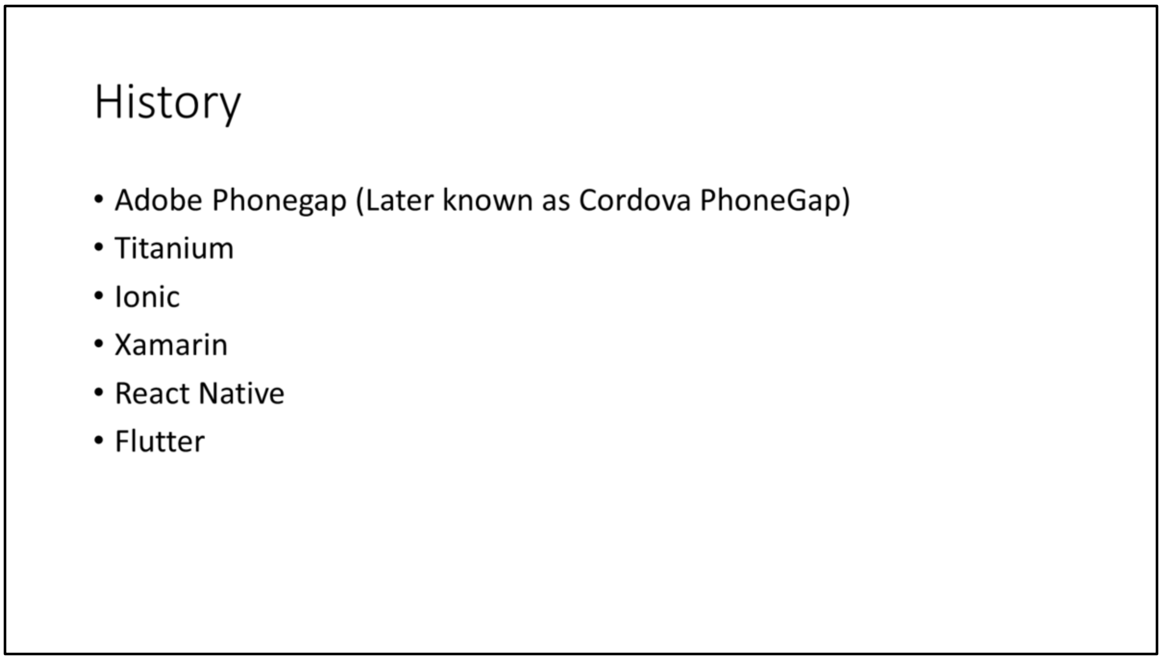
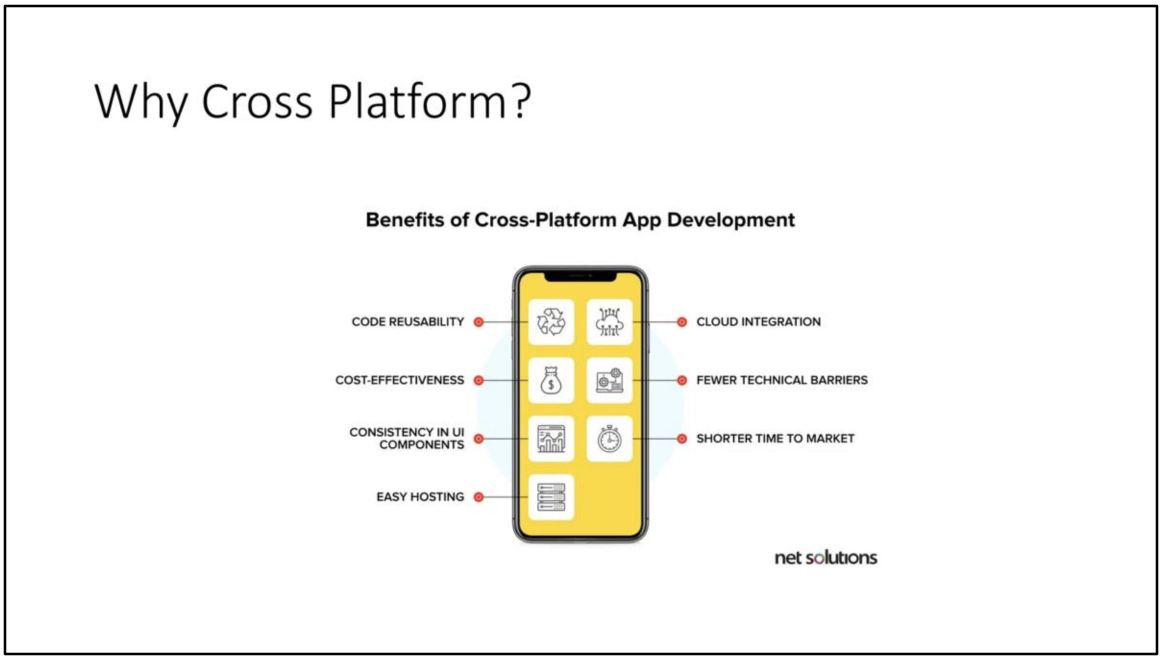
Cross Platform Application Development (Flutter)



History

- Adobe Phonegap (Later known as Cordova PhoneGap)
- Titanium
- Ionic
- Xamarin
- React Native
- Flutter



Why Cross Platform?

1. Maximum Exposure to the Target Audience
Leveraging a mobile cross-platform development approach enables you to build an app and deploy over various platforms, including the web. This means that by building a single app, one can target both – iOS and Android platforms, thus, maximizing their reach.

2. Reduced Development Cost
The cross-platform app development is based on a concept ‘write once, run everywhere’. Reusable codes and agile app development through tools can lessen the cost of development. Therefore, in order to improve your business on multiple platforms and tools in a cost-effective way, there is no other alternative to cross-platform apps.

3. Easier Maintenance & Deployment
Since there’s only one developed app that runs over all platforms, it is easier to maintain as well as deploy code or changes made. Updates can promptly be synced over all platforms and devices, thus saving time and money. Moreover, if a bug is found in the common codebase, it should be fixed once. In this way, developers can save a lot on time and money.

4. Quicker Development Process
The quick development process is another win-win situation when it comes to developing cross-platform apps. Single source code for multiple platforms can help reduce the development efforts by 50 to 80%. It helps you to get a feature-rich business app in less time. The team of developers can meet the expected deadlines in cross-platform app development.

5. Reusable Code
Another good thing about this platform is that the code can be used again and again. Instead of developers developing new codes for every platform, a single code can be reused. This saves time as well as resources because it eliminates repetition in the task of creating codes.

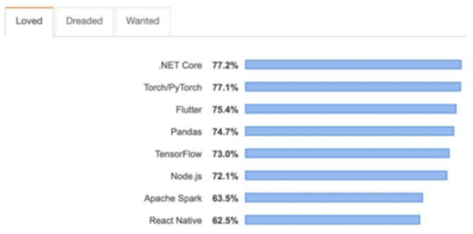
6. Easy Integration with Cloud
Cross-platform mobile apps are totally compatible and can take advantage of various plugins integrated with the cloud settings. In other words, the single source code is coordinated with various plug-ins and extensions to enhance the app's scalability and functionality.

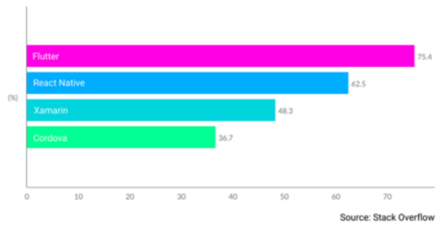
7. Faster Time-to-Market and Customization
As we mentioned above 'write once, run everywhere' is the concept that is followed while building cross-platform app development. It allows app developers to reduce Time-to-Market (TTM) with a quick deployment.
Also, if you need to transform or customize the app, it's easy for the developers to do minor changes in a single code. This, further, helps to deliver products more swiftly than the competitors by improving customer engagement.

8. Uniform Design
Users can recognize user interface (UI) elements, and foresee their interactions over various platforms. Therefore, User Experience (UX) is an important thing to consider for any app or software.
It's hard to sync the various development projects while developing multiple apps. Cross-platform mobile development tools allow developers as well as designers to build an uniform user experience that app users can enjoy.

Which Platform to go for and why?

Most Loved Frameworks, Libraries, and Tools





Xamarin. Built with #C and .Net, Xamarin allows developers to create cross-platform applications for Android, iOS, tvOS, macOS, and Windows. Xamarin applications with shared interfaces are developed using Xamarin.Forms. If your goal is to design a platform-specific interface, Xamarin.iOS and Xamarin.Android are the go-to tools. Xamarin provides powerful libraries to access native and 3rd-party APIs, and leverage smartphone hardware and functionality: sensors, camera, text messages, connectivity, etc. Applications created with Xamarin perform on par with native apps — even when it comes to rendering dynamic data in real time.

React Native. Unveiled by Facebook in 2015, React Native has reigned in the cross-platform app development market ever since. The framework is based on React — a JavaScript library for building highly responsive user interfaces. With React Native, you can create mobile applications that share up to 80% of their codebase and can access certain native features like the accelerometer and smartphone camera, although you might need separate code for iOS and Android for that.

Apache Cordova. Apache Cordova's tech stack features HTML5, CSS3, and JavaScript. The mobile app development framework provides access to a smartphone's built-in accelerometer, file storage, GPS, contact data, media, and notifications. Apache

Cordova boasts several advantages, including a fairly simple API and the opportunity to employ any JS framework. However, the platform visualizes app UIs through a web browser, which might cause lag. Also, some of Cordova's plugins are dated, so developers often have to write custom ones from scratch.

Flutter. Flutter is Google's UI toolkit that allows developers to create natively compiled applications for mobile devices, web browsers, and PCs using the same codebase. Flutter is based on Dart — a relatively new programming language that shares many features with Swift and Kotlin and can be transformed into JavaScript code. With Flutter, you can design applications that render fast and adapt to platform-specific UX logic. The framework is most suitable for MVP development.

Flutter

- Developed and released by Google in 2017.
- Flutter is a popular open source and free cross-platform framework.
- Uses Dart language to develop apps for Android, iOS, Mac, Windows, Linux and the Web

- **Famous Flutter Apps:** Google, eBay
  Alibaba and BMW

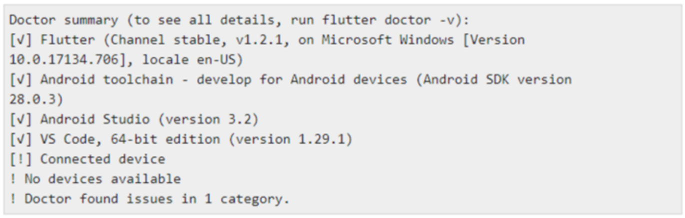
Step by Step - Installation Guide

- **Step 1** – Go to URL, https://flutter.dev/docs/get-started/install/windows and download the latest Flutter SDK.
- **Step 2** – Unzip the zip archive in a folder, say C:\flutter\
- **Step 3** – Update the system path to include flutter bin directory.
- **Step 4** – Flutter provides a tool, flutter doctor to check that all the requirement of flutter development is met.

```
flutter doctor
```

Step by Step - Installation Guide (Contd…)

- **Step 5** – Running the above command will analyze the system and show its report as shown below –



```
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, v1.2.1, on Microsoft Windows [Version
10.0.17134.706], locale en-US)
[√] Android toolchain - develop for Android devices (Android SDK version
28.0.3)
[√] Android Studio (version 3.2)
[√] VS Code, 64-bit edition (version 1.29.1)
[!] Connected device
! No devices available
! Doctor found issues in 1 category.
```

Note: The report says that all development tools are available but the device is not connected. We can fix this by connecting an android device through USB or starting an android emulator.

Step by Step - Installation Guide (Contd…)

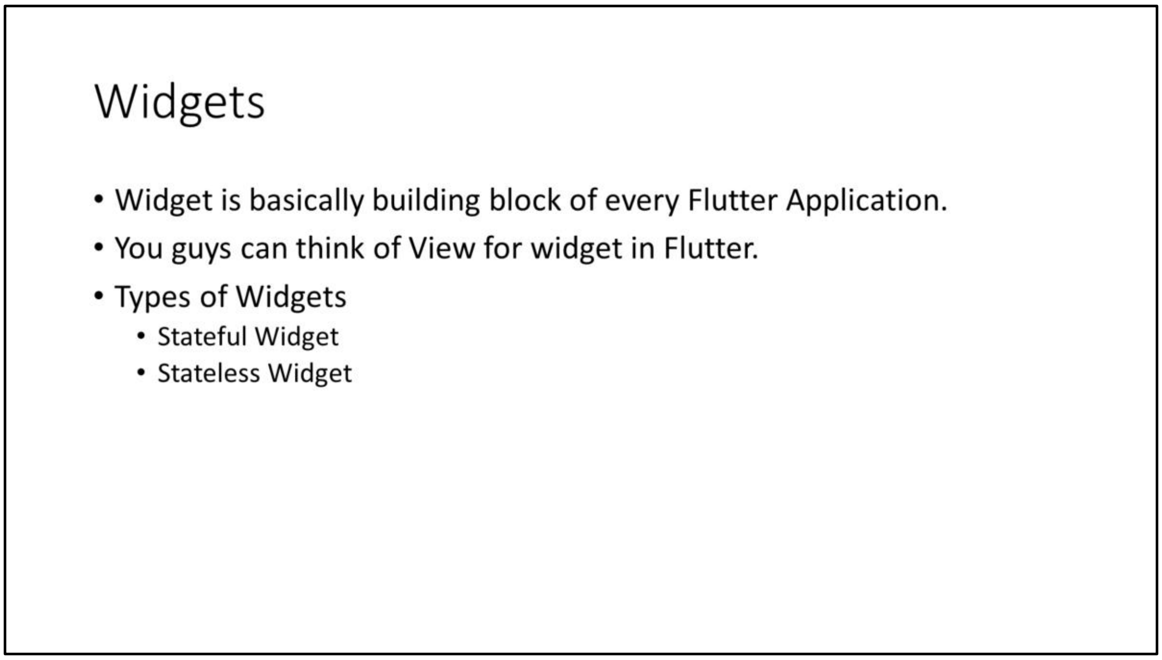
- **Step 6** – Install the latest Android SDK, <u>**if**</u> reported by flutter doctor
- **Step 7** – Install the latest Android Studio, **if** reported by flutter doctor
- **Step 8** – Start an android emulator or connect a real android device to the system.
- **Step 9** – Download the plugins required by the tool you installed

VSCode – Tool Installation

- Install VS Code, latest stable version.
  - Start VS Code.
  - Invoke **View > Command Palette…**.
  - Type “install”, and select **Extensions: Install Extensions**.
  - Type “flutter” in the extensions search field, select **Flutter** in the list, and click **Install**. This also installs the required Dart plugin.

Creating first flutter application

1. Invoke **View > Command Palette**.
2. Type “flutter”, and select the **Flutter: New Application Project**.
3. Create or select the parent directory for the new project folder.
4. Enter a project name, such as `myapp`, and press **Enter**.
5. Wait for project creation to complete and the `main.dart` file to appear

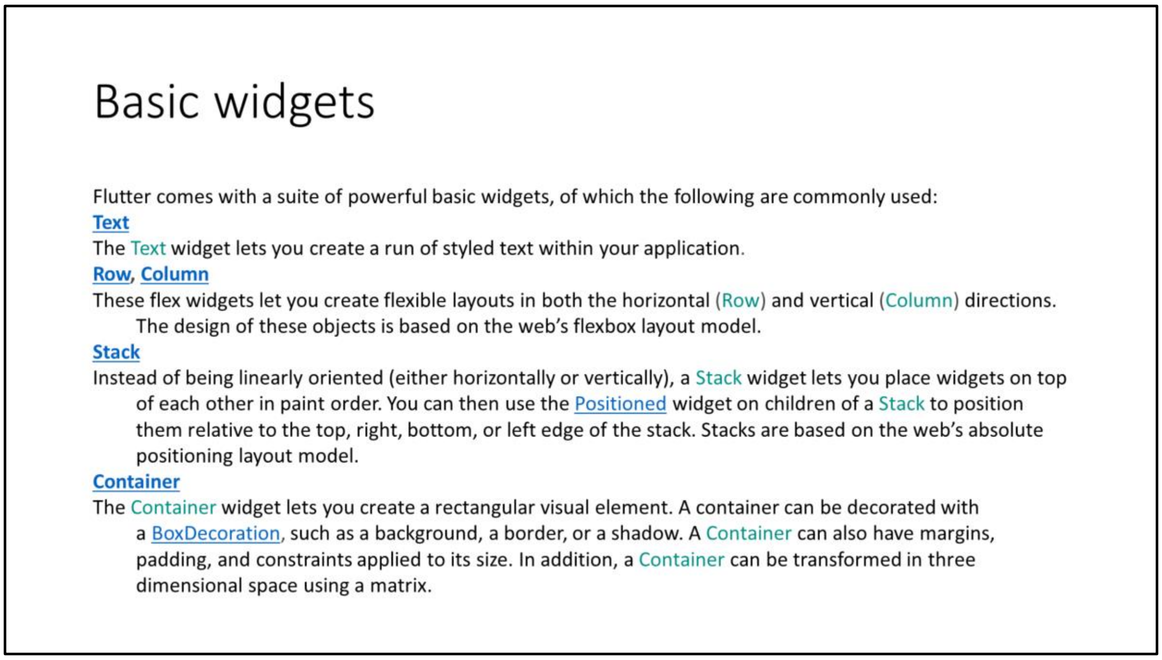


Widgets

- Widget is basically building block of every Flutter Application.
- You guys can think of View for widget in Flutter.
- Types of Widgets
  - Stateful Widget
  - Stateless Widget

Stateful or Stateless Widget

- A stateless widget never changes. Icon, IconButton, and Text are examples of stateless widgets. Stateless widgets subclass StatelessWidget.
- A stateful widget is dynamic: for example, it can change its appearance in response to events triggered by user interactions or when it receives data. Checkbox, Radio, Slider, InkWell, Form, and TextField are examples of stateful widgets. Stateful widgets subclass StatefulWidget.



Basic widgets

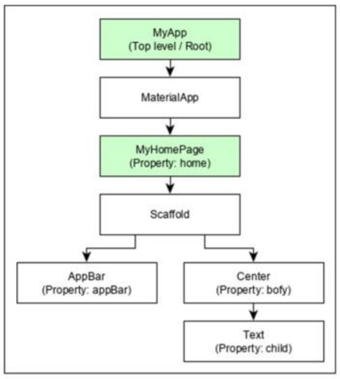
Flutter comes with a suite of powerful basic widgets, of which the following are commonly used:
Text
The Text widget lets you create a run of styled text within your application.
Row, Column
These flex widgets let you create flexible layouts in both the horizontal (Row) and vertical (Column) directions. The design of these objects is based on the web's flexbox layout model.
Stack
Instead of being linearly oriented (either horizontally or vertically), a Stack widget lets you place widgets on top of each other in paint order. You can then use the Positioned widget on children of a Stack to position them relative to the top, right, bottom, or left edge of the stack. Stacks are based on the web's absolute positioning layout model.
Container
The Container widget lets you create a rectangular visual element. A container can be decorated with a BoxDecoration, such as a background, a border, or a shadow. A Container can also have margins, padding, and constraints applied to its size. In addition, a Container can be transformed in three dimensional space using a matrix.

Many Material Design widgets need to be inside of a MaterialApp to display properly, in order to inherit theme data. Therefore, run the application with a MaterialApp.

The MyAppBar widget creates a Container with a height of 56 device-independent pixels with an internal padding of 8 pixels, both on the left and the right. Inside the container, MyAppBar uses a Row layout to organize its children. The middle child, the title widget, is marked as Expanded, which means it expands to fill any remaining available space that hasn't been consumed by the other children. You can have multiple Expanded children and determine the ratio in which they consume the available space using the flex argument to Expanded.

The MyScaffold widget organizes its children in a vertical column. At the top of the column it places an instance of MyAppBar, passing the app bar a Text widget to use as its title. Passing widgets as arguments to other widgets is a powerful technique that lets you create generic widgets that can be reused in a wide variety of ways.
Finally, MyScaffold uses an Expanded to fill the remaining space with its body, which consists of a centered message.

Flutter Application Hierarchy



MyApp is the user created widget and it is build using the Flutter native widget, *MaterialApp*.
MaterialApp has a home property to specify the user interface of the home page, which is again a user created widget, *MyHomePage*.
MyHomePage is build using another flutter native widget, *Scaffold*
Scaffold has two properties – *body* and *appBar*

> ***body*** is used to specify its main user interface and ***appBar*** is used to specify its header user interface

Header *UI* is build using flutter native widget, *AppBar* and *Body UI* is build using *Center* widget.
The ***Center*** widget has a property, *Child*, which refers the actual content and it is build using *Text* widget