

3 Ways to Convert Python App into APK

Concluding Building Android Apps in Python Series!

 [Kaustubh Gupta](#) Dec 3, 2020 · 7 min read

In August 2020, I started a medium blog series called Building Android App in Python, where I explained the usage of Kivy and Kivymd. These libraries allow you to create cross-platform apps using Python. In that series, I described how android apps are configured in Python, it’s limitations, and various key elements that build up an android app. At the end of the series (3rd part), I mentioned that I will convert an app into APK and deploy it on a cloud platform. Here is the article about the same! Here I will discuss three ways in which this conversion can be performed. If you haven’t followed this series or you are unaware of this library then I highly recommend you that check out the first part of the series:

Building Android Apps With Python: Part -1
Step by Step Guide to Build Android Apps using Python
[towardsdatascience.com](#)






Photo by [Daniel Korpai](#) on [Unsplash](#)

The Challenge

The Python apps build with Kivy cannot be directly transferred to android phones as these devices only support APK (Android Application Package) and we need to package them properly. This conversion process is only possible on a Linux system (for now) as the main components of this conversion, [buildozer](#) and [python-for-android](#) are currently supported on Linux based systems only. This adds up a challenge for budding developers

who generally who use a windows machine for all the coding purpose. Other challenges include failed app conversions, app crashing on the start, or not able to connect to the internet. While some issues need extra attention from your side, I will provide you 3 different ways to successfully convert the Python app to APK.

Before moving ahead, let’s look at the flow of the conversion:

- 1. Making sure that the app entry point file is named as main.py
- 2. Installing the dependencies.
- 3. Initialize the buildozer
- 4. Edit the specs file
- 5. Start the process

We will see 1,2 and 4 steps in detail in upcoming sections as they don’t require any explicit change but it is necessary to understand how to configure the buildozer specs properly.

The buildozer spec file is automatically generated while initializing the buildozer. The file contains the whole configuration and the app is built following this only. There are a few lines that need to be modified in that file before proceeding with the next steps (There are a total of 339 lines in the actual file):

```
1  # (str) Title of your application
2  title = <Nice tile of the app>
3
4  # (str) Package name
5  package.name = <A package name of your app>
6
7  # (str) Package domain (needed for android/ios packaging)
8  package.domain = <Name this as: com.<any name> >
9
10 # (list) Application requirements
11 # comma separated e.g. requirements = sqlite3,kivy
12 requirements = python3,kivy, kivymd, <More supported requirements can be added here>
13
14 # (list) Permissions
15 android.permissions = INTERNET <commnet, uncomment depending upon use case>
```

importantLinesBuildozer.spec hosted with ❤ by GitHub

view raw

These lines are not in actual order and have been picked from the generated file

Give your app a suitable title, a package name which can simply be the lower case, concatenated version of the title, package domain which can start with com or org, then list the requirements of your app and finally uncomment the internet permissions line in case your app requires it. A small piece of advice, use kivy and kivymd defined modules to avoid any app crashing error.

1. The Hard Way: Setup Virtual Linux

The first very obvious way is to set up a virtual machine, spin up a Linux distribution, and install all the dependencies. This way is a little tricky because:

1. Not all of us have high-performance machines that can handle virtualization. To run a virtual machine, at least 4GB of RAM would be required to be allotted to this virtual system to properly fire up the machine.
2. The process of configuring the system which involves installing the Linux, setting up the dependencies, transferring the data from the host system to the virtual machine is time-consuming.

You can set up the virtual box and install any Linux distribution in that. The kivy developers have simplified to install all the dependencies at one go by running a bash script that installs python and other requirements to build the app. In the terminal,

```
wget https://github.com/HeaTTheatR/KivyMD-data/raw/master/install-kivy-buildozer-dependencies.sh
chmod +x install-kivy-buildozer-dependencies.sh

./install-kivy-buildozer-dependencies.sh
```

After the script is done installing, navigate to your app directory and run:

```
buildozer init
```

This will create the buildozer spec file which consists of all configurations you can modify. After you are done editing this file, run:

```
buildozer android debug
```

For the first time, this command will take longer than usual to execute (14–19 minutes) and after that, you will have a bin folder in the app directory with the APK. Transfer the app to any android device and see if works!


If you opt this way then you will have a configured VM that can be used for other apps to be converted to APKs in less time. You can also run these installations in WSL (Windows Subsystem Linux) in Windows 10 if you don't want to opt for a virtual box setup.

2. The Google Colab Way!

The conversion requires a Linux environment and Google provides it for free! If you belong to Data Science or Deep learning background then you

must be aware of this platform. It provides you a virtual machine with 75GB space, 12GB RAM, and around 12GB GPU power! You can use this platform to perform model training, checking logs, or running Python codes.

As this is a Linux based system, now the only thing required is to install the dependencies and initiate the process. You can directly use my colab notebook where you just need to run all the cells, that’s it!

 Open in Colab

(<https://colab.research.google.com/gist/kaustubhgupta/0d06ea84760f65888a2488bac9922c25/kivyapp-to-apk.ipynb>)

In []:

!pip install buildozer


In []:

!pip install cython==0.29.19

In []:

!sudo apt-get install -y \
python3-pip \
build-essential \
git \
python3 \
python3-dev \
ffmpeg \
libSDL2-dev \
libSDL2-image-dev \
libSDL2-mixer-dev \
libSDL2-ttf-dev \
libportmidi-dev \
libswscale-dev \
libavformat-dev \
libavcodec-dev \
zlib1g-dev

kivyapp-to-apk.ipynb

 hosted with  by [GitHub](#)

[view raw](#)

The notebook code is originally provided by [Machine Learning Hub](#). Check out [their video](#)!

Before running the cells, make sure to upload your app code to the colab notebook and after running the bulldozer init command, make sure to edit the specs file generated and nothing else!

This is the easiest and most convenient way to build apps without the need for an actual system! But can we go even more simple? Let’s see

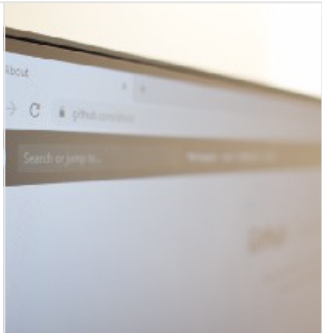
3. The GitHub Action Way!

Ahh, it also provides a Linux system but only when the process is getting executed. Don’t know much about this? After you are done with this article make sure to check out this article!

GitHub Action That Automates Portfolio Generation

Dockerized GitHub Action using Python and Basic Front-end.

towardsdatascience.com



Github Actions are the piece of software that lets you build, test, and deploy your code right from your GitHub repository. You can schedule jobs for

various events occurring in your GitHub repository such as pushing new code, commits, or opening an issue. You can schedule either on basis of these events or use a corn job to set a timer. These GitHub actions run on a Linux system and therefore you can configure the various parameters for the conversion.

The best part is that the developers of Kivy are kind enough that they have created this action too! You just need to include this workflow in your `.github/workflows/<any name>.yml` file:

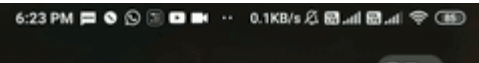
```
1  name: Build
2  on:
3    push:
4
5  jobs:
6    # Build job. Builds app for Android with Buildozer
7    build-android:
8      name: Build for Android
9      runs-on: ubuntu-latest
10
11     steps:
12       - name: Checkout
13         uses: actions/checkout@v2
14
15       - name: Build with Buildozer
16         uses: ArtemSBulgakov/buildozer-action@v1
17         id: buildozer
18         with:
19           workdir: <specify the directory of the app no don't mention this the app files a
20           buildozer_version: stable
21
22       - name: Upload artifacts
23         uses: actions/upload-artifact@v2
24         with:
25           name: package
26           path: ${ steps.buildozer.outputs.filename }}
```

kivy_workflow.yml hosted with ❤ by GitHub view raw

In this method, you need to place the buildozer spec file also with your app files. You can find any spec file online and after that just copy-paste that, edit it, upload the files to the repository and then add this workflow. After you run this workflow, your app will be created as an artifact that can be easily downloaded. You can also configure it to upload the APKs on another branch of the repository. Check out the official [GitHub page](#) for all the configurable parameters.

Any Examples?

I have prepared two apps for this article. One of them is a Machine Learning Deployment which predicts a song genre based on the given parameters. The second one is a very basic authentication app we build in the [second part of the series](#). Here is the video where I am running these apps on my personal Android phone:





GIF by Author

The source code for both the apps (Python file, APK, and buildozer spec) can be found at this [GitHub repository](#).

If you don’t want to use the APK generated into your devices then you simply host them online using a free service called [Appetize.io](#) which allows you to run android apps in browsers!

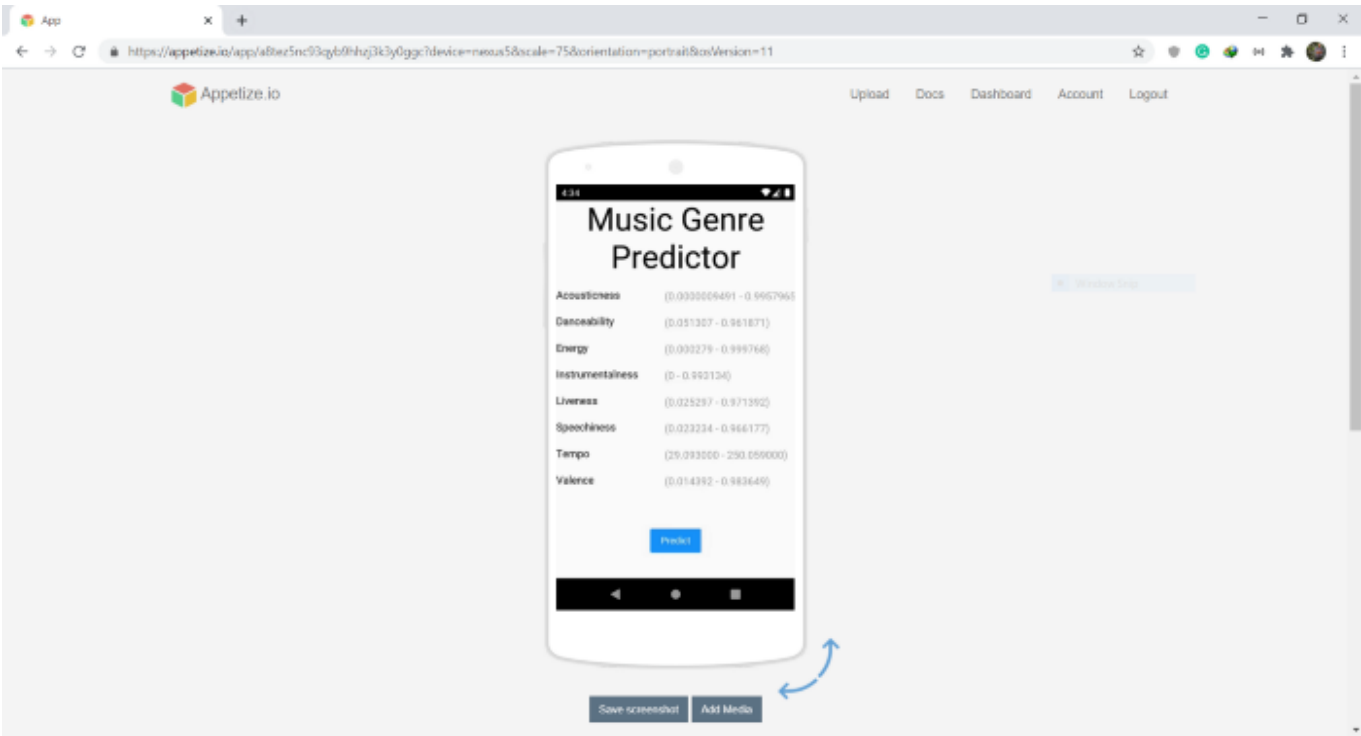


Image by Author. Try this app [here](#).

Conclusion

In this article, we saw how to configure the deployment file needed to convert the Python file into APK. Then we saw 3 different ways to do this conversion. A local machine is the best option but if don’t want to mess with your system then I would suggest Google Colab as it has great processing power. If you are looking for CI/CD type solution then GitHub actions will suit your requirements. With this, we come to an end of this article as well as the development series.


If you don’t want to miss such articles then make sure to follow me on medium to receive all the notifications. With that said, Sayonara!

You can connect with me here:

Kaustubh Gupta - Machine Learning Writer - upGrad | LinkedIn

Hi, I am a Python Developer capable of Web Scraping, Automations, Data Science, Backend Web Development with knowledge...

www.linkedin.com




My other popular articles:

Run Python Code on Websites: Exploring Brython

JavaScript Equivalent Scripting in Python

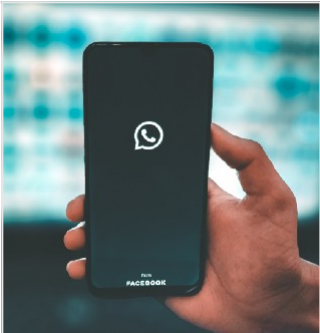
towardsdatascience.com



Is Family Group That Bad? Results Will Shock You

Analyzing WhatsApp Group Chats & Building the Web App


towardsdatascience.com



How Pandemic Has Affected College Scores: Analysis On Real Dataset

A deep dive into exploring my college scores and finding trends in it


towardsdatascience.com



Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

 Get this newsletter

You'll need to sign in or create an account to receive this newsletter.

- Python

Android

Google Colab

Github

Linux