Dart is a client-optimized language for developing fast apps on any platform. Its goal is to offer the most productive programming language for multi-platform development, paired with a flexible execution runtime platform for app frameworks.

Dart is designed for a technical envelope that is particularly suited to client development, prioritizing both development (sub-second stateful hot reload) and high-quality production experiences across a wide variety of compilation targets (web, mobile, and desktop).

Dart also forms the foundation of Flutter. Dart provides the language and runtimes that power Flutter apps, but Dart also supports many core developer tasks like formatting, analyzing, and testing code.

Dart: The language

The Dart language is type safe; it uses static type checking to ensure that a variable's value *always* matches the variable's static type.

Sometimes, this is referred to as sound typing.

Although types are mandatory, type annotations are optional because of type inference.

The Dart typing system is also flexible, allowing the use of a dynamic type combined with runtime checks, which can be useful during experimentation or for code that needs to be especially dynamic.

Dart offers sound null safety, meaning that values can't be null unless you say they can be. With sound null safety, Dart can protect you from null exceptions at runtime through static code analysis. Unlike many other null-safe languages, when Dart determines that a variable is non-nullable, that variable is *always* non-nullable. If you inspect your running code in the debugger, you'll see that non-nullability is retained at runtime (hence *sound* null safety).

Dart: The libraries

Dart has a rich set of core libraries, providing essentials for many everyday programming tasks:

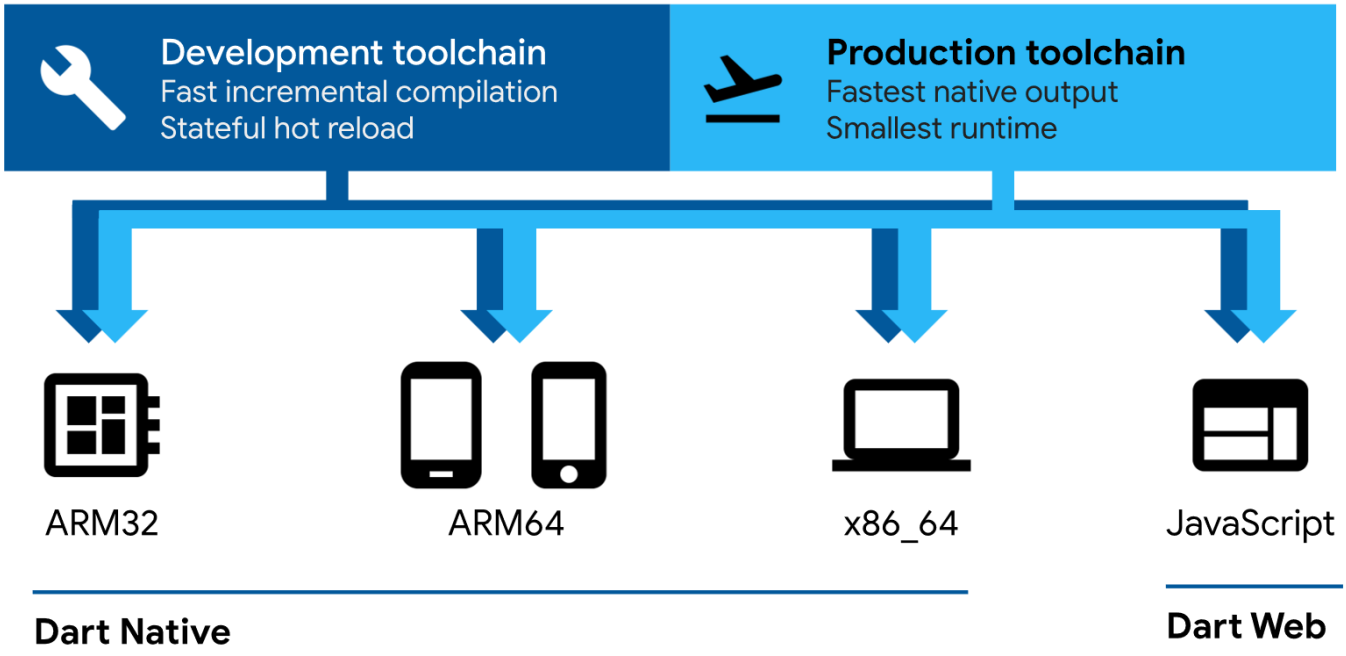
- Built-in types, collections, and other core functionality for every Dart program (dart:core)
- Richer collection types such as queues, linked lists, hashmaps, and binary trees (dart:collection)
- Encoders and decoders for converting between different data representations, including JSON and UTF-8 (dart:convert)
- Mathematical constants and functions, and random number generation (dart:math)
- File, socket, HTTP, and other I/O support for non-web applications (dart:io)

- Support for asynchronous programming, with classes such as Future and Stream (dart:async)
- Lists that efficiently handle fixed-sized data (for example, unsigned 8-byte integers) and SIMD numeric types (dart:typed_data)
- Foreign function interfaces for interoperability with other code that presents a C-style interface (dart:ffi)
- Concurrent programming using *isolates*—independent workers that are similar to threads but don't share memory, communicating only through messages (dart:isolate)
- HTML elements and other resources for web-based applications that need to interact with the browser and the Document Object Model (DOM) (dart:html)

Dart: The platforms

Dart's compiler technology lets you run code in different ways:

- **Native platform**: For apps targeting mobile and desktop devices, Dart includes both a Dart VM with just-in-time (JIT) compilation and an ahead-of-time (AOT) compiler for producing machine code.
- **Web platform**: For apps targeting the web, Dart includes both a development time compiler (dartdevc) and a production time compiler (dart2js). Both compilers translate Dart into JavaScript.



The Flutter framework is a popular, multi-platform UI toolkit that's powered by the Dart platform, and that provides tooling and UI libraries to build UI experiences that run on iOS, Android, macOS, Windows, Linux, and the web.

Dart Native (machine code JIT and AOT)

During development, a fast developer cycle is critical for iteration. The Dart VM offers a just-in-time compiler (JIT) with incremental recompilation (enabling hot reload), live metrics collections (powering DevTools), and rich debugging support.

When apps are ready to be deployed to production—whether you're publishing to an app store or deploying to a production backend—the Dart AOT compiler enables ahead-of-time compilation to native ARM or x64 machine code. Your AOT-compiled app launches with consistent, short startup time.

The AOT-compiled code runs inside an efficient Dart runtime that enforces the sound Dart type system and manages memory using fast object allocation and a generational garbage collector.

Dart Web (JavaScript dev & prod)

Dart Web enables running Dart code on web platforms powered by JavaScript. With Dart Web, you compile Dart code to JavaScript code, which in turn runs in a browser—for example, V8 inside Chrome.

Dart web contains both an incremental dev compiler enabling a fast developer cycle, and an optimizing production compiler, dart2js, which compiles Dart code to fast, compact, deployable JavaScript using techniques such as dead-code elimination.

Regardless of which platform you use or how you compile your code, executing the code requires a Dart runtime. This runtime is responsible for the following critical tasks:

- Managing memory: Dart uses a managed memory model, where unused memory is reclaimed by a garbage collector (GC).
- Enforcing the Dart type system: Although most type checks in Dart are static (compile-time), some type checks are dynamic (runtime). For example, the Dart runtime enforces dynamic checks by type check and cast operators.
- Managing isolates: The Dart runtime controls the main isolate (where code normally runs) and any other isolates that the app creates.

On native platforms, the Dart runtime is automatically included inside self-contained executables, and is part of the Dart VM provided by the "`dart run`" command.