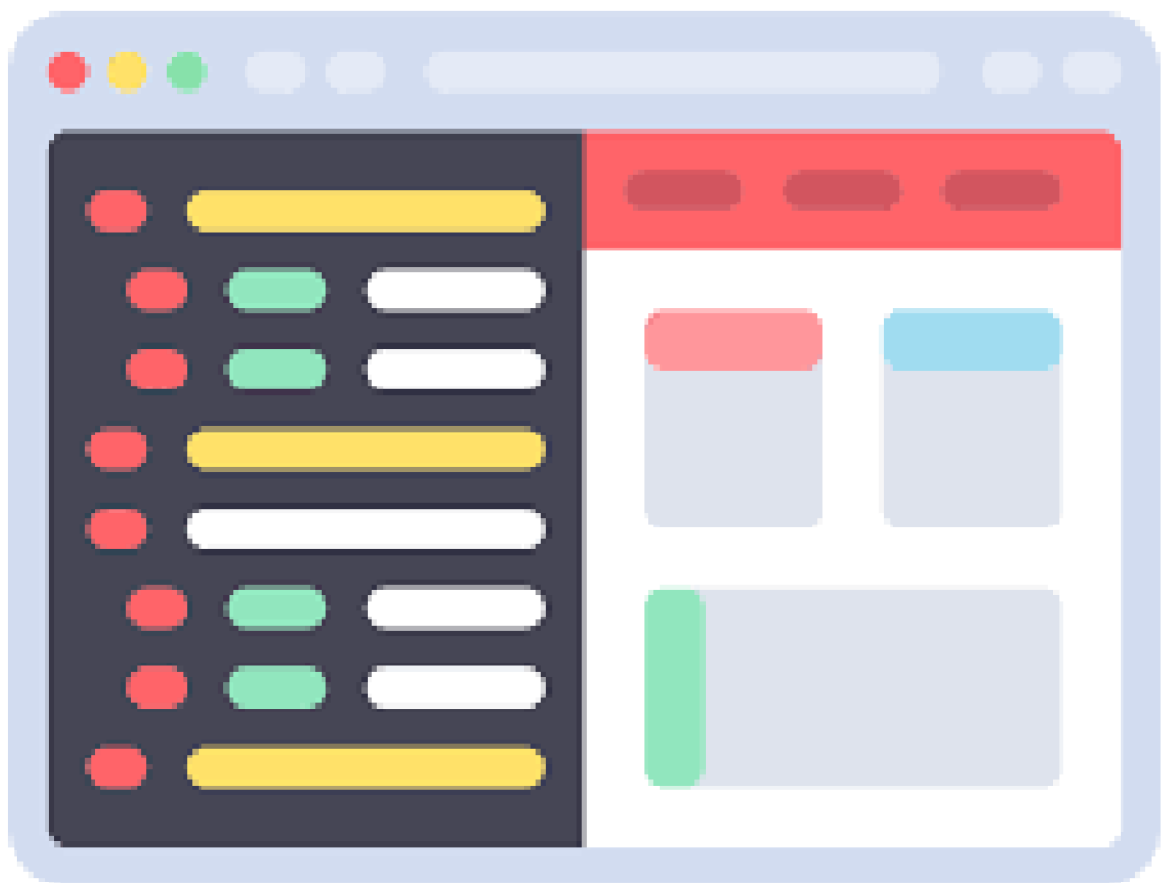


# Ionic 4 - Build APK e Android App Bundle

O guia passo a passo que vai lhe mostrar como você pode gerar o arquivo final da sua aplicação Ionic.



EDIGLEYSSON SILVA

# Índice

Sobre este material .....	3
Sobre o autor .....	4
Capítulo 1 – Conhecendo as tecnologias .....	5
Capítulo 2 – Preparando o ambiente .....	7
Capítulo 3 – Configurando o app .....	13
Capítulo 4 – APK em Debug e em Release .....	15
Capítulo 5 – Build APK em Release .....	17
Apêndice A – Android App Bundle .....	19
Referências .....	21

## Sobre este material

O Ionic Framework é uma excelente escolha no desenvolvimento de aplicativos híbridos, principalmente agora que conta com suporte a ReactJS e VueJS. Além disso conta com uma comunidade muito ativa e que só cresce.

Com o intuito de complementar a documentação e facilitar a utilização dessa excelente ferramenta, tanto para desenvolvedores iniciantes quanto experientes, tenho iniciado a produção de alguns conteúdos sobre o Ionic 4. Dentre estes conteúdos está esse material que se originou de uma minissérie de três vídeos que tratam sobre o build de uma aplicação para a plataforma Android.

Esse é o material que eu gostaria de ter encontrado quando concluí o meu primeiro app. Aqui há um passo a passo de como você pode gerar sua APK e vamos fazer isso sem precisar do famigerado Android Studio IDE.

### Por que não utilizaremos o Android Studio?

O Android Studio é uma excelente IDE e vai ser um belo atalho para a instalação da maioria dos componentes que serão instalados. É excelente para desenvolvimento de aplicações nativas, mas não se faz realmente necessária para aplicativos híbridos.

Outro ponto é que o Android Studio necessita de alguns recursos que você talvez não tenha no momento. Portanto, utilizar uma IDE tão complexa para tal propósito pode não ser a melhor opção.

### Tenho o Android Studio instalado, posso seguir este material?

Sim, se você tem os recursos necessários para o Android Studio você poderá ainda pular para o capítulo 3. Pois se o Android Studio já está instalado é possível que todo o ambiente já esteja pronto.

## Sobre o Autor

Me chamo Edigleysson Silva, sou Analista de Sistemas com mais de seis anos de experiência de mercado.



Sempre tenho buscado aprender coisas novas e cada vez mais compartilhar conhecimentos com a comunidade. Tenho focado mais na área de desenvolvimento web e mobile, sempre atuando tanto no front-end quanto no back-end.

Sempre gostei bastante do Angular e isso me levou a conhecer o Ionic e com ele desenvolvi um aplicativo que esteve presente em um evento na Clinton Foundation, em Chicago.

Se quiser saber mais sobre mim e acompanhar meus conteúdos pode acessar o meu site em <https://codesilva.github.io>

# Capítulo 1 – Conhecendo as tecnologias

Semelhante ao futebol, onde os jogadores fazem um reconhecimento do campo antes de um jogo. Faremos aqui um reconhecimento de campo, observando e entendendo como funcionam as ferramentas que serão necessárias para a geração da nossa APK.

## 1.1 Java, JDK e JRE

### 1.1.1 Java

Java é uma linguagem de programação muito popular e muito utilizada por grandes players do mercado. Um bem famoso e comum em nosso cotidiano é o *Mercado Livre*. Uma das características mais fortes do Java é o seu paradigma orientado a objetos, que permite uma série de padrões para reuso de código.

Outra característica que vale a pena destacar aqui é que o código Java compilado é convertido em um bytecode que executa na JVM (Java Virtual Machine). A JVM por sua vez pode ser portada para qualquer plataforma permitindo que aplicações Java possam ser executadas em diferentes ambientes.

### 1.1.2 JDK

JDK (Java Development Kit) é todo o ferramental necessário para o desenvolvimento de aplicações Java. Ferramentas como: plugins, compiladores, etc.

### 1.1.3 JRE

JRE (Java Runtime Environment) que é o ferramental necessário para a execução de aplicações Java. Inclusive é nesta runtime que está contida a JVM citada anteriormente.

## 1.2 Android Studio e SDK do Android

### 1.2.1 Android Studio

O Android Studio é uma IDE (Integrated Development Environment) oficial para desenvolvimento Android. Possui vários recursos que vão auxiliar do início ao fim.

### 1.2.2 SDK do Android

A SDK (Software Development Kit) do Android é um conjunto de ferramentas necessárias ao desenvolvimento de aplicações para a plataforma Android.

O Android Studio, citado anteriormente, possui já vem com essas ferramentas que permitem a instalação de pacotes, criação de emuladores, build e muito mais.

## 1.3 Gradle

O Gradle é um sistema de automação de compilação baseado no Apache Ant e Apache Maven. Possui uma linguagem de domínio baseada no *Groovy* em vez do XML, este último muito comum em aplicações Java.

Resumindo o Gradle será o responsável por gerenciar a compilação do nosso projeto Android. Ele é mais um dos componentes que já estão presentes na IDE do Android Studio.

## Capítulo 2 – Preparando o ambiente

Agora que já fizemos o reconhecimento de campo e sabemos onde estamos pisando é chegada a hora do jogo. Neste capítulo iniciaremos a preparação do nosso ambiente, onde vamos instalar todos os componentes que vão nos ser úteis para o build da APK da nossa aplicação Ionic.

### 2.1 Instalação do Java

Primeiro de tudo precisamos ter o ambiente Java. Para conseguirmos isso precisamos fazer a instalação dos dois componentes citados no **Capítulo 1**, que são a JDK e JRE.

Um detalhe que não foi citado sobre a JDK e a JRE é que JDK já possui a JRE portanto para a instalação do Java basta que seja instalada a JDK. Você pode fazer o download pelo link <https://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html>.

No site da Oracle você encontrará versões para todas as plataformas, seja Windows, Linux ou Mac. Veja a imagem abaixo.

Java SE Development Kit 8u221		
You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.		
<input type="radio"/> Accept License Agreement <input checked="" type="radio"/> Decline License Agreement		
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.9 MB	<a href="#">jdk-8u221-linux-arm32-vfp-hflt.tar.gz</a>
Linux ARM 64 Hard Float ABI	69.81 MB	<a href="#">jdk-8u221-linux-arm64-vfp-hflt.tar.gz</a>
Linux x86	174.18 MB	<a href="#">jdk-8u221-linux-i586.rpm</a>
Linux x86	189.03 MB	<a href="#">jdk-8u221-linux-i586.tar.gz</a>
Linux x64	171.19 MB	<a href="#">jdk-8u221-linux-x64.rpm</a>
Linux x64	186.06 MB	<a href="#">jdk-8u221-linux-x64.tar.gz</a>
Mac OS X x64	252.52 MB	<a href="#">jdk-8u221-macosx-x64.dmg</a>
Solaris SPARC 64-bit (SVR4 package)	132.99 MB	<a href="#">jdk-8u221-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	94.23 MB	<a href="#">jdk-8u221-solaris-sparcv9.tar.gz</a>
Solaris x64 (SVR4 package)	133.66 MB	<a href="#">jdk-8u221-solaris-x64.tar.Z</a>
Solaris x64	91.95 MB	<a href="#">jdk-8u221-solaris-x64.tar.gz</a>
Windows x86	202.73 MB	<a href="#">jdk-8u221-windows-i586.exe</a>
Windows x64	215.35 MB	<a href="#">jdk-8u221-windows-x64.exe</a>

Imagem 1 - Página de download do Java

Pelo terminal Linux você pode fazer a instalação da JDK com o comando:

```
sudo apt-get install openjdk-8-jdk.
```

O comando acima instalará a versão 8, mas você pode ser livre para definir a versão desejada. Após a instalação é importante que você cheque se tudo correu bem, para isso você pode executar o comando **java -version** e se tudo estiver OK você receberá uma saída semelhante a que é mostrada abaixo:

```
edy@edy-POSITIVO-MASTER-N130i ~ $ java -version
java version "1.8.0_171"
Java(TM) SE Runtime Environment (build 1.8.0_171-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.171-b11, mixed mode)
```

Imagem 02 - Versão do Java via terminal

## 2.2 Instalação da SDK do Android

No Capítulo 1 vimos que a SDK do Android contém ferramentas necessárias ao desenvolvimento de aplicações Android.

Essa SDK já vem em conjunto com o Android Studio e é com ela que faremos instalações de pacotes como level APIs e ferramentas de build. Aqui não utilizaremos a IDE do Android Studio portanto para que tenhamos as ferramentas disponíveis precisaremos baixar o pacote de linha de comando da SDK do Android.

Para fazer o download basta acessar o link <https://developer.android.com/studio>. Ao acessar o link você verá algumas sessões que tratam da IDE do Android, mas você deve procurar pela sessão **Command line tools only** como na imagem abaixo:

Developers Platform **Android Studio** Google Play Jetpack Kotlin Mais ▾  LANGUAGE ▾ FAZER LOGIN

### Command line tools only

If you do not need Android Studio, you can download the basic Android command line tools below. You can use the included [sdkmanager](#) to download other SDK packages.

These tools are included in Android Studio.

Platform	SDK tools package	Size	SHA-256 checksum
Windows	<a href="#">sdk-tools-windows-4333796.zip</a>	148 MB	7e81d69c303e47a4f0e748a6352d85cd0c8fd90a5a95ae4e076b5e5f960d3c7a
Mac	<a href="#">sdk-tools-darwin-4333796.zip</a>	98 MB	ecb29358bc0f13d7c2fa0f9290135a5b608e38434aad9bf7067d0252c160853e
Linux	<a href="#">sdk-tools-linux-4333796.zip</a>	147 MB	92fee5a1d98d856634e8b71132e8a95d96c83a63fde1099be3d86df3106def9

See the [SDK tools release notes](#).

System requirements

Imagem 03 - Página de download da SDK do Android

A SDK por linha de comando está disponível para Windows, Mac e Linux. Faça o download de acordo com o seu sistema operacional.

O conteúdo virá em uma pasta compactada, descompacte a pasta e coloque em um diretório que achar pertinente. Para esse exemplo baixamos e o arquivo veio como **sdk-tools-linux-4333796.zip** e após a extração colocamos na pasta **/opt**.

Você pode colocar o conteúdo onde achar melhor, pois o que vai interessar mesmo é poder setar o caminho para a SDK com uma variável de ambiente.



**NOTA:** A pasta descompactada tem o nome de **tools** apenas. Para o nosso exemplo descompactamos essa pasta dentro de **/opt/android-sdk-linux**.

### 2.2.1 Instalando pacotes com o *sdkmanager*

Só baixar a SDK do Android não será suficiente, é preciso instalar alguns componentes que permitirão o build da aplicação. Esses pacotes são **build-tools**, **platforms** e **platform-tools**.

Para instalarmos esses pacotes faremos a utilização do utilitário **sdkmanager**, ele será o responsável por gerenciar os pacotes e licenças pertinentes ao desenvolvimento de aplicações Android.

O **sdkmanager** poderá ser encontrado no diretório **tools/bin**. Para a instalação de pacotes basta que se execute um comando semelhante ao que é mostrado abaixo:

```
./sdkmanager "platform-tools"
```

O comando acima deve ser executado dentro do diretório **tools/bin**. Esse comando fará a instalação do **platform-tools**. Basicamente você deverá executar o **sdkmanager** e entre aspas o pacote que deseja instalar.

Para a instalação de todos os componentes que precisamos basta executar o comando:

```
./sdkmanager "platform-tools" "platforms;android-28" "build-  
tools;29.0.0"
```

Após a execução, alguns diretórios serão criados dentro da pasta da sua SDK. As pastas são **platforms**, **platform-tools** e **build-tools**. Essas pastas devem ter subpastas de suas respectivas versões.

Para mais informações sobre o **sdkmanager**, acesse:

<https://developer.android.com/studio/command-line/sdkmanager>.

### 2.3 Variáveis de ambiente

As variáveis de ambiente vão permitir que nossa aplicação **ionic**, no momento do build, encontre as ferramentas necessárias para o processo. Nós precisaremos basicamente de duas variáveis principais, a **JAVA\_HOME**, **ANDROID\_HOME** e **ANDROID\_SDK\_HOME**. Essas duas variáveis devem somente apontar para a instalação Java e SDK do Android respectivamente.

**NOTA:** **ANDROID\_HOME** está caindo em desuso e a nova forma de definir o caminho para a SDK é com a variável **ANDROID\_SDK\_HOME**. Mas para não ter problemas, defina as duas, assim funcionará de um jeito ou de outro.

Antes de fazer qualquer coisa cheque se as variáveis já existem. Você pode fazer isso com o comando:

```
echo $JAVA_HOME
```

Use o mesmo comando para as outras. Se aparecer o conteúdo então é porque já está definida. Se o caminho estiver correto, pode ir direto para o capítulo 3. Caso não estejam definidas ou com caminho incorreto precisaremos definir nós mesmos.

### No Linux e no Mac

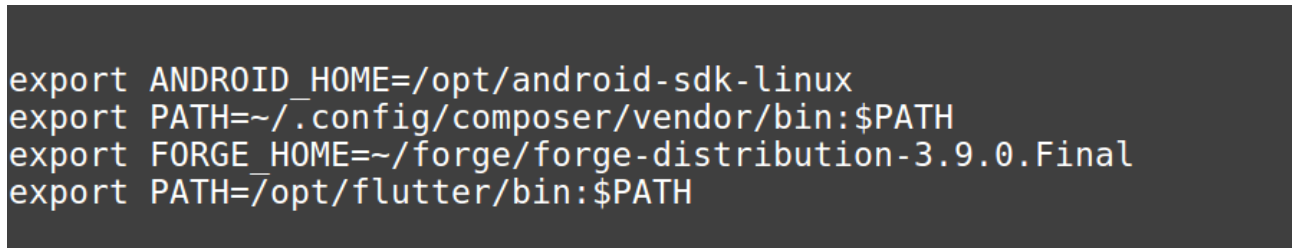
Uma forma bem simples de definir essas variáveis nessas duas plataformas é com o comando **export**, que serve justamente para este fim. A criação da ANDROID\_HOME seria parecido com:

```
export ANDROID_HOME=/opt/android-sdk-linux
```

Onde “/opt/android-sdk-linux” é o caminho para a pasta descompactada que foi baixada na seção anterior. Esse comando será suficiente para a criação da sua variável de ambiente.

No entanto, só executar este comando no terminal não vai permitir que essa variável persista. Ou seja, você sempre terá de recriá-la. Uma forma de contornar isso é adicionando este mesmo comando ao arquivo **.bashrc** no caso do Linux ou **.bash\_profile** no caso do Mac. Este arquivo ficará na sua pasta de usuário. No meu caso é **/home/edy**.

A imagem abaixo mostra um trecho do arquivo **.bashrc** que estamos utilizando:

A imagem mostra um terminal com o seguinte conteúdo: export ANDROID\_HOME=/opt/android-sdk-linux, export PATH=~/.config/composer/vendor/bin:\$PATH, export FORGE\_HOME=~/.forge/forge-distribution-3.9.0.Final, export PATH=/opt/flutter/bin:\$PATH. O fundo é escuro e o texto é branco.

```
export ANDROID_HOME=/opt/android-sdk-linux
export PATH=~/.config/composer/vendor/bin:$PATH
export FORGE_HOME=~/.forge/forge-distribution-3.9.0.Final
export PATH=/opt/flutter/bin:$PATH
```

Imagem 04 - Trecho do arquivo **.bashrc**

### No Windows

Aqui o processo será bem mais simples. Com alguns cliques as variáveis já estarão criadas. Vá ao menu iniciar e procure por **Variáveis do Usuário**. Você verá uma janela semelhante a esta:

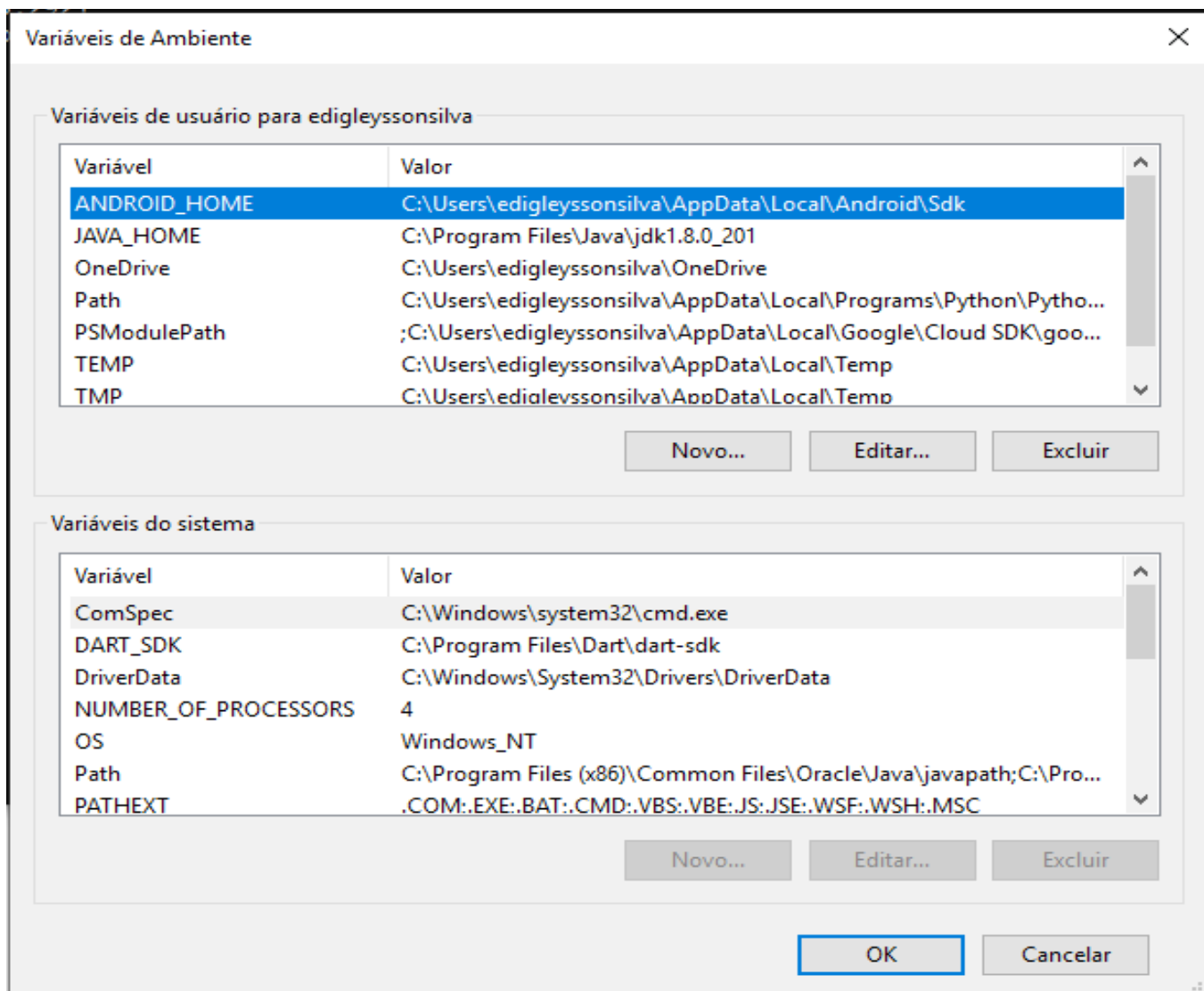


Imagem 05- Variáveis de ambiente no Windows

Clique em **ADICIONAR** e defina o valor da variável, que no caso é o caminho para a instalação do Android, no caso da ANDROID\_HOME. Veja como fica:

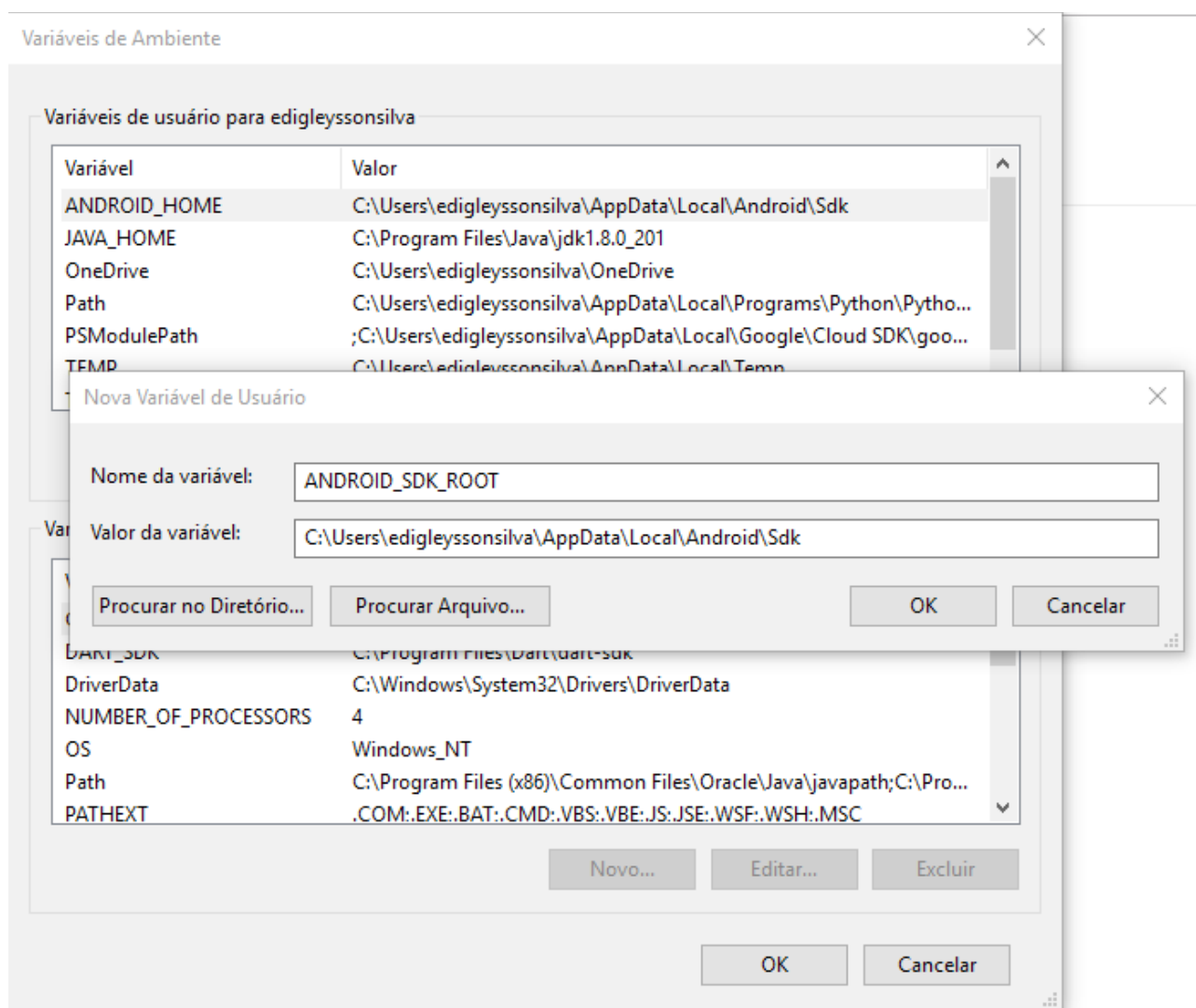


Imagem 06 - Setando a variável ANDROID\_SDK\_ROOT

## Capítulo 3 – Configurando o app

Pronto! Nosso ambiente está preparado. Foi bem cansativo não é? Sabemos que isso realmente é um pouco mais complicado, mas o lado bom é que você não terá que fazer isso sempre. Você já fez uma vez, de agora em diante só deve se preocupar com o seu app.

Portanto, chegamos no momento de configurar nossa aplicação para o build. Vamos definir alguns metadados e ícones para que tudo fique do nosso jeito.

### 3.1 Metadados da aplicação no config.xml

Nossa aplicação deve ser única e deve possuir a nossa cara. Portanto é necessário definir algumas informações que farão com isso aconteça. Informações como, **nome**, **versão**, **descrição**, etc.

Todas essas características poderão ser definidas no arquivo **config.xml** fica na raiz do projeto Ionic. Abaixo há uma listagem de alguns parâmetros:

- widget[id] – Identificador do app em domínio reverso. Algo como io.github.codesilva.
- widget[version] – Versão do app em notação **major/minor/patch** (Veja mais em <https://semver.org/>).
- author
  - author[email] – Email do autor
  - author[href] – Website do autor
- name – Essa tag contém o nome formal da sua aplicação como, WhatsApp, Telegram, etc.
- description – Nessa tag você deve adicionar uma descrição da sua aplicação. Essa descrição será exibida nas lojas.

Esses metadados são os mais comuns. Há algumas outras tags que podem ser utilizadas no config.xml. Algumas delas vão variar dependendo de que plugins existem na sua aplicação. Veja a lista completa de parâmetros no link [https://cordova.apache.org/docs/en/latest/config\\_ref/](https://cordova.apache.org/docs/en/latest/config_ref/).

### 3.2 Setando os ícones da aplicação

Após setar os metadados da sua aplicação é hora de definir os ícones da aplicação. Afinal uma imagem fala mais que mil palavras e é com esse recurso que o seu app será reconhecido entre tantos outros em uma loja.

Sobre os recursos de imagens temos no Ionic dois tipos, Ícones e Splashscreens. Ícone será a imagem exibida na Activity, na loja e no lançador do sistema operacional. Splashscreen é a imagem que será exibida no carregamento do seu app. Ele será exibida em quanto sua aplicação inicia.

Sua aplicação deve estar apta a executar em dispositivos de tamanhos variados. Portanto

é preciso que você tenha imagens em tamanhos distintos.

Você pode gerar todas as imagens de forma muito simples e rápida com o comando **ionic cordova resources**. Esse comando pega duas imagens de referência e as replica em imagens com as dimensões que são necessárias. Porém, para que isso possa ser feito as imagens devem atender aos seguintes critérios:

- A imagem de origem para os ícones deve estar em **resources/icon.png** e deve ter dimensão mínima de **1024x1024px**
- A imagem de origem para a splashscreen deve estar localizada em **resources/splash.png** e deve ter dimensão mínima de **2732x2732px**.

Ao criar o projeto com o comando **ionic start** é possível ver as imagens padrão no diretório **resources**.

Após atender aos critérios exigidos execute o comando abaixo para gerar as suas imagens:

**ionic cordova resources**

É possível fazer algumas variações nesse comando. Como aqui estamos trabalhando apenas para a plataforma Android podemos especificar a plataforma que desejamos com a variação do comando ficando da seguinte forma:

**ionic cordova resources android**

Há também outras variações que você pode ver em <https://ionicframework.com/docs/cli/commands/cordova-resources>

### 3.2.1 Android Assets Icon

Para encerrar o capítulo gostaria de lhe mostrar essa ferramenta bem interessante e que pode ser muito útil no tratamento das imagens e até dos ícones da sua aplicação.

Dentro da IDE do Android Studio existe uma ferramenta chamada Android Asset. Essa ferramenta permite a criação e edição de ícones. No entanto não estamos utilizando a IDE aqui.

Porém, existe uma ferramenta que permite que façamos a mesma coisa em uma página web. É a ferramenta **Android Asset Studio**, veja no link <https://romannurik.github.io/AndroidAssetStudio/>.

## Capítulo 4 – APK em Debug e em Release

Nos preparamos para chegar nesse momento. É hora de finalmente gerarmos o arquivo final, aquele que será instalado nos dispositivos, o arquivo **APK**.

Antes de tudo precisamos entender que há duas formas de build e ao passo que formos entendendo cada uma delas vamos também gerar o APK.

### 4.1 Debug

O build em modo debug é exclusivamente para fazer a depuração da aplicação. Onde você poderá testar em um emulador ou em um dispositivo real. Portanto essa modalidade possui algumas aberturas que lhe permitirão vários testes com logs e depurações.

#### Gerando APK em Debug

Para gerar uma APK em debug no Ionic basta que se execute o seguinte comando:

**ionic cordova build android**

O processo de build da aplicação ocorrerá normalmente e você terá um output semelhante ao que é mostrado abaixo.

```
> Task :app:compileDebugSources
> Task :app:mergeDebugShaders
> Task :app:compileDebugShaders
> Task :app:generateDebugAssets
> Task :CordovaLib:mergeDebugShaders
> Task :CordovaLib:compileDebugShaders
> Task :CordovaLib:generateDebugAssets
> Task :CordovaLib:packageDebugAssets
> Task :app:mergeDebugAssets
> Task :app:validateSigningDebug
> Task :app:signingConfigWriterDebug
> Task :app:transformClassesWithDexBuilderForDebug
> Task :app:transformDexArchiveWithExternalLibsDexMergerForDebug
> Task :app:transformDexArchiveWithDexMergerForDebug
> Task :app:mergeDebugJniLibFolders
> Task :CordovaLib:compileDebugNdk NO-SOURCE
> Task :CordovaLib:mergeDebugJniLibFolders
> Task :CordovaLib:transformNativeLibsWithMergeJniLibsForDebug
> Task :CordovaLib:transformNativeLibsWithIntermediateJniLibsForDebug
> Task :app:transformNativeLibsWithMergeJniLibsForDebug
> Task :app:processDebugJavaRes NO-SOURCE
> Task :app:transformResourcesWithMergeJavaResForDebug
> Task :app:packageDebug
> Task :app:assembleDebug
> Task :app:cdvBuildDebug

BUILD SUCCESSFUL in 8m 45s
42 actionable tasks: 42 executed
Built the following apk(s):
/home/edy/workspaces/tutoriais/ionic/EbookApp/platforms/android/app/build/outputs/apk/debug/app-debug.apk
```

Lembra que falamos do Gradle no capítulo 1? Você deve ter notado que não fizemos nenhuma instalação desse componente. Pois bem, não se faz necessário instalar o Gradle, pois assim que você executar o comando de build, uma versão do Gradle será baixada e será utilizada no build da aplicação.

### 4.2 Release

A versão release é aquela que vai para loja, que no caso aqui é a Play Store. Essa versão

é otimizada e assinada com uma chave própria, o que garante uma melhor performance e segurança da aplicação.

Diferente da modalidade debug, a release não permite depuração. Gerar uma APK em release requer alguns passos a mais e por isso preferimos dedicar um capítulo somente para isso.

**NOTA:** Atualmente a Play Store está exigindo que se utilize o Android App Bundle (**.aab**). Se quiser passar direto para esta parte, vá para o **Apêndice A**, onde tratamos sobre esse assunto.



## Capítulo 5 – Build APK em Release

No capítulo anterior fizemos o build de uma APK em debug e você viu que é muito simples. Agora vamos percorrer o caminho necessário para a geração do APK em release.

### 5.1 Gerando a APK

O comando para a geração da APK vai ser semelhante ao que mostramos no capítulo anterior. Na verdade é o mesmo, só que com algumas opções a mais. O comando para a geração da APK em release será o seguinte:

```
ionic cordova build android --prod --release
```

O output será semelhante ao que foi mostrado no capítulo anterior. Com a diferença de que a APK gerada estará em outro diretório, que será mostrado ao final.

### 5.2 Assinando a APK

A APK gerada precisa ser assinada e para isso você precisa ter uma chave. Assim é preciso que se tenha uma chave, vamos então criar uma chave.

#### 5.2.1 Criando a chave

Para a criação da chave nós utilizaremos o comando abaixo:

```
keytool -genkey -v -keystore my-release-key.keystore -alias  
alias_name -keyalg RSA -keysize 2048 -validity 10000
```

O comando acima criará a chave **my-release-key.keystore**. Com algumas características como algoritmo, tamanho e validade. Durante a criação serão solicitadas algumas informações, uma delas é a senha da sua chave. Preencha todas as informações e a chave será criada com sucesso.

Não vamos nos estender muito sobre este comando, mas você pode ter mais informações em

#### 5.2.2 Assinando com jarsigner

Agora que temos uma chave, podemos então fazer a assinatura da nossa aplicação com o seguinte comando:

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore  
my-release-key.keystore  
platforms/android/app/build/outputs/apk/release/app-release-  
unsigned.apk alias_name
```

Note que passamos tanto a nossa chave, quando a nossa APK não assinada. É importante comentar que este comando foi executado na raiz do nosso projeto Ionic que é

onde está a nossa chave. Dependendo de onde o comando é executado e de onde encontram-se a chave e a APK esse comando pode ter variações.

Pronto, nossa APK está quase liberada para a publicação na Play Store.

### 5.3 Otimização

Finalmente vamos otimizar nossa APK e faremos isso com as ferramentas de build que estão no diretório da nossa SDK do android na pasta **build-tools**. Acesse a pasta na versão que tiver disponível e faça a utilização da ferramenta **zipalign**. O comando será semelhante ao que é mostrado abaixo.

```
zipalign -v 4 HelloWorld-release-unsigned.apk HelloWorld.apk
```

Em nosso caso mudamos o diretório para a APK, que ainda está no mesmo diretório que foi mostrado na seção anterior, e executamos o comando zipalign da seguinte forma:

```
/opt/android-sdk-linux/build-tools/28.0.3/zipalign -v 4 app-release-unsigned.apk ebook-app.apk
```

Veja que passamos todo o caminho para o zipalign. O resto do comando é normal, onde passamos a APK de entrada e a APK de saída.

Em nosso caso o arquivo **.apk** gerado foi **ebook-app.apk**.

## APÊNDICE A - Android App Bundle

O intuito de uma aplicação é ser compartilhada com o mundo. É bacana ver pessoas utilizando aplicativos que foram desenvolvidos por você. Então, um passo além deste ebook é fazer a publicação do seu projeto na Play Store.

A Google adotou um novo formato de upload de apps e tudo caminha para que este seja, pelo menos por um tempo, o modo padrão de publicação. Este novo formato é o Android App Bundle. Uma das principais características é o desenvolvimento modular.

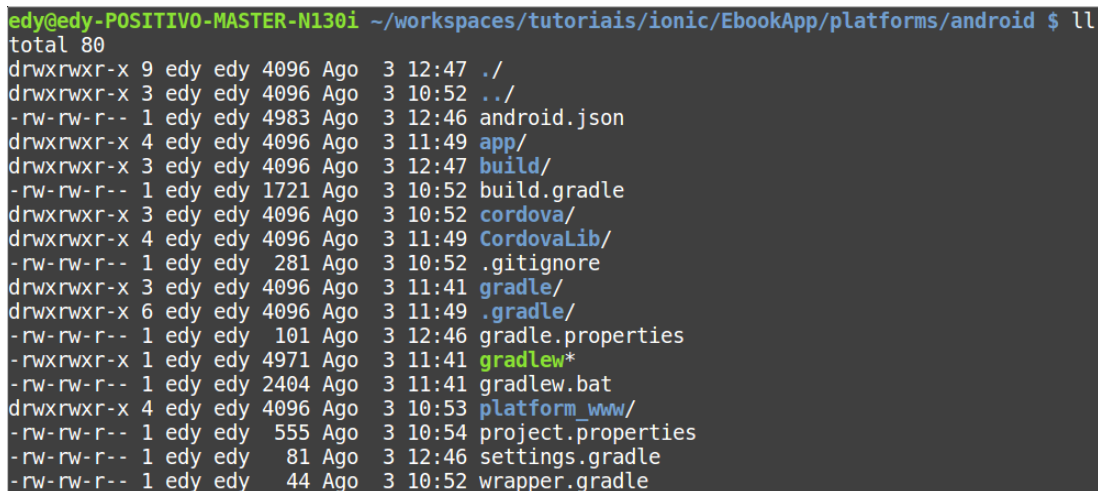
Veja mais informações sobre esse novo método no link <https://developer.android.com/platform/technology/app-bundle>.

A pergunta que vem é: **Como gerar o Android App Bundle do meu projeto Ionic sem o Android Studio?**

Bom, chegamos até aqui sem ele e vamos continuar. Vamos então gerar o arquivo **Android App Bundle (.aab)** da nossa aplicação Ionic.

### Acessando o Gradlew

Bom, anteriormente vimos que ao executar o comando de build uma versão do Gradle é baixada. O gradle baixado estará no subdiretório que fica na raiz do seu app **platforms/android**. Em nosso caso fica em **/home/edy/workspaces/tutoriais/ionic/EbookApp/platforms/android**. Se você executar um comando para listar os arquivos dessa pasta poderá ver que os arquivos do gradle, como mostra a imagem abaixo.



```
edy@edy-POSITIVO-MASTER-N130i ~/workspaces/tutoriais/ionic/EbookApp/platforms/android $ ll
total 80
drwxrwxr-x 9 edy edy 4096 Ago 3 12:47 ./
drwxrwxr-x 3 edy edy 4096 Ago 3 10:52 ../
-rw-rw-r-- 1 edy edy 4983 Ago 3 12:46 android.json
drwxrwxr-x 4 edy edy 4096 Ago 3 11:49 app/
drwxrwxr-x 3 edy edy 4096 Ago 3 12:47 build/
-rw-rw-r-- 1 edy edy 1721 Ago 3 10:52 build.gradle
drwxrwxr-x 3 edy edy 4096 Ago 3 10:52 cordova/
drwxrwxr-x 4 edy edy 4096 Ago 3 11:49 CordovaLib/
-rw-rw-r-- 1 edy edy 281 Ago 3 10:52 .gitignore
drwxrwxr-x 3 edy edy 4096 Ago 3 11:41 gradle/
drwxrwxr-x 6 edy edy 4096 Ago 3 11:49 .gradle/
-rw-rw-r-- 1 edy edy 101 Ago 3 12:46 gradle.properties
-rwxrwxr-x 1 edy edy 4971 Ago 3 11:41 gradlew*
-rw-rw-r-- 1 edy edy 2404 Ago 3 11:41 gradlew.bat
drwxrwxr-x 4 edy edy 4096 Ago 3 10:53 platform_www/
-rw-rw-r-- 1 edy edy 555 Ago 3 10:54 project.properties
-rw-rw-r-- 1 edy edy 81 Ago 3 12:46 settings.gradle
-rw-rw-r-- 1 edy edy 44 Ago 3 10:52 wrapper.gradle
```

O arquivo importante aqui será o **gradlew**, que é utilizado para compilação com wrapper. Veja mais informações sobre Gradle Wrapper no link [https://docs.gradle.org/current/userguide/gradle\\_wrapper.html](https://docs.gradle.org/current/userguide/gradle_wrapper.html)

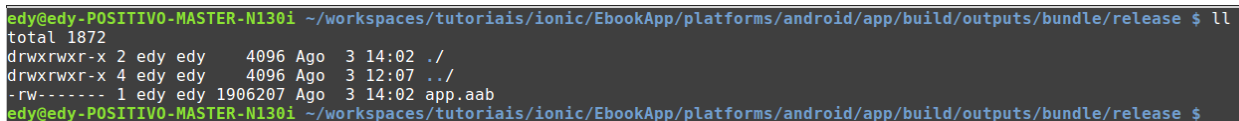
Esse arquivo vai ser o que vamos utilizar para gerar o nosso Android App Bundle. E por incrível que pareça isso será muito fácil.

Ainda no diretório onde se encontra o gradlew execute o seguinte comando:

```
./gradlew bundle
```

A execução será semelhante ao build comum e ao final dela o arquivo **.aab** será gerado. O arquivo gerado poderá ser encontrado no diretório **platforms/android/app/build/outputs/bundle/release/app.aab**.

Em nosso caso o arquivo gerado está no diretório **/home/edy/workspaces/tutoriais/ionic/EbookApp/platforms/android/app/build/outputs/bundle/release/app.aab**. Veja a imagem abaixo.



```
edy@edy-POSITIVO-MASTER-N130i ~/workspaces/tutoriais/ionic/EbookApp/platforms/android/app/build/outputs/bundle/release $ ll
total 1872
drwxrwxr-x 2 edy edy 4096 Ago 3 14:02 ./
drwxrwxr-x 4 edy edy 4096 Ago 3 12:07 ../
-rw-r----- 1 edy edy 1906207 Ago 3 14:02 app.aab
edy@edy-POSITIVO-MASTER-N130i ~/workspaces/tutoriais/ionic/EbookApp/platforms/android/app/build/outputs/bundle/release $
```

Para assinar seu Android App Bundle você utilizará o mesmo comando que foi mostrado no Capítulo 5.

## REFERÊNCIAS

1. Android Developers - <https://developer.android.com/studio/command-line/sdkmanager>
2. Chaves e assinatura - <https://stablekernel.com/creating-keystores-and-signing-android-apps>
3. Documentação Ionic - <https://ionicframework.com/docs/publishing/play-store>
4. Issue sobre Android App Bundle - <https://github.com/apache/cordova-android/issues/729/>