**Ethereum Virtual Machine (EVM)**

Ethereum is a programmable blockchain. Rather than give users a set of pre-defined operations (e.g. bitcoin transactions), Ethereum allows users to create their own operations of any complexity they wish. In this way, it serves as a platform for many different types of decentralized blockchain applications, including but not limited to cryptocurrencies.

Ethereum in the narrow sense refers to a suite of protocols that define a platform for decentralised applications. At the heart of it is the [Ethereum Virtual Machine (“EVM”)](http://ethdocs.org/en/latest/contracts-and-transactions/developer-tools.html#the-evm), which can execute code of arbitrary algorithmic complexity. In computer science terms, Ethereum is “Turing complete”. Developers can create applications that run on the EVM using friendly programming languages modelled on existing languages like JavaScript and Python.

Like any blockchain, Ethereum also includes a peer-to-peer network protocol. The Ethereum blockchain database is maintained and updated by many nodes connected to the network. Each and every node of the network runs the EVM and executes the same instructions.

# Web 3: A platform for decentralized apps

Many have come to believe that an open, trustless blockchain platform like Ethereum is perfectly suited to serve as the shared “back end” to a decentralized, secure internet - Web 3.0. An internet where core services like DNS and digital identity are decentralized, and where individuals can engage in economic interactions with each other.

As intended by the Ethereum developers, Ethereum is a blank canvas and you have the freedom to build whatever you want with it. The Ethereum protocol is meant to be generalized so that the core features can be combined in arbitrary ways. Ideally, dapp projects on Ethereum will leverage the Ethereum blockchain to build solutions that rely on decentralized consensus to provide new products and services that were not previously possible.

Ethereum is perhaps best described as an ecosystem: the core protocol is supported by various pieces of infrastructure, code, and community that together make up the Ethereum project. Ethereum can also be understood by looking at the projects that use Ethereum. Already, there are a number of high-profile projects built on Ethereum such as Augur, Digix, Maker, and many more (see [Dapps](http://ethdocs.org/en/latest/contracts-and-transactions/developer-tools.html#dapps)). In addition, there are development teams that build open source components that anyone can use. While each of these organizations are separate from the Ethereum Foundation and have their own goals, they undoubtedly benefit the overall Ethereum ecosystem.

### Relation Between Web3, RPC, Geth

Dapp development requires an understanding of the (1)Web3 Javascript API, the (2)JSON RPC API, and the (3)Solidity programming language.

1. [Web3 JavaScript API](https://github.com/ethereum/wiki/wiki/JavaScript-API) - This is the main JavaScript SDK to use when you want to interact with an Ethereum node.

To make your Ðapp work on Ethereum, you can use the web3 object provided by the [web3.js library](https://github.com/ethereum/web3.js). Under the hood it communicates to a local node through [RPC calls](https://github.com/ethereum/wiki/wiki/JSON-RPC). web3.js works with any Ethereum node, which exposes an RPC layer.

web3 contains the eth object - web3.eth (for specifically Ethereum blockchain interactions) and the shh object - web3.shh (for Whisper interaction)

## JavaScript API

|  |  |
| --- | --- |
| C++ | http://localhost:8545 |
| Go | http://localhost:8545 |
| Parity | http://localhost:8545 |
| Py | http://localhost:4000 |

To talk to an ethereum node from inside a JavaScript application use the [web3.js](https://github.com/ethereum/web3.js) library, which gives an convenient interface for the RPC methods. See the [JavaScript API](https://github.com/ethereum/wiki/wiki/JavaScript-API) for more.

## JSON-RPC Endpoint

Default JSON-RPC endpoints:

About Private Network

### The genesis file

The genesis block is the start of the blockchain - the first block, block 0, and the only block that does not point to a predecessor block. The protocol ensures that no other node will agree with your version of the blockchain unless they have the same genesis block, so you can make as many private testnet blockchains as you’d like!

CustomGenesis.json

{ "nonce": "0x0000000000000042", "timestamp": "0x0", "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000", "extraData": "0x0", "gasLimit": "0x8000000", "difficulty": "0x400", "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000", "coinbase": "0x3333333333333333333333333333333333333333", "alloc": { } }

Save a file called CustomGenesis.json. You will reference this when starting your geth node using the following flag:

--genesis /path/to/CustomGenesis.json

### Command line parameters for private network

There are some command line options (also called “flags”) that are necessary in order to make sure that your network is private. We already covered the genesis flag, but we need a few more. Note that all of the commands below are to be used in the geth Ethereum client.

--nodiscover

Use this to make sure that your node is not discoverable by people who do not manually add you. Otherwise, there is a chance that your node may be inadvertently added to a stranger’s blockchain if they have the same genesis file and network id.

--maxpeers 0

Use maxpeers 0 if you do not want anyone else connecting to your test chain. Alternatively, you can adjust this number if you know exactly how many peers you want connecting to your node.

--rpc

This will enable RPC interface on your node. This is generally enabled by default in Geth.

--rpcapi "db,eth,net,web3"

This dictates what APIs that are allowed to be accessed over RPC. By default, Geth enables the web3 interface over RPC.

**IMPORTANT: Please note that offering an API over the RPC/IPC interface will give everyone access to the API who can access this interface (e.g. dapp’s). Be careful which API’s you enable. By default geth enables all API’s over the IPC interface and only the db,eth,net and web3 API’s over the RPC interface.**

--rpcport "8080"

Change 8000 to any port that is open on your network. The default for geth is 8080.

--rpccorsdomain "http://chriseth.github.io/browser-solidity/"

This dictates what URLs can connect to your node in order to perform RPC client tasks. Be very careful with this and type a specific URL rather than the wildcard (\*) which would allow any URL to connect to your RPC instance.

--datadir "/home/TestChain1"

This is the data directory that your private chain data will be stored in (under the nubits . Choose a location that is separate from your public Ethereum chain folder.

--port "30303"

This is the “network listening port”, which you will use to connect with other peers manually.

--identity "TestnetMainNode"

This will set up an identity for your node so it can be identified more easily in a list of peers. Here is an example of how these identities show up on the network.

### Launching geth

After you have created your custom genesis block JSON file and created a directory for your blockchain data, type the following command into your console that has access to geth:

geth --identity "MyNodeName" --genesis /path/to/CustomGenesis.json --rpc --rpcport "8080" --rpccorsdomain "\*" --datadir "C:\chains\TestChain1" --port "30303" --nodiscover --rpcapi "db,eth,net,web3" --networkid 1999 console

**Note**

Please change the flags to match your custom settings.

You will need to start your geth instance with your custom chain command every time you want to access your custom chain. If you just type “geth” in your console, it will not remember all of the flags you have set.