Deploy Smart Contracts using Node.JS. DIFFERENCE BETWEEN TRANSACTION ACCOUNT AND CONTRACT ACCOUNT

ABI

Constructor Args

Gas Amount

ByteCode

Deployment Process

Ganache

Live Ethereum

Test Networks (Rinkeby,Ropsten,Kovan)

Contract Address

As we saw in previous article (Link to Article 3) Compile.js spits out Bytecode and ABI when compiled.

In this article we will write Node.js which takes all input (Bytecode, ABI, Constructor Arguments, Gas amount) and outputs Contract address. Later contract address and ABI can be used by other applications or Smart Contracts to build dApps. Above diagram describes what I just said. We will write Node.js for deploying to local blockchain environment, Ganache. It makes deployment process just so easy to test. Later in other article I will show how to deploy to test networks. Create a folder deploy and file deploy.js in this folder. We will use [web3.js](https://web3js.readthedocs.io/en/1.0/) libraries which allows to connect with local or test Ethereum networks. Web3.js is a JavaScript framework to build distributed apps (dApps). Follow below code. In bold are actual commands

**const Web3 = require('web3'); //Web3 is a constructor**

Using above constructor, we create an instance of web3

ganache-cli is a provider for connection to local blockchain. Consider it as communication layer between web3 and a particular Ethereum network. Each network (Test or Live) has its own provider. This provider sends and receive messages between network and web3

**const ganache = require('ganache-cli');**

Below require refers to compile.js and captures Interface and Bytecode from compile.js. Remember module.exports as a last statement within compile.js

**const { interface, bytecode } = require('../compile');**

While running deploy script in case you land into error “possible [EventEmitter](https://nodejs.org/api/events.html#events_emitter_setmaxlisteners_n) memory leak detected. 11 error listeners added. Use emitter.setMaxListeners() to increase limit.” Use below command

**require('events'). EventEmitter.prototype.\_maxListeners = 100;**

Write a function deploySC which is deploying smart contract. Remember every function in web3 is of asynchronous nature. That’s why function is wrapped with async. Read Promises, async and await in Java script in case you are not familiar. For asynchronous programming getting grasp of Promise and async/await is quite an important. Below script is also printing 10 accounts provided by Ganache

**const deploySC = async () => {**

**accounts = await web3.eth.getAccounts();** //eth is module of web3 library

**console.log ('Total number of accounts provided by Ganache '+accounts.length);**

**console.log ('List of Accounts :');**

**console.log(accounts);**

In below Java script captured interface from compile.js is parsed and bytecode is passed as data. Interface is ABI layer for other applications to access functions of SC. Bytecode is what EVM understand and runs SC. Remember constructor of our contract expects an integer. That’s why 2000 is passed as an argument. Every deployment of contract is a transaction and transaction in Ethereum network demands gas. That’s why out of 10 accounts provided by Ganache one of the accounts [0] is passed as from address. This account already contains fake ethers. One of the argument is gas in .send function. I have given an arbitrary big value of 5 mil so that deployment does not run out of gas. Any transaction in Ethereum demands gas. We set gas limit (or some call gas start) and provide gas fee for transaction to succeed.

**const deployContract = await new web3.eth.Contract(JSON.parse(interface))**

**.deploy({data: bytecode, arguments: [2000]})**

**.send({ from: accounts[0], gas:5000000});**

**console.log('deployment done');**

**//This print contract address on //successful deployment**

**console.log('SC Address '+deployContract.options.address);**

**}**

**deploySC(); //Call function to deploy contract**

From command prompt run **node deploy.js** from same folder where deploy.js is residing. When contract gets deployed, contract address is generated. In Ethereum world there are 2 kinds of addresses.

Externally owned Account (EOA) and Contract address. In above script accounts [0] is EOA or public address through which one can send and receive ethers. Every account address resides in a Wallet and linked to its unique private key. Contract address can also receive Ethers, but main distinction from EOA is Contract address can also receive and send messages/data. I will cover Wallets, EOA, Public/Private key in some other article.

Useful links for Promise, Async/Await

<https://www.youtube.com/watch?v=QO4NXhWo_NM>

<https://www.youtube.com/watch?v=104J7_HyaG4&list=RDQM6NzaS65Pznk&index=16>