**Continuous Integration & Continuous Deployment (CI/CD)**

**Continuous Integration(CI):** systems let you automatically build and test your app every time you commit into a shared repository.

**Continuous Deployment(CD): A**utomated deployment system let you automatically deploy your app on the app store or play store (in the case of the mobile apps) every time when you commit into a shared repository or your trigger([Triggering pipeline](https://docs.gitlab.com/ee/ci/triggers/)) is fired.

***Benefits of CI/CD***

i. Code versioning & control

ii. Reduce efforts and time

iii. Reduce overheads

iv. Reduce the risk of wrong commit and missing files.

v. Build & Deployment automation

**CI/CD Tools**

1. GitLab
2. Jenkin

**Jenkin:** **in Jenkins** is an open source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery. It is a server-based system that runs in servlet containers such as Apache Tomcat.

**Jenkins** is a popular open source automation server. It's used to implement Continuous Integration (CI) and Continuous Delivery (CD) for any development project. CI/CD, a key component of a DevOps strategy, allows you to shorten the development lifecycle while maintaining quality by automating tasks like testing.

1. Create Project: We create a project & push it on git/guthub,
2. Kenkin Configure: configure & setup the jenkin. jenkin create & build the APK and jenkin push the build/apk on app centre.
3. APK: Tester will download app from app centre for testing

Steps:

1. Install Java-8 or latter
2. Install Android Studio
3. Install graddle command line tool
4. Install git/gitHub for code hosting(optional) or you can manage it though local codebase
5. Install Jenkin
6. configure App centre
   1. Add New App:> Create the new app by selecting the OS(Android/IOS) and Plateform(Java/kotlin and Swift/Objective-c)
   2. Create Tester group:> Click on Distribute & select Group-> create group and add user/tester
   3. Copy the ID & User API Token from account setting
7. Configure the Jenkin

i. Android PATH

Manage Jenkin-> Configure system -> Global Properties -> Environment variable-> Add SDK :=> Name: ANDROID\_HOME and Value: Path of android SDK

ii. Java Path

Manage Jenkin-> Global Tool Configure -> JDK :=> Name: Java and JAVA\_HOME: Path of JDK

iii. Java Path

Manage Jenkin-> Global Tool Configure ->Gradle:=> Name: latest and version: gradle version

iv. Create Project

**Create Project:** New Item-> Name of your Project

**Add Git URL:** **Source Code Management->** Git URL(git credentials if required)

**Build-> Build Environment:**> Time out action :> Abort the build

**Build-> Build:**> Invoke Gradle => Gradle version:> latest

**Build-> Build:**> Use Gradle Wrapper :> Tools:> clean

**Build-> Build:**> Use Gradle Wrapper :> Tools:> clean and assembleDebug

**Post** **Build Action->** Archive the artifacts :> File to archive:> \*\*/\*.apk

**Post** **Build Action->** Upload App to app centre :> Add your app centre token & owner/name/user ID, App & Group name, Path to app(\*\*/\*.apk)

v. Build Action

Build Now: your build/apk will appear

1. configure App centre in Jenkin
   1. Go To manage jenkin -> Manage Plugging -> Search & Install the APK centre

**GitLab:** Getting Started with GitLab CI/CD with android

[GitLab](https://about.gitlab.com/stages-devops-lifecycle/) is an open source collaboration platform that provides powerful features beyond hosting a code repository. You can track issues, host packages and registries, maintain Wikis, set up continuous integration (CI) and continuous deployment (CD) pipelines, and more.

In this tutorial you’ll build a continuous deployment pipeline with GitLab. You will configure the pipeline to build a Docker image, push it to the GitLab container registry, and deploy it to your server using SSH. The pipeline will run for each commit pushed to the repository.

1. Create project/repo on gitlab

ii. **.gitlab-ci.yml:** GitLab CI/CD pipeline are configured using YAML file called .gitlab-ci.yml in each project - .gitlab-ci.yml file defines the structure and order of the pipeline & determines. Add this file in your root folder of your project/repo and in this file we configure our pipeline.

<https://docs.gitlab.com/ee/ci/yaml/>

Note: After writing scrip in .yml file, pls commit & push this on gitLab

iii. **Gradle Play Publisher plugin:** Gradle Play Publisher is a Gradle plugin that allows you to upload your App Bundle or APK and other app details to the Google Play Store.

<https://github.com/Triple-T/gradle-play-publisher>

iiv. **Setup Gradle Play Publisher Plugin:** open your project level build.gradle (not app/build.gradle) file and set the classpath

iv. **Setup** **build.gradle(app/build.gradle):** -file and add below line at the top of the file

apply plugin: ‘com.github.triplet.play’

vi. **Set Sign In Config:** if no publishing tasks were created, you most likely haven’t added a valid signing configuration to your release build. Be sure to [add one](https://developer.android.com/studio/publish/app-signing#gradle-sign).

*Note: add this in****app/build.gradle****and make sure to put the jks file in the same directory like****app/my-release-key.jks***

**Step to setup Gitlab CI**

1. Create Gitlab project and setup repository for the android project.
2. Setup Continous Integration using the gitlab default CI android template.
3. Automatically build android application and generate APK when pushing the code to the particular branch.
4. Run the Link check and Test Cases
5. Deploy the android APK to Firebase App Distrubution.
6. Create GitLab project and setup repository  
   Create a new project on gitlab with entering details of like project name, url, slug, description, visibility(private/public) and project configuration. Also, initialized the readme file and enable static application security testing(SAST) for the project details.
7. Setup GitLab project in Android Studio
   1. Install GitLab in your local system
   2. Create Android Project
   3. Run, set-up, Push, Commit the code on GitLab

git init

git remote add origin git@gitlab.com:GITLABUSERNAME/YOURGITPROJECTNAME.git

git add .

git commit -m "Initial commit"

git push -u origin master

After the git push, you can able to see the initial android project code pushed to the git master branch of the GitLab project.

1. Setup Continuous Integration on GitLab

Already we have completed the GitLab repository setup, Let’s create a GitLab pipeline to automate test and deployment. To get started we need to create**.gitlab-ci.yml** in the root direct of the project.

# To contribute improvements to CI/CD templates, please follow the Development guide at:

# https://docs.gitlab.com/ee/development/cicd/templates.html

# This specific template is located at:

# https://gitlab.com/gitlab-org/gitlab/-/blob/master/lib/gitlab/ci/templates/Android.gitlab-ci.yml

# Read more about this script on this blog post https://about.gitlab.com/2018/10/24/setting-up-gitlab-ci-for-android-projects/, by Jason Lenny

# If you are interested in using Android with FastLane for publishing take a look at the Android-Fastlane template.

image: openjdk:11-jdk

variables:

# ANDROID\_COMPILE\_SDK is the version of Android you're compiling with.

# It should match compileSdkVersion.

ANDROID\_COMPILE\_SDK: "30"

# ANDROID\_BUILD\_TOOLS is the version of the Android build tools you are using.

# It should match buildToolsVersion.

ANDROID\_BUILD\_TOOLS: "30.0.3"

# It's what version of the command line tools we're going to download from the official site.

# Official Site-> https://developer.android.com/studio/index.html

# There, look down below at the cli tools only, sdk tools package is of format:

# commandlinetools-os\_type-ANDROID\_SDK\_TOOLS\_latest.zip

# when the script was last modified for latest compileSdkVersion, it was which is written down below

ANDROID\_SDK\_TOOLS: "7583922"

# Packages installation before running script

before\_script:

- apt-get --quiet update --yes

- apt-get --quiet install --yes wget tar unzip lib32stdc++6 lib32z1

# Setup path as ANDROID\_SDK\_ROOT for moving/exporting the downloaded sdk into it

- export ANDROID\_SDK\_ROOT="${PWD}/android-home"

# Create a new directory at specified location

- install -d $ANDROID\_SDK\_ROOT

# Here we are installing androidSDK tools from official source,

# (the key thing here is the url from where you are downloading these sdk tool for command line, so please do note this url pattern there and here as well)

# after that unzipping those tools and

# then running a series of SDK manager commands to install necessary android SDK packages that'll allow the app to build

- wget --output-document=$ANDROID\_SDK\_ROOT/cmdline-tools.zip https://dl.google.com/android/repository/commandlinetools-linux-${ANDROID\_SDK\_TOOLS}\_latest.zip

# move to the archive at ANDROID\_SDK\_ROOT

- pushd $ANDROID\_SDK\_ROOT

- unzip -d cmdline-tools cmdline-tools.zip

- pushd cmdline-tools

# since commandline tools version 7583922 the root folder is named "cmdline-tools" so we rename it if necessary

- mv cmdline-tools tools || true

- popd

- popd

- export PATH=$PATH:${ANDROID\_SDK\_ROOT}/cmdline-tools/tools/bin/

# Nothing fancy here, just checking sdkManager version

- sdkmanager --version

# use yes to accept all licenses

- yes | sdkmanager --licenses || true

- sdkmanager "platforms;android-${ANDROID\_COMPILE\_SDK}"

- sdkmanager "platform-tools"

- sdkmanager "build-tools;${ANDROID\_BUILD\_TOOLS}"

# Not necessary, but just for surity

- chmod +x ./gradlew

# Basic android and gradle stuff

# Check linting

lintDebug:

interruptible: true

stage: build

script:

- ./gradlew -Pci --console=plain :app:lintDebug -PbuildDir=lint

# Make Project

assembleDebug:

interruptible: true

stage: build

script:

- ./gradlew assembleDebug

artifacts:

paths:

- app/build/outputs/

# Run all tests, if any fails, interrupt the pipeline(fail it)

debugTests:

interruptible: true

stage: test

script:

- ./gradlew -Pci --console=plain :app:testDebug

**Understanding .gitlab-ci.yml**

first, we will see the important key terms in yml file.

**Image:**This tag is used to specify a Docker image to be used for execution. Gitlab runners will use this Docker image to run the pipeline.

**Variables:** This variable tag is used to set the variables used in the script execution.

**Before Script:** This tag is used to execute the scripts before the main script execution. Mainly used to set up the environment.

**Script:** We need to use a script tag to execute the main functions like building the APK or running the test cases.

**Stages:** This tag is used to specify the globally defined level of the pipeline run. This stage will execute on the given order. Each stage can contain different jobs. Every stage has the following tags.

* **Stage:** used to define the which stage this running job belongs to.
* **only:**This only tag is used to define on which condition this particular stage should execute. we can use this tag to execute the stage only on master branch changes or etc.
* **artifacts:** Jobs can output an archive of files and directories. This output is known as a job artifact. You can download job artifacts by using the GitLab UI or the API.

Now we know most of the tags used in the pipeline. below I have explained the pipeline yml file in detail.

**Defining the Docker Image:** Docker containers are very fast to create and destroy instances, also a good choice for setting up a temporary environment for building and testing.

**image: openjdk:11-jdk**

**Defining the variables:** lets set up the variables needed to build the android applications.

ANDROID\_COMPILE\_SDK: “30"

ANDROID\_SDK\_TOOLS: "7583922"

ANDROID\_BUILD\_TOOLS: “30.0.3"

**Setup android environment:** on the before\_script, we need to set up the android environment by setting up ANDROID\_SDK\_ROOT and installing the android SDK manager.

- apt-get --quiet update --yes

- apt-get --quiet install --yes wget tar unzip lib32stdc++6 lib32z1

# Setup path as ANDROID\_SDK\_ROOT for moving/exporting the downloaded sdk into it

- export ANDROID\_SDK\_ROOT="${PWD}/android-home"

# Create a new directory at specified location

- install -d $ANDROID\_SDK\_ROOT

# Here we are installing androidSDK tools from official source,

# (the key thing here is the url from where you are downloading these sdk tool for command line, so please do note this url pattern there and here as well)

# after that unzipping those tools and

# then running a series of SDK manager commands to install necessary android SDK packages that'll allow the app to build

- wget --output-document=$ANDROID\_SDK\_ROOT/cmdline-tools.zip https://dl.google.com/android/repository/commandlinetools-linux-${ANDROID\_SDK\_TOOLS}\_latest.zip

# move to the archive at ANDROID\_SDK\_ROOT

- pushd $ANDROID\_SDK\_ROOT

- unzip -d cmdline-tools cmdline-tools.zip

- pushd cmdline-tools

# since commandline tools version 7583922 the root folder is named "cmdline-tools" so we rename it if necessary

- mv cmdline-tools tools || true

- popd

- popd

- export PATH=$PATH:${ANDROID\_SDK\_ROOT}/cmdline-tools/tools/bin/

# Nothing fancy here, just checking sdkManager version

- sdkmanager --version

# use yes to accept all licenses

- yes | sdkmanager --licenses || true

- sdkmanager "platforms;android-${ANDROID\_COMPILE\_SDK}"

- sdkmanager "platform-tools"

- sdkmanager "build-tools;${ANDROID\_BUILD\_TOOLS}"

# Not necessary, but just for surity

- chmod +x ./gradlew

**Building the app:** To start the build job, we need to define the job name and the stage. in the script tag we need to mention the Gradle script to build the android app. Also, In the artifacts tag, we need to define the output apk file location. So, that we can download the output of the build.

assembleDebug:

interruptible: true

stage: build

script:

- ./gradlew assembleDebug

artifacts:

paths:

- app/build/outputs/

**Running tests**: same as building the app, first need to define the job name and the stage. Then, the script tag needs to provide the Gradle script to execute the test cases.

debugTests:

interruptible: true

stage: test

script:

- ./gradlew -Pci --console=plain :app:testDebug

1. How to run the gitlab CI setup

After you’ve added your new **.gitlab-ci.yml** file to the root of your directory, just push your changes to the repository. Then the build will be triggered automatically, you can see the process in the Pipelines tab of your project. Also, you have an option to run the pipeline manually in the pipeline tab.