Distribución de builds VeriX

Builds de VeriX

Builds de VeriX (`scripts\build.ps1`)

Cómo generar un build de VeriX con versión y nombre de archivo automáticos, para `dev`, `test` o `prod`.

Qué hace este script

`build.ps1` es un wrapper alrededor de `eas build`. Dispara el build **en la nube de Expo** (los servidores de Expo compilan, no tu máquina — es exactamente lo mismo que correr `eas build -p android --profile test` a mano), espera a que termine, y descarga el `.apk`/`.aab` ya renombrado. No reemplaza nada de tu flujo normal, solo lo automatiza.

> `eas build --local` **NO se usa aquí.** Si en algún lado ves referencias a builds "locales" en commits viejos de este repo, ya no aplica — el flujo actual es 100% en la nube.

Uso

Desde la raíz del repo (`nexus33-app-verix`), en PowerShell:

```
.\scripts\build.ps1 -Env test
.\scripts\build.ps1 -Env dev
.\scripts\build.ps1 -Env prod
```

Por defecto compila Android. Para iOS:

```
.\scripts\build.ps1 -Env test -Platform ios
```

El build en la nube tarda normalmente **15-20+ minutos** — el script se queda esperando, no hace falta hacer nada mientras tanto.

Prerrequisitos

- Sesión iniciada en EAS: `npx eas-cli login` (si nunca lo has hecho en esta máquina).
- Conexión a internet (el build corre en los servidores de Expo, no localmente).
- Git instalado y este repo clonado con al menos un commit (el script usa el hash del commit actual).

No hace falta Android SDK, JDK, ni Xcode instalados — eso lo tiene Expo del lado de sus servidores.

Troubleshooting

`An Expo user account is required to proceed` / **`eas build exited with code 1`** No hay sesión iniciada en EAS en esta máquina (el script corre con `--non-interactive`, así que no te va a pedir login solo — falla directo). Soluciónalo con:

```
npx eas-cli login
```

y confirma con `npx eas-cli whoami`. Después vuelve a correr el script normalmente. Ver también la nota sobre `app.json` más abajo si el build falló después de que la versión ya se bumpeó.

`eas : term not recognized` Estás intentando correr `eas` directo en vez de `npx eas-cli`. El script usa `npx eas-cli` internamente, así que no necesitas tener `eas-cli` instalado global — pero si quieres el comando corto disponible en tu terminal: `npm install -g eas-cli`.

Qué hace el script, paso a paso

1. Lee la versión actual de `app.json` (`expo.version`).
2. Calcula la siguiente versión — ver "Versionado" abajo.
3. Escribe la nueva versión en `app.json` (**en disco, sin commit/push** — eso lo sigues haciendo tú manualmente, como siempre).
4. Obtiene el hash corto del commit actual (`git rev-parse --short HEAD`).
5. Genera el timestamp del build.
6. Mapea `-Env` al perfil real de `eas.json`: `dev`→`development`, `test`→`test`, `prod`→`production`.
7. Corre `eas build -p <platform> --profile <perfil> --wait` (equivalente a hacerlo a mano) y espera a que termine.
8. Cuando termina, toma el link de descarga que devuelve Expo y baja el archivo a `builds\<nombre-generado>`.

Si algo falla en cualquier paso, el script se detiene y muestra el error — no deja archivos de build a medias con nombres incorrectos.

> ⚠ **Ojo con fallos después del paso 3.** El bump de versión en `app.json` ocurre *antes* de disparar el build (paso 3 vs. paso 7). Si el build falla después de eso — por ejemplo, por no tener sesión iniciada en EAS — `app.json` queda con la versión ya incrementada en disco, sin build real asociado. Antes de reintentar, revisa `git diff app.json`: puedes revertirlo (`git checkout -- app.json`) para no "quemar" un número de versión, o simplemente dejarlo y correr el script de nuevo (no pasa nada grave si el contador salta un número).

Formato del nombre de archivo

```
VeriX-{env}-{version}-{yyyyMMddHHmm}+{commitHash}.{apk|aab|ipa}
```

Ejemplo:

```
VeriX-test-1.0.1-202607241005+7bc21f0.apk
```

Parte	Significado	Origen
`env`	`dev`, `test` o `prod`	Parámetro `-Env`
`version`	Versión semántica	`app.json` (`expo.version`), auto-bumpeada por el script
`yyyyMMddHHmm`	Fecha/hora del build	Generado al momento de compilar
`commitHash`	Hash corto del commit actual	`git rev-parse --short HEAD`
Extensión	`.apk`, `.aab`, `.ipa` (o `.tar.gz` en builds de simulador iOS)	Tomada del link real que devuelve Expo, no adivinada

`prod` en Android genera `.aab` (Android App Bundle, lo que exige Google Play para publicar). `dev` / `test` generan `.apk` (instalable directo en un dispositivo). Esto ya estaba definido en `eas.json`, el script no lo cambia.

Versionado (`app.json` ? `expo.version`)

Formato `major.minor.patch`:

- **`major`** — siempre manual. Solo tú lo cambias, para indicar un cambio grande/breaking.

- `minor` y `patch` — semi-automáticos, con un contador **global** compartido entre `dev`, `test` y `prod` (no es un conteo separado por ambiente):
  - Cada build (sin importar el ambiente) suma 1 al `patch`.
  - Al llegar a 20 builds, el `patch` vuelve a 0 y el `minor` sube en 1.

Build #	Versión
1 (manual, punto de partida)	`1.0.0`
2	`1.0.1`
…	…
20	`1.0.19`
21 (rollover)	`1.1.0`

`versionCode` (Android) / `buildNumber` (iOS) — campos aparte

Estos **no** son lo mismo que `expo.version` de arriba. Son 2 campos separados (uno por plataforma) que las tiendas usan para exigir que cada build subido sea estrictamente más nuevo que el anterior:

- `android.versionCode` (entero: 1, 2, 3…)
- `ios.buildNumber` (string: "1", "2", "3"…)

Hoy están configurados para auto-incrementarse **solo en el perfil** `production` (`eas.json` → `"autoIncrement": true`). Cada build de `production` los sube en 1 automáticamente — EAS los escribe en `app.json` solo, sin que el script tenga que hacer nada. `test`/`development` no los tocan (no lo necesitan: no se suben a las tiendas, y reinstalar un `.apk` con el mismo `versionCode` sobre uno viejo funciona bien en Android).

Salida

Los builds quedan en `nexus33-app-verix\builds\` (carpeta ignorada por git, ver `.gitignore` — no se commitean).

Commit del bump de versión

El script deja `app.json` modificado en disco pero **no commitea ni pushea nada**. Cuando quieras dejar ese cambio de versión en git, hazlo tú manualmente como siempre (`git add app.json`, `git commit`, `git push`).

Pendiente / no cubierto todavía por este flujo

- **`eas submit`** — subir el `.aab`/`.ipa` a Play Store / App Store todavía no está automatizado (`eas.json` → `submit.production` está vacío). Hoy la subida a las tiendas se hace manual con el archivo que este script genera. Los pasos completos para instalar/distribuir cada build (pruebas y producción) están documentados aparte: ver `distribucion-android-eas.md` y `distribucion-ios-eas.md`.
- **Credenciales de firma** (keystore Android / certificados iOS) — no verificadas todavía. EAS las pide de forma interactiva la primera vez que corras `eas build --profile production`; no requiere trabajo previo, solo tenerlo en cuenta la primera vez.
- **Fichas de la app en Play Console / App Store Connect** — no es código, hay que crearlas manualmente (screenshots, descripción, política de privacidad, data safety / privacy nutrition label) antes de poder subir el primer build.
- **OTA updates (`expo-updates`)** — decisión consciente de dejarlo pendiente (2026-07-23). Para el primer lanzamiento no aporta lo suficiente frente a la complejidad que agrega, y al ser una app de cumplimiento regulatorio (DOT) hay valor en que cada cambio pase por el review formal de la tienda. Revisar de nuevo en 1-2 meses con datos reales de cadencia de updates/hotfixes. No requiere ningún cambio previo — es 100% aditivo cuando se decida hacerlo.

Distribución de builds Android con EAS — VeriX

A diferencia de iOS, Android **no requiere registrar dispositivos** ni certificados de por vida — un APK se puede instalar en cualquier teléfono con "orígenes desconocidos" habilitado. Esto hace que el flujo de pruebas sea más simple.

> Para generar el build en sí (`.apk` / `.aab`) con versión y nombre automáticos, ver `BUILD.md` (`scripts\build.ps1`). Esta guía asume que ya tienes ese archivo listo.

Opción A: Test / instalación directa (APK)

Ideal para pruebas rápidas en cualquier dispositivo del equipo, sin pasar por Google Play.

1. Verificar el perfil en `eas.json`

Para pruebas, el perfil `test` debe generar un **APK** (instalable directo), no un AAB (que es exclusivo para Play Store):

```
"test": {
  "distribution": "internal",
  "android": {
    "buildType": "apk"
  }
}
```

2. Correr el build

```
.\scripts\build.ps1 -Env test -Platform android
```

El resultado es un `.apk` en la carpeta `builds\`.

3. Instalar en el dispositivo

Dos formas:

a) Link directo (más fácil para compartir)

- El build queda disponible en expo.dev con un link de descarga.
- Abre ese link **desde el navegador del teléfono Android** y descarga el APK.
- Si es la primera vez, Android pedirá habilitar "Instalar apps desconocidas" para ese navegador — se acepta una sola vez.

b) Cable USB (si tienes el .apk en la PC)

```
adb install builds\VeriX-test-<version>.apk
```

(requiere `adb` — viene con Android Studio / platform-tools, y el dispositivo con depuración USB activada)

Notas

- No hay expiración de certificado ni límite de dispositivos, a diferencia de iOS ad hoc.
- Cada instalación reemplaza la anterior si usas el mismo `applicationId` — no necesitas desinstalar manualmente entre builds.

Opción B: Producción (publicación en Google Play)

1. Requisitos previos

- Cuenta de Google Play Console (pago único de $25, una sola vez).
- App creada en Play Console con su ficha (descripción, screenshots, política de privacidad, clasificación de contenido).
- Una **cuenta de servicio (service account)** de Google Cloud con permisos en Play Console, y su archivo JSON de credenciales — necesario para que `eas submit` pueda subir builds automáticamente.
  - Se configura una vez en `eas.json` → `submit.production.android.serviceAccountKeyPath`, apuntando al archivo `.json` (no se sube a git — agregarlo a `.gitignore`).

2. Ajustar el perfil de producción en `eas.json`

Play Store requiere formato **AAB** (Android App Bundle), no APK:

```
"production": {
  "distribution": "store",
  "android": {
    "buildType": "app-bundle"
  }
}
```

3. Generar el build

```
.\scripts\build.ps1 -Env prod -Platform android
```

4. Subir el build a Play Console

```
npx eas-cli submit -p android --latest
```

Esto sube el `.aab` automáticamente al track que definas (por defecto, `internal` — se puede cambiar a `alpha`, `beta` o `production` en `eas.json` bajo `submit.production.android.track`).

5. Testing interno en Play Console (recomendado antes de producción)

- En Play Console → tu app → **Testing → Internal testing**.
- Agrega testers por email o crea un link de opt-in.
- Los testers instalan la app **desde el Play Store** (no requiere APK manual) una vez aceptan la invitación.
- Esto evita mandar builds directo a producción sin pasar por revisión previa de tu equipo.

6. Promover a producción

- Una vez validado en internal testing, en Play Console → **Production → Create new release** → selecciona el build ya subido (o promuévelo directo desde el track de testing con "Promote release").
- Completa el rollout (puede ser gradual, ej. 20% → 50% → 100%, para reducir riesgo).
- Google revisa el release (usualmente horas, a veces hasta un par de días en la primera publicación de una app nueva).

Notas

- El `versionCode` de Android debe ser mayor en cada release nuevo — si usas `"autoIncrement": true` en `eas.json` (como ya está configurado en VeriX), EAS lo maneja automáticamente.
- No puedes volver a subir el mismo `versionCode` una vez publicado, ni siquiera si cancelas el rollout.

¿Cuál usar?

	Test (APK directo)	Producción (Play Store)
Setup inicial	Ninguno	Cuenta de Play Console + service account
Requiere registrar dispositivo	No	No
Formato de build	APK	AAB
Pasa por revisión de Google	No	Sí (horas a días)
Ideal para	Prueba rápida en cualquier Android	Lanzamiento público / testers vía Play Store

---

Comparación rápida con iOS

	Android	iOS
Pruebas rápidas	APK directo, sin registrar nada	Requiere registrar UDID (ad hoc)
Testing con más gente	Internal testing track (Play Store)	TestFlight
Producción	Play Console (AAB)	App Store Connect (revisión completa)
Costo de cuenta developer	$25 único	$99/año

Distribución de builds iOS con EAS — VeriX

Dos formas de instalar un build de prueba en un iPhone físico sin pasar por el simulador.

> Para generar el build en sí (`.ipa` / `.tar.gz`) con versión y nombre automáticos, ver `BUILD.md` (`scripts\build.ps1`). Esta guía asume que ya tienes ese archivo listo.

Opción A: Ad Hoc (directo al dispositivo, sin TestFlight)

Rápido, pero limitado a dispositivos registrados manualmente (máx. 100 UDIDs/año en tu cuenta Apple Developer).

1. Ajustar el perfil en `eas.json`

```
"test": {
  "distribution": "internal",
  "ios": {
    "simulator": false
  }
}
```

2. Registrar el iPhone (obtener su UDID)

```
npx eas-cli device:create
```

- Genera un link/QR.
- Ábrelo **desde Safari en el iPhone**.
- Instala el perfil de configuración que se descarga — esto registra el UDID automáticamente en Expo/Apple.

3. Correr el build

```
.\scripts\build.ps1 -Env test -Platform ios
```

- EAS genera (o reutiliza) el certificado de distribución y un provisioning profile ad hoc que incluye ese dispositivo.
- El resultado es un `.ipa` (no `.tar.gz`).

4. Instalar en el iPhone

- El build aparece en expo.dev o en el link que EAS imprime en consola al terminar.
- Abre ese link **desde el navegador del iPhone**.
- Toca instalar — no requiere Mac, App Store ni TestFlight.

Notas

- Solo dispositivos registrados *antes* de generar el provisioning profile pueden instalar el build.
- Si el dispositivo se agregó después de un build viejo, puede que necesites regenerar el profile:

  ```
  eas build --clear-provisioning-profile
  ```

- El provisioning profile expira (normalmente ~1 año).

Opción B: TestFlight

Mejor para testers externos o cuando no quieres registrar UDIDs uno por uno. Pasa por App Store Connect, así que toma un poco más de tiempo (revisión automática, no manual, suele ser minutos-horas).

1. Requisitos previos

- Cuenta de Apple Developer Program (paga, $99/año) vinculada al proyecto en EAS.
- App creada en App Store Connect.

2. Ajustar el perfil de build en `eas.json`

Para TestFlight normalmente se usa el perfil de `production` (o uno dedicado), con distribución **store**:

```
"production": {
  "distribution": "store",
  "ios": {
    "simulator": false
  }
}
```

3. Generar el build

```
.\scripts\build.ps1 -Env prod -Platform ios
```

(o el ambiente que corresponda al perfil que uses para TestFlight)

4. Subir el build a App Store Connect

```
npx eas-cli submit -p ios --latest
```

- Sube el `.ipa` más reciente automáticamente a App Store Connect.
- Espera a que Apple procese el build (aparece en la sección TestFlight de App Store Connect, usualmente en minutos).

5. Agregar testers

- En App Store Connect → tu app → pestaña **TestFlight**.
- Agrega testers por email (grupo interno) o crea un **link público** de invitación si quieres que cualquiera con el link pueda unirse.
- Los testers instalan la app **TestFlight** desde el App Store, y desde ahí instalan tu build usando el link/invitación.

Notas

- No requiere registrar UDIDs — cualquier tester invitado puede instalar sin configuración previa.
- Los builds de TestFlight expiran a los 90 días.
- Testers externos (fuera de tu equipo de Apple Developer) requieren que Apple apruebe el build para "external testing" (revisión ligera, normalmente rápida).

Opción C: Producción (publicación en App Store)

El build es técnicamente el mismo que para TestFlight (`distribution: "store"`), pero además de subirlo hay que enviarlo a **revisión de Apple** y publicarlo.

1. Build y submit (igual que TestFlight, pasos 3-4)

```
.\scripts\build.ps1 -Env prod -Platform ios
npx eas-cli submit -p ios --latest
```

Esto sube el `.ipa` a App Store Connect — mismo build que usarías para TestFlight, solo que ahora lo vas a enviar a revisión en vez de (o además de) dejarlo en testing interno.

2. Completar la ficha en App Store Connect

Antes de enviar a revisión, en App Store Connect → tu app → **App Store** tab necesitas tener listo:

- Screenshots (por tamaño de dispositivo)
- Descripción, keywords, categoría
- Política de privacidad (URL)
- Clasificación de contenido (age rating)
- Precio/disponibilidad por país

3. Enviar a revisión

- Selecciona el build que subiste (paso 1) en la sección **Build**.
- Click en **Add for Review** → **Submit for Review**.
- Apple revisa (normalmente 24-48h, puede variar).

4. Publicación

- Si aprueban, puedes elegir publicación **manual** (tú decides cuándo sale al público) o **automática** (sale apenas Apple aprueba) — se configura antes de enviar a revisión.
- También puedes usar **phased release** (lanzamiento gradual a % de usuarios en 7 días) para reducir riesgo si algo sale mal.

Notas

- El número de versión (`app.json` → `version`) y el `buildNumber` deben ser mayores a la última versión publicada — tu script ya incrementa el patch automáticamente, pero revisa que no choque con una versión ya en revisión.
- Una vez enviado a revisión, no puedes subir un build nuevo sin cancelar el envío anterior.
- Si tu app usa capacidades especiales (push notifications, in-app purchases, etc.), Apple puede pedir información adicional o rechazar por incumplimiento de guidelines — vale la pena revisar el build en TestFlight primero antes de enviarlo a producción.

¿Cuál usar?

	Ad Hoc	TestFlight	Producción
Setup inicial	Rápido	Más pasos (App Store Connect)	Igual que TestFlight + ficha completa
Requiere registrar UDID	Sí	No	No
Límite de dispositivos	100/año	Sin límite práctico	Sin límite (público)
Expiración del build	~1 año	90 días	No expira (versión publicada)
Pasa por revisión de Apple	No	Ligera (external testing)	Sí (revisión completa, 24-48h)
Ideal para	Prueba rápida en 1-2 dispositivos del equipo	Testers externos, distribución más amplia	Lanzamiento público