Celo/BAF Celo for Eth Devs Feedback

Charlie Timmins
`https://github.com/catimmins/celo-feedback`

Jan. 2020

Introduction

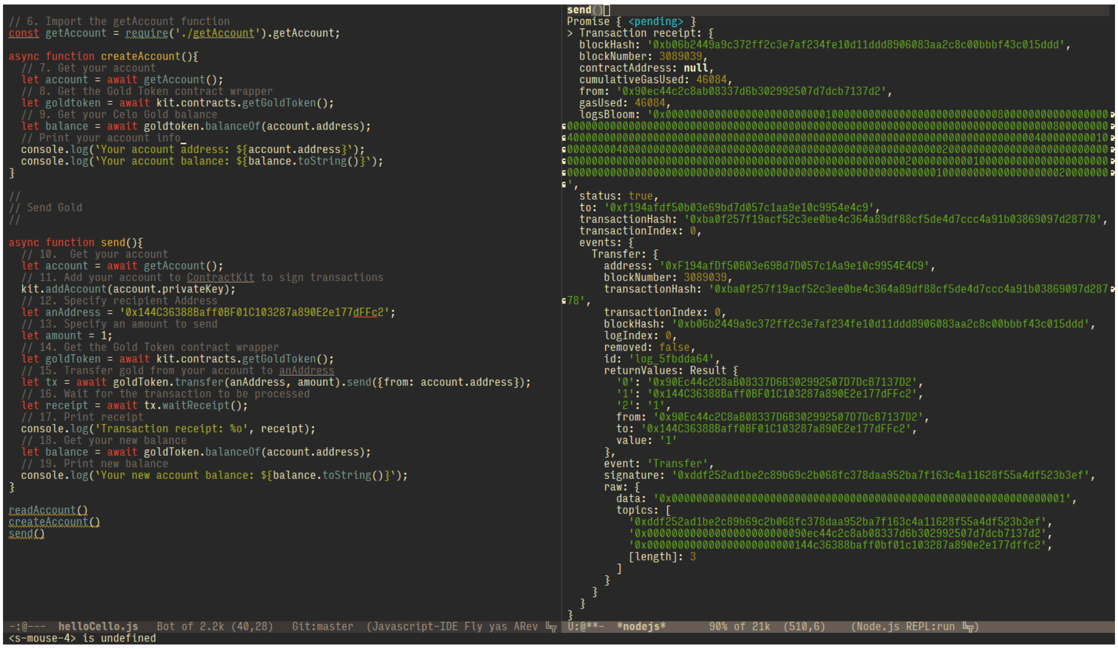
I'm a software, and more recently Ethereum, developer working in a blockchain-based startup and am personally interested in how blockchain technologies can be used to preserve and transmit information. My previous experience is in writing smart contracts written in Solidity and I'm excited to explore how this translates to the Celo protocol.

Process

List the actions that you took during the exercise and include screenshots of problems your encountered

I began by become familiar with Celo through the documentation at `https://docs.celo.org/`. The layout of these pages was clear and I appreciated that the links to start using the testnet immediately followed the highlighted features as I was keen to get started after reading these. The documentation strongly encouraged me to use the Android client, however I chose to generate an account on the command line using the Docker container. I was able to do this, add funds to the account, and verify the balance using the Blockscout without any difficulty.

It's essential in development that basic operations on the blockchain can be performed programatically so I next opened the developer docs to explore how accounts can be created in this way. The helloCelo tutorial at `https://docs.celo.org/developer-guide/start/hellocelo` provided a simple, but effective introduction to how the Celo blockchain might be used in practice and was clear for anyone familiar with JavaScript. Using it I was able to create another account send a minute amount of CELO to the testnet account I had created earlier [Fig. 1]. I did note that there didn't appear to be any way to request funds from the faucet using the JS SDK. I assume this is to prevent it being flooded by a deluge of automated requests, however I couldn't see any alternatives for automated testing.

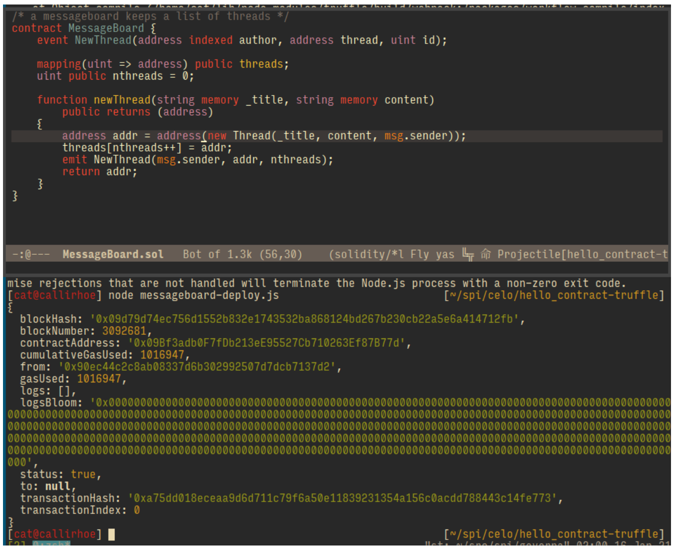


```
// 6. Import the getAccount function
const getAccount = require('./getAccount').getAccount;

async function createAccount(){
  // 7. Get your account
  let account = await getAccount();
  // 8. Get the Gold Token contract wrapper
  let goldtoken = await kit.contracts.getGoldToken();
  // 9. Get your Celo Gold balance
  let balance = await goldtoken.balanceOf(account.address);
  // Print your account info
  console.log(`Your account address: ${account.address}`);
  console.log(`Your account balance: ${balance.toString()}`);
}

//
// Send Gold
//

async function send(){
  // 10.  Get your account
  let account = await getAccount();
  // 11. Add your account to ContractKit to sign transactions
  kit.addAccount(account.privateKey);
  // 12. Specify recipient Address
  let anAddress = '0x144C36388Baff0BF01C103287a890E2e177dFFc2';
  // 13. Specify an amount to send
  let amount = 1;
  // 14. Get the Gold Token contract wrapper
  let goldToken = await kit.contracts.getGoldToken();
  // 15. Transfer gold from your account to anAddress
  let tx = await goldToken.transfer(anAddress, amount).send({from: account.address});
  // 16. Wait for the transaction to be processed
  let receipt = await tx.waitReceipt();
  // 17. Print receipt
  console.log('Transaction receipt: %o', receipt);
  // 18. Get your new balance
  let balance = await goldToken.balanceOf(account.address);
  // 19. Print new balance
  console.log(`Your new account balance: ${balance.toString()}`);
}

readAccount()
createAccount()
send()

-:@---  helloCello.js   Bot of 2.2k (40,28)   Git:master  (Javascript-IDE Fly yas ARev
<s-mouse-4> is undefined
```

```
send()
Promise { <pending> }
> Transaction receipt: {
  blockHash: '0xb06b2449a9c372ff2c3e7af234fe10d11ddd8906083aa2c8c00bbbf43c015ddd',
  blockNumber: 3089039,
  contractAddress: null,
  cumulativeGasUsed: 46084,
  from: '0x90ec44c2c8ab08337d6b302992507d7dcb7137d2',
  gasUsed: 46084,
  logsBloom: '0x00000000000000000000000000100000000000000000000000000008000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000800000000
4000000000000000000000000000000000000000000000000000000000000000000000000000400000000010
0000000040000000000000000000000000000000000000000000000200000000000000000000000000
0000000000000000000000000000000000000000000000000000000200000000001000000000000000000000
0000000000000000000000000000000000000000000000000000000000001000000000000000000002000000
',
  status: true,
  to: '0xf194afdf50b03e69bd7d057c1aa9e10c9954e4c9',
  transactionHash: '0xba0f257f19acf52c3ee0be4c364a89df88cf5de4d7ccc4a91b03869097d28778',
  transactionIndex: 0,
  events: {
    Transfer: {
      address: '0xF194afDf50B03e69Bd7D057c1Aa9e10c9954E4C9',
      blockNumber: 3089039,
      transactionHash: '0xba0f257f19acf52c3ee0be4c364a89df88cf5de4d7ccc4a91b03869097d287
78',
      transactionIndex: 0,
      blockHash: '0xb06b2449a9c372ff2c3e7af234fe10d11ddd8906083aa2c8c00bbbf43c015ddd',
      logIndex: 0,
      removed: false,
      id: 'log_5fbdda64',
      returnValues: Result {
        '0': '0x90Ec44c2C8aB08337D6B302992507D7DcB7137D2',
        '1': '0x144C36388Baff0BF01C103287a890E2e177dFFc2',
        '2': '1',
        from: '0x90Ec44c2C8aB08337D6B302992507D7DcB7137D2',
        to: '0x144C36388Baff0BF01C103287a890E2e177dFFc2',
        value: '1'
      },
      event: 'Transfer',
      signature: '0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef',
      raw: {
        data: '0x0000000000000000000000000000000000000000000000000000000000000001',
        topics: [
          '0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef',
          '0x00000000000000000000000090ec44c2c8ab08337d6b302992507d7dcb7137d2',
          '0x000000000000000000000000144c36388baff0bf01c103287a890e2e177dffc2',
          [length]: 3
        ]
      }
    }
  }
}
U:@**-  *nodejs*       90% of 21k  (510,6)    (Node.js REPL:run )
```

Figure 1: Sending CELO

At this point I was onboard the Celo network with a funded testnet account and sought to start deploying contracts. I chose to follow the tutorial on deploying a contract using a remote node via Forno as I have experience working this way on Ethereum and wished to compare the workflow on Celo. I was pleased to discover that there's very little difference in the process and many Ethereum tools can be used unmodified. In addition to the example contract provided I was able to deploy a simple contract I had written some time ago for Ethereum without any changes and using only the short deployment scripts provided [Fig. 2].



```
/* a messageboard keeps a list of threads */
contract MessageBoard {
    event NewThread(address indexed author, address thread, uint id);

    mapping(uint => address) public threads;
    uint public nthreads = 0;

    function newThread(string memory _title, string memory content)
        public returns (address)
    {
        address addr = address(new Thread(_title, content, msg.sender));
        threads[nthreads++] = addr;
        emit NewThread(msg.sender, addr, nthreads);
        return addr;
    }
}

-:@---  MessageBoard.sol   Bot of 1.3k (56,30)    (solidity/*l Fly yas ╚╦ 🏠 Projectile[hello_contract-t
```

```
mise rejections that are not handled will terminate the Node.js process with a non-zero exit code.
[cat@callirhoe] node messageboard-deploy.js                     [~/spi/celo/hello_contract-truffle]
{
  blockHash: '0x09d79d74ec756d1552b832e1743532ba868124bd267b230cb22a5e6a414712fb',
  blockNumber: 3092681,
  contractAddress: '0x09Bf3adb0F7fDb213eE95527Cb710263Ef87B77d',
  cumulativeGasUsed: 1016947,
  from: '0x90ec44c2c8ab08337d6b302992507d7dcb7137d2',
  gasUsed: 1016947,
  logs: [],
  logsBloom: '0x0000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000',
  status: true,
  to: null,
  transactionHash: '0xa75dd018eceaa9d6d711c79f6a50e11839231354a156c0acdd788443c14fe773',
  transactionIndex: 0
}
[cat@callirhoe] █                                               [~/spi/celo/hello_contract-truffle]
```

Figure 2: Deploying a minimal blockchain messageboard

Deploying Complex Contracts

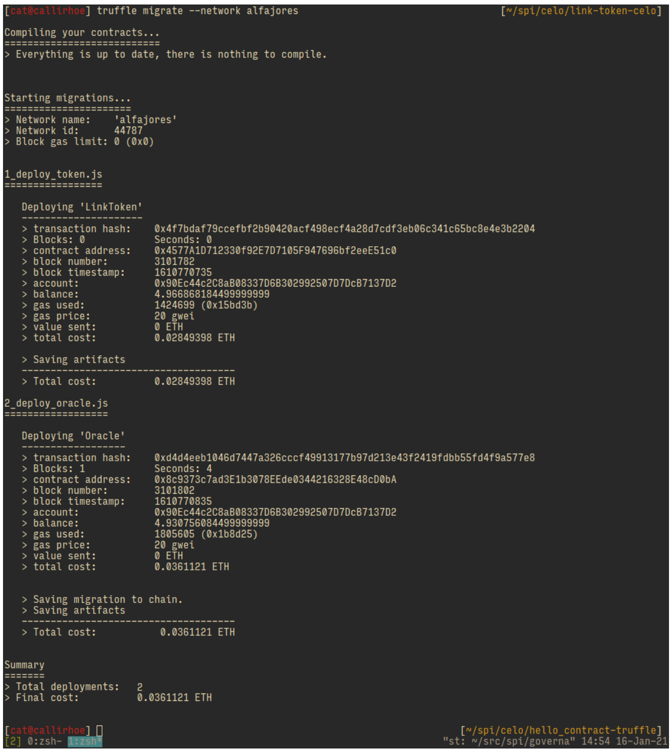
I followed this up by deploying a more complicated set of contracts to assess what adaptions, if any, would be required to move to Celo. I chose the Chainlink ERC20 token and oracle as as it's a very mature project tightly integrated with the Ethereum platform. I expected that this would expose deficiencies in Celo, however I was very pleasantly surprised that it was only a matter of configuring truffle. Far more time was spent tracking down dependencies and figuring out the structure of Chainlink than working with Celo. Deployment seceded without any problems as seen in [Fig. 3] The Oracale can be found at `0x8c9373c7ad3E1b3078EEde0344216328E48cD0bA` and associated LINK token contract at `0x4577A1D712330f92E7D7105F947696bf2eeE51c0`, both on Alfajores.

I really liked:

- Compatibility with Ethereum
- Documentation encourages users to start exploring the network straight away.
- The Javascript SDK appears to be robust and provides a large amount of common functionality.

I had a hard time with:

- The documentation often reminds the reader that Celo is not 100% compatible with Ethereum without elaborating on what the differences are or where they lie. This led to some apprehension in using Ethereum tools and handling potential errors.



```
[cat@callirhoe] truffle migrate --network alfajores                                  [~/spi/celo/link-token-celo]

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



Starting migrations...
======================
> Network name:    'alfajores'
> Network id:      44787
> Block gas limit: 0 (0x0)


1_deploy_token.js
=================

   Deploying 'LinkToken'
   ---------------------
   > transaction hash:    0x4f7bdaf79ccefbf2b90420acf498ecf4a28d7cdf3eb06c341c65bc8e4e3b2204
   > Blocks: 0            Seconds: 0
   > contract address:    0x4577A1D712330f92E7D7105F947696bf2eeE51c0
   > block number:        3101782
   > block timestamp:     1610770735
   > account:             0x90Ec44c2C8aB08337D6B302992507D7DcB7137D2
   > balance:             4.966868184499999999
   > gas used:            1424699 (0x15bd3b)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.02849398 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:          0.02849398 ETH

2_deploy_oracle.js
==================

   Deploying 'Oracle'
   ------------------
   > transaction hash:    0xd4d4eeb1046d7447a326cccf49913177b97d213e43f2419fdbb55fd4f9a577e8
   > Blocks: 1            Seconds: 4
   > contract address:    0x8c9373c7ad3E1b3078EEde0344216328E48cD0bA
   > block number:        3101802
   > block timestamp:     1610770835
   > account:             0x90Ec44c2C8aB08337D6B302992507D7DcB7137D2
   > balance:             4.930756084499999999
   > gas used:            1805605 (0x1b8d25)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.0361121 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:           0.0361121 ETH


Summary
=======
> Total deployments:   2
> Final cost:          0.0361121 ETH


[cat@callirhoe]                                                            [~/spi/celo/hello_contract-truffle]
[2] 0:zsh- 1:zsh*                                                   "st: ~/src/spi/governa" 14:54 16-Jan-21
```

Figure 3: Deploying LINK and Oracle

- Being able to use Ethereum tooling is a definite advantage, however it can lead to a confusing mismatch of information, Truffle still reports value and gas prices in ETH and gwei, respectively. I was unsure what these meant in terms of CELO.

I encountered these errors along the way:

- There was some inconsistency in the examples that initially disoriented me. The helloCelo tutorial is very hands on, encouraging the user to fill in each line according to the given instructions, while the remote deployment tutorial is really just a matter of pulling a repository and running the code. They explain creating an address on Alfajores in very different terms to the point where I initially didn't realise I already had the required account.

- The Celo for Ethereum developers document provides a good high-level overview of the differences in the protocol, however it doesn't provide any examples or links on how the unique features can be used in contracts.

General Impression

How long did this take you?

I spent roughly 5-7 hours reading through the Celo documentation, testing code, and deploying contracts. This is about right for getting acquainted with a new technology in my experience.

How could this exercise be improved?

There are huge amount of features and tools listed in the Celo documentation, some of which I didn't get the opportunity to go through. A broader range of tasks could have better demonstrated what Celo has to offer. In particular I was very impressed with the quality of the SDK and would have liked to explore how it can be used to build blockchain-based applications.