Lecture Notes

Smart Contract Development in Ethereum

Setting Up a Private Ethereum Blockchain

In this session, you learnt about:

- Ethereum networks and clients
- Geth Ethereum client
- Installing the Geth client on your system
- Familiarising yourself with the Geth CLI and its commands
- Creating a private blockchain and connecting to it using the Geth client
- Creating an account on the network
- Performing transactions from one account to the other
- Creating another node and adding it to the same network

Pre-Requisites

As a prerequisite to proceed with this session, you need to either have Mac OS or some Linux OS like Ubuntu installed on your machine.

For Windows users, before you move ahead with the module, ensure that you have access to Ubuntu using any one of the following three approaches:

1. You can dual boot your system. Please refer to **this** link for detailed instructions on how to install Ubuntu and dual boot with Windows and Ubuntu. The link works for users of Windows 10. This is the most recommended approach.
2. You can use virtualisation applications like Oracle VirtualBox. Please refer to **this** link for detailed instructions on how to install and run Ubuntu on VirtualBox.
3. You can install the Ubuntu app as well. This will also serve the purpose. Please refer to the following video for the instructions on the same. This is the least recommended approach.

Ethereum Networks

Ethereum is an open-source, blockchain-based decentralised software platform used for its own cryptocurrency ether. It enables smart contracts and distributed applications (DApps) to be built and run without any downtime, fraud, control, or interference from a third party. Such multiple Ethereum nodes together form an Ethereum network. There are multiple such Ethereum networks. They are independent of each other, and anyone can create their own private Ethereum network. They are defined or identified by a unique parameter called **chain ID**. You can build your own private Ethereum network with a chain ID that does not belong to any of the existing networks on Ethereum.

Following are the three main public Ethereum networks:

- **Frontier:** Frontier is the main public network. This is the blockchain network that you refer to when talking about Ethereum. It is this network where ethers have real values. Frontier's chain ID is 1.
- **Ropsten:** Ropsten is the most popular test network. Before deploying your applications on the main Frontier network, you should test them on Ropsten. Ropsten is identical to the Frontier network, but the ethers here have no real value. Ropsten's chain ID is 3.
- **Rinkeby:** Rinkeby is the most popular test network with the proof of authority (PoA) mining algorithm. It is similar to the Ropsten network, except that the mining algorithm used in Rinkeby is PoA. You can test your smart contracts or applications on Rinkeby before deploying them on a blockchain network that uses the PoA mining algorithm. Rinkeby's chain ID is 4.

Ethereum Clients

In order to communicate with blockchain, we must use a blockchain **client**. The client is a piece of software capable of establishing a P2P communication channel with other clients, signing and broadcasting transactions, mining, deploying and interacting with smart contracts, etc. The client is often referred to as a **node**.
There are multiple Ethereum networks in the world with a unique chain ID. Each network consists of several interconnected nodes. To connect to a network, you need to become a node to this network. This node is called the client. You need to download an Ethereum client on your machine and then connect to the network through that. This is important because you can only connect to a network as an Ethereum client.
Ethereum developers did not create an Ethereum client. Instead, they released a yellow paper called Protocol. Anyone can create his/her own version of an Ethereum client using the instructions mentioned in the Protocol.

Following are the most popular Ethereum client implementations:

- **Go Ethereum (Geth)**
- **Parity**
- **Cpp-Ethereum (Aleth)**

- **Trinity**

Introduction to Go Ethereum (Geth)

Go **Ethereum**, also known as **Geth** is a command line interface which allows us to run a full **Ethereum** node. **Geth** is implemented in GO and it allows us to mine blocks, generate **ether**, deploy and interact with smart contracts, transfer funds, block history inspection, create accounts, etc.
You can install Geth in any operating system. Using Geth, you can connect to any Ethereum network.

Geth Installation and Geth CLI

This is the link to find the instructions/commands to install Geth on your machine.

This is the link to the page containing the list of Geth command line options (CLIs).

This is the link to the list of management APIs.

Setting Up A Private Ethereum Blockchain

Genesis Block

The Genesis **block** is the first block in the chain. The Genesis **file** is a JSON file that defines the characteristics of that initial block and subsequently the rest of the blockchain.
In this file, you defined the different parameters for the genesis block, which indicates the starting state of your blockchain. The genesis block is almost always hardcoded.

Following are the parameters needed to specify in genesis.json:

- **alloc**: This is used to list down various Ethereum accounts that you want to allot a certain amount of ether. This will not help you create an account. So, ensure it is an account you already have control of.

- **Config:**
  - chainId: This is the id of the private Ethereum network.
  - homesteadBlock, eip155Block and eip158Block: These relate to chain forking and versioning. So, in our case, let’s leave them at 0 since we are starting a new blockchain.

- **difficulty:** This dictates how difficult it is to mine a block. Setting this value low (~10–10000) is helpful in a private blockchain as it lets you mine blocks quickly (which equals fast transactions) and provides plenty of ETH to test with. For comparison, the Ethereum mainnet genesis file defines a difficulty of 17179869184.

- **gasLimit:** This is the total amount of gas that can be used in each block. With such a low mining difficulty, blocks will be moving pretty quickly, but you should still set this value pretty high to avoid hitting the limit and slowing down your network.
- **Nonce:** It is a mining nonce which increases by 1 every time you mine a block.
- **Mixhash:** It is the hash of the current block.
- **Coinbase:** It is the account address where rewards will get stored.
- **Parenthash:** It is the hash of the parent block.
- **Timestamp:** It is the creation time of the block.

Start an Ethereum network using this *genesis* file

1. Create a Geth client: Connect to a private network
   - First create a private chain data for your blockchain
   - Start the private network and console for this network
2. Create two accounts on this Geth client
   - Create two accounts on this network and start the mining process
   - Check the balance of an account
   - Convert the balance from *Wei to ether*

3. Transfer ethers from one account to the other
   - Send transactions from one account to another
     - Unlock the account from which you are trying to send the ethers
     - Successfully transfer ethers from one account to another

4. Create a second peer/node: Join the same private network
   - Create another peer
   - Using the *enode* ID of the first peer, connect the second peer to the first peer

Refer this document for detailed steps with commands used in each step.

JSON files

JSON stands for JavaScript Object Notation. JSON objects are used for transferring data between the server and the client. XML serves the same purpose. However, JSON objects have several advantages over XML and we are going to discuss them in this segment along with JSON concepts and its usages.

Let’s have a look at the piece of the following JSON data, which basically has key value pairs:

```
var chaitanya = {
   "firstName" : "Chaitanya",
   "lastName" : "Singh",
   "age" :  "28"
};
```

Features of JSON

- It is lightweight.
- It is language independent.
- It is easy to read and write.
- It is a text-based, human-readable data exchange format.

Let’s see how JSON looks when you store the records of four students in a text-based format so as to retrieve it later when required.

JSON style

```
{"students":[
   {"name":"John", "age":"23", "city":"Agra"},
   {"name":"Steve", "age":"28", "city":"Delhi"},
   {"name":"Peter", "age":"32", "city":"Chennai"},
   {"name":"Chaitanya", "age":"28", "city":"Bangalore"}
]}
```

Important Data Type Used in JSON

Data Type	Description
Number	It includes real numbers, integer or floating numbers.
String	It consists of any text or Unicode double-quoted with backslash escapement.
Boolean	The boolean data type represents either true or false values.
Null	The null value denotes that the associated variable does not have any value.
Object	It is a collection of key value pairs and is always separated by a comma and enclosed in curly brackets.
Array	It is an ordered sequence of values separated by a ',' (comma).

Number

- The number is a double-precision floating-point format which depends on its implementation method.
- In JSON you cannot use hexadecimal and octal formats.

Syntax:

```
var json-object-name = { string : number_value,......}
```

Example:

```
var obj = {salary: 2600}
```

String:

It is a series of double-quoted Unicode characters and has backslash escaping.
The following table shows various string types:

Syntax:

```
var json-object-name = { string : "string value",…..}
```

Example:

```
var obj= {name: 'Andy'}
```

Boolean

It stores only true or false values.

Syntax:
```
var json-object-name = {string : true/false, …..}
```

Example:
```
var obj = {active: 'true'}
```

Array

- It is an ordered collection of values.
- You should use an array when the key names are sequential integers.
- It should be enclosed inside square brackets which should be separated by a ',' (comma).

Syntax:
```
[value, .......]
```

Example showing an array storing multiple objects:
```
{
   "eBooks":[
      {
         "language":"Pascal",
         "edition":"third"
      },
      {
         "language":"Python",
         "edition":"four"
      },
      {
         "language":"SQL",
         "edition":"second"
      }
   ]
}
```

Object

- An object should be enclosed in curly braces.
- It should be an unordered set of name or value pairs.
- Name should be followed by ":" (colon) and the name/value pairs need to be separated using "," (comma).
- You can use it when key names are arbitrary strings.

Syntax:

```
{ string :  value, ….. }
```

Example:

```
{
"id": 110,
"language": "Python",
"price": 1900,
}
```

Whitespace

You can insert whitespace between a pair of tokens.

Syntax:

```
{string:"    ",….}
```

Example:

```
var a = " Alex"; var b = "Steve";
```