Android CI/CD Implementation with Gitlab and Firebase

Ramdan Nurul · Follow
5 min read · Feb 20, 2022

As we know, the process of building an application often spends a lot of time when developing an android application, especially if the scope of the feature is large enough. Fortunately, **CI/CD** can automate these process and distribute APKs to testers. **Continuous Integration (CI)** is one of the most common development practices used in software development by implementing automation in buildings. **Continuous delivery (CD)** is a step after Continuous Integration, used to deploy or distribute applications.

Android CI/CD with Gitlab and Firebase

GitLab has a simple and free CI/CD system that can automate the process of building, testing, and deploying in our applications. In this article, I will share from my experience how to configure Android CI/CD in a GitLab repository, with the goal after the build is to deploy the APK to the Firebase app distribution without using third party services (with Gitlab Runner).

Signing Config

Prepare the **keystore** of your application and create a file `key.properties` to save the credentials data of the keystore. Don't forget to add this file to `.gitignore`.

```
storePassword=<your_store_password>
keyPassword=<your_key_password>
keyAlias=<your_key_alias>
storeFile=<your_kestore_location_file>
```

Then change the signing config in the **gradle app** by pointing to the `key.properties` file's just created.

```
def keystoreProperties = new Properties()
def keystorePropertiesFile = rootProject.file('key.properties')
if (keystorePropertiesFile.exists()) {
    keystoreProperties.load(new
FileInputStream(keystorePropertiesFile))
}
```

Add the following script in the `android` section :

```
signingConfigs {
  file(rootProject.file('key.properties')).with { propFile ->
       if (propFile.canRead()) {
            config {
                keyAlias keystoreProperties['keyAlias']
                keyPassword keystoreProperties['keyPassword']
                storeFile file(keystoreProperties['storeFile'])
                storePassword keystoreProperties['storePassword']
            }
        }
        else {
            print('not signed')
        }
```

```
    }
}
```

Then, add the following script to the `defaultConfig` section :

```
file(rootProject.file('key.properties')).with { propFile ->
  if (propFile.canRead()) {
        signingConfig signingConfigs.config
    }
}
```

For more details how to signing config, you can visit: https://developer.android.com/studio/publish/app-signing.

Store as Variable

To protect **sensitive keys** and **credential data**, we can store them as variables in Gitlab CI/CD settings. For files such as keystore, `key.properties` dan `google-services.json` , we can convert it to base64 encoding first before storing into a variable (Settings > CI/CD > Variables).

Type	↑ Key	Value	Protected	Masked	Environments	
Variable	FIREBASE_APP_ID	**********	✓	✕	All (default)	
Variable	FIREBASE_CI_TOKEN	**********	✓	✕	All (default)	
Variable	GOOGLE_SERVICE_JSON	**********	✓	✕	All (default)	
Variable	KEY_STORE_PROP	**********	✓	✕	All (default)	
Variable	KEYSTORE_FILE	**********	✓	✕	All (default)	

Environment Variables

For more details how to get firebase ci token, you can visit: https://firebase.google.com/docs/cli.

Gitlab CI/CD Pipeline

GitLab CI/CD pipelines are configured using a **YAML** file `.gitlab-ci.yml` in the root of the project folder. The configuration file consists of several tags :

- **Image**, to define a Docker Image that will be used for execution in CI/CD Jobs.
- **Variable**, is to set a value to a variable used during script execution.
- **Before Script**, used to initialize setup or define the command that should be run before main script execution.
- **Cache**, to specify a list of files and directories to cache between jobs.
- **Stages**, is to defines the order of job stage for pipeline.
- **Stage**, used to specify the stages of work in progress, the pipeline will execute all stages with the same name first.
- **Only**, is to applied only when there is a push on a specific branch.
- **Script**, is to execute the main script where we build or sign the application, with the credentials previously stored in a variable.
- **Artifacts**, used to store or upload the file (APK) and passed to the next job and are also available for download from repo.

We can add CI/CD configuration to any branch we want. In my case, I create a new branch with the named `app-distribute` to run the pipeline.

Build Stage

Here is the workflow build script for the **debug** :

```
assembleDebug:
  stage: build
  only:
    - app-distribute
  script:
    - ./gradlew assembleDebug
```

If you want the **production (release)**, here is the workflow build script :

```
assembleRelease:
  stage: build
  only:
    - app-distribute
```

```
    script:
      - echo ${GOOGLE_SERVICE_JSON} | base64 -d > app/google-
services.json
      - echo ${KEY_STORE_PROP} | base64 -d > key.properties
      - echo ${KEYSTORE_FILE} | base64 -d > app/keystore.jks
      - ./gradlew assembleRelease
    artifacts:
      expire_in: 7 days
      paths:
        - app/build/outputs/apk/**/*.apk
```

It's depending on your needs.

Sometimes the Docker Image OpenJDK script from Gitlab CI has problems because the license of the Android SDK changes causing the build to fail. An alternative is to use another prebuild Docker Image.

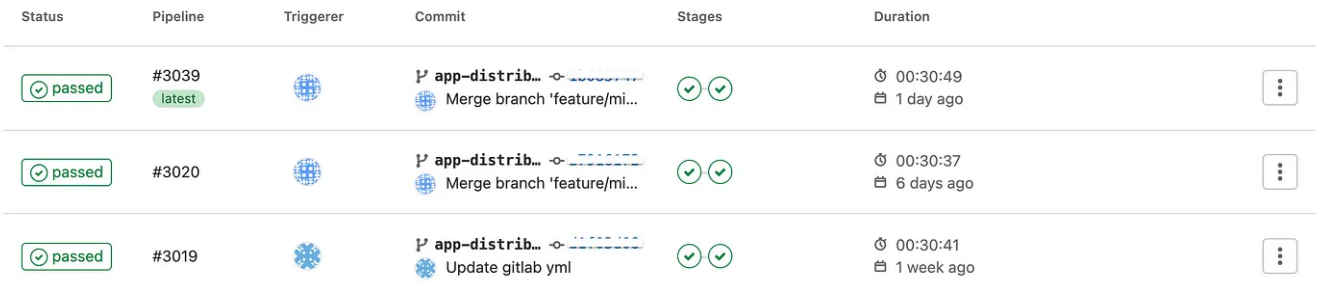
Deploy Stage

You first need to create a test **group** in the firebase console, then add the file containing the new **release update notes** `release-notes.txt` at the root of the folder structure.

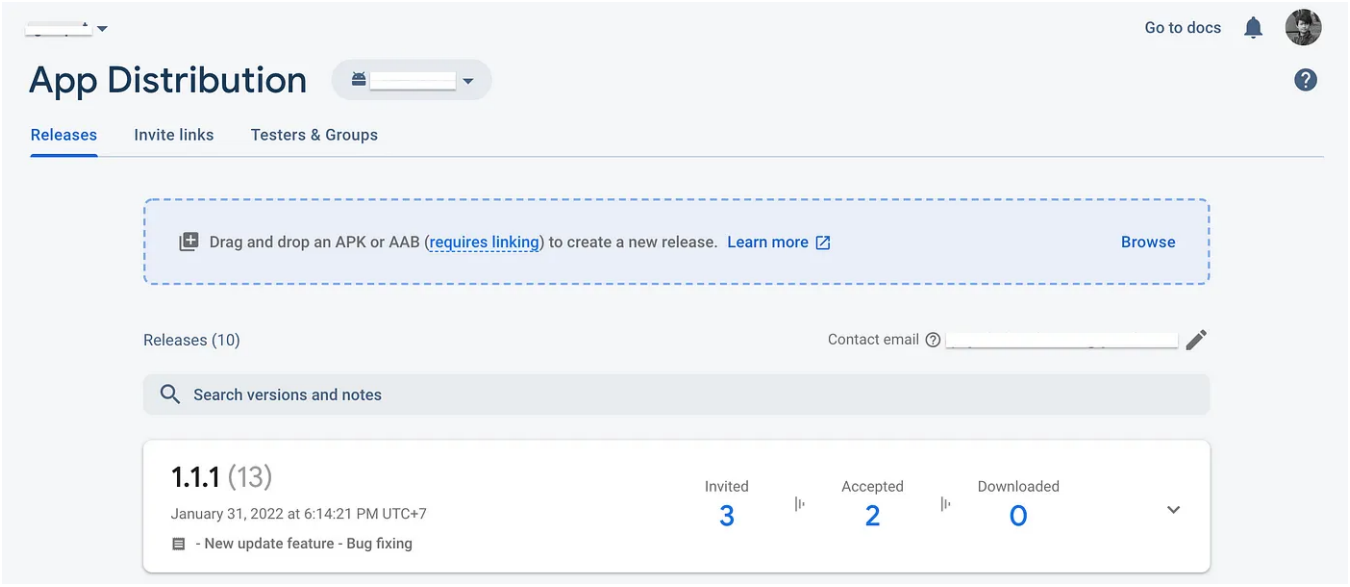
```
  deployFirebase:
    stage: deploy
    image: node:latest
    before_script:
      - export GRADLE_USER_HOME=$(pwd)/.gradle
      - export JAVA_HOME="/usr/bin/java"
      - apt-get update -y && apt-get install wget -y
    dependencies:
      - assembleRelease
    only:
      - app-distribute
    script:
      - npm install -g firebase-tools
      - if [ -f "app/build/outputs/apk/release/app-release.apk" ];
  then firebase appdistribution:distribute
  app/build/outputs/apk/release/app-release.apk --app $FIREBASE_APP_ID
  --release-notes-file release-notes.txt --groups "gitlab-group" --
  token "$FIREBASE_CI_TOKEN"; fi
```

Implementation

Here's a sample result of how I implement Gitlab CI/CD in one of the Android projects in my repo.



Gitlab CI/CD Pipeline



Firebase App Distribution Console

Full code can be seen here: https://gitlab.com/-/snippets/2258348

Conclusion

To prevent a mobile developers from generate APKs **repeatedly** which sometimes disturbs their sleep or weekends, I think CI/CD can be applied, especially in developing android applications using the Gitlab repo. Some of the better alternative third-party service tools like CodeMagic might also be good, but they are need more cost. So far, Gitlab CI/CD is **not bad and quite powerful.** However, maybe some repo have a size limitation for uploading APKs.

If you have any questions or something that you don't understand from the explanation above, feel free to ask in the comments below.

Android Gitlab Firebase App Distribution Apk Ci Cd Pipeline

Follow

Written by Ramdan Nurul

16 Followers

A Software Engineer, Living in Bandung.

More from Ramdan Nurul

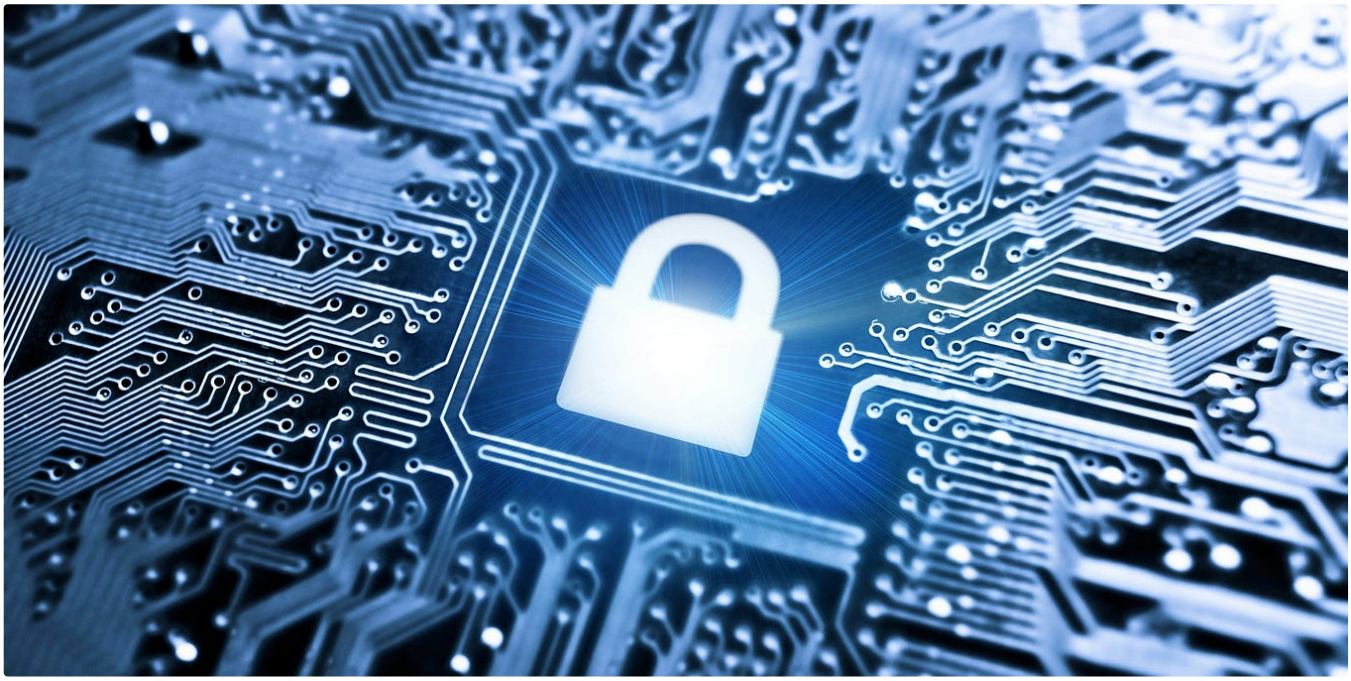
Ramdan Nurul

Teknologi Membuat Orang Pintar atau Sebaliknya?

Sampai saat ini teknologi adalah salah satu bidang yang paling cepat dalam perkembangannya, tidak ada seorang pun yang dapat...

3 min read · Dec 29, 2018

4 1



Ramdan Nurul

Perbedaan Enkoding, Enkripsi, dan Hash?

Bagi seorang programmer mungkin tidak asing dengan istilah enkoding, enkripsi, dan hash. meskipun begitu tidak sedikit dari mereka yang...

3 min read · Apr 19, 2018

105

Ramdan Nurul

Migrasi Data : Proses Pemindahan Database

Apakah anda pernah mendengar Istilah Migrasi Data atau Data Migration? Apa pengertian dari kata tersebut? dan apa manfaat yang bisa kita...

3 min read · Mar 15, 2018

8

Ramdan Nurul

Seni Menyembunyikan Pesan Steganography

Pada artikel sebelumnya, saya sudah pernah memberikan sedikit penjelesan mengenai kriftografi, salah satu fungsi dari kriftografi yaitu...

6 min read · May 26, 2020

3

See all from Ramdan Nurul

Recommended from Medium

Asmae ziani

Reverse Engineering and Analyzing Android Apps: A Step-by-Step Guide

Reverse engineering is the process of taking apart a software application to understand how it works and possibly modify it. This can be...

· 6 min read · Dec 26, 2022

263 2

Maxi Rosson in Dipien

10 ideas to reduce your APK size [Part III]

Less is More: A Comprehensive Guide to Reducing APK Size and Optimizing Your Android App's User Experience

✦ · 7 min read · Mar 13

10

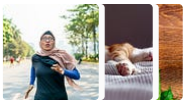
Lists

Staff Picks

304 stories · 66 saves

Stories to Help You Level-Up at Work

19 stories · 24 saves

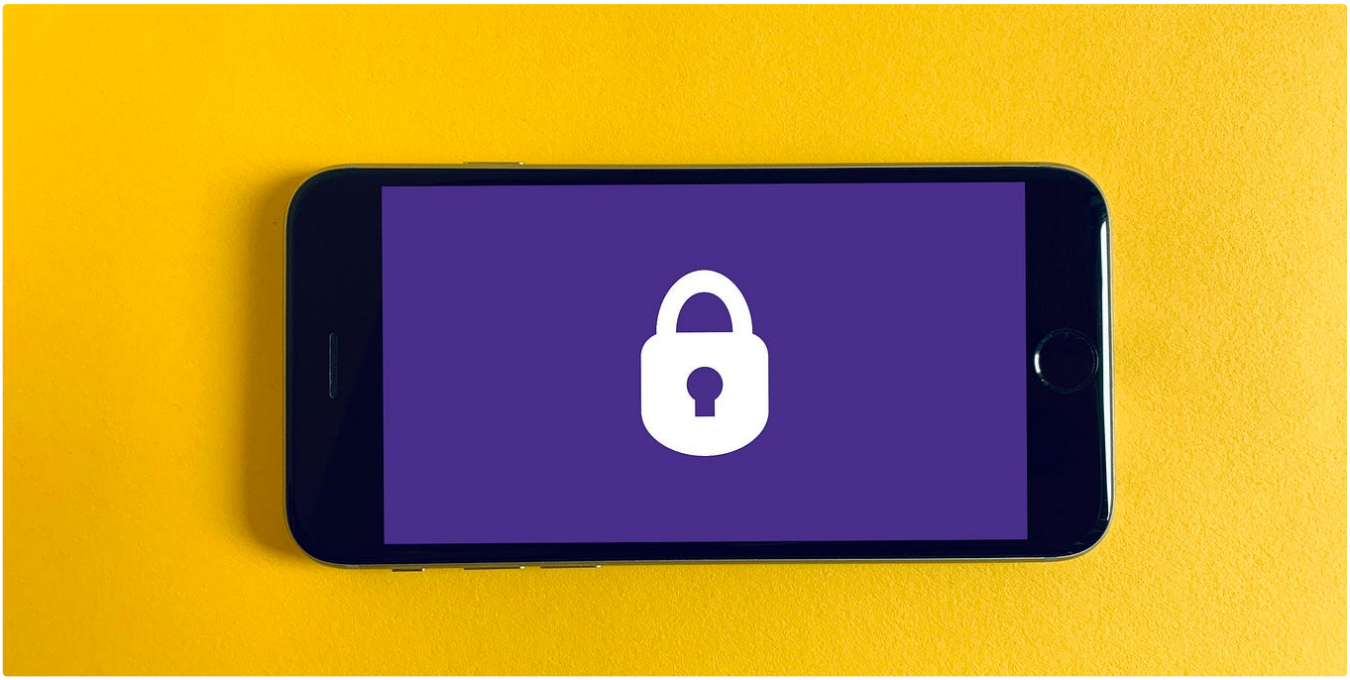


Self-Improvement 101

20 stories · 63 saves

Productivity 101

20 stories · 53 saves



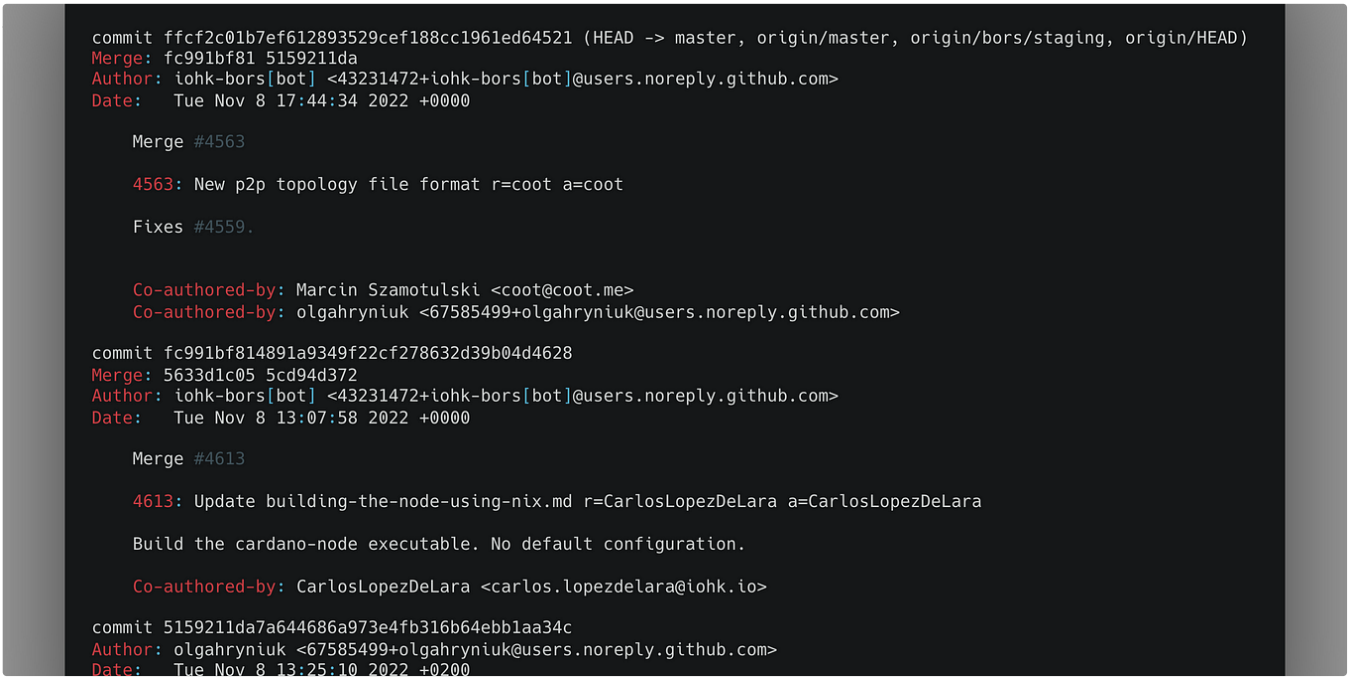
Patryk Kosieradzki in CodeX

Top 10 Mobile App Security Risks #1—Improper Platform Usage on Android

The first article in a series dedicated to the OWASP Mobile Top 10 — a comprehensive list of the most common and significant security...

✦ · 13 min read · Mar 24

8



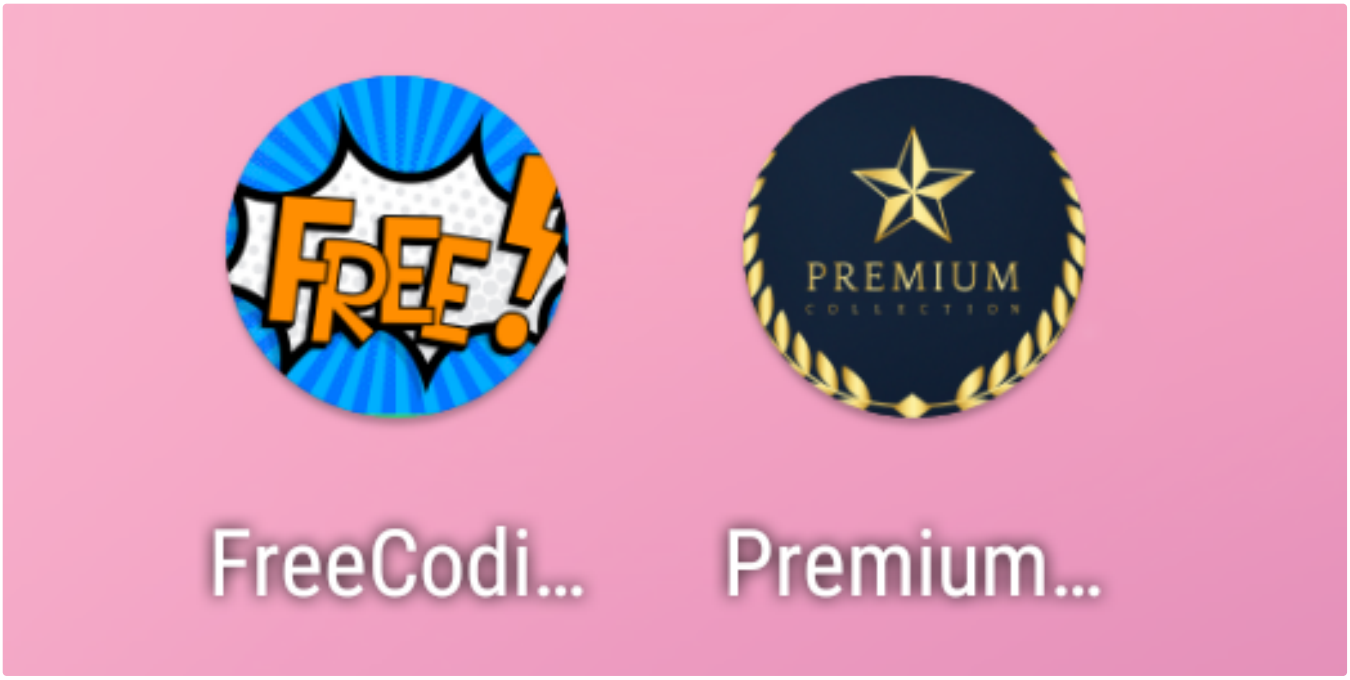
Jacob Bennett in Level Up Coding

Use Git like a senior engineer

Git is a powerful tool that feels great to use when you know how to use it.

✦ · 4 min read · Nov 15, 2022

4.6K 51



Shaik Ahron in Level Up Coding

Explaining Android Product Flavors using a real-life example

The flow of Article:1. What are Product Flavors? 2. Scenarios Where we can use Product Flavors. 3. Example App

5 min read · Nov 23, 2022

154

inVerita

12 Best Android Development Tools & Software

The Google Play store offers more than 2.67 million apps available for download as of March 2023, according to Statista, beating Apple App...

· 9 min read · Mar 22

9

See more recommendations