**CI/CD FOR FLUTTER APP USING GITHUB ACTIONS**

**What is CI/CD?**

**CI/CD** stands for Continuous Integration/Continuous Delivery (or Continuous Deployment). It's a set of practices and methodologies used in software development to automate the process of integrating code changes into a shared repository (Continuous Integration) and then automatically deploying those changes into a production environment (Continuous Delivery/Deployment).

**CI/CD Terms:**

* **Continuous Integration (CI):** Developers frequently merge their code changes into a shared repository, where automated builds and tests are triggered. This ensures that integration issues are identified and resolved early in the development cycle.
* **Continuous Delivery (CD):** Once code changes pass the automated tests in the CI pipeline, they are automatically prepared for release to production. The aim is to have code always in a deployable state, allowing teams to release changes to customers quickly and safely.
* **Continuous Deployment:** This takes CD a step further by automatically deploying code changes to production environments after passing through the CI pipeline. This approach is suitable for teams looking to achieve rapid and frequent releases with minimal manual intervention.

CI/CD practices help streamline the software development process, improve code quality, reduce manual errors, and accelerate the delivery of new features and updates to end-users.

**Benefits of using CI/CD:**

* **Rapid Feedback Loop:** CI/CD pipelines provide rapid feedback on code changes. As a Flutter developer, you can quickly identify issues, bugs, or regressions, allowing you to address them promptly before they escalate into larger problems.
* **Cross-Platform Testing:** Flutter's strength lies in its cross-platform capabilities. CI/CD enables automated testing across various platforms, devices, and configurations, ensuring consistent functionality and appearance across all supported environments without the need for specific hardware or device.
* **Faster Time-to-Market:** By automating build, test, and deployment processes, you can significantly reduce the time it takes to get new features and updates into the hands of users. This agility can give you a competitive edge in the market.
* **Enhanced Collaboration:** CI/CD fosters collaboration among team members. As a Flutter developer, you can collaborate seamlessly with designers, QA engineers, and other stakeholders, knowing that the automated processes maintain code quality and consistency.
* **Versioning and Traceability:** CI/CD integrates with version control systems, providing clear traceability between code changes and their corresponding builds and deployments. This helps in understanding the evolution of the app over time.

**What are GitHub Actions?**

GitHub Actions is a feature provided by GitHub that allows you to automate workflows directly within your GitHub repository. It enables you to build, test, and deploy your code right from your repository, eliminating the need for external continuous integration and deployment tools.

**Key features of GitHub Actions include:**

* **Workflow Automation:** You can define custom workflows using YAML syntax directly within your repository. Workflows consist of one or more jobs, which can run sequentially or in parallel.
* **Event-Driven Triggers:** Workflows can be triggered by various events, such as pushes to a repository, pull requests, issue creation/closure, and more. This allows you to automate actions based on specific activities in your repository.
* **Rich Ecosystem of Actions:** GitHub Actions provides a marketplace of pre-built actions that you can use within your workflows. These actions cover a wide range of tasks, from building and testing code to deploying applications and notifying team members.
* **Integration with GitHub:** Since GitHub Actions is tightly integrated with GitHub, you can access repository secrets, pull request information, and other GitHub-specific data directly within your workflows.
* **Scalability and Customization:** GitHub Actions scales with your needs, allowing you to run workflows on various virtual environments, including Linux, macOS, and Windows. You can also customize workflows to fit your specific requirements.

Overall, GitHub Actions simplifies the process of automating software development workflows, making it easier for teams to build, test, and deploy their code efficiently.

**Demo:**

**Step-1:**

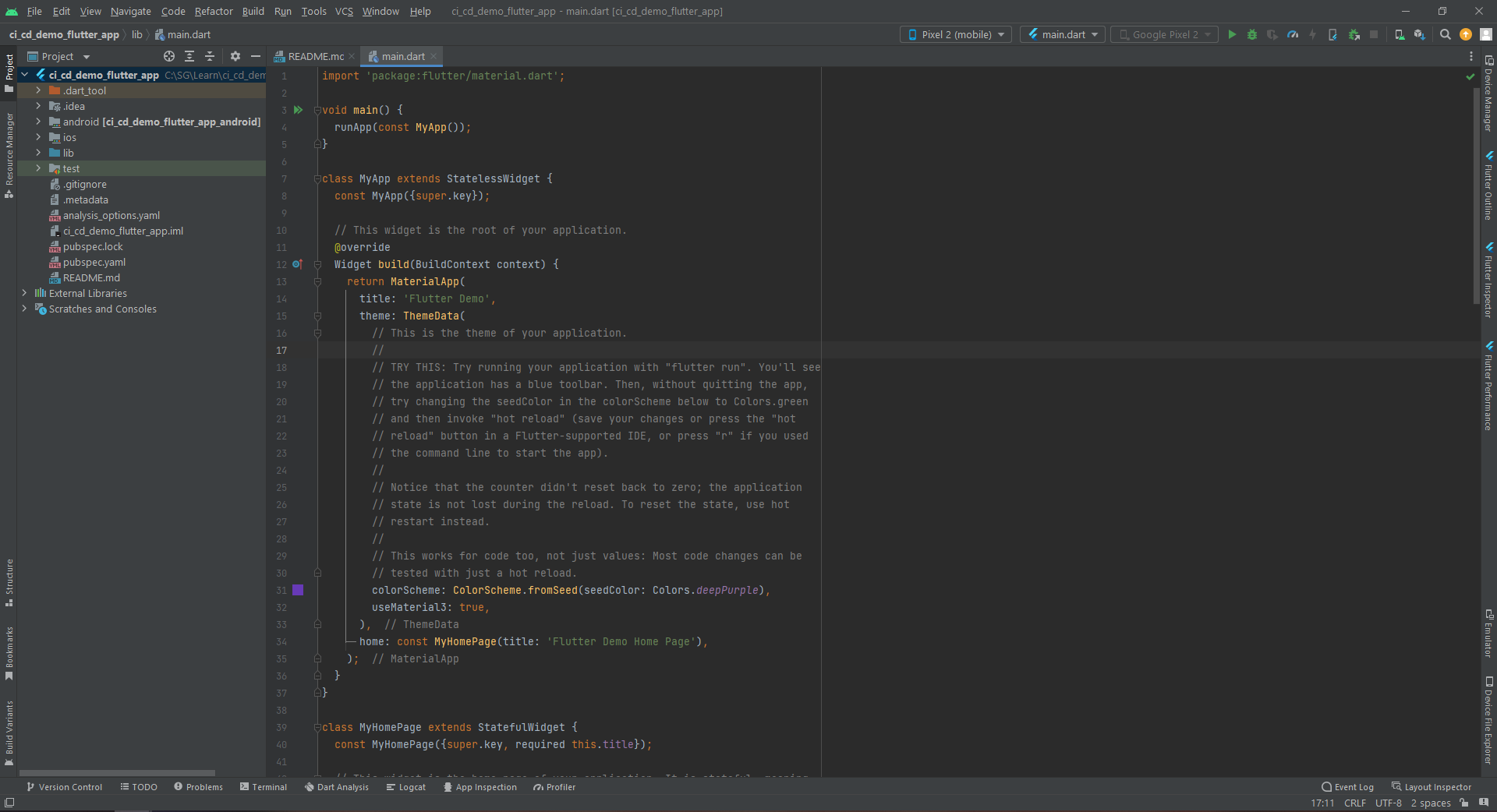
* We need firstly to create a GitHub Repo. for our project. LET'S CALL IT **CI\_CD\_DEMO\_FLUTTER\_APP**.

A screenshot of a computer

Description automatically generated

**Step-2:**

* Then, Create Flutter Demo App LET’S GIVE A NAME **“ci\_cd\_demo\_flutter\_demo\_app”**.



**Step-3:**

* Connect The App with Remote GitHub Repository and Push the Initial Commit to the Repository.

**C:\SG\Learn\ci\_cd\_demo\_flutter\_app>** git init

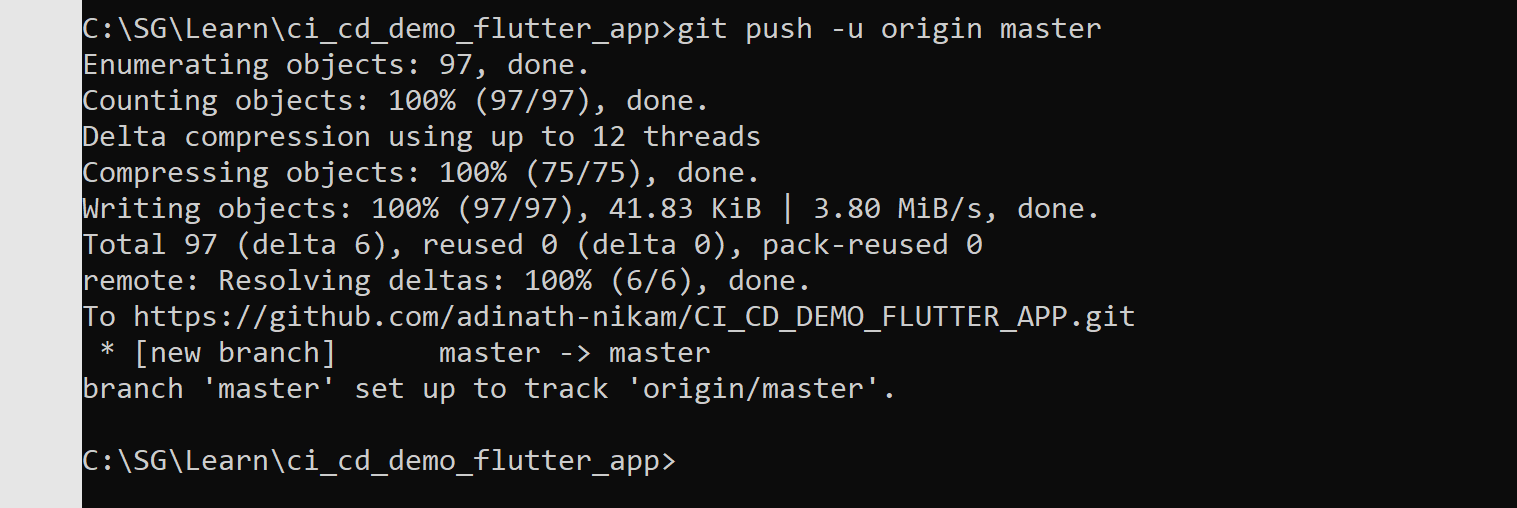
**C:\SG\Learn\ci\_cd\_demo\_flutter\_app>** git add .

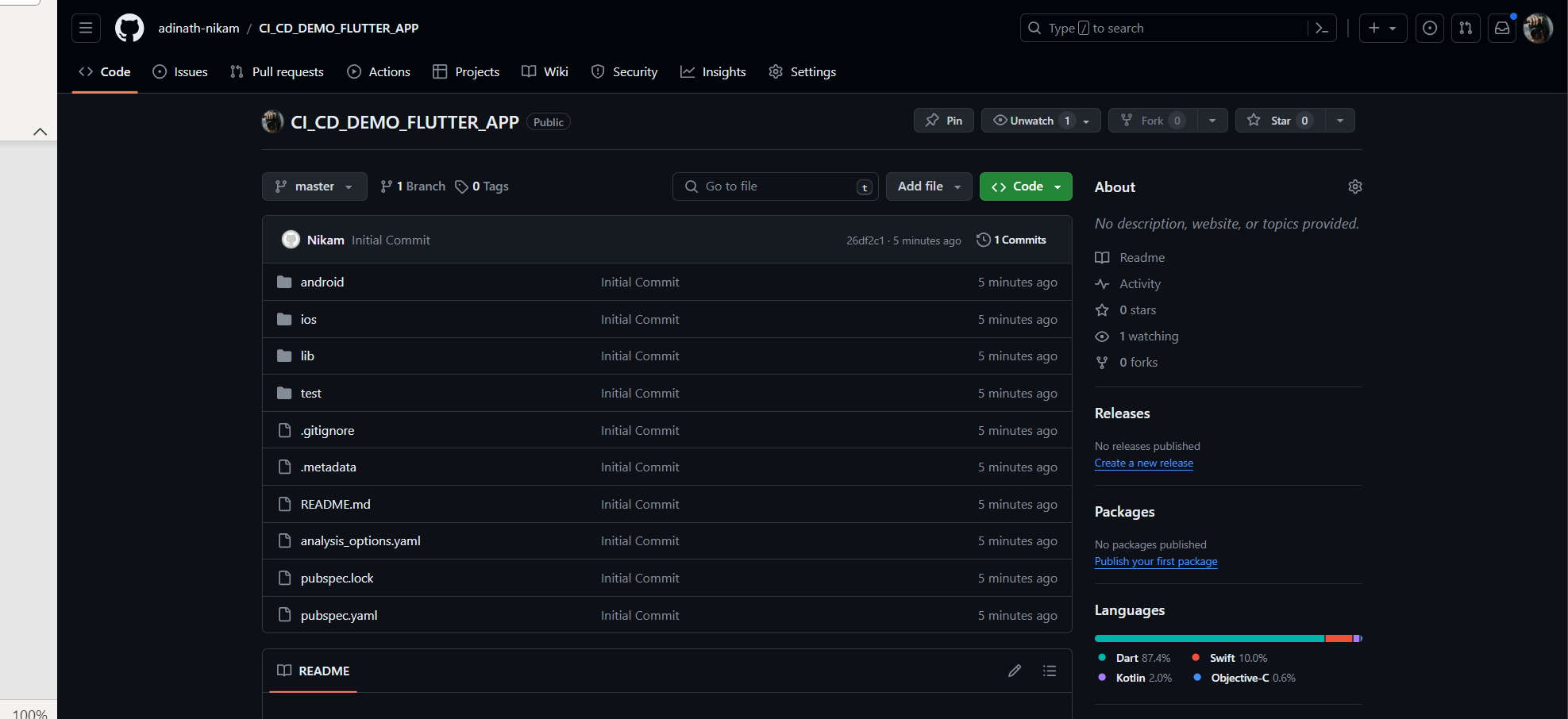
**C:\SG\Learn\ci\_cd\_demo\_flutter\_app>** git commit -m "init project"

**C:\SG\Learn\ci\_cd\_demo\_flutter\_app>** git remote add origin

<https://github.com/adinath-nikam/CI_CD_DEMO_FLUTTER_APP.git>

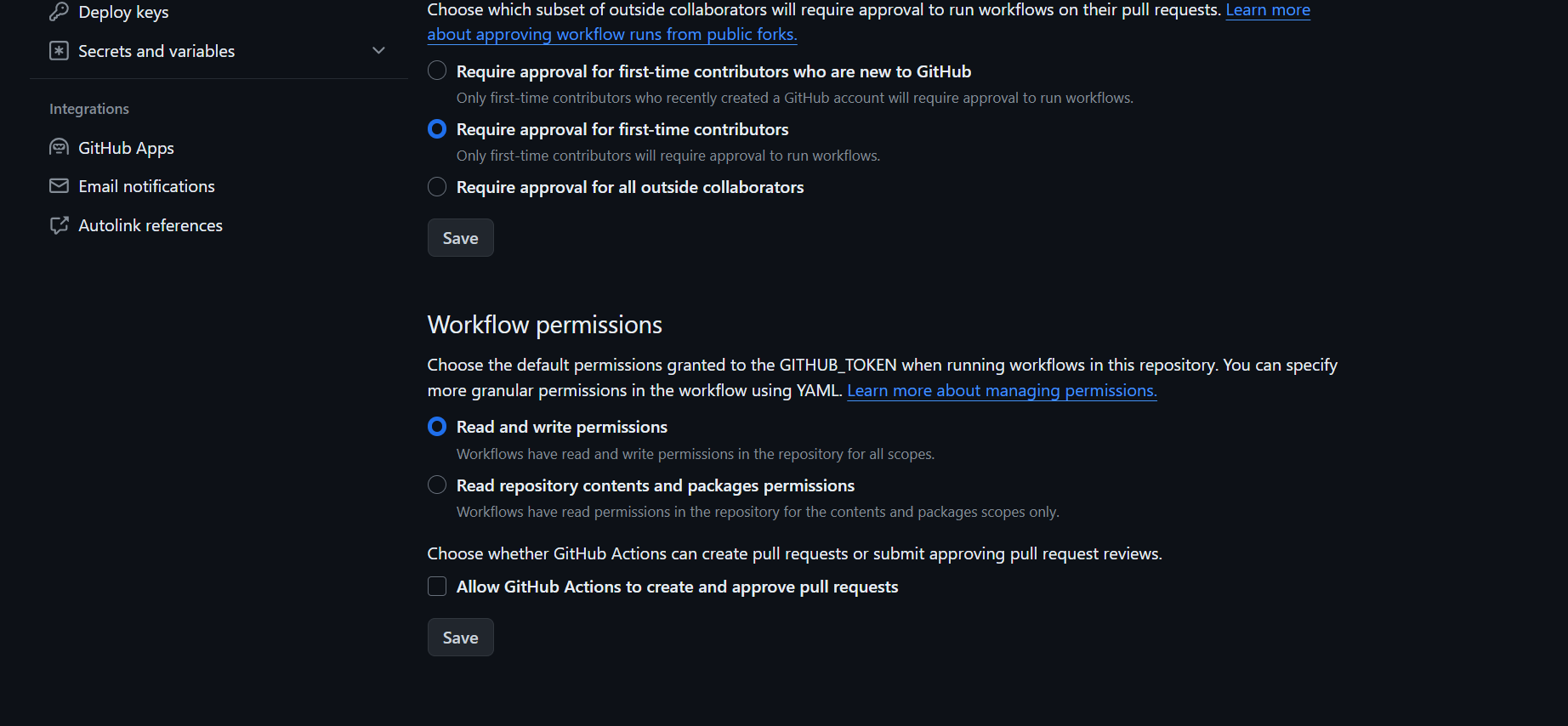
**C:\SG\Learn\ci\_cd\_demo\_flutter\_app>** git push -u origin main

****

****

**Step-4:**

* Then Grant Read/Write Permissions for Workflows on your Repository to Create and Upload Releases.
  + Go to repository "Settings".
  + After that it will show you a left pane where you will find "Actions"
  + Expand "Actions" tab.
  + Click on "General" under options tab.
  + Now on new page scroll down and you will fine "Workflow Permissions”.
  + Select "Read and Write" under "Workflow Permissions".

****

**Step-4:**

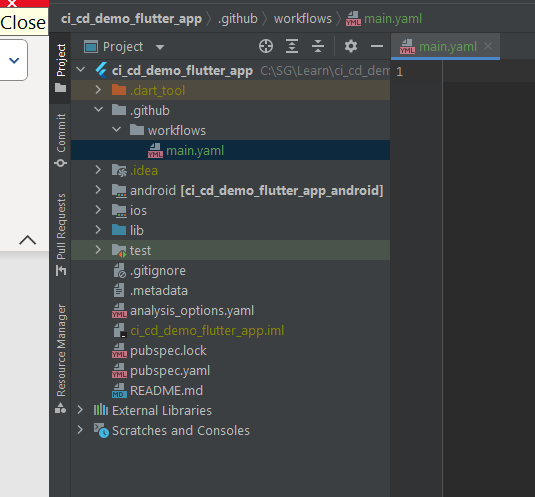
* As we can see above the Repo name, there is a tap called Actions (This is what we need)

A screenshot of a computer

Description automatically generated

Now Let's back to our Android Studio and start working in it.

1. In the Root folder of your project, create a new Directory called "**.github** ".
2. under the newly created directory, create another new directory and name it " **workflows** ".
3. finally, inside workflows directory create a new File and let's name it " **main.yml** ".



**Step-5:**

* Let's start to write the workflow we need to happen.

.**github/workflows/main.yaml**

**# Code Starts from here**

name: "CI/CD Demo Build & Release Flutter App"

on:

push:

branches:

- master

jobs:

# Job 1

build:

name: Build Flutter App

runs-on: ubuntu-latest

if: github.ref == 'refs/heads/master'

steps:

#1 Checkout Repository

- name: Checkout Repository

uses: actions/checkout@v3

#2 Setup Java

- name: Set-up Java

uses: actions/setup-java@v3.12.0

with:

distribution: 'oracle'

java-version: '17'

#3 Setup Flutter

- name: Set-up Flutter

uses: subosito/flutter-action@v2

with:

flutter-version: '3.13.0'

channel: 'stable'

#4 Install Dependencies

- name: Install Dependencies

run: flutter pub get

#6 Run Test

- name: Test Flutter App

run: flutter test

#7 Build APK

- name: Build APK

run: flutter build apk --release

#8 Build AAB

- name: Build appBundle

run: flutter build appbundle

#9 Create Release in GitHub

- name: Create Release

id: create\_release

uses: actions/create-release@v1

env:

GITHUB\_TOKEN: ${{ secrets.GITHUB\_TOKEN }}

with:

tag\_name: ${{ github.run\_id }}

release\_name: Release ${{ github.run\_id }}

draft: false

prerelease: false

#10 Upload Release to GitHub Release

- name: Upload Releases

id: upload\_release

uses: actions/upload-release-asset@v1

env:

GITHUB\_TOKEN: ${{ secrets.GITHUB\_TOKEN }}

with:

upload\_url: ${{ steps.create\_release.outputs.upload\_url }}

asset\_path: ./build/app/outputs/flutter-apk/app-release.apk

asset\_name: app-release.apk

asset\_content\_type: application/zip

**# Code Ends Here**

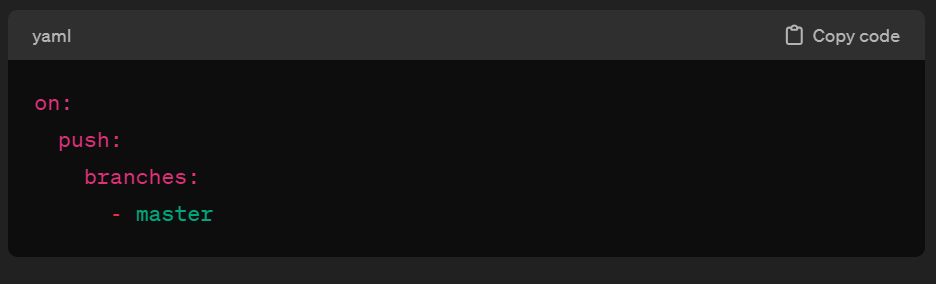
****

**Breakdown of Workflow File:**

This GitHub Actions workflow is designed to automate the build and release process of a Flutter application. Let's break down each part:

1. **Trigger:**

This workflow is triggered by the push event on the master branch. This means that whenever a new commit is pushed to the master branch of the repository, this workflow will be executed.



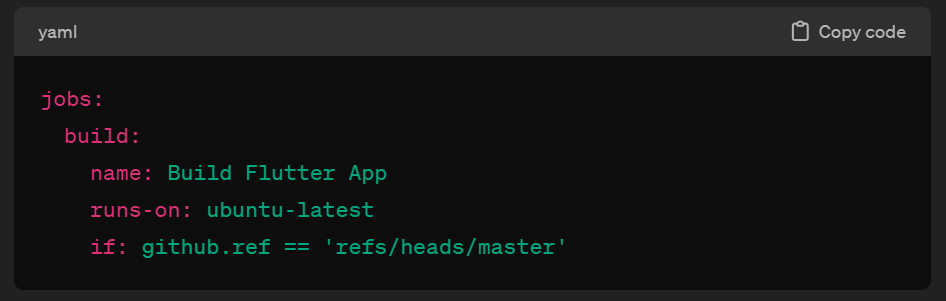
1. **Jobs:**

**Build Job:**

**Name:** This job is named "Build Flutter App".

**Runs on:** This job runs on the latest version of Ubuntu.

**Condition:** It only runs if the branch being pushed to is the master branch. This is specified using the if condition.



1. **Steps:**

* **Checkout Repository:**
  + This step checks out the repository code.
  + It uses the actions/checkout action to clone the repository into the runner machine.

A black and green screen with text

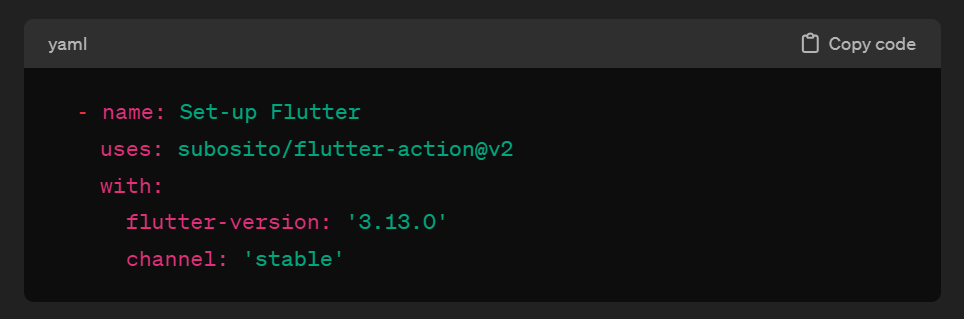
Description automatically generated

* **Set-up Java:**
  + This step sets up the Java environment.
  + It uses the actions/setup-java action to set up Java with Oracle distribution and version 17.

A screenshot of a computer

Description automatically generated

* **Set-up Flutter:**
  + This step sets up the Flutter environment.
  + It uses the subosito/flutter-action action to set up Flutter with version 3.13.0 from the stable channel.



* **Install Dependencies:**
  + This step installs dependencies for the Flutter app.
  + It runs the command flutter pub get to fetch and install the dependencies listed in the pubspec.yaml file.

A screen shot of a computer

Description automatically generated

* **Test Flutter App:**
  + This step runs tests for the Flutter application.
  + It runs the command flutter test to execute tests and verify the integrity of the code.

A black rectangular object with green text

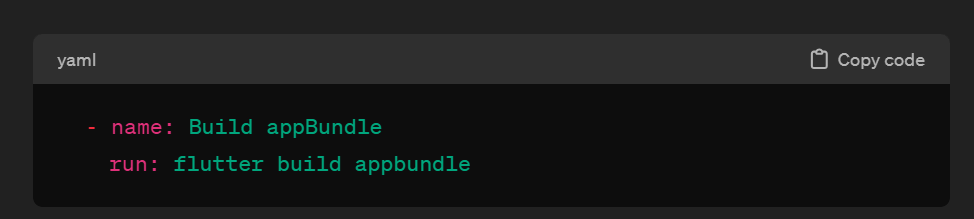
Description automatically generated

* **Build APK:**
  + This step builds the release APK for the Flutter application.
  + It runs the command flutter build apk --release to generate the APK file.

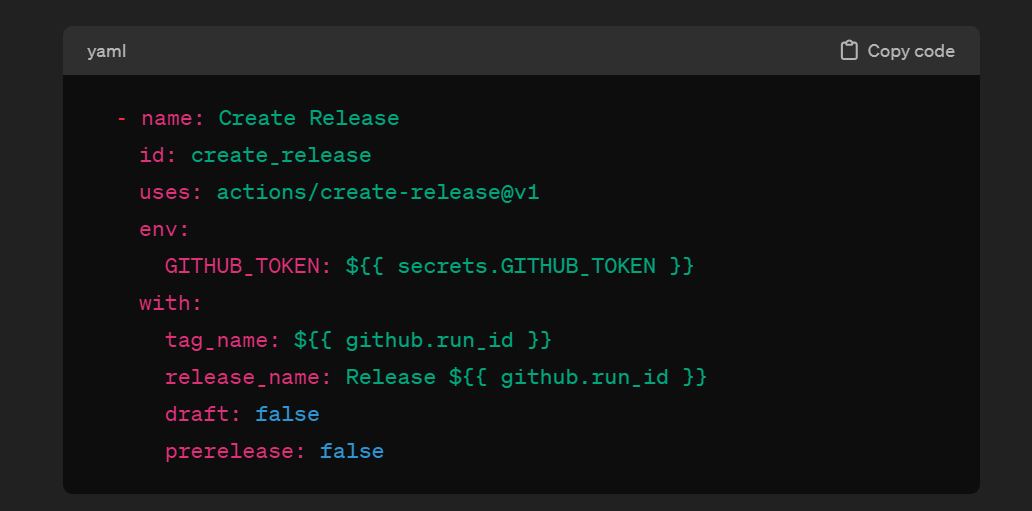
A screen shot of a computer

Description automatically generated

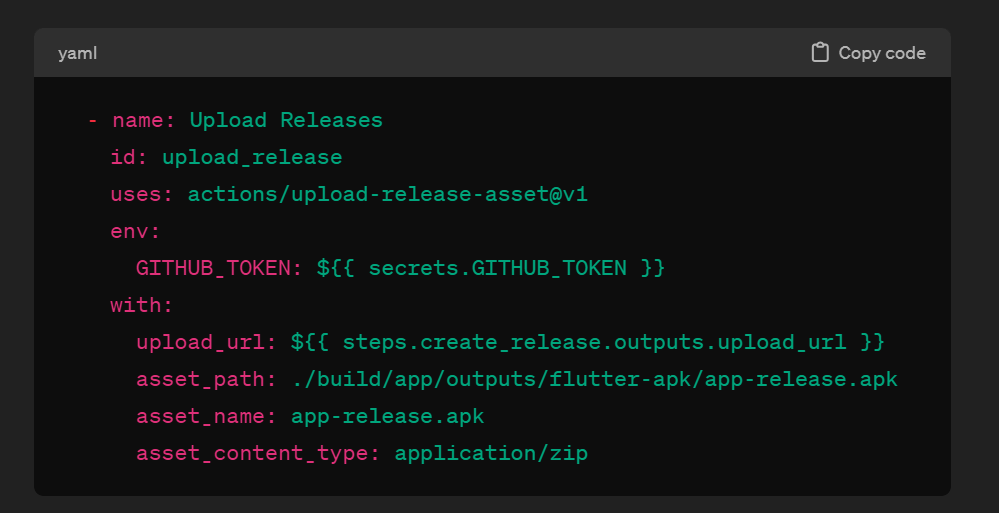
* **Build AAB:**
  + This step builds the Android App Bundle (AAB) for the Flutter application.
  + It runs the command flutter build appbundle to generate the AAB file.



* **Create Release in GitHub:**
  + This step creates a GitHub release for the built artifacts.
  + It uses the actions/create-release action to create a new release.
  + The tag\_name and release\_name is set to the run ID of the GitHub Actions workflow.
  + The release is not marked as a draft or prerelease.



* **Upload Release to GitHub Release:**
  + This step uploads the built APK to the GitHub release assets.
  + It uses the actions/upload-release-asset action to upload the APK file.
  + The upload\_url is obtained from the output of the previous step (create\_release).
  + The asset\_path, asset\_name, and asset\_content\_type are specified.



This workflow automates the process of building, testing, and releasing a Flutter application on GitHub. It ensures that the latest version of the app is built and released whenever changes are pushed to the master branch.

**Step-6:**

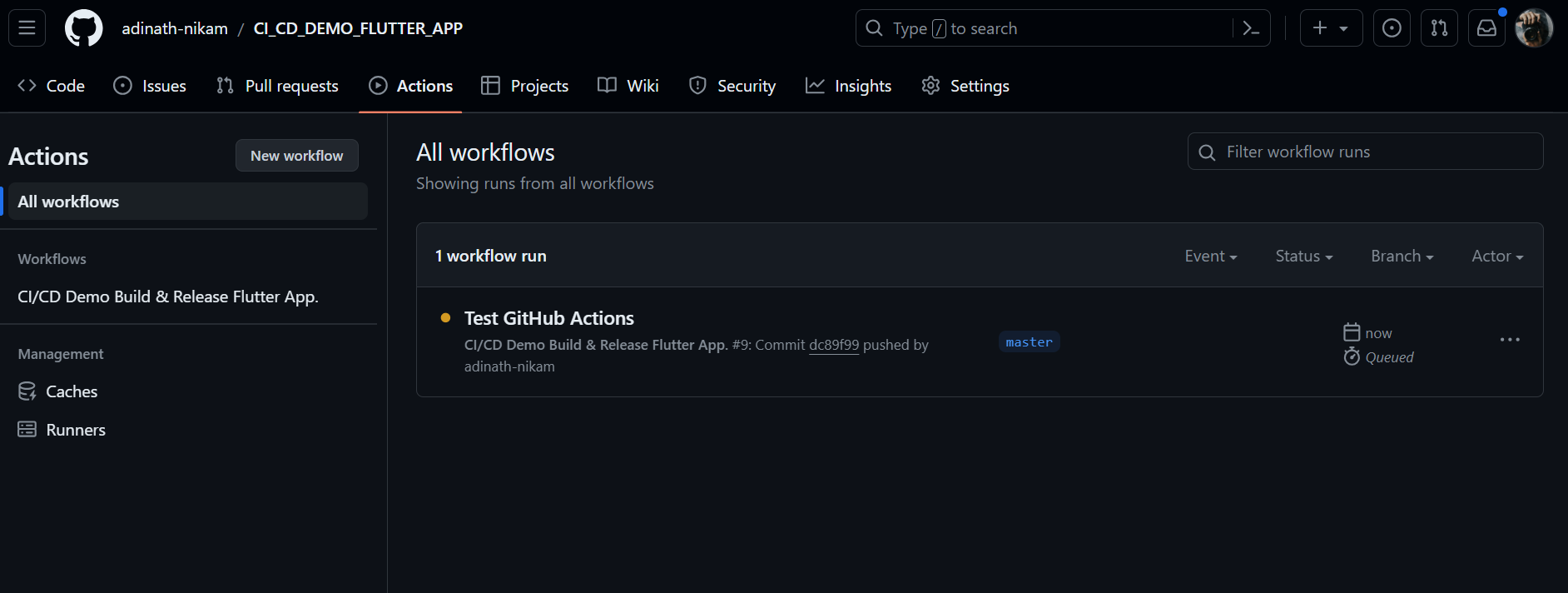
* **Now Test if Workflow Works.**

**C:\SG\Learn\ci\_cd\_demo\_flutter\_app>** get add .

**C:\SG\Learn\ci\_cd\_demo\_flutter\_app>** git commit -m "Test GitHub Actions"

**C:\SG\Learn\ci\_cd\_demo\_flutter\_app>** git push -u origin dev

* Push the changes to the master branch and let's see it works.
* Navigate to Actions tab on your repo and see it.

****

**A screenshot of a computer

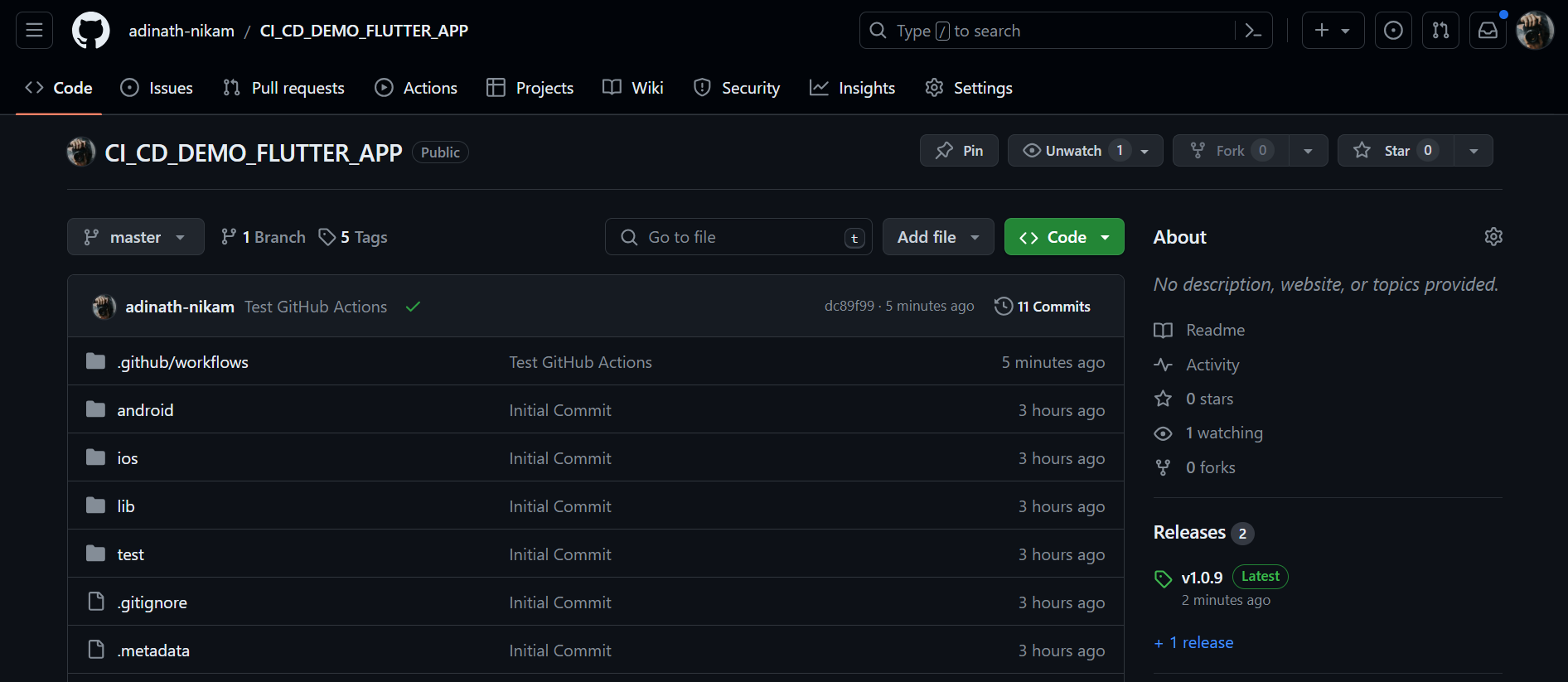
Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

****

**Workflow File with Multiple Jobs Example Demo:**

**name: "CI/CD Demo Build & Release Flutter App"**

**on:**

**push:**

**branches:**

**- master**

**jobs:**

**# Job 1**

**build:**

**name: Build Flutter App**

**runs-on: ubuntu-latest**

**if: github.ref == 'refs/heads/master'**

**steps:**

**#1 Checkout Repository**

**- name: Checkout Repository**

**uses: actions/checkout@v3**

**#2 Setup Java**

**- name: Set-up Java**

**uses: actions/setup-java@v3.12.0**

**with:**

**distribution: 'oracle'**

**java-version: '17'**

**#3 Setup Flutter**

**- name: Set-up Flutter**

**uses: subosito/flutter-action@v2**

**with:**

**flutter-version: '3.13.0'**

**channel: 'stable'**

**#4 Install Dependencies**

**- name: Install Dependencies**

**run: flutter pub get**

**#5 Run Test**

**- name: Test Flutter App**

**run: flutter test**

**#6 Build APK**

**- name: Build APK**

**run: flutter build apk --release**

**#7 Build AAB**

**- name: Build appBundle**

**run: flutter build appbundle**

**#8 Upload APK Artifact**

**- name: Upload APK Artifact**

**uses: actions/upload-artifact@v2**

**with:**

**name: my\_artifacts**

**path: |**

**build/app/outputs/flutter-apk/app-release.apk**

**# Job 2**

**release:**

**needs: build**

**name: Release Flutter App in GitHub Release**

**runs-on: ubuntu-latest**

**if: github.ref == 'refs/heads/master'**

**steps:**

**#1 Download APK Artifact from Job 1**

**- name: Download APK Artifact**

**uses: actions/download-artifact@v2**

**with:**

**name: my\_artifacts**

**#2 Create Release in GitHub**

**- name: Create Release**

**id: create\_release**

**uses: actions/create-release@v1**

**env:**

**GITHUB\_TOKEN: ${{ secrets.GITHUB\_TOKEN }}**

**with:**

**tag\_name: ${{ github.run\_id }}**

**release\_name: Release ${{ github.run\_id }}**

**draft: false**

**prerelease: false**

**#3 Upload Release to GitHub Release**

**- name: Upload APK to GitHub Release**

**uses: actions/upload-release-asset@v1**

**env:**

**GITHUB\_TOKEN: ${{ secrets.GITHUB\_TOKEN }}**

**with:**

**upload\_url: ${{ steps.create\_release.outputs.upload\_url }}**

**asset\_path: /home/runner/work/CI\_CD\_DEMO\_FLUTTER\_APP/CI\_CD\_DEMO\_FLUTTER\_APP/app-release.apk**

**asset\_name: app-release.apk**

**asset\_content\_type: application/zip**