

devfest



```
// You'll need  
// com.google.  
ListRef.listAl  
.addOn  
pixels.toList  
// All  
// You  
}  
it  
each { item  
the items  
}
```

Ten Years into Android Gradle Plugin

El Zhang @2BAB / Android GDE / Singapore

 Google Developer Groups
Taipei

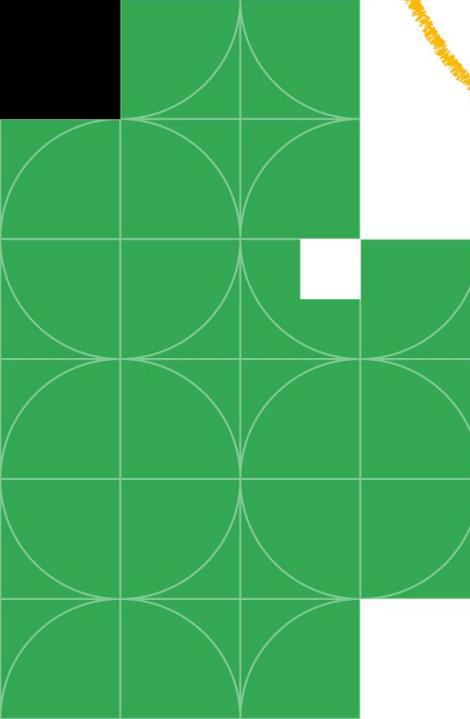


```
text  
  'Section Title',  
  style: TextStyle(  
    color: Colors.green[200],  
  ),  
,
```

devfest

```
s.star,  
r: Colors.green[500],
```

```
Text('23'),
```

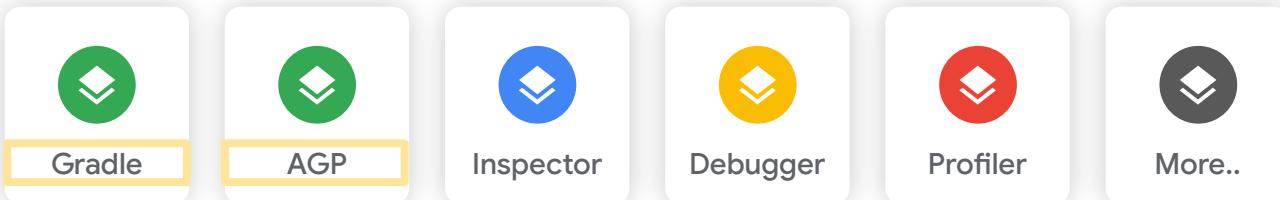


Google Developer Groups

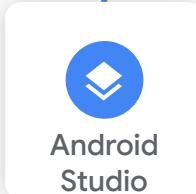
Android Gradle Plugin (AGP) and Android Studio (AS)

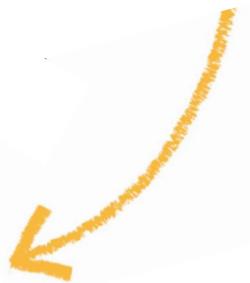
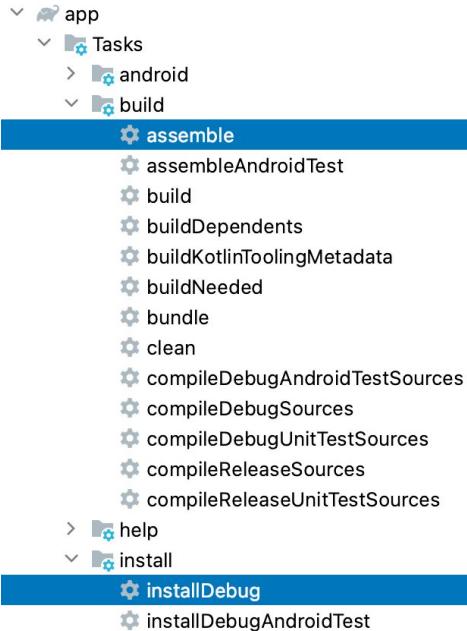
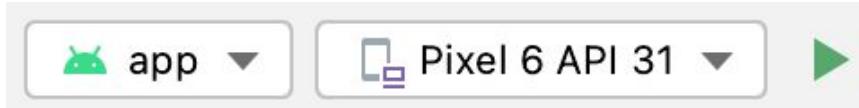
Android Studio - Features

Background Services



UI / Console





android gradle plugin



How about CI/CD? What should we expect from AGP and Gradle?



android gradle plugin



Extended use cases of the Gradle and AGP



Business-as-usual

Triple-T/gradle-play-publisher

GPP is Android's unofficial release automation Gradle Plugin. It can do anything from building, uploading, and then promoting your App Bundle or APK to publishing app listings and other metadata.

Kotlin 4.0k 342



Architecture

2BAB/Seal

A Gradle Plugin for resolving AndroidManifest.xml merge conflicts.

Kotlin 228 23



Troubleshooting

akaita/easylauncher-gradle-plugin

Add a different ribbon to each of your Android app variants using this gradle plugin. Of course, configure it as you will

Java 1.0k 55

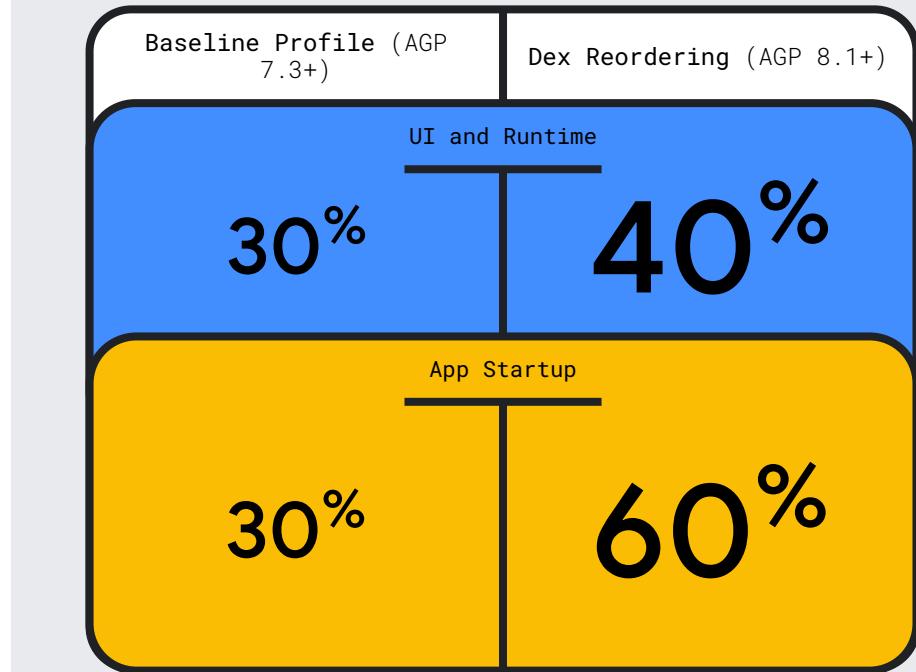


Baseline Profiles

Baseline profiles further improves startup time and overall UI performance with R8 and Dex reordering. Integrating them requires latest AGP with a bunch of advanced configurations.

goo.gle/android-perf-news-io23

* Google internal research data. Note that this does not affect network and non-code disk access



Productivity boosts

By adapting to the Configuration Cache, Square's Developer Experience (DX) team has reclaimed an estimated \$1.1 million in annual productivity losses.



DECEMBER 13TH, 2022 | 3 MINUTE READ

Saving 5,400 hours a year with Gradle's Configuration Cache

Recovering an estimated \$1.1 million in lost productivity annually

The code we don't understand

1. Limited understanding of the underlying API in Groovy DSL, including source code navigation challenges.
2. Uncertainty about modifying AGP intermediates and finding intervention points.
3. Lack of clarity on where to place the logic or a new Gradle plugin.



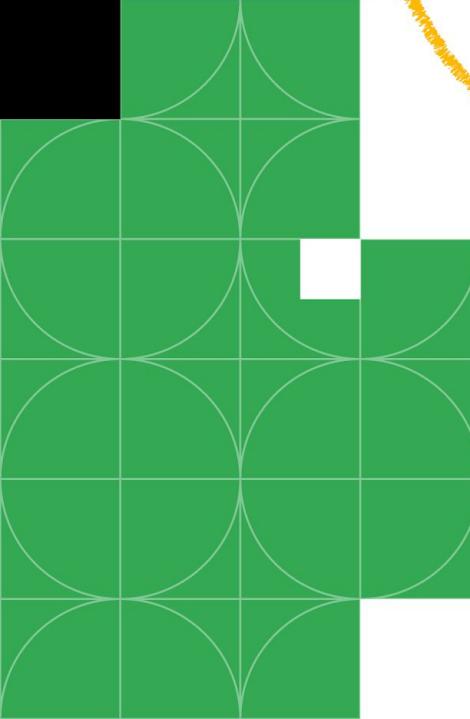
Copy, paste, and it works like a charm!

```
text  
  'Section Title',  
  style: TextStyle(  
    color: Colors.green[200],  
  ),  
,
```

devfest

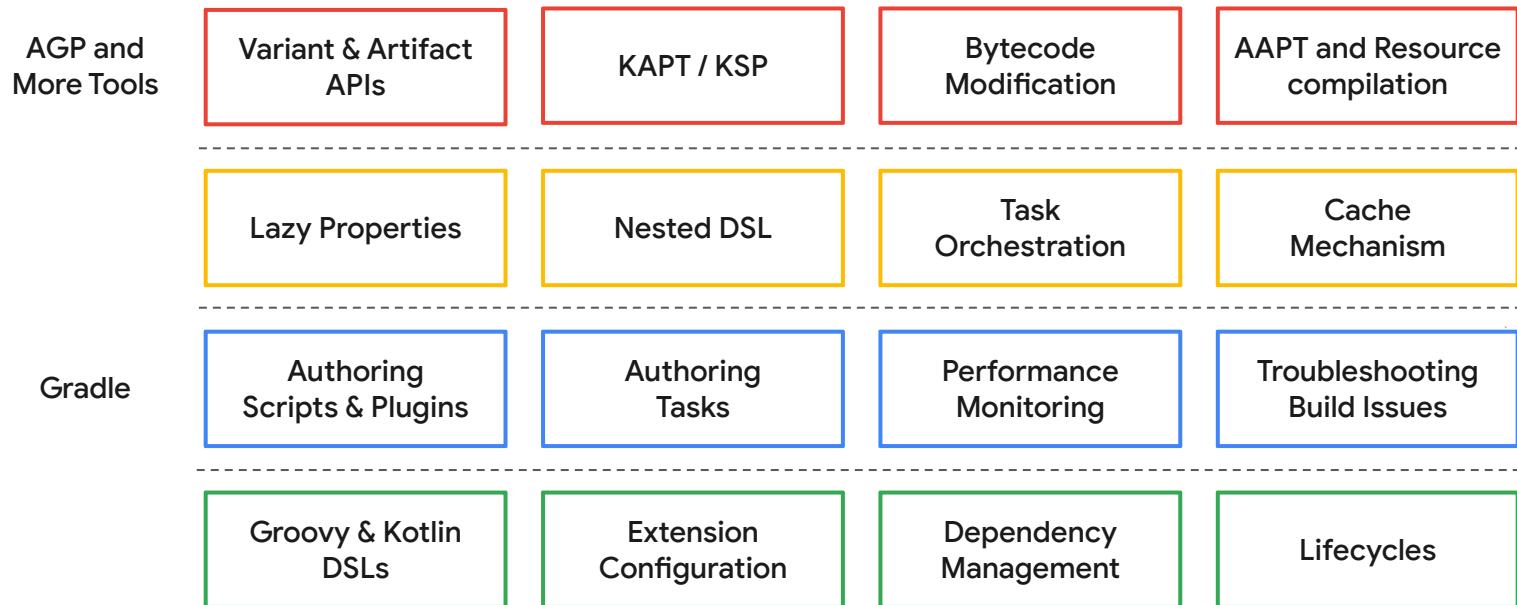
```
s.star,  
r: Colors.green[500],
```

```
Text('23'),
```



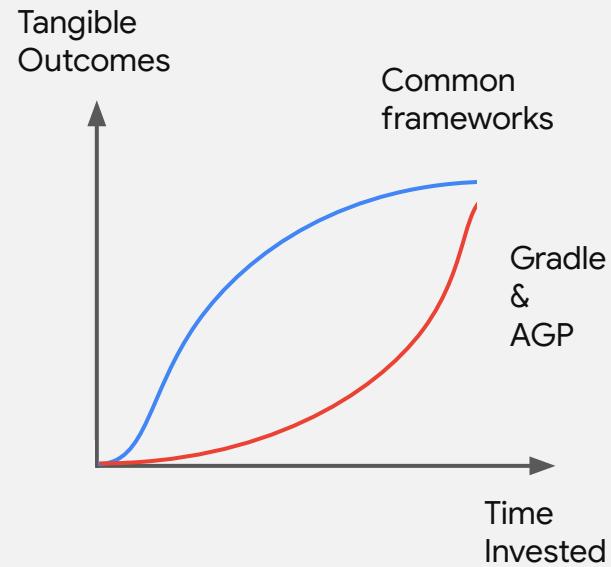
Obstacles on Gradle and AGP Learning Paths

Knowledge Graph of Gradle and AGP



Investment & Returns

Gradle and AGP typically require a greater investment of time to yield corresponding returns. They cover two stages: learning the configuration and customized plugin.



Groovy DSL and Raw API



Unexpected: Groovy DSL make it hard to raw APIs.



Expected: List out the comparison of DSL and raw API differences, especially for Gradle managed properties/classes.

The screenshot shows an IDE interface with two code snippets. The top snippet is a Groovy DSL configuration for an Android application:

```
android {  
    compileSdkVersion 30  
    buildToolsVersion "30.0.3"  
}
```

A red tooltip box with the text "Cannot find declaration to go to" points to the word "compile". A numbered callout "①" is positioned to the right of the code. The bottom snippet shows a Groovy DSL configuration for build types:

```
buildTypes { this: NamedDomainObjectContainer<ApplicationBuildType>  
    getByName( name: "debug" ) { this: ApplicationBuildType  
        isMinifyEnabled = false  
    }  
}
```

A numbered callout "②" is positioned to the right of the code. At the bottom of the screenshot, there is a yellow-bordered box containing the text "Since 2010".

Lack of Tutorial



Unexpected: Simple API / Java docs are provided, however **tutorials** are not.



Expected: More use cases and comparison in both Groovy and Kotlin languages to be friendly for beginners.

```
buildTypes { this: NamedDomainObjectContainer<ApplicationBuildType>
    getByName( name: "debug" ) { this: ApplicationBuildType
        isMinifyEnabled = false
    }
}
```

Package org.gradle.api

Interface **NamedDomainObjectContainer<T>**

Type Parameters:

T - The type of objects in this container.

All Superinterfaces:

java.util.Collection<T>, Configurable<NamedDomainObjectContainer<T>>

All Known Subinterfaces:

ArtifactTypeContainer, AuthenticationContainer, BuildTypeContainer, C
MutableVersionCatalogContainer, NativeToolChainRegistry, PlatformCont

Since 2010

Lack of Tutorial



Unexpected: Simple API / Java docs are provided, however **tutorials** are not.



Expected: More use cases and comparison in both Groovy and Kotlin languages to be friendly for beginners.

```
buildTypes { this: NamedDomainObjectContainer<ApplicationBuildType>
    getByName( name: "debug" ) { this: ApplicationBuildType
        isMinifyEnabled = false
    }
}
```

Example 8. Managing a collection of objects

DownloadExtension.java

```
public interface DownloadExtension {
    NamedDomainObjectContainer<Resource> getResources();
}

public interface Resource {
    // Type must have a read-only 'name' property
    String getName();

    Property<URI> getUri();

    Property<String> getUserName();
}
```

Since 2019

Lack of Extensibility



Unexpected: Challenging to engage with the intermediary products of AGP.



Expected: Revealing as many APIs as possible to meet the requirements of developers.



Build Tools for *FrontEnd

* 'Frontend' refers to all client interfaces that interact directly with end users.

Lack of Extensibility



Unexpected: Using a **Task** as an API to **connect two plugins** is not ideal, as it requires **awareness** of the other's **implementation**.



Expected: The public API should be separate from its internal implementation.

```
processManifestTask.doLast( actionName: "postUpdateAction" ) { this: Task<Unit>
    val task = this as ProcessApplicationManifest
    val targetValue = task.inputs.properties["replace_value"].toString()
    val modifiedManifest = task.mergedManifest.get().asFile.readText()
        .replace( oldValue: "allowBackup=\"true\"", targetValue )
    task.mergedManifest.get().asFile.writeText(modifiedManifest)
}
```

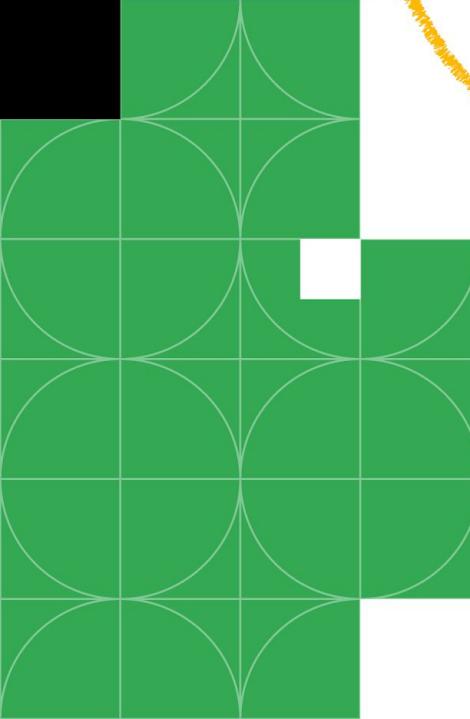
TaskA.doLast{...}

```
text  
  'Section Title',  
  style: TextStyle(  
    color: Colors.green[200],  
  ),  
,
```

devfest

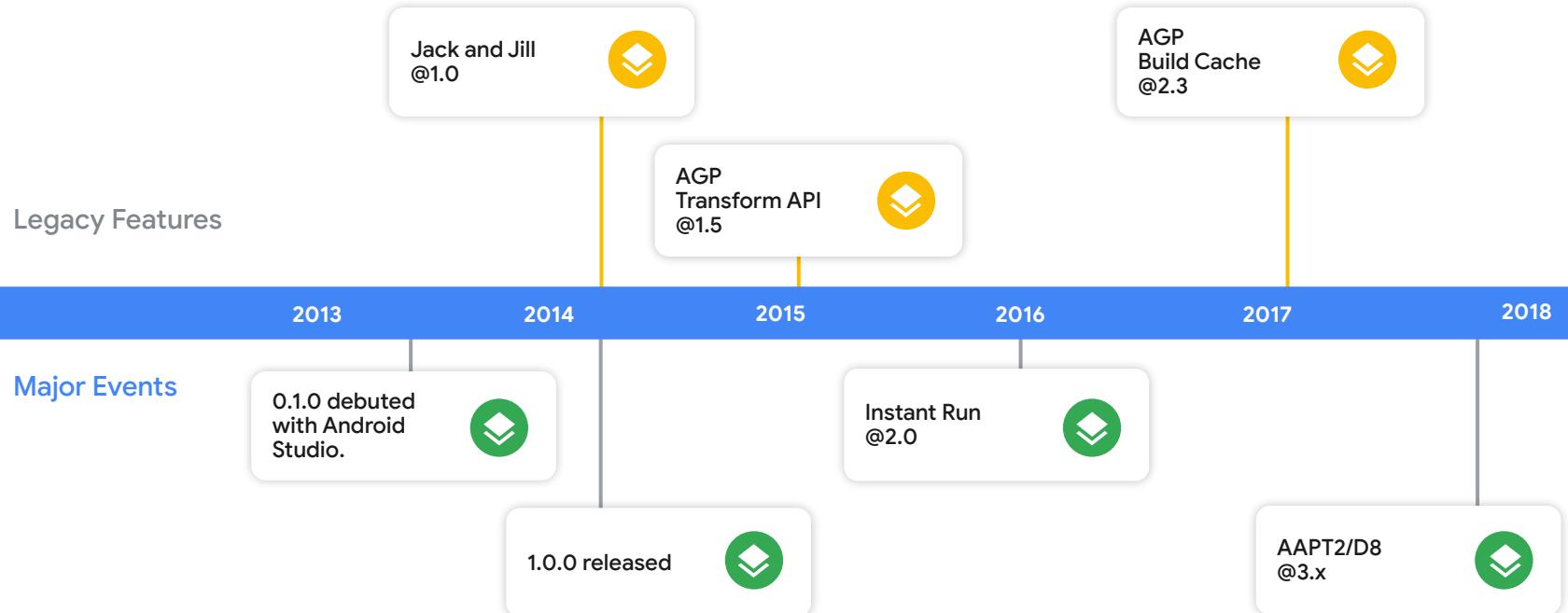
```
s.star,  
r: Colors.green[500],
```

```
Text('23'),
```

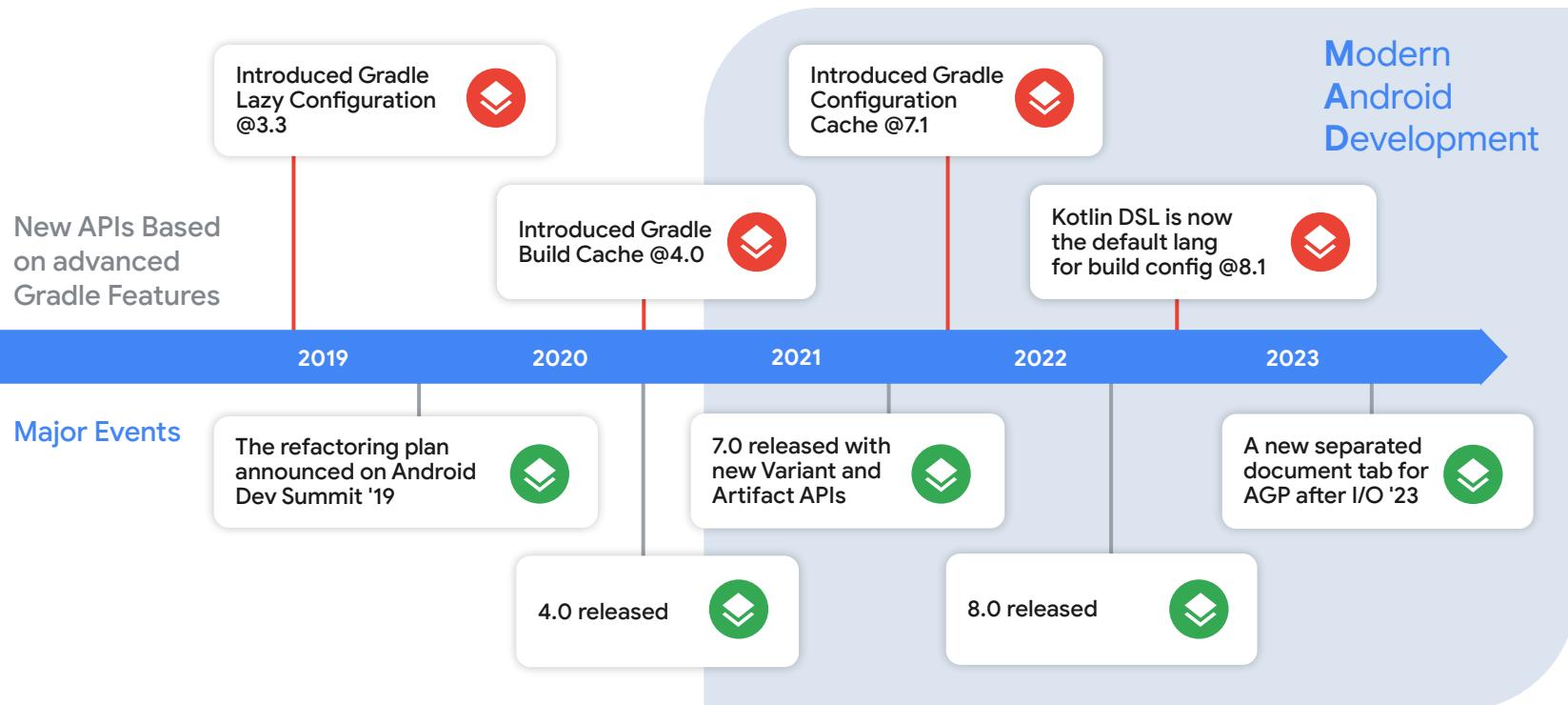


Evolution of Android Gradle Plugin

Milestones of AGP 2013-2018



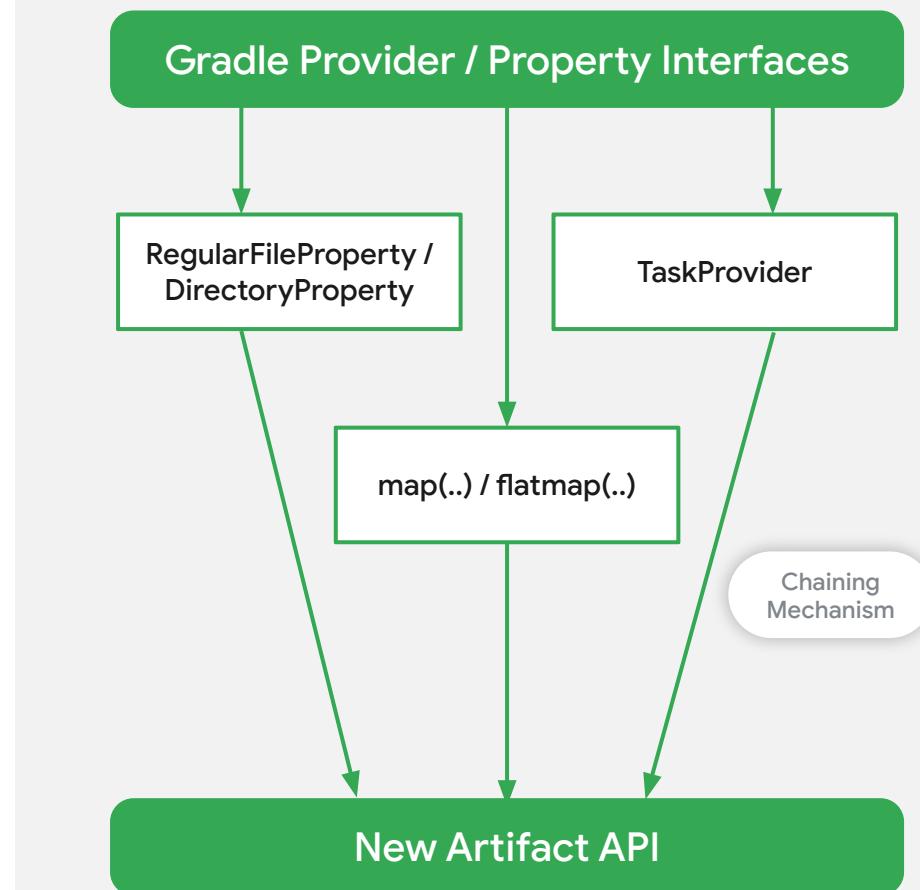
Milestones of AGP 2019-2023



Lazy Configuration



It's the cornerstone of the modern AGP architecture, offering **separation of focus** for **read and write operations** and decoupling direct dependencies between Tasks.



New Artifact APIs



A series of standardized public APIs for third-party developers, eliminating the dependency on Task API and allowing for sustainable expansion.

```
val postUpdateTask = project.tasks.register<ManifestAfterMergeTask>(  
    name: "postUpdate${variantCapitalizedName}Manifest")  
  
variant.artifacts Artifacts  
    .use(postUpdateTask) TaskBasedOperation<ManifestAfterMergeTask>  
    .wiredWithFiles(  
        ManifestAfterMergeTask::mergedManifest,  
        ManifestAfterMergeTask::updatedManifest  
    ) InAndOutFileOperationRequest  
    .toTransform(SingleArtifact.MERGED_MANIFEST)
```

toCreate(..) / toAppendTo(..) / toTransform(..)

* Submit your use case for the proposed Artifact to the issue tracker using the following paths.

Android Public Tracker > App Development > Android Studio >

Kotlin is Default



1. For scripts, it offers enhanced **auto-completion** and **source navigation** in Android Studio, along with **accurate and understandable syntax**.
2. For binary plugins, it offers better compatibility assurances.

The screenshot shows a blog post from the Android Developers Blog. The title is "Kotlin DSL is Now the Default for New Gradle Builds". The post was published on April 13, 2023, by Paul Merlin. It discusses the introduction of Kotlin DSL in version 3.0 of the Gradle Build Tool in August 2016, its evolution to version 1.0 in Gradle 5.0, and its current status as the default build language. The post highlights the benefits of Kotlin DSL, such as improved productivity and maintainability, and its integration with other tools like IntelliJ IDEA and Android Studio. It also mentions the continued support for Groovy DSL for existing projects. The page includes social sharing icons (Twitter, Facebook, LinkedIn, Email, Link) and a sidebar for the Jet Brains blog.

Kotlin DSL is Now the Default for New Gradle Builds

April 13, 2023 | Paul Merlin | General

Kotlin DSL for Gradle was introduced in version 3.0 of the Gradle Build Tool in August 2016 and released as 1.0 in Gradle 5.0. Since then, it's been growing in popularity and has greatly improved the authoring experience of many Gradle builds.

Kotlin DSL is now the default choice for new Gradle builds. This means that when creating a new project with Gradle, including in IntelliJ IDEA (starting with 2023.1) and Android Studio (starting with Giraffe), Kotlin DSL is the default option. Support for Groovy DSL will continue for existing projects or those who prefer to use it.

In this post, we will explore the benefits of Kotlin DSL and why it is becoming the recommended option for new Gradle builds. We will also discuss some of the improvements that are planned for the future to make Kotlin DSL even better.

Posted by [James Ward](#), Product Manager, Kotlin and [Boris Farber](#), Developer Relations Engineer

Android has been Kotlin-first for four years and many Android developers have made the switch resulting in higher productivity and more stable apps. However the default language to define builds has been Groovy (`build.gradle`), even though a Kotlin (`build.gradle.kts`) option has existed in Gradle for a number of years.

Anton Yalyshev
April 13, 2023

Read this post in other languages:
Français, 日本語, 한국어, 简体中文

We always strive to help developers write better-structured and more maintainable builds. Given this aim, applying Kotlin to writing Gradle build

Kotlin is Default



1. For scripts, it offers enhanced **auto-completion** and **source navigation** in Android Studio, along with **accurate and understandable syntax**.
2. For binary plugins, it offers better compatibility assurances.

```
buildTypes {  
    ① getByName("debug") {  
        isMinifyEnabled = false  
    }  
    getByName("release") {  
        isMinifyEnabled = false  
    }  
    ② create("staging") {  
        isMinifyEnabled = false  
    }  
    // register("staging") {  
    //     isMinifyEnabled = false  
    // }  
}
```

Kotlin is Default



1. For scripts, it offers enhanced auto-completion and source navigation in Android Studio, along with accurate and understandable syntax.
2. For binary plugins, it offers **better compatibility assurances**.

The recommendation to use a statically-typed language is independent from the language choice for writing tests for your plugin code. The use of dynamic Groovy and (its very capable testing and mocking framework) [Spock](#) is a very viable and common option.

Developers Develop More Search English Android Studio S

Download Android Studio editor Android Gradle Plugin SDK tools Preview

Filter

What's new in Android build

Configure your build

Optimize your build

Extend your build

Write Gradle plugins

Integrate a custom C/C++ build system

Publish your library

Troubleshoot

Android Gradle plugin API reference

```
onVariants(selector().withBuildType("release")) { variant ->
    // Step 1. Register the task.
    val resCreationTask =
        project.tasks.register<ResCreatorTask>("create${variant.name}Resources")
    resCreationTask.configure {
        variant.sources.res?.addGeneratedSourceDirectory(
            resCreationTask,
            ResCreatorTask::outputDirectory
        )
    }

    ...
}

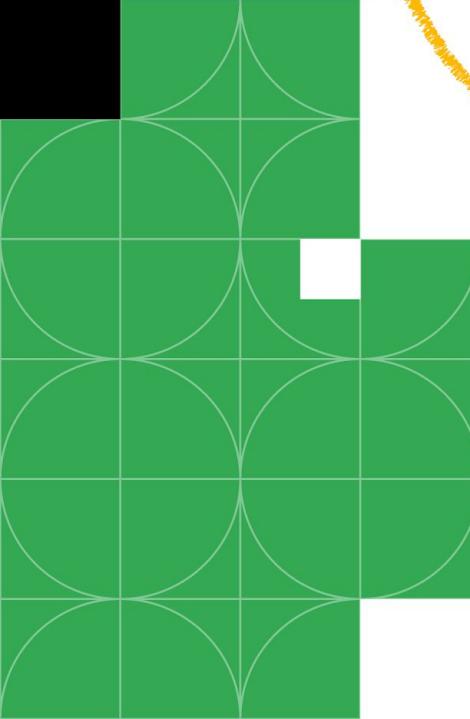
// Step 3. Define the task.
abstract class ResCreatorTask: DefaultTask() {
    @get:OutputFiles
    abstract val outputDirectory: DirectoryProperty

    @TaskAction
    fun taskAction() {
        // Step 4. Generate your resources.
        ...
    }
}
```

```
text  
  'Section Title',  
  style: TextStyle(  
    color: Colors.green[200],  
  ),  
,
```

devfest

```
s.star,  
r: Colors.green[500],  
  
Text('23'),
```

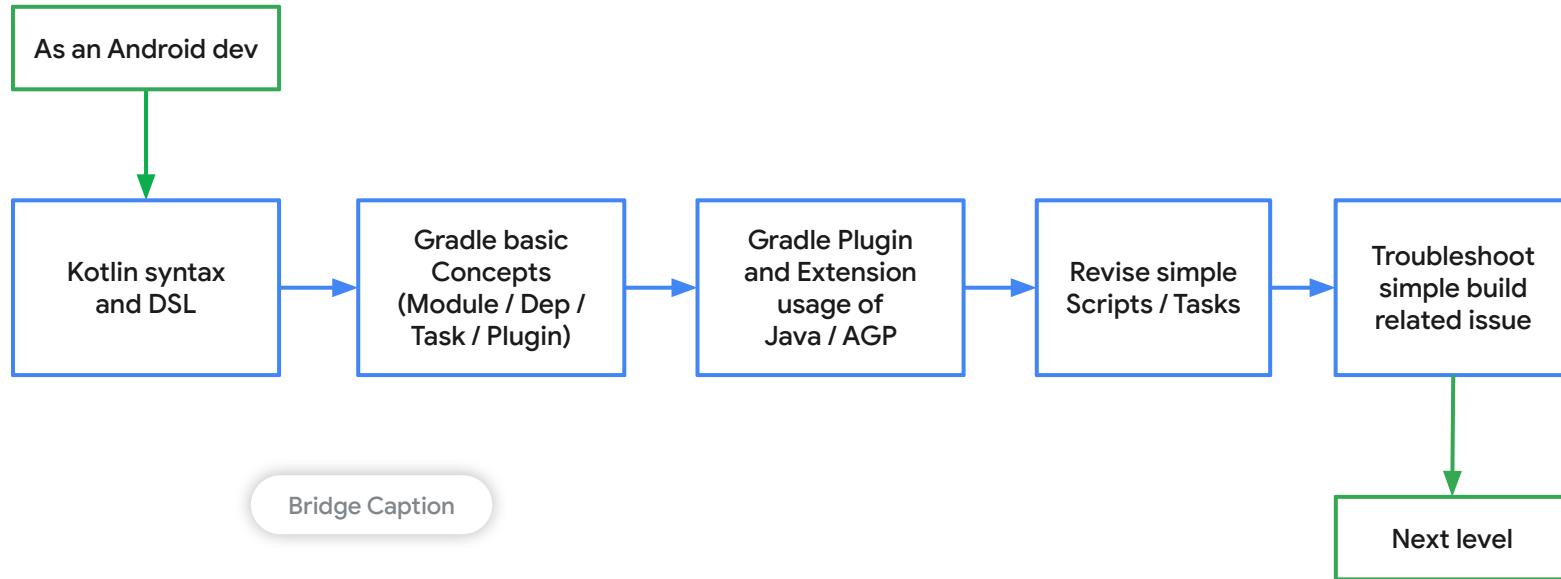


Google Developer Groups

Learning Strategies

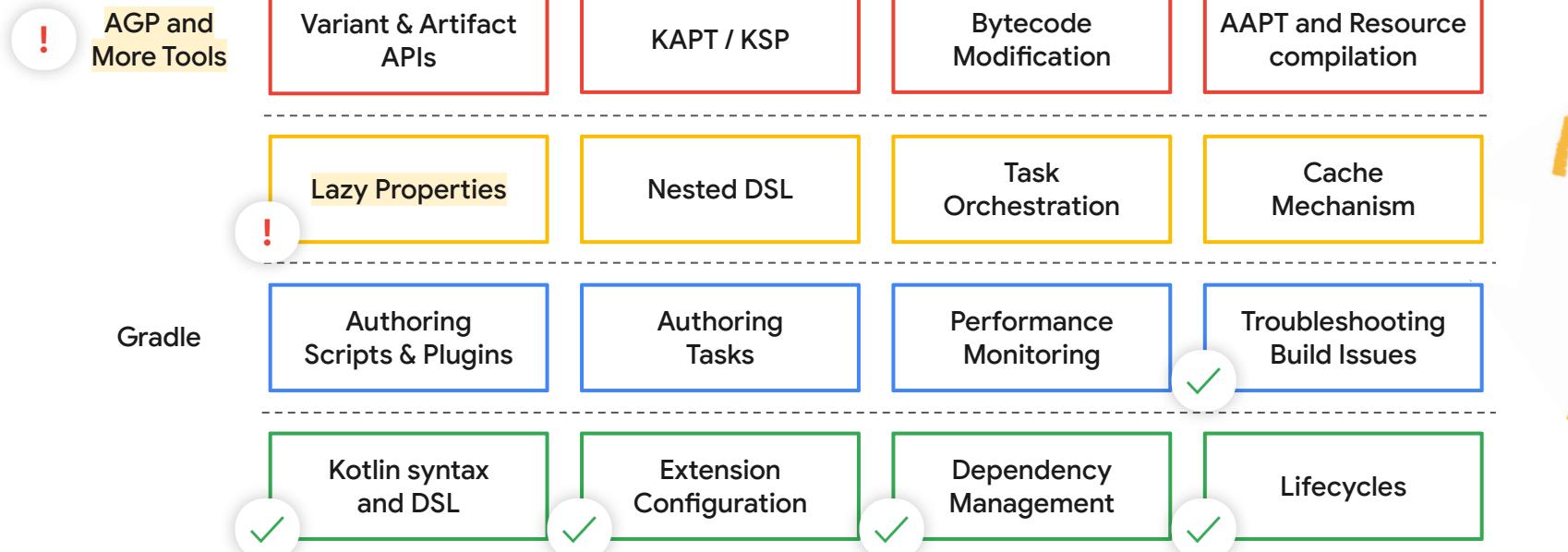
Unlock better understanding of Gradle with Kotlin

For Beginners (Kotlin-oriented)



Remember we are building
plugins over AGP

For Intermediate Level (Kotlin-oriented)



And More

1. [From Gradle properties to AGP APIs](#)
[\(Android Dev Summit '19\)](#) 

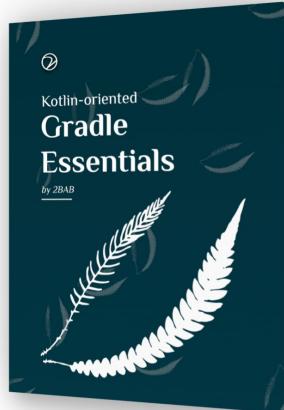
2. [Gradle and AGP Build APIs - MAD Skills](#) 

3. [android/gradle-recipes](#) 

4. [Document -> Android Gradle Plugin -> Write Gradle plugins](#) 

5. [Issue Tracker -> Android Public Tracker -> Android Development -> Android Studio](#)

(Submit here and mention it's AGP related,
then it will be assigned to a sub-track for AGP)



(Scan me to view all resources)

- **Introduction**

- Basis

- Getting Started

- Kotlin DSL

- DSL Configuration

- Dependency Management

- Regular Tasks

- Lifecycle

- Customization

- Advanced Scripts

- Arguments

- Customized Task

- Performance Optimization

- More

**There's no time
like the present**



(Scan me to view all resources)



El Zhang he/his

Android GDE



xx2bab@gmail.com



github.com/2bab



2bab.com



(Scan me to view all resources)