devfest

Third-party App Stores on Android

Google Developer Groups

Raipur

Agenda

1. App stores
2. Third-party app stores
3. Building an app store
4. Q/A

Google Developer Groups

Raipur

App Stores

App Stores

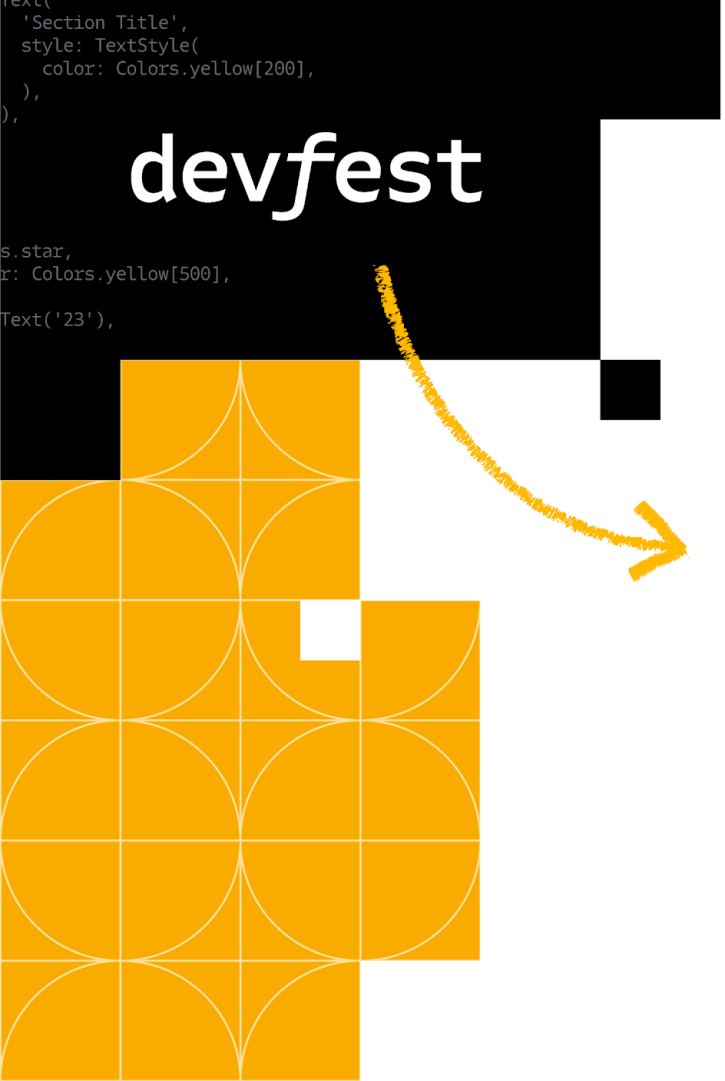
- Digital distribution/hosting platform for software
- Provides users with a catalogue to browse & install software
- Allows updating software already installed
- Shares information about the software being distributed
- Optionally allows you to purchase software

Google Play

- Primary first-party app store and repo
- Developed & distributed by Google
- Privileged system app, present on all Android devices
- Allows users to purchase software & related content/services

Other App Stores

- Amazon's Amazon Appstore
- Samsung's Galaxy store
- Huawei's AppGallery



Google Developer Groups

Raipur

Third-party App Stores

Third-party App Stores

- Alternate distribution platform for software than the bundled one in OS
- Uses either an existing repo or self-hosted
- Access to same system APIs as first-party app store is limited or absent

F-Droid

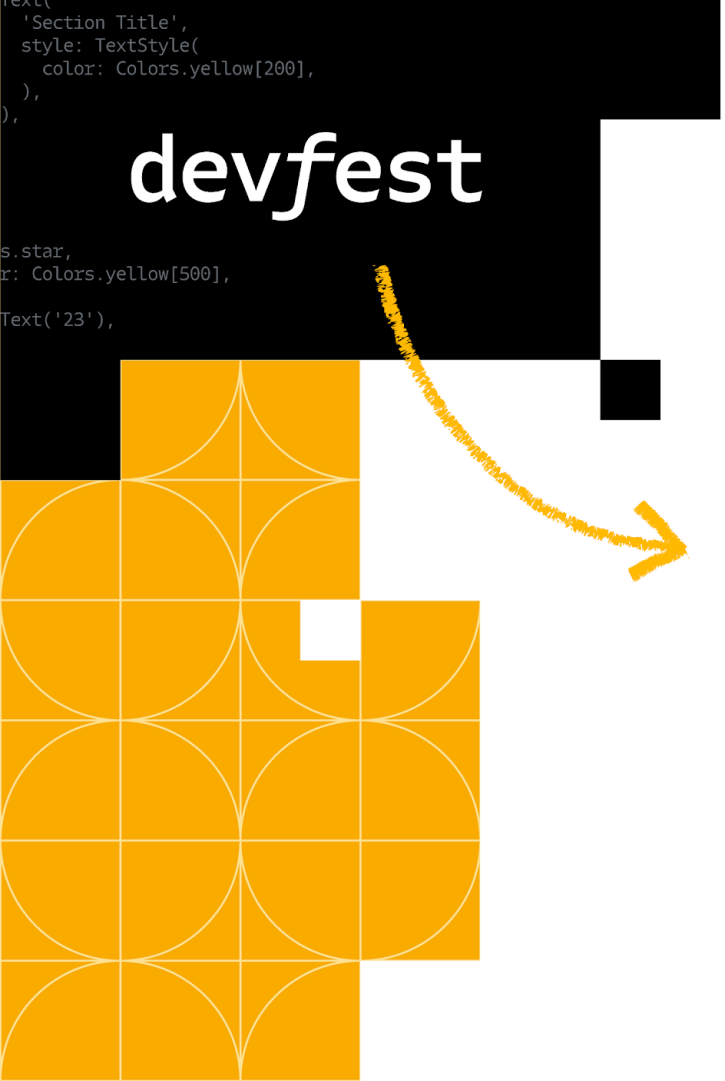
- FOSS-only android app distribution platform
- Maintains owns repo and client app
- Can work with privileged system APIs via extension
- 3k+ stars on GitHub/GitLab

Aurora Store

- Alternative FOSS client to Google Play
- Uses repo hosted by Google Play to distribute same apps
- Can work with system APIs via SU privileges or third-party installers
- 1.5k+ stars on GitLab

Other App Stores

- Droid-ify
- Neo-Store



Google Developer Groups

Raipur

Building an App Store

Pre-requisites

- Must have an existing repo source
- Familiarity with system APIs will be a plus

Displaying Apps

- Present relevant apps on home screen
- Allow to search for specific apps
- Divide and allow browsing apps by categories
- Present all required information about app
  - Description, supported Android version
  - Permissions and data shared
  - Screenshots, reviews and other relevant information
- Optionally consider supporting opening repo links within app

Downloading Apps

- Apps can be either single APK, split-APK and might depend upon shared libs
- Apps can be downloaded easily using either DownloadManager API or WorkManager
- DownloadManager API is simple but allows less configuration
- WorkManager can be configured to download and trigger app installation as required

```
private suspend fun downloadFile(request: Request): Result {
    return withContext(Dispatchers.IO) {
        val requestFile = File(request.filePath)
        try {
            requestFile.createNewFile()
            URL(request.url).openStream().use { input ->
                requestFile.outputStream().use {
                    input.copyTo(it, request.size).collectLatest { p -> onProgress(p) }
                }
            }
            // Ensure downloaded file exists
            if (!File(request.filePath).exists()) {
                Log.e(TAG, "Failed to find downloaded file at ${request.filePath}")
                notifyStatus(DownloadStatus.FAILED)
                return@withContext Result.failure()
            }
            return@withContext Result.success()
        } catch (exception: Exception) {
            Log.e(TAG, "Failed to download ${request.filePath}!", exception)
            requestFile.delete()
            notifyStatus(DownloadStatus.FAILED)
            return@withContext Result.failure()
        }
    }
}
```

Installing Apps

- Apps can be installed on Android devices using either Native or Session Installer
- Intent-based installer uses Intent based installation but is legacy and has been deprecated
- PackageInstaller is the new & default installer, allows more granular control over the install and after install process including updates

Intent-based Installer

- Native Installer remained default till Android 4.4
- Apps can be installed using ACTION_INSTALL_PACKAGE Intent
- ACTION_VIEW can also be used with data and type set as "application/vnd.android.package-archive"
- Other flags can be specified as per convenience such as EXTRA_NOT_UNKNOWN_SOURCE, EXTRA_INSTALLER_PACKAGE_NAME, etc
- It has been deprecated in favour of Session Installer

```
private fun xInstall(file: File) {
    val intent: Intent

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
        intent = Intent(Intent.ACTION_INSTALL_PACKAGE)
        intent.data = getUri(file)
        intent.flags = Intent.FLAG_GRANT_READ_URI_PERMISSION or
Intent.FLAG_ACTIVITY_NEW_TASK
    } else {
        intent = Intent(Intent.ACTION_VIEW)
        intent.setDataAndType(Uri.fromFile(file),
"application/vnd.android.package-archive");
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
    }

    intent.putExtra(Intent.EXTRA_NOT_UNKNOWN_SOURCE, true)
    intent.putExtra(Intent.EXTRA_INSTALLER_PACKAGE_NAME, context.packageName)
    context.startActivity(intent)
}
```

PackageInstaller

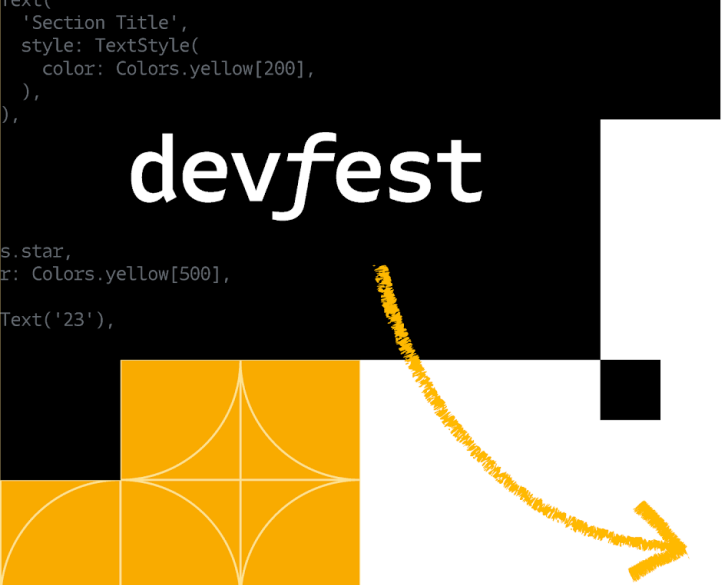
- Introduced on Android 5.0 and is now the default installer
- Allows installing APKs, Split-APKs, Shared libraries and more
- Developer creates a session, sets required information and flags
- Once ready, the session can be committed to the system

```kotlin
fun install(packageName: String, files: List<Any>) {
    val packageInstaller = context.packageManager.packageInstaller
    val sessionParams = SessionParams(SessionParams.MODE_FULL_INSTALL).apply {
        setAppPackageName(packageName)
        if (isOAndAbove()) {
            setInstallReason(PackageManager.INSTALL_REASON_USER)
        }
        if (isNAndAbove()) {
            setOriginatingUid(android.os.Process.myUid())
        }
        if (isSAndAbove()) {
            setRequireUserAction(SessionParams.USER_ACTION_NOT_REQUIRED)
        }
        if (isTAndAbove()) {
            setPackageSource(PACKAGE_SOURCE_STORE)
        }
        if (isUAndAbove()) {
            setInstallerPackageName(context.packageName)
        }
    }

        val sessionId = packageInstaller.createSession(sessionParams)
        val session = packageInstaller.openSession(sessionId)

        xInstall(sessionId, session, packageName, files)
}
```

```kotlin
fun xInstall(sessionId: Int, session: PackageInstaller.Session, packageName: String, files: List<Any>) {
    files.forEach {
        context.contentResolver.openInputStream(it)?.use { input ->
            session.openWrite("${packageName}_${System.currentTimeMillis()}", 0, -1).use {
                input.copyTo(it)
                session.fsync(it)
            }
        }
    }

    val callBackIntent = Intent(context, InstallReceiver::class.java).apply {
        action = InstallReceiver.ACTION_INSTALL_STATUS
        setPackage(context.packageName)
        putExtra(PackageInstaller.EXTRA_PACKAGE_NAME, packageName)
        addFlags(Intent.FLAG_RECEIVER_FOREGROUND)
    }
    val flags = if (isSAndAbove())
        PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_MUTABLE else
        PendingIntent.FLAG_UPDATE_CURRENT

    val pendingIntent = PendingIntent.getBroadcast(
        context,
        sessionId,
        callBackIntent,
        flags
    )

    Log.i("Starting install session for $packageName")
    session.commit(pendingIntent.intentSender)
    session.close()
}
```

Android 14

- Multiple new APIs for PackageInstaller for better install and updates process
- requestUserPreapproval() method to request install approval before downloading apps
- setRequestUpdateOwnership() to claim responsibilities for future updates without asking for confirmation
- commitSessionAfterInstallConstraintsAreMet() method allows to constraint installs to not disrupt users

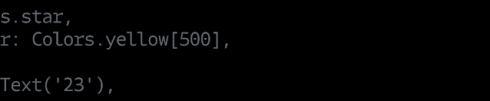


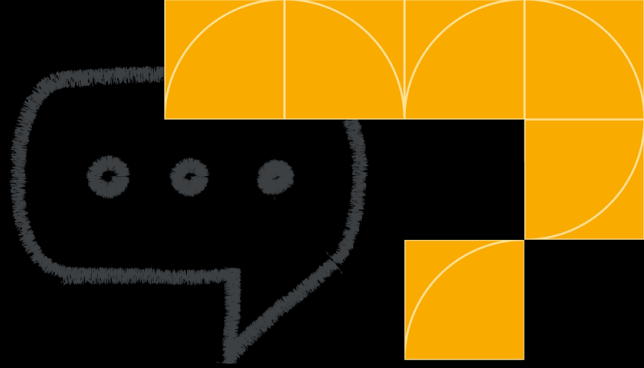
Google Developer Groups

Raipur

Q/A

devfest





Thank You