Phase3 : instructions to start a private blockchain and deploying smart contract on it:

1. **Creating a private network using geth:**

**Step 1**:

You first need to create a new folder using the following command:

mkdir KYC-Blockchain

Then you need to move to this path using the command:

cd KYC-Blockchain /

After this, you need to open the folder which you have created in any text editor of your choice.

Inside this folder create a file named genesis.json. Inside of this file, you will define the different parameters for genesis block of a private network.

**Step 2:**

After you have defined the parameters in genesis.json file you need to go back to your terminal and just make sure that you are inside the project directory which you had created in the previous step.

Next, you need to issue the following command and press enter:

geth --datadir ./datadir init ./genesis.json

This command will create a private chain data for our blockchain.

When you go back to your code-editor you will see that there is a new folder inside your project directory called datadir which is where our chain data is going to be.

Step 3: Now you need to go back to your terminal and here you will start this particular network.

You need to type the following command and press enter:

geth --datadir ./datadir/ --networkid 2039 console

This command is going to create a new blockchain network and it will start a console with which you can access the different APIs that are available for this private blockchain network you have created. After you press enter the network will start with a network id of 2039.

**Step 4:**

The next step is to now create an account which can be created using personal API.

You need to type in the following command inside the console:

personal.newAccount(‘cricket’)

Inside the parentheses, you need to type the password for this new account.

When you press enter it is going to create a new account for you and is going to list down the address of this new account.

When you go inside of your project directory and inside the keystore folder there is a new keystore file which is the file for this new account which you have created.

By using the command “eth.coinbase” you can check the coinbase for your private network. Coinbase is that account which will collect the ether rewards for mining on your particular geth node. By default, the first account that you created on the node will become the coinbase account.

If you want to change the coinbase you can do so using the command miner.setEtherbase(‘new account ID that is to be made new coinbase account’).

**Step 5:**

To check ether balance in accounts you can use the following command: eth.getBalance(‘account ID’) This will return you the balance in Wei and not in ether.

You can also use the following command: eth.getBalance(eth.coinbase)

This will return the balance of your coinbase in Wei.

**Step 6:**

The next step is to start the mining process.

The command for the same is miner.start()

To stop the mining process you need to issue the following command: miner.stop()

**Step 7:**

Using the following command eth.blockNumber you can see what is the last block number on your private network.

**Step 8:**

To check the balance in ethers you use the following command: web3.fromWei(eth.getBalance(eth.coinbase))

This will return to you the ether value instead of the Wei value.

Using eth.accounts command we can get the list of accounts on your network.

1. **Deploying smart contract on the private network created previously using truffle:**

**Step 1:**

The next step is to start a new Truffle project by creating a new directory inside your computer.

Enter the following command to create a new directory:

mkdir KYC-SC

Next, enter the following command to get inside the directory created above:

cd KYC-SC

**Step 2:**

The command to initiate a new Truffle project inside any folder is

truffle init

When you run this command Truffle prepares a framework of files that you would require to be able to successfully compile, migrate and test your smart contract projects using the Truffle environment.

Next, inside any text editor of your choice open the project folder that you just created.

Inside this folder, you would see that there are some files and folders that have now been added.

These have been added because of the truffle init command that you ran previously in this step.

The first file which you would see is truffle-config.js file. One of the three folders that you will see as part of the project directory is the contracts folder which will by default have one contract which is already available to you called Migrations.sol.

The second folder will be migrations folder inside which there will be one file available by default named 1\_initial\_migration.js.

The third folder is the test folder and this is an empty folder.

This is where all of your test files would go. You are not supposed to add any code or delete these files in any way.

**Step 3:**

If you want to start writing your own smart-contracts inside this project directory, you need to add a new file inside the contracts directory.

Copy the KYC.sol contract from remix to this directory with the name “KYC.sol”

**Step 4:**

The next step is to write the migration script to be able to deploy this new smart contract on the blockchain network.

For this, you need to create a new file inside the migrations directory and name it as 2\_KYC\_migration.js. You can use any name which you wish but the 1st character has to be a number and it should be one greater than the last number of the various migration scripts inside this folder

**Step 5:**

Inside this script, you are going to write a deployment script to be able to migrate the voting contract on to the blockchain network.

You just need to copy the script present inside the migration script which was available to you by default. Then you need to change this script slightly.

Change the first line of this script. It should be as follows:

const KYC = artifacts.require(“KYC”);

You also need to make one more change.Change the line which reads as

deployer.deploy(Migrations) to deployer.deploy(KYC).

This will deploy the voting smart contract on top of the blockchain.

**Step 6:**

The next step is to define the various parameters or the configurations inside the truffle-config.js.

Open the truffle-config.js file. The most important parameter that you have to define is the network’s property.

It will have a list of various networks that your truffle environment can connect to and one of these networks would be used to deploy your smart contracts on the blockchain.

Now you need to uncomment the development network which is the first network that you are going to use to connect our truffle environment.

You also need to define another network and call it geth which would be the geth network that you wish to connect to.

You just need to copy the development network and change the name of the network to geth. You also need to change the port number to 30303 and change the network\_id to 2039

**Step 7:**

Now you are going to compile the smart contract code and make sure it works properly.

For this, you need to go back to your Terminal.

Now you need to run the following command:

truffle compile

With this command, you will see that truffle is going to compile all the smart contracts available inside your contracts directory and is going to create the byte-code that needs to be put on top of the blockchain.

When the command finishes its execution you will see that all the smart contracts inside the contracts directory have been compiled and artifacts have been written down inside a new project directory called build.

If you go inside your text editor you will see that you have a new folder inside your project directory called build. Inside this folder, you will have a contracts folder where you will have a JSON file for each of the different smart contracts that you have written.

**Step 8:**

Open a new tab in your Terminal and go to the KYC-Blockchain directory. You are going to use the same private ethereum network that you created before to deploy the newly compiled smart contract on top of this network

You need to first restart the private ethereum network which you have created before and keep it ready so that you can deploy smart contracts on top of this network.

You need to start the geth private network using the following command:

geth --datadir ./datadir --networkid 2039 --rpc --rpcport 30303 --allow-insecure-unlock console

After you press enter the geth environment is up and running.

Use the command eth.accounts to make sure that you have few accounts on this geth node.

By default, when you use truffle to connect to this geth network it uses the 1st account that is defined inside of this geth node to be the sender account for all transactions.

**Step 9:**

You need to unlock the first account so that it is accessible to the truffle console environment when you want to send your smart contracts on the blockchain.

You need to run the following command to do this:

personal.unlockAccount(‘Address of 1st Account’, ‘password of the account’, 0)

In this case it will be,

personal.unlockAccount(‘0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db’, ‘cricket’, 0)

The last parameter is the amount of time in seconds for which you want to keep this account unlocked. If you pass in 0 as the last parameter, then it keeps the account unlocked for an infinite amount of time.

**Step 10:**

The final step to prepare your geth network to be able to accept the truffle connection and install the smart contract is to start the mining process.

Enter the following command to start the mining process:

miner.start()

At this point, your geth network is properly set up, it is mining and it is ready to accept any HTTP access to be able to install the smart contracts on top of this network.

**Step 11:**

Move to the 2nd tab of the Terminal and run the following command:

truffle migrate --network geth --network geth

using this option you can pass the network where you want to migrate your smart contracts.

In this case, the network is geth.

If you don’t specify any network name, then by default it picks up a network name called development.

When you run the above command, it is going to migrate the smart contracts on to the geth network that you have earlier created.

In the end, you can see the cost in ethereum for running this contract transaction.

Both of the smart contracts will now be installed on this private geth network.

After you have compiled and deployed the smart contract on the private network, let’s now move on to the last step of this process which is to be able to access the smart contracts and to run its functions and test them.

**Step 12:**

To access the smart contracts, you are going to use the truffle’s console environment.

To access the truffle console run the following command:

truffle console --network geth

After you run this command, you will be taken inside the truffle console connected to the geth environment.

Inside this console, you can write the JavaScript statements that you need to get the instance of smart contracts and then use that instance to access the functions of smart contracts.

**Step 13:**

To get the instance of the smart contract that you have deployed on the blockchain you need to write the following JavaScript statement inside the truffle console and press enter:

let kyc = await KYC.deployed()

When this statement is executed, you will have the value of the smart contract available inside this voting variable

Step 14:

Now, when you use the following command kyc. and press Tab twice you will see the various methods that are available to this instance of the smart contract.

To access one of the functions run the following command:

Kyc.AddBank(“SBI”,” 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4,”AB123”,{from:accounts[0]})

When this command is executed you will get the receipt and the transaction hash, which is the response for executing this function on the blockchain.

You can also run some other functions as part of this smart contract instance and those will run perfectly as well.

This is how you can access your smart contract from truffle’s console and test out the various functions that you have defined inside of your smart contract