Desarrollo de Interfaces  
Unidad 08. Introducción a Flutter - Instalación y comandos prácticos

short line

Autor: Sergi García

Actualizado Agosto 2025

Licencia

**Reconocimiento - No comercial - CompartirIgual** (BY-NC-SA): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se ha de hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán diferentes símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

📖 **Importante**

❕ **Atención**

💬 **Interesante**

**Índice**

[**1. Introducción a Flutter 3**](#_4i7dfnou906)

[**2. Instalación de Flutter y Configuración del Entorno 4**](#_lhef2njw1i2t)

[**3. Estructura básica de una app Flutter 4**](#_gxh5ssstv4a8)

[**4. Comandos principales de Flutter 6**](#_m36t2vqhr3si)

[**5. Gestión de Paquetes y Herramientas en Flutter 8**](#_bch0rck3o71h)

[**6. Extensiones recomendadas en VS Code / Android Studio 10**](#_4yqu6klhxffq)

[**7. Despliegue de Flutter en diferentes plataformas 10**](#_1bg0zy25bp7x)

[**8. Recursos recomendados para aprender Flutter 11**](#_yl6ohe946aee)

Unidad 08. Introducción a Flutter

# 1. Introducción a Flutter

**🔷 ¿Qué es Flutter?**

Flutter es un framework de código abierto desarrollado por Google para crear interfaces nativas multiplataforma (móvil, web y escritorio) desde una única base de código. Su primera versión estable se lanzó en diciembre de 2018, y desde entonces ha ganado popularidad por su rendimiento, flexibilidad y productividad.

**Características clave:**  
✔ UI declarativa: La interfaz se construye en función del estado actual de la aplicación.  
✔ Motor de renderizado propio (Skia): No depende de componentes nativos del sistema, lo que garantiza consistencia visual en todas las plataformas.  
✔ Alto rendimiento: Compila a código nativo (ARM, x64) y JavaScript (para web).  
✔ Hot Reload: Permite ver cambios al instante sin reiniciar la app, acelerando el desarrollo.  
✔ Widgets altamente personalizables: Ofrece una amplia biblioteca de componentes adaptables.

**🔷 ¿Qué es Dart y por qué Flutter lo usa?**

Dart es el lenguaje de programación detrás de Flutter, diseñado por Google para ser:  
✔ Rápido: Compilación AOT (ahead-of-time) para producción y JIT (just-in-time) para desarrollo ágil.  
✔ Productivo: Sintaxis clara y moderna, similar a JavaScript/TypeScript, Java y C#.  
✔ Orientado a UI: Ideal para aplicaciones reactivas gracias a su manejo eficiente de estados y eventos.

¿Por qué Flutter eligió Dart?

* Rendimiento cercano al nativo (evita el "puente JavaScript" de otros frameworks).
* Capacidad de compilación multiplataforma (móvil, web y desktop).
* Hot Reload nativo, algo difícil de lograr con otros lenguajes.

**🔷 Ventajas del desarrollo con Flutter**

✅ Código único para múltiples plataformas:

* Escribe una vez y despliega en Android, iOS, web, Windows, macOS y Linux.

✅ Productividad elevada:

* Hot Reload acelera las iteraciones de desarrollo.
* Amplio ecosistema de paquetes (pub.dev) y herramientas integradas (Dart DevTools).

✅ UI consistente y personalizable:

* Los widgets de Flutter se ven y funcionan igual en todas las plataformas, sin inconsistencias entre Android/iOS.

✅ Comunidad activa y soporte de Google:

* Más de 150,000 paquetes disponibles en pub.dev.
* Documentación oficial detallada y actualizada.

✅ Rendimiento competitivo:

* Supera a soluciones basadas en JavaScript (como React Native) al evitar el "puente" entre lenguajes

**🔷 Flutter vs React Native vs Apps Nativas**

| **Característica** | **Flutter** | **React Native** | **Nativo (Kotlin/Swift)** |
| --- | --- | --- | --- |
| **Lenguaje** | Dart | JavaScript | Kotlin / Swift |
| **Rendimiento** | Alto (compilación nativa) | Medio-Alto | Excelente |
| **UI** | 100% personalizada con widgets | Bridged con componentes nativos | Componentes nativos |
| **Hot Reload** | Sí | Sí | No |
| **Comunidad y soporte** | Alta y creciendo | Muy grande | Alta pero separada por plataforma |
| **Acceso a funciones nativas** | Completo con plugins y canales | Requiere puente con código nativo | Directo |
| **Estabilidad** | Alta | Media-Alta | Alta |

# 2. Instalación de Flutter y Configuración del Entorno

**🔧 Requisitos generales**

Los requisitos para instalar Flutter son:

* Un sistema operativo compatible: Windows, macOS o Linux
* Espacio en disco: Al menos 2.8 GB (sin contar dependencias)
* Un editor de texto o IDE: Visual Studio Code, Android Studio, etc.
* Git instalado y accesible desde la terminal

Para instalar Flutter, sigue los pasos actualizados en <https://docs.flutter.dev/get-started/install>

# 3. Estructura básica de una app Flutter

En Flutter, toda aplicación empieza por el archivo **main.dart** que se encuentra dentro de la carpeta lib/.  
 Este archivo contiene la función principal (main) que arranca la aplicación con el método runApp().

Ejemplo mínimo:

| **import** 'package:flutter/material.dart';  **void** main() {  runApp(MyApp()); *// Punto de arranque de la app* }  **class** **MyApp** **extends** **StatelessWidget** {  **@override**  Widget build(BuildContext context) {  **return** MaterialApp( *// Widget raíz de la app*  title: 'Mi primera app Flutter',  home: HomePage(),  );  } }  **class** **HomePage** **extends** **StatelessWidget** {  **@override**  Widget build(BuildContext context) {  **return** Scaffold( *// Estructura básica de una pantalla*  appBar: AppBar(title: Text('Inicio')),  body: Center(child: Text('¡Hola Mundo!')),  );  } } |
| --- |

**Widgets principales: MaterialApp y Scaffold**

En este ejemplo aparecen dos widgets fundamentales:

1. MaterialApp
   * Es el contenedor raíz de toda la aplicación.
   * Define la configuración global:
     + Título
     + Tema (colores, tipografías)
     + Rutas de navegación
     + Pantalla inicial (propiedad home)

📌 Analogía: Piensa en MaterialApp como el diseñador de interiores de una casa: establece el estilo, los colores y la distribución general.

1. Scaffold
   * Proporciona la estructura visual estándar de cada pantalla.
   * Incluye los elementos más comunes de una interfaz en Material Design:
     + AppBar: barra superior
     + Body: contenido principal
     + Drawer: menú lateral
     + FloatingActionButton: botón flotante

📌 Analogía: Scaffold es como el andamiaje de una casa: paredes, techo y suelo donde colocas los muebles (otros widgets).

**Analogías prácticas**

1. **La casa 🏠**
   * main.dart → La puerta de entrada de la casa.
   * MaterialApp → El interiorista que decide colores, distribución y estilo.
   * Scaffold → Las paredes y el esqueleto de cada habitación.
   * Widgets (Text, Button, etc.) → Los muebles y objetos dentro de la casa.
2. **Las muñecas rusas 🎎**
   * Los widgets se anidan unos dentro de otros.
   * Ejemplo:

| MaterialApp(  home: Scaffold(  body: Center(  child: Text('Hola'),  ),  ), ) |
| --- |

* + Aquí vemos:
    - MaterialApp contiene → Scaffold
    - Scaffold contiene → Center
    - Center contiene → Text

📌 Como las muñecas rusas, cada widget es una pieza que envuelve a otra hasta llegar al más pequeño.

**✅ Ideas clave para recordar:**

1. El archivo main.dart siempre es el inicio de la aplicación.
2. MaterialApp configura la app entera.
3. Cada pantalla suele estar envuelta en un Scaffold.
4. Los widgets se organizan en forma de árbol, de padres a hijos.

# 4. Comandos principales de Flutter

Flutter ofrece una CLI (Command Line Interface) muy potente para gestionar todo el ciclo de vida de una aplicación: desde la creación del proyecto hasta el despliegue en las diferentes plataformas.

A continuación, repasamos los comandos más importantes:

**flutter create → Crear un proyecto**

Este comando genera la estructura básica de una aplicación Flutter.

| flutter create mi\_app |
| --- |

📌 Esto crea una carpeta con:

* lib/main.dart → Punto de entrada de la app
* pubspec.yaml → Archivo de configuración de dependencias y assets
* Carpetas específicas para Android (android/), iOS (ios/), web (web/), etc.

**flutter run → Ejecutar en dispositivo/emulador**

Permite compilar y lanzar la app en un emulador o dispositivo físico conectado.

| flutter run |
| --- |

**Opciones comunes:**

* flutter run -d chrome → Ejecutar en navegador Chrome
* flutter run -d emulator-5554 → Lanzar en un emulador Android específico
* flutter run -d ios → Ejecutar en un simulador de iOS

📌 Además, soporta Hot Reload (r) y Hot Restart (R) directamente desde la terminal.

**📦 flutter build → Generar binarios para distintas plataformas**

Compila la aplicación en modo release para distribuirla.

Ejemplos más usados:

| flutter build apk *# Genera un APK para Android* flutter build appbundle *# Genera un AAB (obligatorio en Play Store)* flutter build ios *# Genera un IPA para iOS* flutter build web *# Compila para web (HTML/CSS/JS)* flutter build windows *# Compila ejecutable para Windows* flutter build linux *# Compila para Linux* flutter build macos *# Compila para macOS* |
| --- |

📌 Importante:

* Para Android → necesitarás configurar firma en key.properties
* Para iOS → requiere Xcode y cuenta de desarrollador de Apple

**📚 flutter pub get y flutter pub add → Gestión de dependencias**

Las dependencias en Flutter se gestionan en el archivo **pubspec.yaml.**

* Actualizar dependencias

| flutter pub get |
| --- |

Descarga las librerías definidas en pubspec.yaml.

* Añadir una dependencia

| flutter pub add http |
| --- |

Añade el paquete http automáticamente al **pubspec.yaml** e instala la librería.

**Ejemplo manual:**  
Fichero **pubspec.yaml**

| dependencies:  http: ^0.13.6  provider: ^6.1.1 |
| --- |

Luego ejecutar:

| flutter pub get |
| --- |

👉 Las librerías se obtienen del repositorio oficial: [pub.dev](https://pub.dev).

**🧪 flutter test → Ejecución de tests unitarios**

Flutter incluye un framework de testing integrado.

| flutter test |
| --- |

📌 Busca automáticamente todos los archivos terminados en \_test.dart dentro de la carpeta test/.

Ejemplo de test unitario (test/suma\_test.dart):

| **import** 'package:flutter\_test/flutter\_test.dart';  int sumar(int a, int b) => a + b;  **void** main() {  test('La función sumar debería retornar la suma de dos enteros', () {  expect(sumar(2, 3), 5);  }); } |
| --- |

Al ejecutar flutter test, la salida mostrará si el test pasó o falló ✅❌.

**✅ Resumen rápido**

| **Comando** | **Uso principal** | **Ejemplo** |
| --- | --- | --- |
| **flutter create** | Crear un nuevo proyecto | flutter create mi\_app |
| **flutter run** | Ejecutar en dispositivo/emulador | flutter run -d chrome |
| **flutter build** | Generar binarios para distribución | flutter build apk |
| **flutter pub get** | Descargar dependencias definidas | flutter pub get |
| **flutter pub add** | Añadir nueva dependencia | flutter pub add provider |
| **flutter test** | Ejecutar tests unitarios | flutter test |

# 5. Gestión de Paquetes y Herramientas en Flutter

Una de las mayores ventajas de Flutter es su ecosistema de paquetes en [pub.dev](https://pub.dev), donde encontramos más de 150.000 librerías que extienden las funcionalidades de nuestras aplicaciones (bases de datos, networking, autenticación, animaciones, etc.).

**Añadir dependencias al pubspec.yaml**

El archivo pubspec.yaml es el corazón de la configuración de un proyecto Flutter. Aquí definimos:

* Nombre y versión de la app
* Dependencias externas (paquetes de pub.dev)
* Assets (imágenes, fuentes, traducciones)

Ejemplo básico de **pubspec.yaml**:

| name: mi\_app description: Mi primera aplicación en Flutter version: 1.0.0+1  environment:  sdk: ">=3.0.0 <4.0.0"  dependencies:  flutter:  sdk: flutter   *# Dependencias externas*  http: ^0.13.6  provider: ^6.1.1  dev\_dependencies:  flutter\_test:  sdk: flutter |
| --- |

📌 Cada vez que edites el archivo debes ejecutar:

| flutter pub get |
| --- |

**Herramientas prácticas para desarrollo**

**🔹 Hot Reload vs Hot Restart**

Una de las funcionalidades estrella de Flutter:

* Hot Reload (r)
  + Recarga solo el código modificado.
  + Mantiene el estado actual de la app.
  + Ideal para ajustes visuales (UI, estilos).
  + Ejemplo: cambiar color de un botón → se refleja al instante.
* Hot Restart (R)
  + Reinicia toda la aplicación desde cero.
  + Se pierde el estado.
  + Útil cuando cambias variables globales, inicialización o dependencias.

📌 Ambos pueden ejecutarse desde la terminal o desde VS Code/Android Studio.

**🔹 Dart DevTools (Debugging y Profiling)**

[Dart DevTools](https://docs.flutter.dev/tools/devtools) es un conjunto de herramientas que se integran con Flutter para analizar y depurar:

* Inspector de Widgets → Ver jerarquía de widgets en tiempo real.
* Performance → Analizar tiempos de renderizado y detectar cuellos de botella.
* Memory → Ver consumo de memoria y detectar fugas.
* Logs → Inspeccionar errores y prints.

👉 Para abrir DevTools:

| flutter pub global activate devtools flutter run --observatory-port=9200 devtools |
| --- |

Se abrirá en el navegador como un dashboard visual.

# 6. Extensiones recomendadas en VS Code / Android Studio

**📌 VS Code**

* Flutter (Dart-Code.flutter): soporte completo para Flutter (snippets, autocompletado, debugging).
* Dart (Dart-Code.dart-code): soporte para Dart.
* Flutter Widget Snippets: autocompletado rápido de widgets.
* Pubspec Assist: facilita añadir paquetes de pub.dev.

**📌 Android Studio**

* Flutter plugin: integración con emuladores y debugging.
* Dart plugin: soporte de lenguaje.
* Flutter Enhancement Suite: herramientas adicionales para mejorar la productividad.

# 7. Despliegue de Flutter en diferentes plataformas

**📱 Android**

🔗 Documentación oficial para publicar en Google Play Store  
<https://docs.flutter.dev/deployment/android>

* Cómo generar APK/AAB: flutter build appbundle o flutter build apk
* Configuración de firma: keytool y build.gradle
* Proceso de subida a Play Console

**🍏 iOS**

🔗 Guía completa para App Store  
<https://docs.flutter.dev/deployment/ios>

* Requisitos: Cuenta de desarrollador Apple ($99/año)
* Generar IPA: flutter build ipa
* Configurar Xcode: Certificados y provisionamiento

**🖥️ Windows**

🔗 Compilación para Windows  
<https://docs.flutter.dev/deployment/windows>

* Requisitos: Visual Studio 2022 con workloads específicos
* Comando: flutter build windows
* Opciones de empaquetado: MSIX, EXE o instalador

**🐧 Linux**

🔗 Despliegue en Linux  
<https://docs.flutter.dev/deployment/linux>

* Dependencias: GTK, CMake y clang
* Formatos soportados: .deb, .rpm y Snap
* Personalización: Íconos y metadatos

**🌐 Web**

🔗 Publicación para web  
<https://docs.flutter.dev/deployment/web>

* Optimización: flutter build web --web-renderer canvaskit
* Plataformas de hosting recomendadas:
  + Firebase: <https://firebase.google.com/docs/hosting>
  + GitHub Pages: gh-pages branch
  + Netlify/Vercel: Drag-and-drop deployment

**📌 Pasos Comunes**

1. Build general:
2. bash
3. flutter build <platform> # android, ios, windows, linux, web
4. Pruebas locales:
5. bash
6. flutter run -d <device> # chrome, edge, dispositivo físico
7. Requisitos previos:
   * Android: JDK 11+, Android Studio
   * iOS: Xcode 14+, macOS
   * Windows: Visual Studio 2022
   * Linux: GTK 3.0+
   * Web: Chrome para testing

**⚠️ Consideraciones Especiales**

* Android/iOS:
  + Necesitas cuentas de desarrollador
  + Proceso de revisión en tiendas (1-3 días)
* Windows/Linux:
  + No requieren tiendas oficiales
  + Puedes distribuir directamente los ejecutables
* Web:
  + Soporte para PWA (Service Workers)
  + Optimiza assets con --tree-shake-icons

**🚀 Flujo Recomendado**

1. Prueba en modo debug (flutter run)
2. Build en modo release (flutter build --release)
3. Verifica con DevTools <https://docs.flutter.dev/tools/devtools>
4. Sigue la guía específica de cada plataforma
5. Publica y monitorea crashes con Firebase Crashlytics

# 8. Recursos recomendados para aprender Flutter

**📚 Documentación Oficial**

* Flutter Docs  
  🔗 [https://docs.flutter.dev](https://docs.flutter.dev/)
* Dart Language  
  🔗 <https://dart.dev/language>

**🎓 Cursos Gratuitos**

1. Flutter Crash Course (Google)  
   🔗 <https://docs.flutter.dev/get-started/codelab>
2. Dart en Codecademy  
   🔗 <https://www.codecademy.com/learn/learn-dart>
3. Curso Completo de Flutter (YouTube - Fernando Herrera)  
   🔗 <https://www.youtube.com/watch?v=GXIJJkq_H8g&list=PLV6pYUAZ-ZoE6kzN1t9lfV9aYkRFYhwyj>

**📦 Paquetes y bibliotecas**

* Pub.dev (Repositorio Oficial)  
  🔗 [https://pub.dev](https://pub.dev/)
* Flutter Awesome (Inspiración UI)  
  🔗 [https://flutterawesome.com](https://flutterawesome.com/)

**📌 Libros**

1. "Flutter in Action" (Manning)  
   🔗 <https://www.manning.com/books/flutter-in-action>
2. "Dart Apprentice" (Ray Wenderlich)  
   🔗 <https://www.raywenderlich.com/books/dart-apprentice>

**💬 Comunidad**

* Stack Overflow (Flutter Tag)  
  🔗 <https://stackoverflow.com/questions/tagged/flutter>
* Reddit r/FlutterDev  
  🔗 <https://www.reddit.com/r/FlutterDev>
* Flutter Community en Medium  
  🔗 <https://medium.com/flutter-community>

**🛠️ Herramientas Clave**

1. Flutter DevTools (Debugging)  
   🔗 <https://docs.flutter.dev/tools/devtools>
2. Firebase para Flutter  
   🔗 [https://firebase.flutter.dev](https://firebase.flutter.dev/)
3. Riverpod (Gestión de Estado)  
   🔗 [https://riverpod.dev](https://riverpod.dev/)
4. VS Code + Extensión Flutter  
   🔗 <https://marketplace.visualstudio.com/items?itemName=Dart-Code.flutter>

**🌟 Extra: Proyectos Open-Source**

* Repositorio Oficial de Flutter  
  🔗 <https://github.com/flutter/flutter>
* Ejemplos de Apps en GitHub  
  🔗 [https://github.co/flutter/samples](https://github.com/flutter/samples)