Step2 - OpenST Utility Chain Contracts Deployment

1. Open up a new console and start composing our sample contract, create an empty file called `BasicContract.sol`and open it up.

`touch BasicContract.sol`

Add our sample smart contract solidity code and save this file.

```
pragma solidity ^0.4.23;
contract BasicContract {
 bytes32 public constant name = "BasicContract";
}
```

The contract is named `BasicContract` and it contains a public constant, "BasicContract".

2. Compile the above contract using`solc`the solidity compiler, by parsing it into `a``json`format and assigning it to a JavaScript variable`basicContractOutput`with the flags`--optimize`and`--combined-json`. Save this to the present working directory as`basicContract.js`. Check if you have the files (and its contents), before moving forward.

```
echo "var basicContractOutput=`solc --optimize --combined-json
abi,bin,interface BasicContract.sol`" > basicContract.js

cat basicContract.js
```

Account Creation and Obtaining Gas

1. Open up a new console (yes, a third one) and `attach` `geth` with the `IPC` path saved from step 3 of the *OpenST Utility Chain Sync* document. This starts an interactive JavaScript environment - connect to node in the go-ethereum command line interface.

```
geth attach ~/uc_node_1409/geth.ipc //this path will be different
in your case
```

2. Create a new account, for this you will own the private key. So you can deploy your sample smart contract from this account. Enter a passphrase with which only you can access the account. To know about the list of available commands for geth look here

```
> personal.newAccount()
Passphrase:
Repeat passphrase:
```

3. This newly created account should be your coinbase account (if you haven't already previously created an account). Please note this does not have any relation whatsoever to coinbase.com. Check for the coinbase account address and make sure its the same as the one created above, then take a look at your account's balance (this should be 0).

```
> eth.coinbase
> eth.getBalance(eth.coinbase)
```

Copy-paste (save) the account address displayed here for the upcoming steps.

4. Now you need some funds - OST Prime to deploy your sample contract with `geth`. You should use the new transfers api make sure to transfer to the coinbase address you have copied above. For executing this step you will need to register for OST KIT. Complete the steps in this tutorial before moving forward if you have not already.
5. In a separate console spin up an official OST KIT SDK and execute the new Transfers API.

- javascript
- ruby
- php

Here we show the example of the JavaScript SDK. Make sure that the`to_address`is the coinbase address you copied previously. The amount to be sent to this address can be understood from theTransfers API documentation.Please note that you can transfer only as much OST OST Prime as you have available after staking and minting. If you need more, you will have to stake and mint the required amount via the OST KIT dashboard.

```
const transferService = ostObj.services.transfers; // transfer object
creation
transferService.execute({to_address:'0xd2b789293674faEE51bEb2d0338d15401
dEbfdE3', amount:1000000000000000000}).then(function(res) {
console.log(JSON.stringify(res)); }).catch(function(err) {
console.log(JSON.stringify(err)); }); //here the address will be the
coinbase address you copied.
```

Contract Deployment on the UC

1. Provide some time for the balance to show up as transactions are mined and included into blocks by `geth` nodes. One way to check the status of the transfer would be to use the List Transfers API. alternative you can check using the following command in the `geth` console. Also, make sure you are in sync with the UC (using step 4 of the *OpenST Utility Chain Sync* document) so that you are reading from the latest blocks. If you have the balance you requested with the Transfers API, proceed to the next steps.

```
> eth.getBalance(eth.coinbase)
```

2. When the funds you requested appear in your account, proceed with loading the contract scripts into `geth`, check for the key-pair values in `basicContractOutput`, this would display the `abi` - application binary interface and `bin` - binary file, which you will need.

```
> loadScript('basicContract.js')
> basicContractOutput
```

3. Get the contract's `abi` from basicContractOutput and store it into a variable (var) called `basicContractAbi` and check out its contents.

```
> var basicContractAbi =
basicContractOutput.contracts['BasicContract.sol:BasicContract'].ab
i
> basicContractAbi
```

4. Store the contract's `abi` into a variable called `basicContract` by passing it as an argument to the `eth.contract`function.

```
> var basicContract = eth.contract(JSON.parse(basicContractAbi))
```

5. Store the contract's `bin` into a variable called `basicContractBinCode` and concatenate it with the hexadecimal prefix `0x` so it is callable in a transaction. Check out its contents.

```
> var basicContractBinCode = "0x" +
basicContractOutput.contracts['BasicContract.sol:BasicContract'].bi
n
> basicContractBinCode
```

6. Unlock your coinbase account (with OST Prime) to start the deployment process and give in the passphrase you used to create the account. Anytime in the following steps if you get an `Error: authentication needed: password or unlock`, use the above command to unlock your account.

```
> personal.unlockAccount(eth.accounts[0])
> Passphrase:
```

7. Store the deployable transaction object into a variable called `deployTransactionObject` using the `basicContractBinCode` as data and set the gas value to `1000000` for deploying the sample contract.

```
> var deployTransactionObject = { from: eth.accounts[0], data:
basicContractBinCode, gas: 200000 }
```

8. Store the `basicContract` instance in a variable called `basicContractInstance` by using the deployable transaction object as its argument. Deploy the contract by calling the transaction object `basicContractInstance.` A transaction is executed for deploying `B asicContract` on the UC, returning a web3 contract instance, this lacks an address until it is mined by the `geth` nodes.

```
> var basicContractInstance =
basicContract.new(deployTransactionObject)
> basicContractInstance
```

9. Grab the transaction receipt of your deployed contract using the `getTransactionReceipt` function. The address returned here is the unique, immutable address of the contract; it is calculated from the hash of the sender address and the transaction nonce. When you interact with this contract instance you need to mention this address.

```
> eth.getTransactionReceipt(basicContractInstance.transactionHash)
```

10. Store the address of the deployed contract into a `basicContractAddress` variable.

```
> var basicContractAddress =
eth.getTransactionReceipt(basicContractInstance.transactionHash).co
ntractAddress
```

11. Store the `basicContractAddress` variable in a `storage` variable and check out its contents.

```
> var storage = basicContract.at(basicContractAddress)
> storage
```

You have successfully deployed this sample smart contract on the UC. This completes the objective of this tutorial.