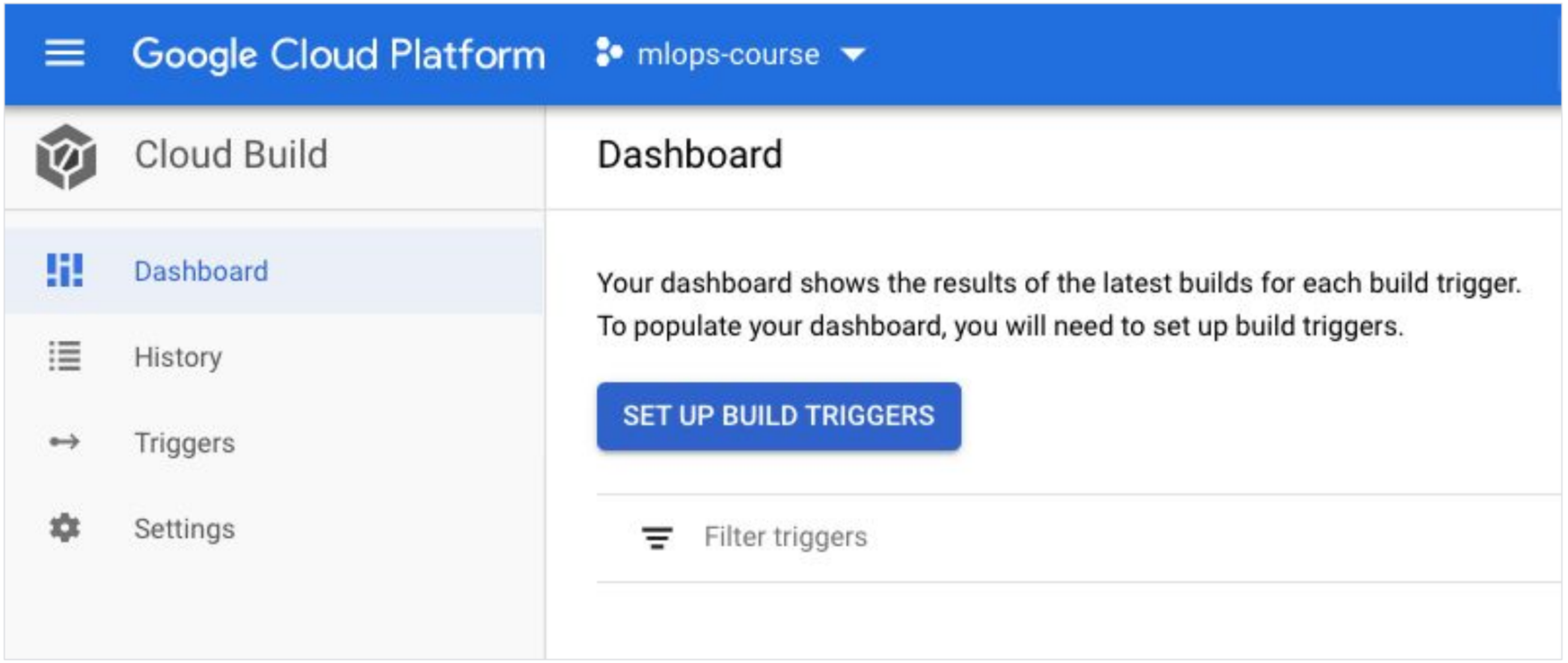
Add your repo to Cloud Build



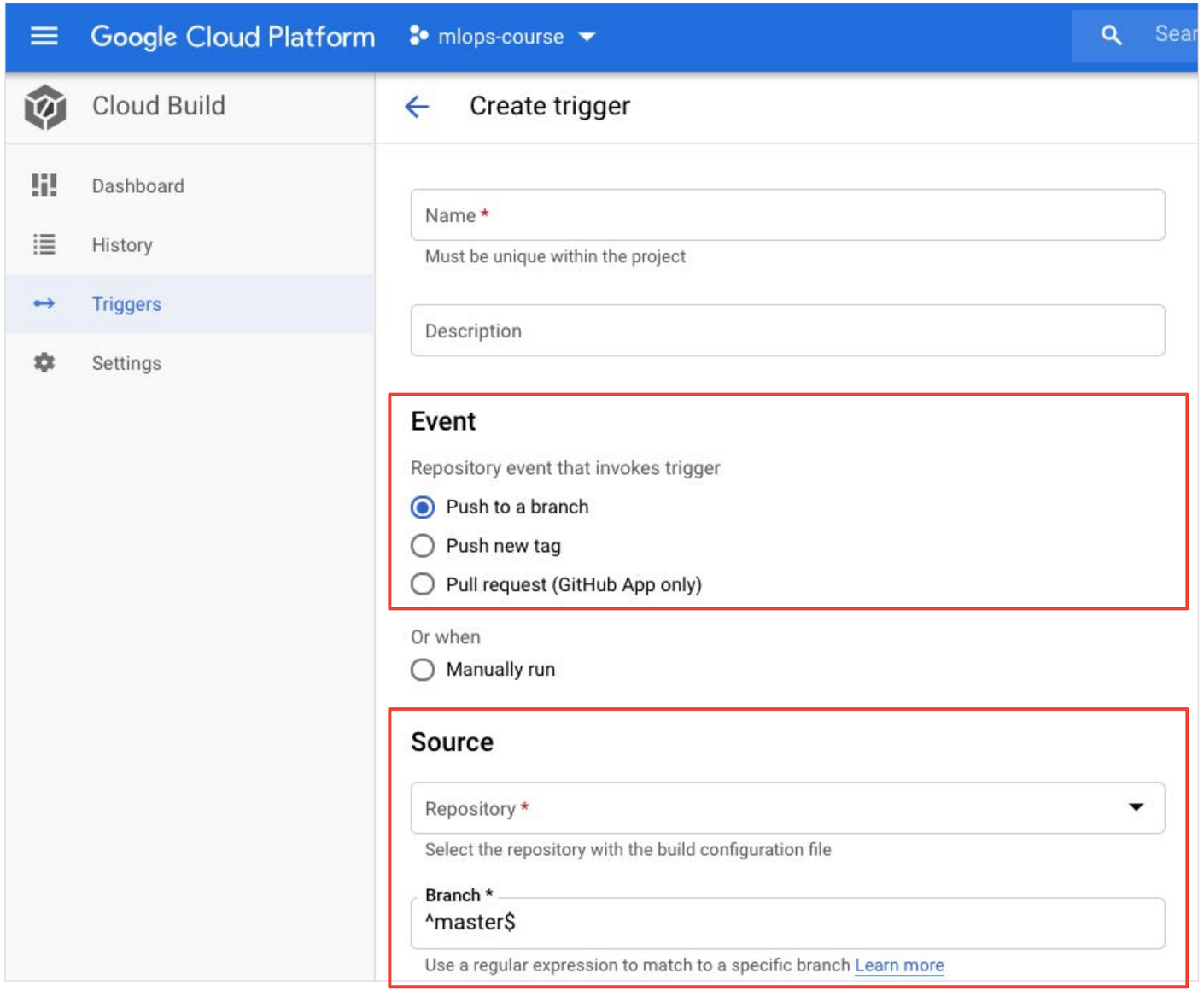
Add your repo to Cloud Build



Set up triggers



Cloud triggers

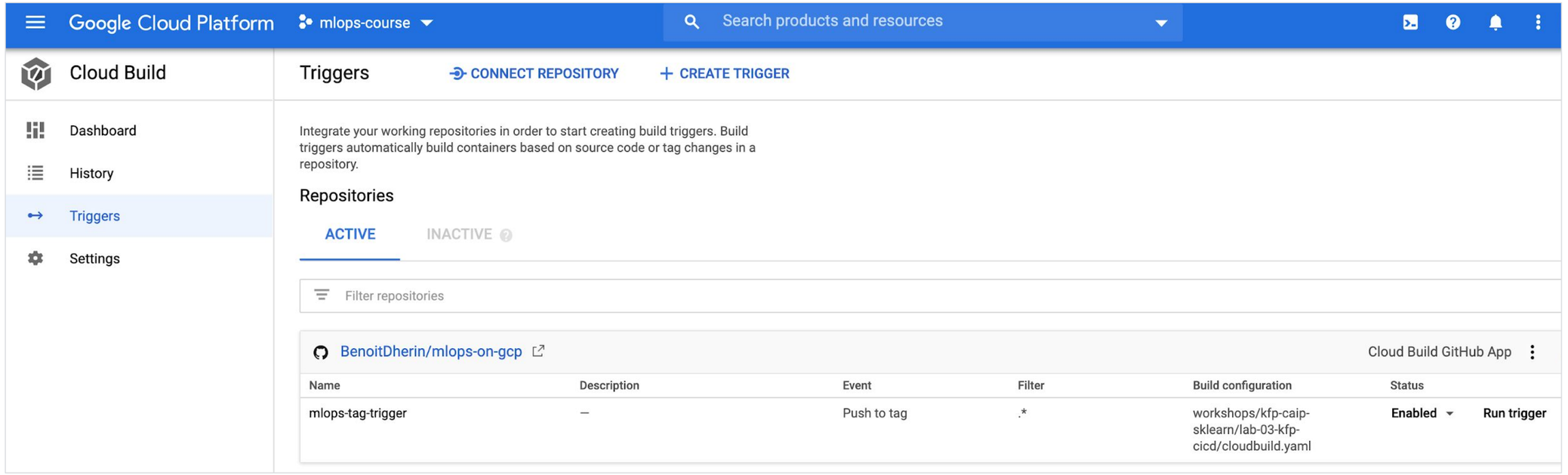


Specify the location of the cloudbuild.yaml file

Field	Value
Name	[YOUR TRIGGER NAME]
Description	[YOUR TRIGGER DESCRIPTION]
Event	Tag
Source	[YOUR FORK]
Tag (regex)	.*
Build Configuration	Cloud Build configuration file (yaml or json)
Cloud Build configuration file location	./immersion/kubeflow_pipelines/cicd/solutions/cloudbuild_vertex.yaml

Set up the substitution variable values

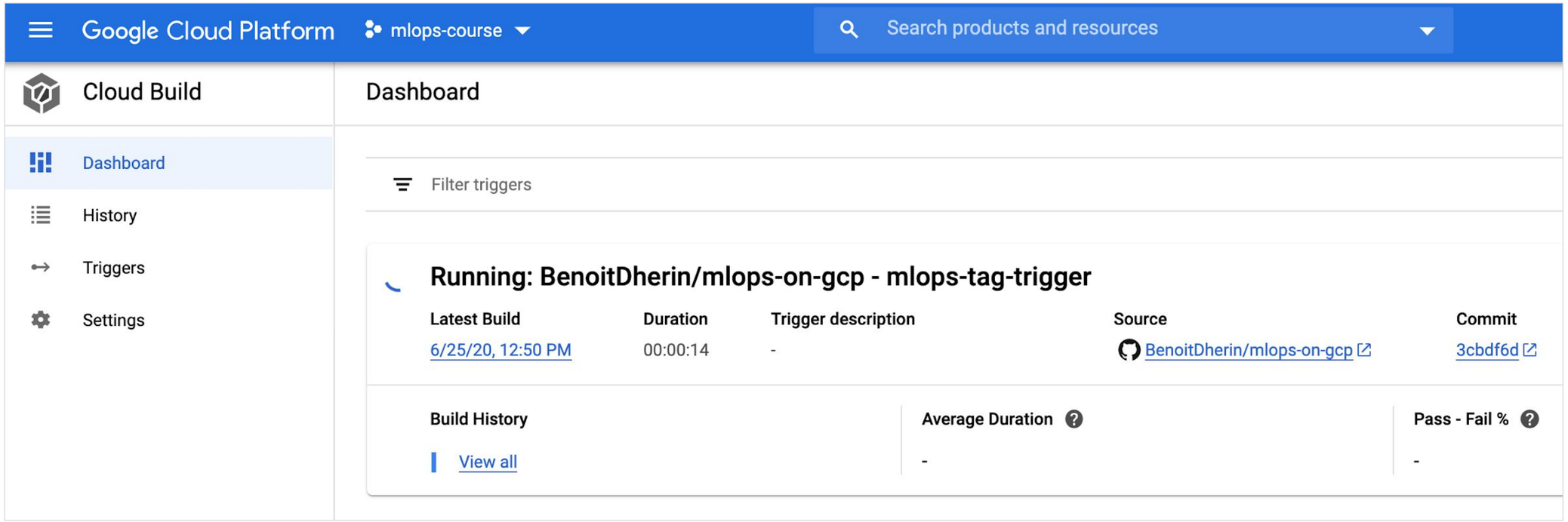
A Cloud Build trigger is now listed



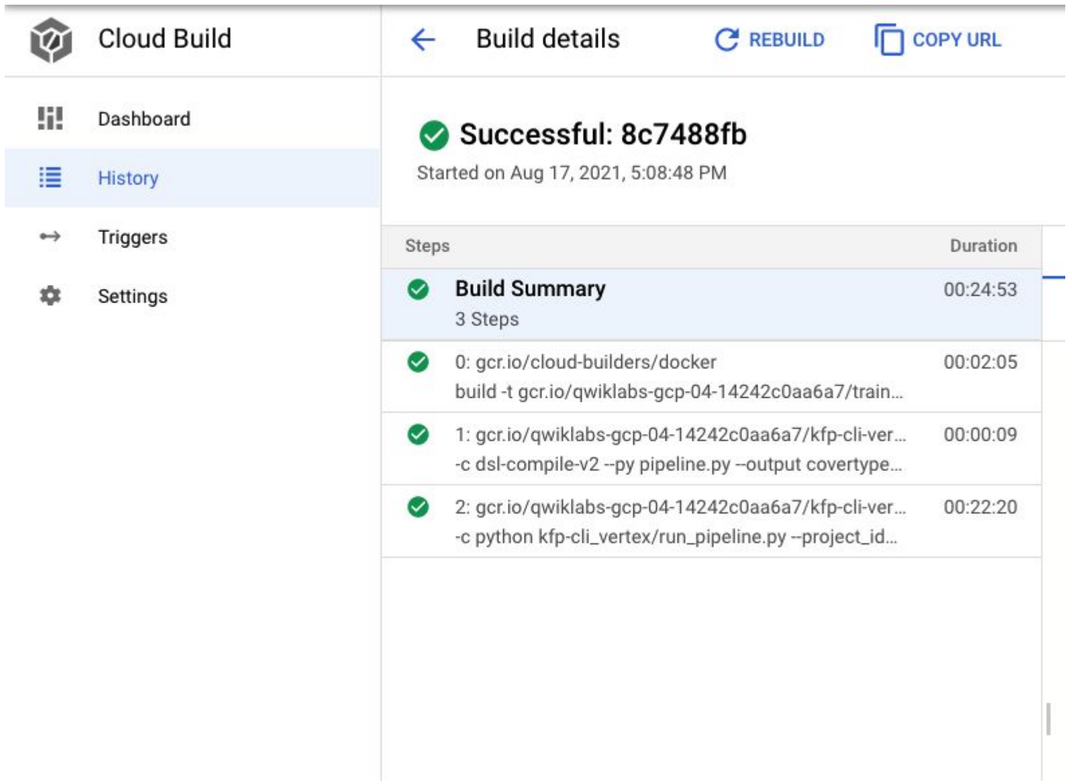
Pushing a new tag will now trigger a Cloud Build

```
git tag [TAG NAME]
git push origin --tags
```

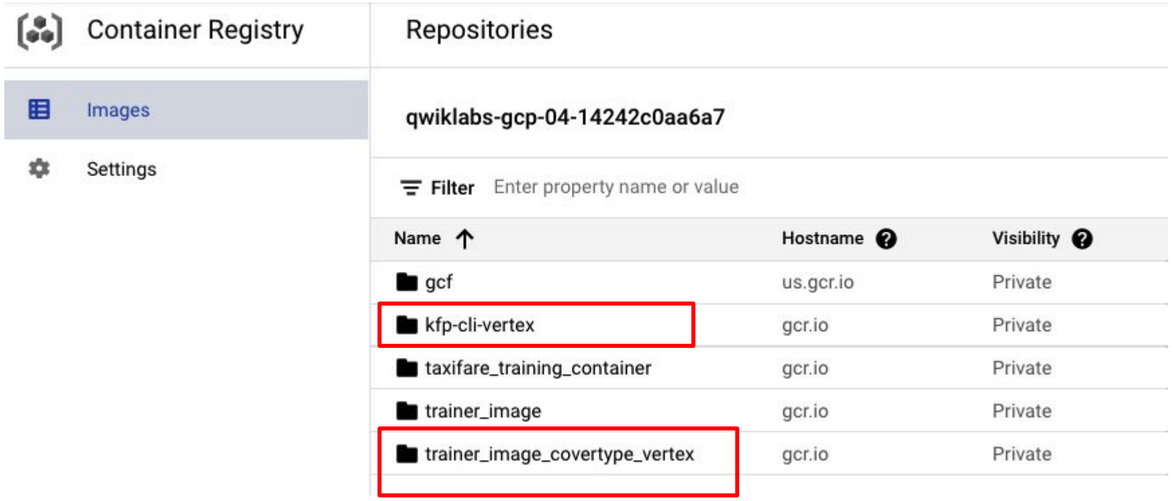
The build can be monitored



The build can be monitored



After the build, new artifacts are available



Lab

CI/CD for a KFP Pipeline

In this lab you will walk through authoring a Cloud Build CI/CD workflow that automatically builds and deploys a KFP pipeline.

notebooks/kubeflow_pipelines/cicd/labs/kfp_cicd_vertex.ipynb

cloud.google.com