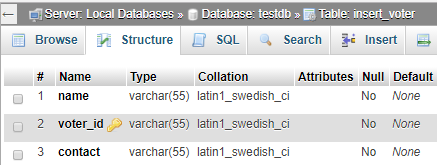
**Part 1: Running the system on localhost:**

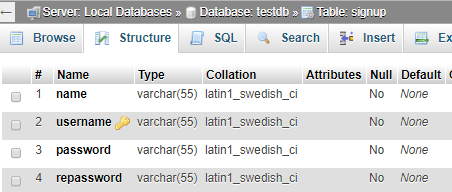
Step 1:

In our system we make also make use of sql server to provide a platform for users (admin and voters) to sign up and login to our system.

We make use of 2 tables, namely:

1. Insert\_voter (name, voter\_id, contact)
2. Signup (name, username, password, repassword)





In order to connect to the sql server, we have a php file named connection.php where all the set up for the connection to the sql is made.



The host, password, username, DB should be redefined according to your localhost/phpmyadmin settings for the database to be connected to the localhost as shown below.



To interact with our sql server from the front end we have php programs in the login folder of the front end of the admin and that of the voter to process the logic of our software.

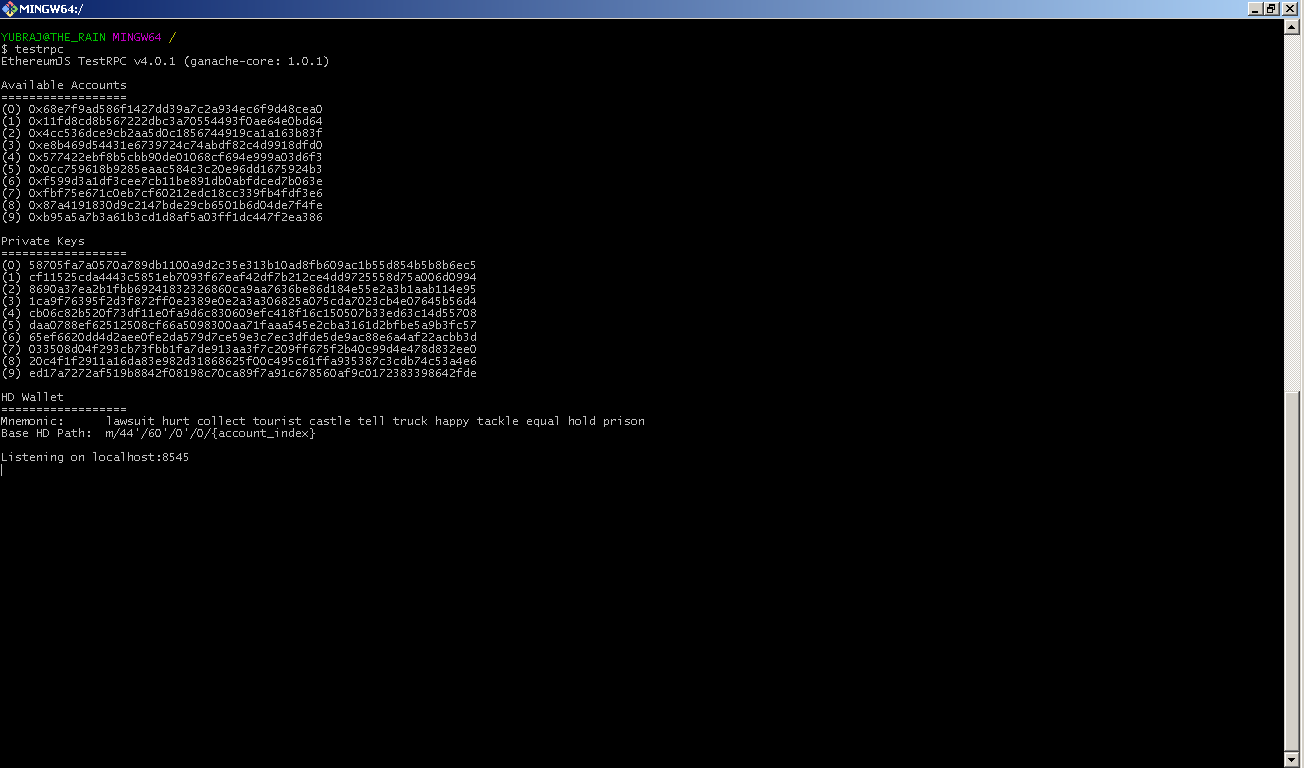
**Step2:**

To deploy your system on your localhost while using the geth console, follow the steps below:

There are a number of ways to run our system. You could either run it locally using simulation ethereum client like testrpc, or you could create a private blockchain using go-ethereum (geth) or you could deploy our system to the Microsoft azure cloud platform. Here are the steps we used to develop, deploy and test our system.

**Step 1. Locally using testrpc, truffle and webpack**

1. First thing, you need to do is get an ethereum client like ethereumjs-testrpc via as follows:  
   *npm install -g ethereumjs-testrpc  .*  
   Run testrpc once you are done with installing it. you can run it using 'testrpc' command .



then, you need to install Truffle development framework, which is used to compile, deploy and test smart contracts. . It is a good choice for the test-driven development of smart contracts and aid for the front-end development later on. Truffle can also be installed via npm as follows:

`npm install -g truffle `

But make sure, before installing truffle , you have the necessary dependencies like nodejs, npm and windows-build-tools are installed. This can be done as follows

npm install-g npm

npm install-g --production windows-build-tools

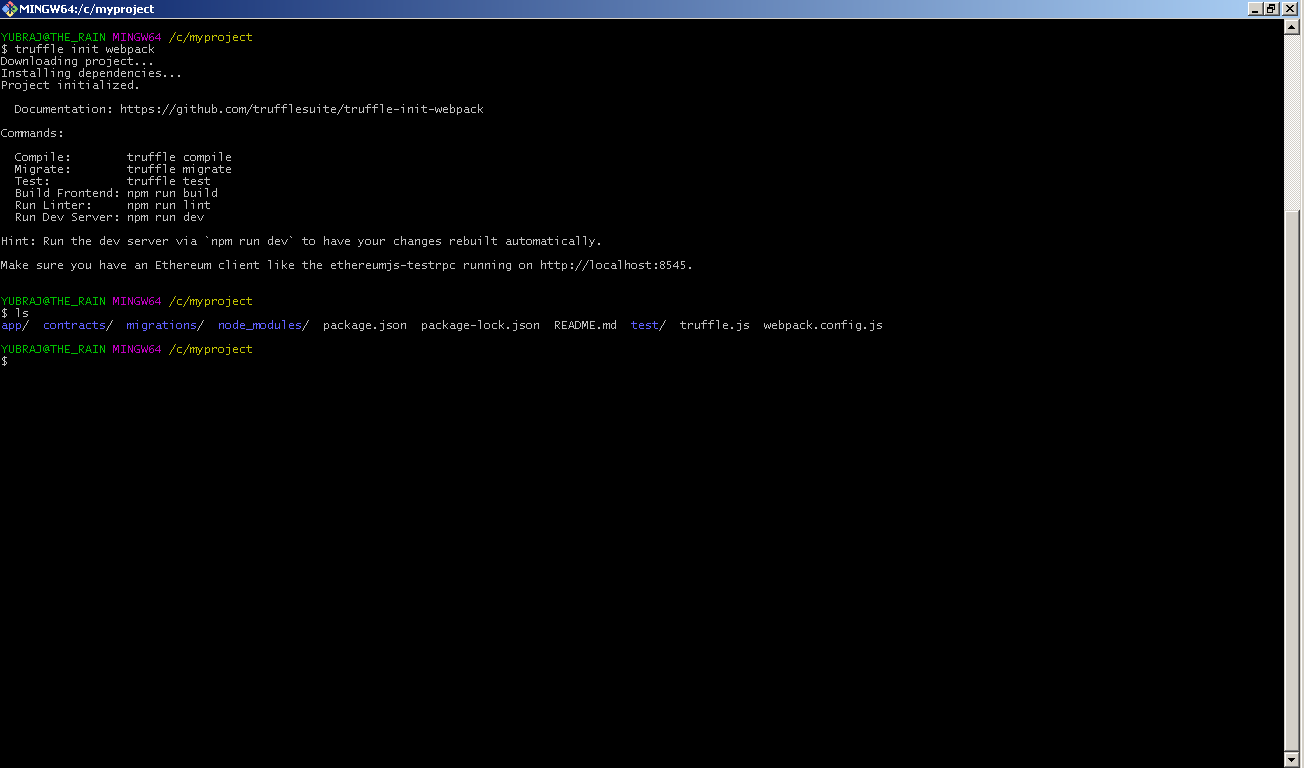
First create a project folder as *`mkdir myproject'*  
Next, initialize your Truffle project by performing the following in the command line:

$ cd myproject

$ truffle init webpack

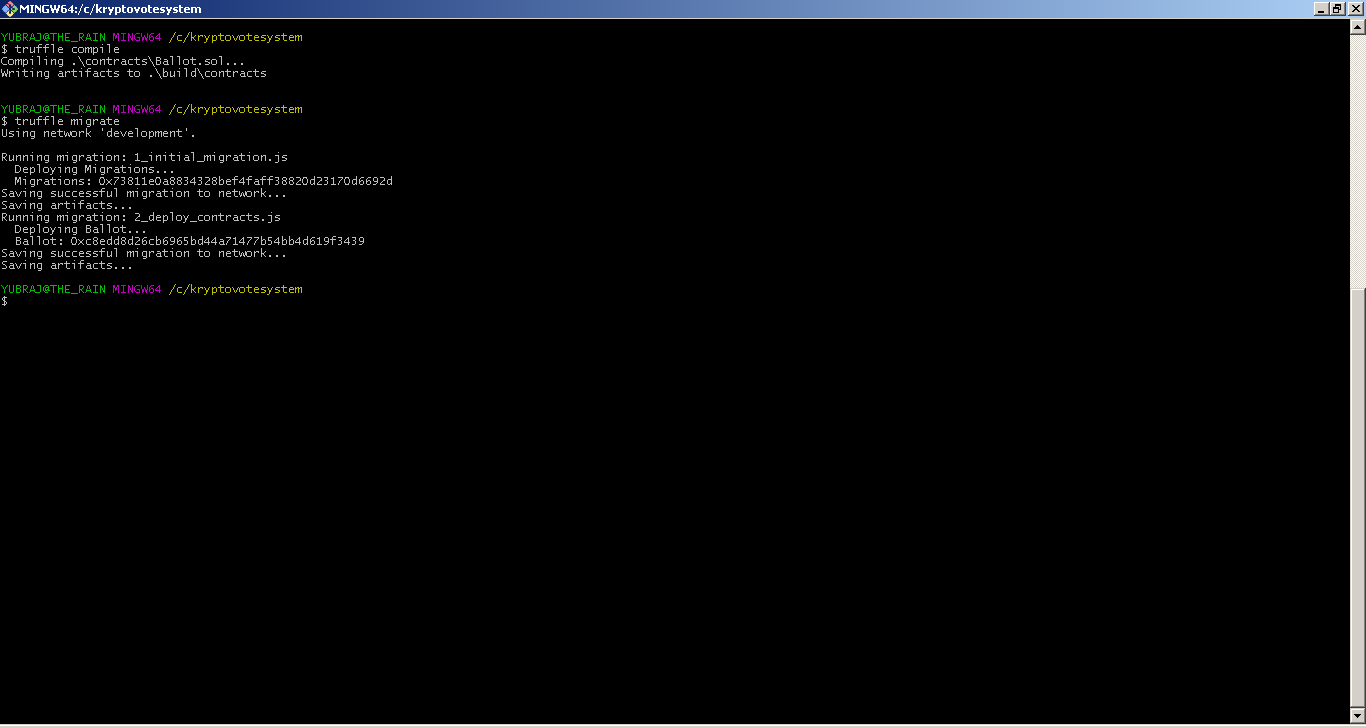
Make sure, your testrpc is running while doing so. Once completed, you'll now have a project structure with the following items:

* contracts/ - smart contracts written in solidity are kept here
* migrations/ - files required while migrating to the blockchain are kept here
* test/ - location of test files for testing your application and contracts.
* truffle.js - main truffle configuration files
* app/ - directory which contains files for the front end design, which contains two sub directories stylesheets and javascripts to keep css and javascript files respectively.
* webpack-config.js - webpack configuration file
* node-modules/ directory for all the npm packages installed
* package.json - json file which includes the necessary dependencies for the project



1. Next,  Under root directory of the system , you will come across the following files structure:

* contracts /
* javascripts /
* kryptovote/
* migrations /
* stylesheets/
* truffle.config
* webpack.config
* kryptovote\_db.sql  
    
  Overwrite the folders contracts, migrations and file webpack.config with the one you created above inside **myproject** directory. in the same way, overwrite javascripts and stylesheets directory which is insde the app folder of the main directory. you dont need to worry about overwriting  truffle.config file now , we will be required later while deploying our smart contract to the private blockchain..  
    
  Now, we got to compile and deploy our smart contract as follows:  Note, the smart contract is inside the contracts/ directory
* truffle comile - to compile the smart contract written in solidity language
* truffle migrate - to deploy the smart contract to the blockchain

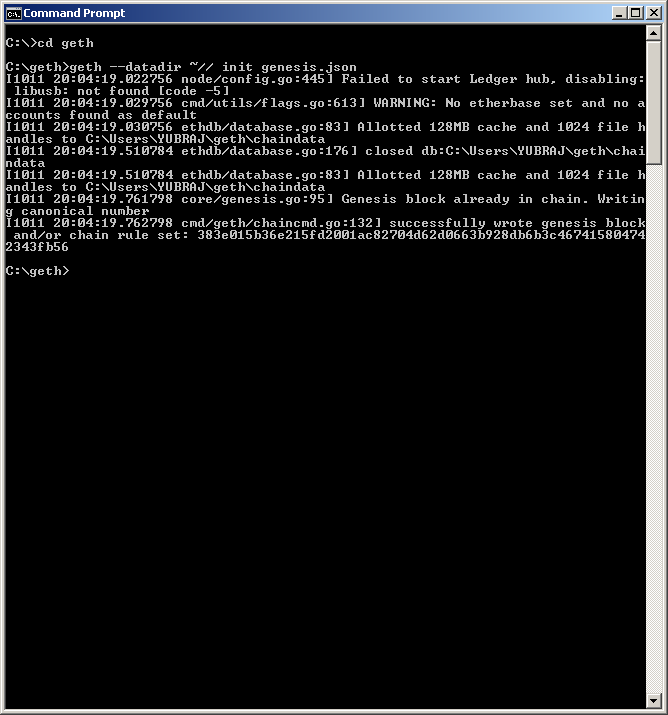


Then, install all required dependencies npm packages using comamand - **'*npm install'*** and ***build webpack*** using 'webpack'We are finally done with configuring the smart contract section and now we need to configure apache server, php and mysql. To do this, You either need to install xamp or wamp in your machine and then copy kryptovote/ directory to the www folder of wamp server. finally, import the database kryptovote\_db.sql using phpmy admin. this file contains the necessary ddl scripts to create necessary tables in your machine. that's it. we are done with installing our system locally.   
You can run our system from the apache **localhost/kryptovote/index1.php**

* **Step 2. Private Blockchain using go-ethereum (geth)**  
  In the above step, we used simulation blockchain using testrpc. However, It is also possible to run our system in a private blockchain using go-ethereum (geth). here , you wont be having 10 accounts with prefunded ether though. following are the steps to run our system in a private blockchain :
* Download and install Geth which is one of the popular ethereum node client.
* Create a new genesis.json file as follows:   
  *mkdir geth && cd geth*  
  *touch genesis.json*
* Edit the genesis.json file as follows:

{  
"config": {  
"chainID" : 10,  
"homesteadBlock": 0,  
"eip155Block": 0,  
"eip158Block": 0  
},  
"nonce": "0xlookatmeimanonce",  
"difficulty": "0x20000",  
"mixhash": "0x00000000000000000000000000000000000000647572616c65787365646c6578",  
"coinbase": "0x0000000000000000000000000000000000000000",  
"timestamp": "0x00",  
"parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",  
"extraData": "0x00",  
"gasLimit": "0x2FEFD8",  
"alloc": {  
}  
}

* Initiate a blockchain , creating  a genesis block as follows:   
  *geth --datadir ~// init genesis.json*



* Next, we use geth to start the first node on our private network  Run the following command to start the node using the genesis block , we initialised in the above step

**geth  --networkid 58342  -- --rpc --rpcport 8545 --rpcaddr 127.0.0.1 --rpccorsdomain "\*" --rpcapi "eth,net,web3"  --nat none --dev**

* Next, open a new terminal and run the following command to attach to the geth console

*geth attach*

* Then, we create an account to use and set it as an coinbase account using the following command:

*personal.newAccount("password")*

*miner.setEtherbase(web3.eth.accounts[1])*

* check if your account has been set as a coin base as follows:

*web3.eth.coinbase*

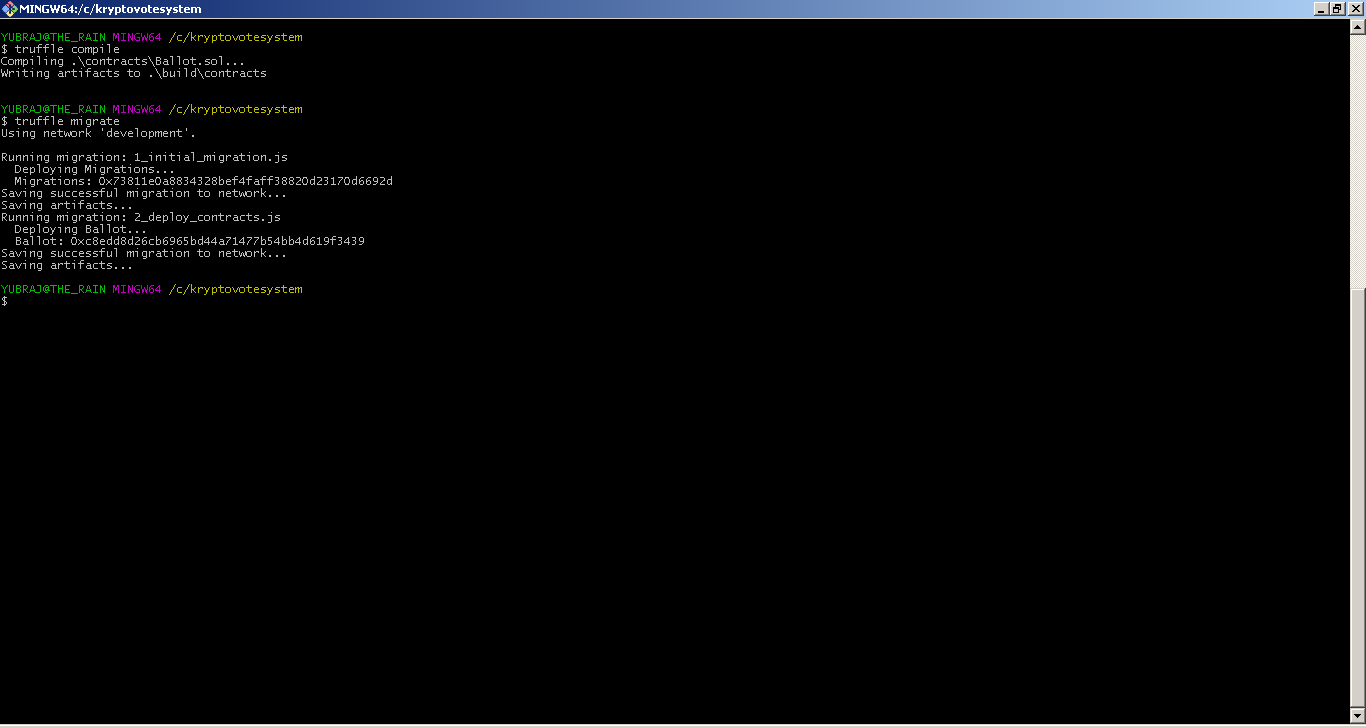
* *After, we need to mine some ether runing following command:*

*miner.start()*which should return true if succeed.

* Deploy your contract to the private blockchain using truffle . to do this, first unlock your coinbase account as follows:

**> personal.unlockAccount(web3.eth.coinbase, "password", 15000)**

* + In the third windows run 'truffle compile' and `truffle migrate ` from your project directory. you should be able to get the result similar as below:

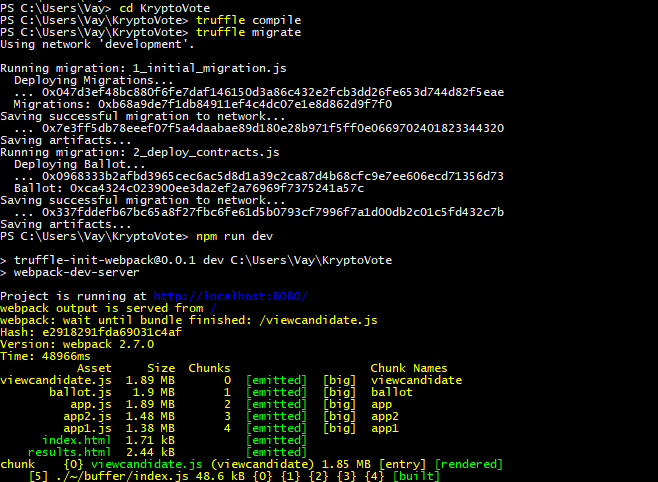


* Now , you can start interacting with the smart contract using truffle console or from the front end.  
  Lastly we need to deploy to Windows Azure to allow multiple voters to use the system online. Use the guidline about the deployment of Window Azre to allow multiple voters to use the system online  to continue and complete the installation.

**Step 3:**

Once the blockchain program is migrated onto the private network, we can unlock the coinbase account using our password and migrate our smart contract on the network using truffle migrate from our folder directory where the blockchain files are located.

We can then type in “npm run dev” so that our files are hosted on localhost:8080.



Step 4:

Since our project has been deployed on azure, all of our front end files have been modified for the system to adapt to the azure private blockchain.

To be able to run the project in the localhost, we need to modify all the php files for admin and voter where scripts of app.js, app1.js, app2.js, ballot.js, viewcandidate.js are present.

For example, the script <script src="./app2.js"></script> should be replace by <script src="http://localhost:8080/app2.js"></script> for the project to run on localhost.



**Part 2: Running the system on azure**

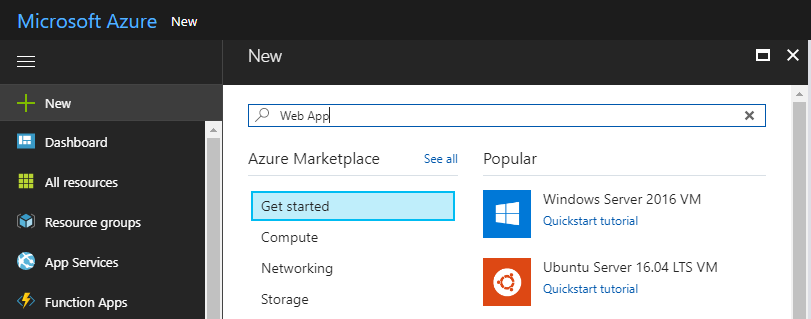
To run our system from azure, make use of the files provided for admin, voter and back end and follow the following guide.

**Using Microsoft azure cloud platform as a blockchain service,Web App Service & Database**

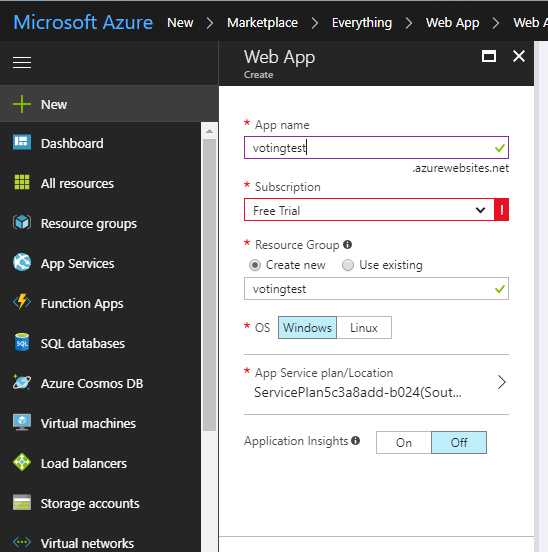
* Firstly, we need to login to <https://azure.microsoft.com/>
* We need to create 3 resources, to successfully create your website online and have the blockchain and database hosted on windows azure:
  + Web App
  + MYSQL DATABASE
  + METATEST 4
* We need to set-up each on the 3 Resources

### ***STEP 1: Setting up Website Service on Windows Azure***

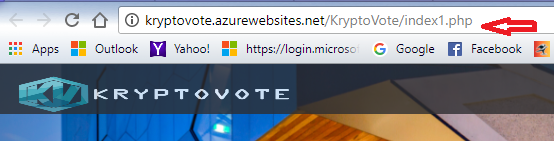
* The very first step to be able to deploy your developed website on Windows Azure, firstly we need to set Up **Web App** Resource  
  Click on New Resource, then type **Web App** as shown below.



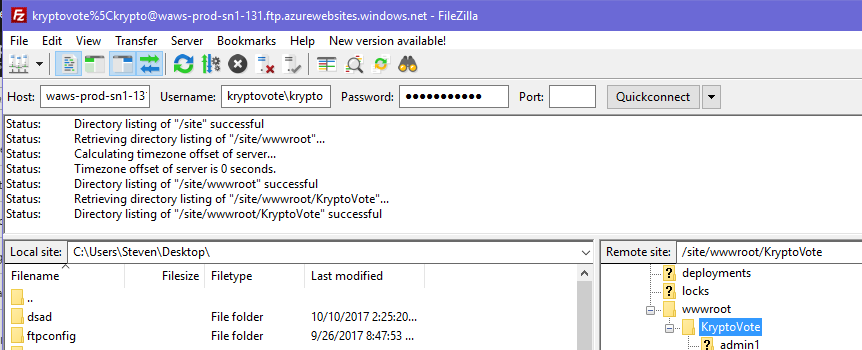
* Then you need to configure the website name and resource group. And then create on Create Button.



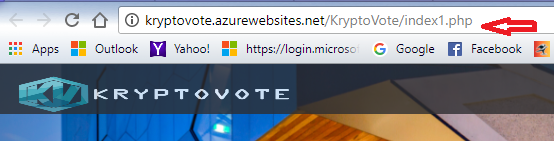
* The Web App has will be generated on the Dashboard. By clicking on the Properties tab, in the Web App created, the URL of the web app will be set and it can be accessed. As shown below, the URL can be accessed.



* Then to put all your website developed for deployment on Windows Azure, we use FileZila and connect to the Web App on Azure using the FTP Username, FTP hostname and the login password. For a successful connection, successful message is displayed.



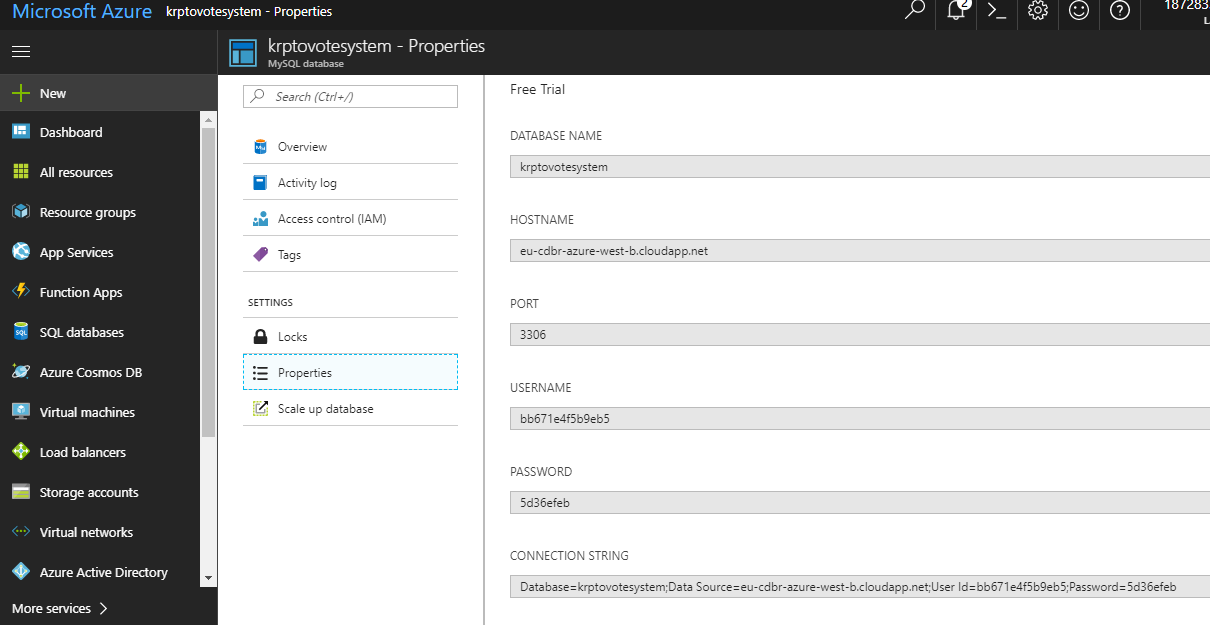
* Then the website can be accessed. By using the URL given by azure as shown previously.
* Below is the website link deployed using Windows Azure. It should work as:  yourwebsitename.[**azurewebsites.net/**](http://azurewebsites.net/)**,**here for us it is: <http://kryptovote.azurewebsites.net/KryptoVote/index1.php>



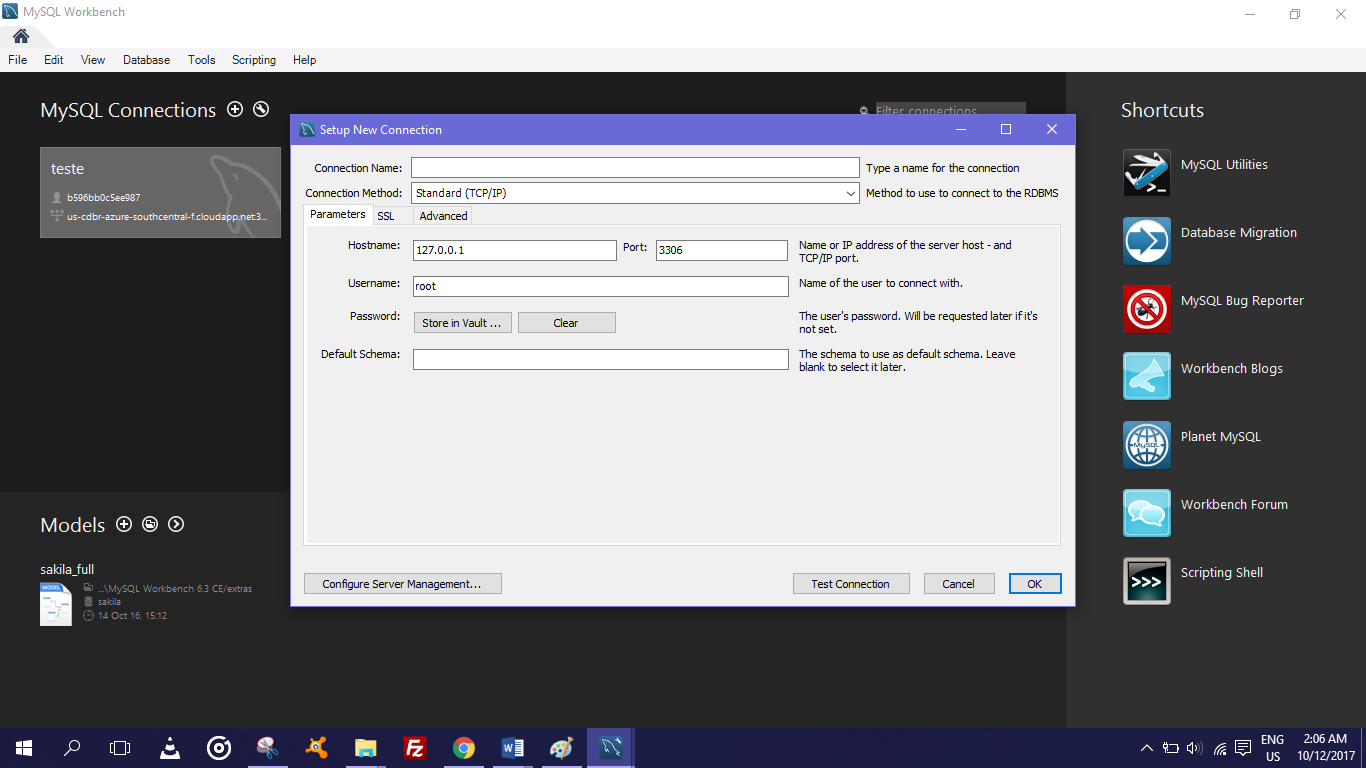
### ***STEP 2: Setting up Database Server on Windows Azure***

Now, we need to set up the Database Server.

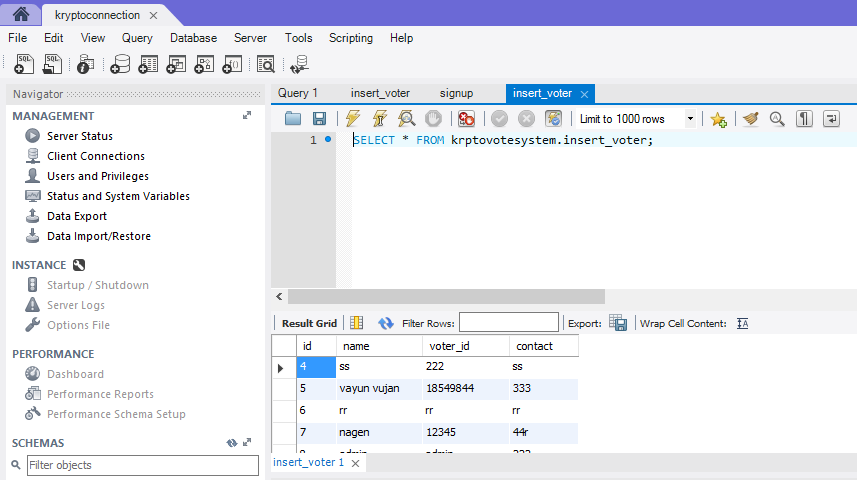
* We need to choose the **MySQL** from the Resource available menu. And then click on create.
* After installation, we need to obtain the Properties of the MySQL database setup like shown above.



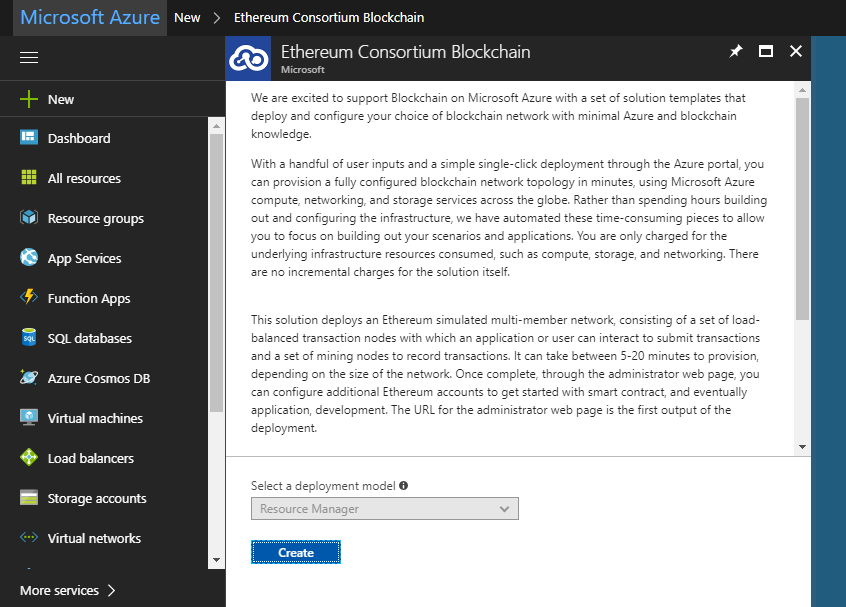
* We need to make a connection of the from the Windows Azure Database to the MySQL database using MYSQL WORK BENCH. Click on the add new connection and then enter the HOSTNAME, USERNAME, PASSWORD from the above screenshot given from azure. Input the credentials and setting into MYSQL WORKBENCH and then click on test connection. A successful message will be displayed for successful connection.



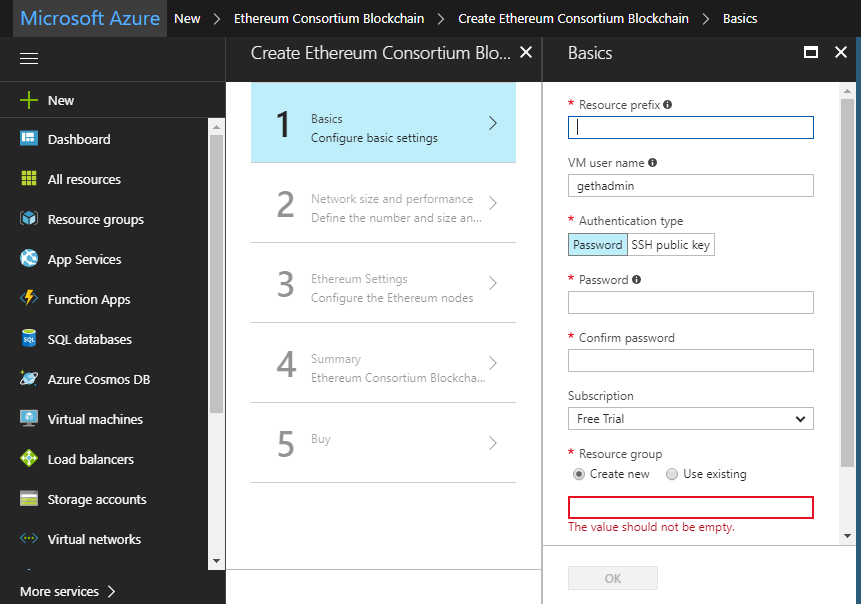
After setting the connection, double click on the connection, a window will open as shown below to manage your database. You can create tables, add data to tables using the SQL statements in the Query pages,   as shown in the screenshot above. You can create a new create page and then to run each query select the query and then click on the FLASH icon.



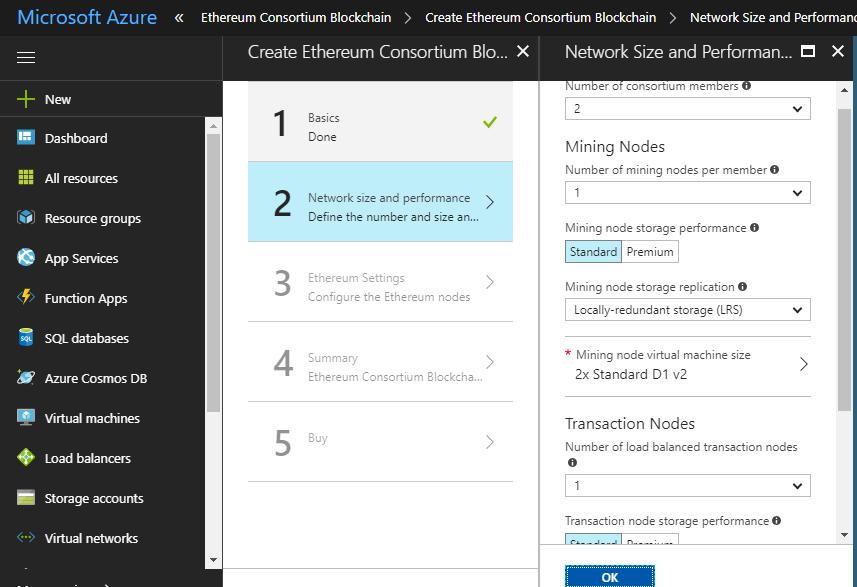
* ***STEP 3: Setting up BlockChain service on Azure for Deployment***
* To set up the Ethereum Consortium Blockchain, we need to create an ethereum consortium blockchain resource group.



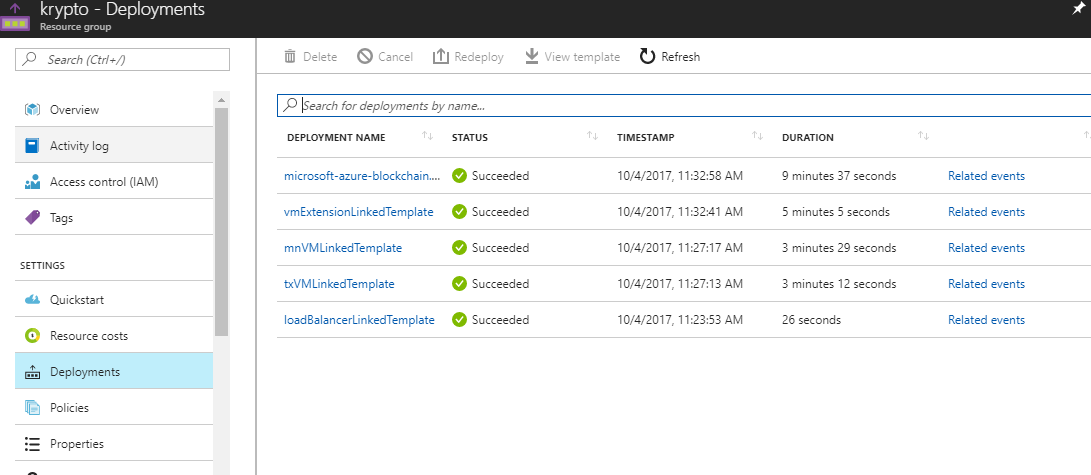
* Now we need to assign resource prefix, create a new password and allocate the resource group.



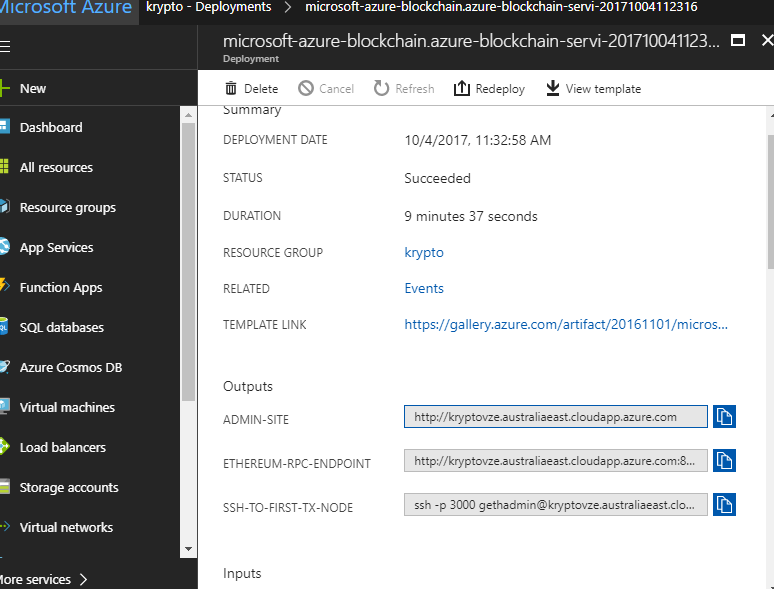
* Following that, we choose our network specifications for the blockchain. In our project, we set up the blockchain for 2 consortium members, 1 mining nodes per member and 2 transaction nodes.

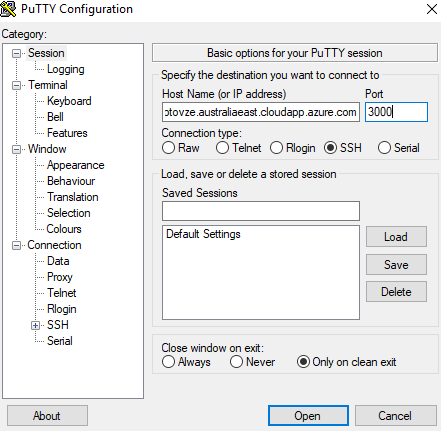


* We then find the resource created for the blockchain and click on deployments. After this process, we select microsoft-azure-blockchain from the list that appears.

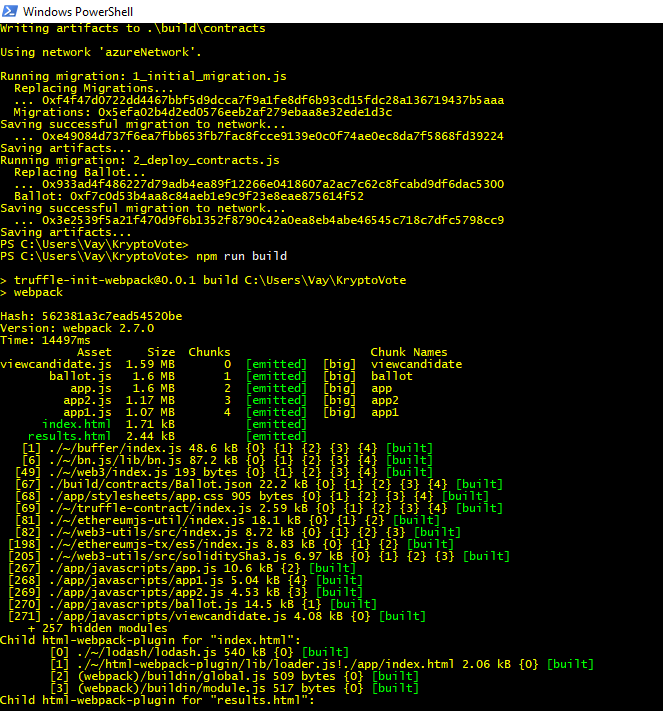


* Following this step, we get the configuration settings of our blockchain and we can use PuTTy to access the transactions nodes using [**"gethadmin@kryptovze.australiaeast.cloudapp.azure.com**](mailto:gethadmin@kryptovze.australiaeast.cloudapp.azure.com)**"**as hostname and 3000 as port.



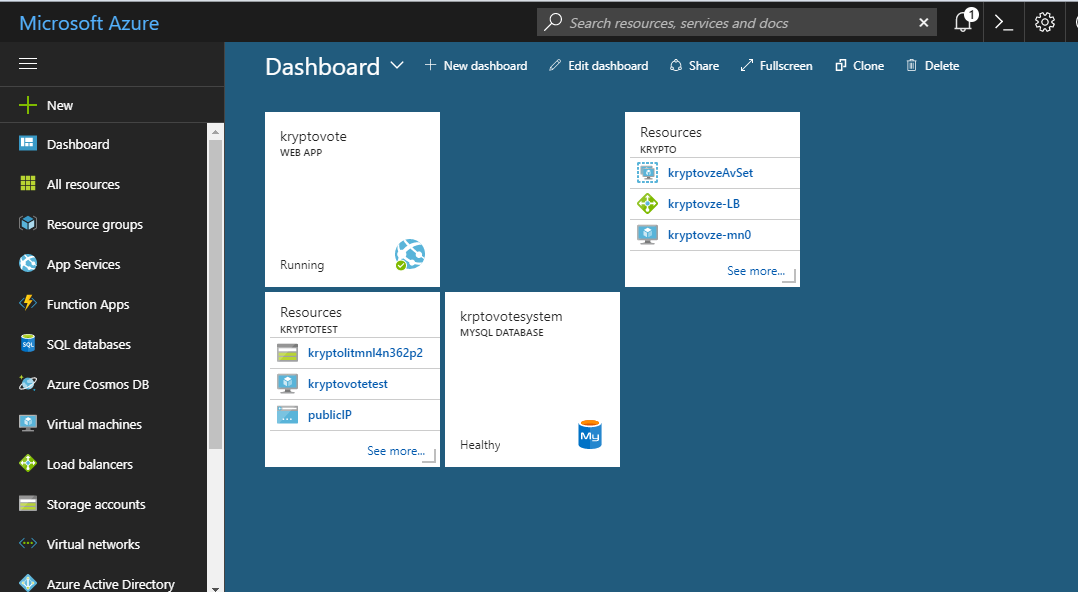


* In the aftermath of this process, we truffle migrate --network azureNetwork so that we put our Ballot.sol smart contract onto the private blockchain.



* an interact with the contract via web3js from the admin and client side.

### After creation of all the 3 services, the dashboard should have all the 3 resources as shown below.



---------------------- END OF FILE --------------------