---

title: Introduction to the Ethereum stack description: A walkthrough of the different layers of the Ethereum stack and how they fit together. lang: en

---

Like any software stack, the complete "Ethereum stack" will vary from project to project depending on your goals.

There are, however, core components of Ethereum that help provide a mental model for how software applications interact with the Ethereum blockchain. Understanding the layers of the stack will help you understand the different ways that Ethereum can be integrated into software projects.

Level 1: Ethereum Virtual Machine {#ethereum-virtual-machine}

The Ethereum Virtual Machine (EVM) is the runtime environment for smart contracts on Ethereum. All smart contracts and state changes on the Ethereum blockchain are executed by transactions. The EVM handles all of the transaction processing on the Ethereum network.

As with any virtual machine, the EVM creates a level of abstraction between the executing code and the executing machine (an Ethereum node). Currently, the EVM is running on thousands of nodes distributed across the world.

Under the hood, the EVM uses a set of opcode instructions to execute specific tasks. These (140 unique) opcodes allow the EVM to be Turing-complete, which means the EVM is able to compute just about anything, given enough resources.

As a dapp developer, you don't need to know much about the EVM other than it exists and that it reliably powers all applications on Ethereum without downtime.

Level 2: Smart contracts {#smart-contracts}

Smart contracts are the executable programs that run on the Ethereum blockchain.

Smart contracts are written using specific programming languages that compile to EVM bytecode (low-level machine instructions called opcodes).

Not only do smart contracts serve as open source libraries, they are essentially open API services that are always running and can't be taken down. Smart contracts provide public functions which users and applications (dapps) may interact with, without needing permission. Any application may integrate with deployed smart contracts to compose functionality, such as adding data feeds or to support token swaps. Additionally, anyone can deploy new smart contracts to Ethereum in order to add custom functionality to meet their application's needs.

As a dapp developer, you'll need to write smart contracts only if you want to add custom functionality on the Ethereum blockchain. You may find you can achieve most or all of your project's needs by merely integrating with existing smart contracts, for instance if you want to support payments in stablecoins or enable decentralized exchange of tokens.

Level 3: Ethereum nodes {#ethereum-nodes}

In order for an application to interact with the Ethereum blockchain, it must connect to an Ethereum node. Connecting to a node allows you to read blockchain data and/or send transactions to the network.

Ethereum nodes are computers running software - an Ethereum client. A client is an implementation of Ethereum that verifies all transactions in each block, keeping the network secure and the data accurate. **Ethereum nodes are the Ethereum blockchain**. They collectively store the state of the Ethereum blockchain and reach consensus on transactions to mutate the blockchain state.

By connecting your application to an Ethereum node (via the JSON-RPC API), your application is able to read data from the blockchain (such as user account balances) as well as broadcast new transactions to the network (such as transferring ETH between user accounts or executing functions of smart contracts).

Level 4: Ethereum client APIs {#ethereum-client-apis}

Many convenience libraries (built and maintained by Ethereum's open source community) allow your applications to connect to and communicate with the Ethereum blockchain.

If your user-facing application is a web app, you may choose to `npm install` a JavaScript API directly in your frontend. Or perhaps you'll choose to implement this functionality server-side, using a Python or Java API.

While these APIs are not a necessary piece of the stack, they abstract away much of the complexity of interacting directly with an Ethereum node. They also provide utility functions (e.g. converting ETH to Gwei) so as a developer you can spend less time dealing with the intricacies of Ethereum clients and more time focused on the functionality specific to your application.

Level 5: End-user applications {#end-user-applications}

At the top level of the stack are user-facing applications. These are the standard applications you regularly use and build today: primarily web and mobile apps.

The way you develop these user interfaces remains essentially unchanged. Often users will not need to know the application they're using is built using a blockchain.

Ready to choose your stack? {#ready-to-choose-your-stack}

Check out our guide to set up a local development environment for your Ethereum application.

Further reading {#further-reading}

- The Architecture of a Web 3.0 application - *Preethi Kasireddy*

Know of a community resource that helped you? Edit this page and add it!