Private Blockchain Creation Process
Introduction

We will go through a private blockchain setup using Geth. Geth is an open-source command-line Ethereum implementation using the Go programming language.

Building a private blockchain is done for special use cases where we want to use Ethereum technology but keep the blockchain limited to a set of participants.

Please look at the video in the *Practitioner's Perspective* section for a detailed walkthrough.

Environment Setup (Step 1)

Go to the downloads section of Ethereum geth - https://geth.ethereum.org/downloads/ - and download the binary for your operating system. Please always pick the **'Geth and Tools'** option so that you get other tools like puppeth as well.

It's a straightforward installation but you can look at the installation guides if needed - https://geth.ethereum.org/docs/install-and-build/installing-geth

Additionally, you'll need:

- A command terminal (Use the terminal in VS Code for convenience, if you don't have an existing preference)
- Google Chrome with Metamask plugin installed and set up
- And, of course, a working internet connection for the installation and remix/metamask interaction

Once everything is installed, create a folder for your private blockchain, let's say gethpoa.

Create an Account on Node 1 (Step 2)

1. Open a terminal, whether in VS Code or otherwise
2. Go to the gethpoa folder
3. Create a node1 folder using **mkdir node1** command
4. Go to the node1 folder using **cd node1** command
5. Run the command **geth --datadir ./data account new**
   This will create a new account under the node1 folder with the data being saved in the data folder, within node1
6. Enter and re-enter the password
7. You'll get the public address of the key on the console which will look something like '0xb6514b5085fE8dbA97519532Af6B61f77bBF7BEa'
   Save that and the password/passcode under Account1 heading in an info.txt file, under your gethpoa folder

Create an Account on Node 2 (Step 3)

1. Navigate to the gethpoa folder from your current node1 folder by going up a level (**cd ..**)
2. Create a node2 folder using **mkdir node2** command
3. Go to the node2 folder using **cd node2** command
4. Run the command **geth --datadir ./data account new**
5. This will create a new account under the node2 folder with the data being saved in the data folder, within node2
6. Enter and re-enter the password
7. You'll get the public address of the key on the console which will look something like '0xb6514b5085fE8dbA97519532Af6B61f77bBF7BEa'
8. Save that and the password/passcode under Account2 heading in the info.txt file, under your gethpoa folder

Configure the Genesis File (Step 4)

The genesis file defines the initial conditions of the blockchain including selecting the consensus protocol, pre-funding specific accounts, defining the sealers (equivalent to miners) in case of Proof-of-Authority, and more.

It is used to initialize the blockchain. The very first block, called the genesis block, is crafted based on the parameters in the genesis.json file .

We will use a tool called *puppeth*, which comes pre-built with *geth*, to configure the genesis file.

1. Go back to the gethpoa folder and type **puppeth**
2. Pick any network name (we'll pick **blockpoa**) and hit enter
3. Select option 2 - 'Configure new genesis', and hit enter
4. Select option 1 - 'Create new genesis from scratch', and hit enter
5. Select option 2 - 'Clique - proof-of-authority', and hit enter. We are picking PoA here so that blocks can be created faster and with less computation. It's a private blockchain so we can define only trusted nodes as block creators (called sealers in PoA)
6. Enter 10 - 'How many seconds should blocks take? (default = 15)'. This will define that a block will be created every 10 seconds. You can, of course, choose any value
7. 'Which accounts are allowed to seal? (mandatory at least one)' - Take the public address of Account1 saved earlier and enter it as a sealing account
8. 'Which accounts should be pre-funded? (advisable at least one)' - Enter both Account1 and Account2 public addresses so that both get some initial cryptocurrency
9. 'Should the precompile-addresses (0x1 .. 0xff) be pre-funded with 1 wei? (advisable yes)' - Type yes, and hit enter
10. 'Specify your chain/network ID if you want an explicit one (default = random)' - This will be the chain/network ID for your private blockchain. You can pick any random high number or even let it generate one randomly by default. We'll type 14333 here, but anything is fine. We don't pick low numbers so that we don't clash with public mainnet and testnets, in case we use the same node to interact with those blockchains separately.
11. It would confirm that new genesis block configuration is done and will ask you for the next steps. Select option 2 - 'Manage existing genesis', and hit enter
12. Select option 2 - 'Export genesis configurations', and hit enter. This will put the genesis file in your top-level folder in various formats
13. You can open blockpoa.json and observe the config file properties. Please make sure not to change anything

Initializing Both Nodes (Step 5)

1. Go to the node1 folder - **cd node1**
2. Run the command - **geth --datadir ./data init ../blockpoa.json**
   This will initiate this node with your genesis file configuration setting up your private blockchain on node 1
3. Go to the node2 folder - **cd node2**
4. Run the command - **geth --datadir ./data init ../blockpoa.json**
   This will initiate this node with your genesis file configuration setting up your private blockchain on node 2

Create and Start the Boot Node (Step 6)

A boot node is a stripped-down version of an Ethereum node, mainly running the network node discovery protocol and helping other nodes connect to each other. It doesn't participate in mining or transaction processing.

If you don't run a boot node, then you have to manually add the peers in each node using the addPeer and other relevant commands.

1. Go back to the gethpoa folder
2. Create a bnode folder using the **mkdir bnode** command
3. Go to the bnode folder using the **cd bnode** command
4. Run the command - **bootnode -genkey boot.key** - to initialize your bootnode. This would print the node address of your boot node. Please take and save it in your info.txt file, under a Bootnode Enode heading. It'll look something like this:
   enode://c1f11d32ccc9f68ac48a2b3a6a2cd4b5828454b3f5fa8cce66a50e7d24e2e8ab1a3c72ca065a06e6951729f53e433954e714f2cd3be4ac48dfd5029071293df9@127.0.0.1:0?discport=30301
5. Run the command - **bootnode -nodekey ./boot.key -verbosity 7 -addr 127.0.0.1:30301** - to start the boot node This will start the boot node with a high verbosity level so that it'll print all the interaction with other nodes

Start Nodes 1 and 2 (Step 6)

Now, we'll start both node1 and node2 after appropriately modifying their argument values.

1. Create **password.txt** in both node1 and node2 directory, saving just the password/passcode value of Account1 and Account 2 in each one of them respectively.
2. In a new terminal, we'll start node 1. First go to the node1 folder. The command to start node 1 will be something like this:
   geth --networkid 14333 --datadir ./data --bootnodes enode://7526eabd19c67ef1d0596259e6567059bc55f0b938186207d080d07f8d46538e159279ab99768e80c3f02112c98d28c7a799c09309ad74fd1c53c1e1ede436a7@127.0.0.1:30301 --port 30303 --ipcdisable --syncmode 'full' --http --allow-insecure-unlock –http.corsdomain "*" --http.port 8545 --unlock '0xc143ff9cf9457f814f72fed89eaae7b0103353cc' --password password.txt --mine console
   a. networkid should be the id that you gave earlier. We gave 14333 in this example
   b. datadir will remain data under your node folder
   c. bootnodes - Here we'll specify the enode value we saved earlier from our bootnode setup
   d. port specifies your local IPC port that can be used by other nodes on this machine to interact with your node. http.port specifies your RPC port that nodes on other machines can use to interact with your node. As you can see, we need to set different values for these for different nodes running on the same machine.
   e. password tells the node to get the passcode from a specific file whenever it needs to unlock credentials of an account for mining or transaction functionality.
   f. mine signifies that this node should automatically be started with the mining functionality based on the genesis-defined consensus protocol
   g. unlock defines which account to unlock by default using the passcode
3. Change the value of bootnodes, and of the unlock account (Account1 public address) and run the command

4. You'll see that Node 1 gets started and starts communicating with the boot node. It'll also start mining a block every ten seconds. You can also see 30303 port number coming up in bootnode's console logs.
5. Go to the node2 folder and follow the same instructions with node 2 information. Please use the following command and make changes in it for bootnode enode value and Account 2 address. This already has different port numbers for Node 2. You'll observe the same effects
   geth --networkid 14333 --datadir ./data --bootnodes enode://7526eabd19c67ef1d0596259e6567059bc55f0b938186207d080d07f8d46538e159279ab99768e80c3f02112c98d28c7a799c09309ad74fd1c53c1e1ede436a7@127.0.0.1:30301 --port 30304 --ipcdisable --syncmode 'full' --http --allow-insecure-unlock –http.corsdomain "*" --http.port 8546 --unlock '0x5a10c665fb52206473dab68790ffc3740df870de' --password password.txt console
6. You should now have the private blockchain running - one boot node and two active nodes (Node 1 and Node 2)

Basic operations (Step 7)

1. You can check the balance of both accounts by running the following command: **eth.getBalance('<address>')**
   Put the address of the account without '0x' in front
2. You can initiate transfer from Account 1 to Account 2 for example: **eth.sendTransaction({from: "<Account 1 address>", to: "<Account 2 address>", value: "<value in Wei"})**
   Put the addresses with 0x in front. Value will be in Wei so put a large value to see the balance change easily
3. Node 1 will automatically mine this transaction and you can see that happening in the logs. If you re-check the account balances now, you can see the appropriate balance change

Contract interaction on your Private Blockchain, using Remix/Metamask (Step 8)

You should follow the videos for detailed steps for the following:

1. Configure localhost with port 8545 in Metamask to allow Node 1 to be used as the processing node
2. Open up Remix and connect to Metamask with the ‘Injected Web3’ option. You’ll see the name and port number right below the options
3. Going as per the video, you can deploy and run a test contract on your Node 1. You can use the default example of ‘Storage.sol’ that comes with Remix, under the contracts folder
4. You can see another transaction in your Node 1 logs which is for the contract creation that you just did
5. You’ll also see further transactions if you try to run it by using the store function in the contract
6. You can do the same process using Node 2 as well