Lecture Notes

Smart Contract Development in Ethereum

Deploying Smart Contracts on a Blockchain

In this session, you learnt about the following:

- How to deploy smart contracts on a private Ethereum network.
- How to use the Truffle framework to compile, deploy and test smart contracts on a blockchain network.
- The different types of wallets and how to install Metamask.

Introduction to Truffle

Truffle is a local development environment based on Ethereum blockchain used to develop DApps, compile contracts, deploy contracts, inject it into a web app, create frontend for DApps and test them.

Features of Truffle

- Built-in support to compile, deploy and link smart contracts
- Automated contract testing
- Support for console as well as web apps
- Network management and package management
- Truffle console to directly communicate with smart contracts
- Supports tight integration

Truffle Installation and Configuration

To install Truffle on your computer, you need to run the following command:

```
npm install -g truffle
```

After installing Truffle on your system, create a folder inside your computer. Then traverse to this folder in the terminal and initiate a new Truffle project inside this folder using the following command:

```
truffle init
```

This prepares a framework of files that are required to compile, migrate and test your smart contracts inside this project directory.

- **Contracts**
  - By default, it has a contract file (Migration.sol). This is used by truffle to handle smart contract deployment and testing.
- **Migrations**
  - By default, it has a migration file (1_initial_migration.js). This file is a part of the migration process supported by the migration contract. This is the file which is related to deploying the migration smart contract on top of the blockchain.
- **Test**
  - This is an empty folder where all the test files or testing framework files (mocha or chai) are kept.

- **Truffle-config.js**
  - It has various configuration options required by the Truffle to decide which network to connect to and what default parameters to pick up when sending transactions or fetching details from a smart contract.

Truffle Migration

Truffle migration is a process by which truffle when you first deploy your smart contract on any network would by default install a migration smart contract by which truffle provides when you initiate a new project.
This migration smart contract keeps track of the latest version of the smart contract that is put on the blockchain.
Since you cannot update smart contracts on the blockchain, truffle puts this layer of abstraction using the migration smart contract, where it actually deploys the new smart contract on the blockchain with a different version number and keeps track of it. This is to give you a sense of updating your smart contract as you go into the development phase. So, at the time of development, you need not to worry about deploying the new versions of smart contracts when you make changes to your code.
In this way, truffle is installing a new smart contract and keeping the track of the latest code. Moreover, all of the test functions are directed towards the latest code of the smart contract using the 'Migration.sol' smart contract which is the first thing that truffle deploys on the blockchain.

Migration Scripts

To handle the migration process, truffle uses the migration scripts. '1_initial_migration.js' is an example of the script truffle uses to deploy the 'Migration.sol' smart contract.
Every time you add a smart contract in the contract folder, you need to add a migration script for that smart contract in the migration folder below the existing migration script. The name of the script should start with the number followed by underscore(_) and the number should be one more than the previously existing script.

*** Do not edit or delete the default files created by truffle in the contract and the migration folders.

Voting Use Case

- Add *Voting.sol & AccessControlled.sol* smart contract in the contract folder.
- Create a migration script to deploy the newly added smart contract.
    - *2_voting_deployment.js*

```
const Voting = artifacts.require("Voting");
module.exports = function(deployer) {
  deployer.deploy(Voting);
};
```

Here, you need not to deploy **AcessControlled** smart contract as it is inherited by the **Voting** smart contract.

Configuration – (truffle-config.js)

- Network: It defines the development or Geth network and its parameters.

Apart from defining network settings, you can also define the test or compiler settings.

Smart Contract Deployment on a Private Blockchain

In the project directory, compile the smart contract code using the following command:

```
truffle compile
```

Now, truffle will compile the smart contracts available in the contract directory and create a bytecode that needs to be put on the top of the blockchain.
The result of the command would be the artifacts of the smart contracts written in the new folder 'Build'.
Inside the 'Build; folder, there is a contract folder containing a 'JSON' file for each of the smart contracts.
An important property called 'ABI' is defined inside this JSON file. This is required when you want to access these smart contracts from the frontend application.

ABI is the JSON representation of the entire smart contract.

After this, perform the following steps:,

- Restart the private Ethereum network that you created earlier.
- Keep it ready to deploy smart contracts on this network.

- Then unlock the first account to make it accessible to the Truffle console environment.
- Start the mining process.
- Issue the *migrate* command to migrate your smart contracts to the *geth* network.
- Access the console of your truffle environment and create an instance of a smart contract.
- Run the various functions present inside the smart contract.

Start Geth Setup

- Now, go to the terminal and start your previously configured Geth client using cmd.

```
geth --datadir ./datadir --networkid 252601 --rpc --rpcport 30303 --allow-insecure-unlock console
```

- Now, unlock the account.

```
personal.unlockAccount("accountID", "pwd")
```

- Start the mining process.

```
miner.start()
```

- Migrate your smart contracts.

```
truffle migrate --network geth
```

- To access the smart contracts, use the following:

```
truffle console --network geth
```

- To get the instance of the smart contract, use the following:

```
let voting = await Voting.deployed()
```

- To access one of the functions, use the following:

```
voting.startVoting()
```

Refer this document for detailed steps and commands used.

Wallets in Ethereum

Wallets are services that are installed on your computer and are available as an extension to your regular browsers like Chrome and Firefox.

Some of the important points regarding the concept of wallets can be summarised as follows:

- Just like Paytm, wallets in Ethereum can be used to access the ethers stored in your Ethereum account for transactions.

- The public private key pair stored in the Ethereum wallet is used to authenticate and access/transfer ethers.
- Each wallet has a username and a password associated with it.

Let's now take a look at types of wallets. Some of the different types of wallets are as follows:

- Hardware and Software Wallets

Hardware Wallets	**Software Wallets**
• These are wallets that are used to store accounts in hardware such as pen drives. • Example: Trezor, etc.	• A software wallet can be a desktop or a mobile application or a browser extension. • Example: Trust, Mist, Metamask, etc.

- Hot and Cold Wallets

Hot Wallets	**Cold Wallets**
• Hot wallets are connected to the internet. Their keys are stored online. • They are less secure. • Some examples of hot wallets include Metamask and Ledger.	• Cold wallets are not connected to the internet. They can be as simple as a piece of paper. • They are more secure because they are stored offline. • Multisignature is an example of a cold wallet.

- Single Currency and Multicurrency Wallets

Single Currency Wallets	**Multicurrency Wallets**
• These are wallets that store only one type of currency. • Example: Bitcoin CLI is used to store Bitcoins only. Likewise, Metamask is used to store the private keys of Ethereum accounts only.	• These are wallets that can store multiple types of currencies. Example: Wallets that can store both Bitcoin and Ether. • One of the examples include Trust.

One of the most popular services for a wallet is Metamask. It is one of the most popular Ethereum wallets available. It lets the user upload their private keys and have their accounts ready when they land on a frontend application that is a DApp. The application can then see that the user has Metamask installed and can use its services and the user account inside Metamask to sign the transactions on behalf of the user.

Metamask

Installation: Install Metamask from https://metamask.io/ as a Chrome extension.
Once the extension is added, create a wallet, set a password and you will be given a backup phrase. This phrase needs to be kept safe as it will be given only once and you need to have it as a backup in case you forget your password. After you re-enter the phrase, you would have successfully created your Metamask wallet.