

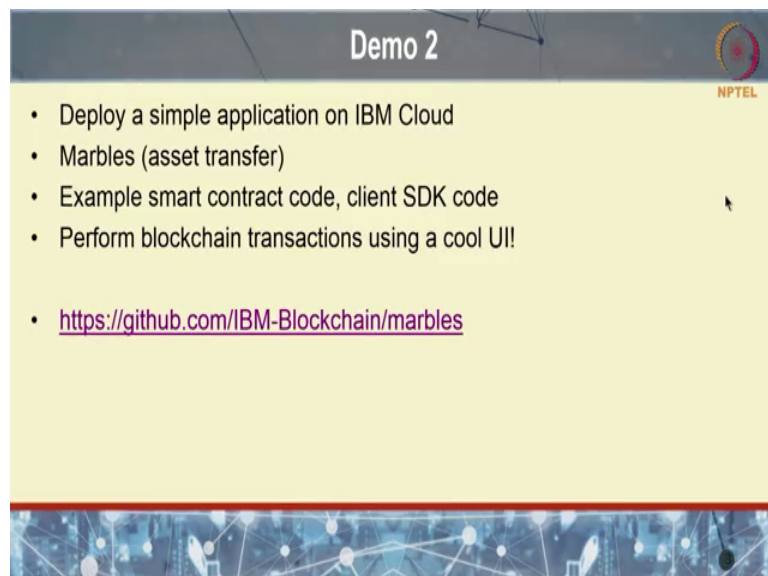
Blockchains Architecture, Design and Use Cases
Prof. Sandip Chakraborty
Prof. Praveen Jayachandran
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 26
Fabric Demo on IBM Blockchain Cloud – II

Hello and welcome back. So, the, we are going to continuous our demos of Hyperledger Fabric. The best way to go through some of these demos, is to work along with me, while you are seeing this video, try it yourself on IBM cloud. So, I would encourage you to do that. So, we in the last lecture we looked at, how to setup your own network, create a channel, add peers um, look at the blocks in your in your network and, and so on in your channel.

In this one going to look a demo application, simple application that will transfer marbles. So, we can think of it is, as an extrapolation of what bitcoin does. Bitcoin is actually transferring assets across, so you going to transfer bitcoins from one person to another person

(Refer Slide Time: 00:59)



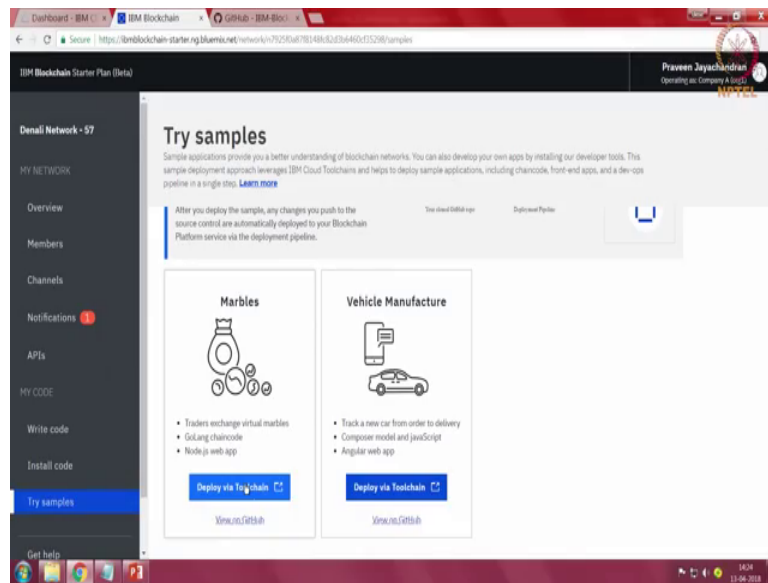
The slide is titled "Demo 2" and features the NPTEL logo in the top right corner. It contains a list of four bullet points: "Deploy a simple application on IBM Cloud", "Marbles (asset transfer)", "Example smart contract code, client SDK code", and "Perform blockchain transactions using a cool UI!". Below the list is a link to a GitHub repository: <https://github.com/IBM-Blockchain/marbles>. The slide has a yellow background and a blue decorative border at the bottom.

But here is a toy example, you going to transfer some marbles right. So, marbles have certain properties it has color, it has size we will look at that right. And this is very

simple application that you can deploy on, on Hyperledger fabric of course, you can think of very much for complex applications as well.

Apart from just deploying that code, we will also look at, what that code looks like, how you would write that smart contract code and also write the client application code, that will interact with the smart contract itself. And all this you also get cool UI with it I will show you that along the way.

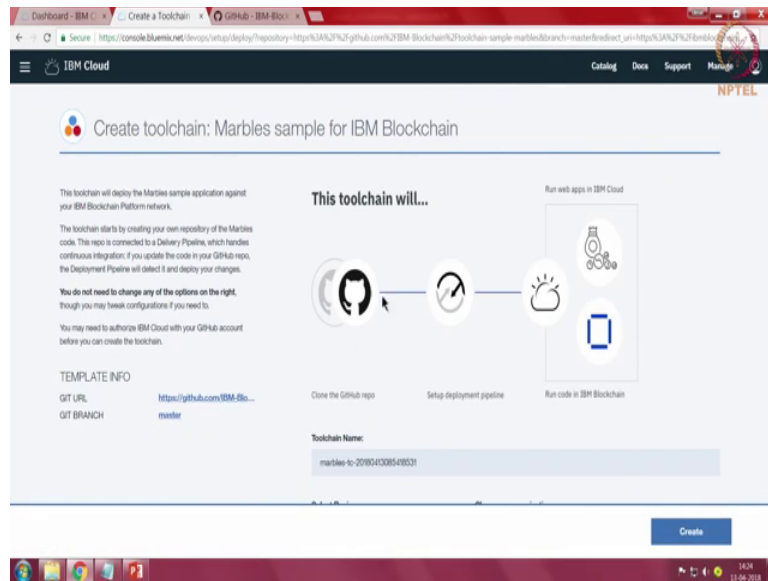
(Refer Slide Time: 01:42)



So, I am going to start by now creating this launching this application first. So, let us go first; if we go to the same network that you created last time the try samples tab here. There are two sample applications that you can deploy. So, let us start with first, the first one is on Hyperledger fabric, the second one is the on composer.

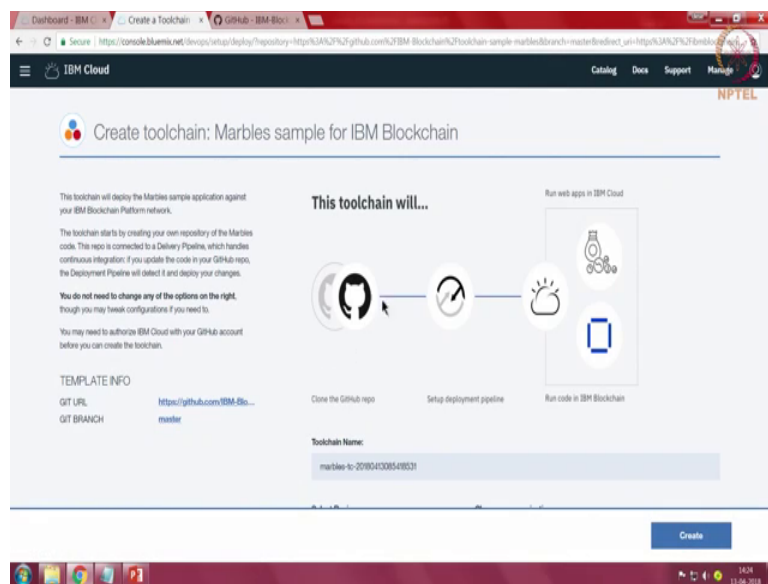
So, we will have the demo of that later, but let us first go deploy this on our network. So, this is going to go deploy, the marbles smart contract on to blockchain, ok. So, I was not a member of the default channels. So, I was trying to do this as company C I did not let me do that. So, let me go do this as company A. So, I am going to try and deploy this as come now company A, I am going to deploy marbles on the default channel. So now, A and B are the ones, who are going to be part of this application.

(Refer Slide Time: 02:31)



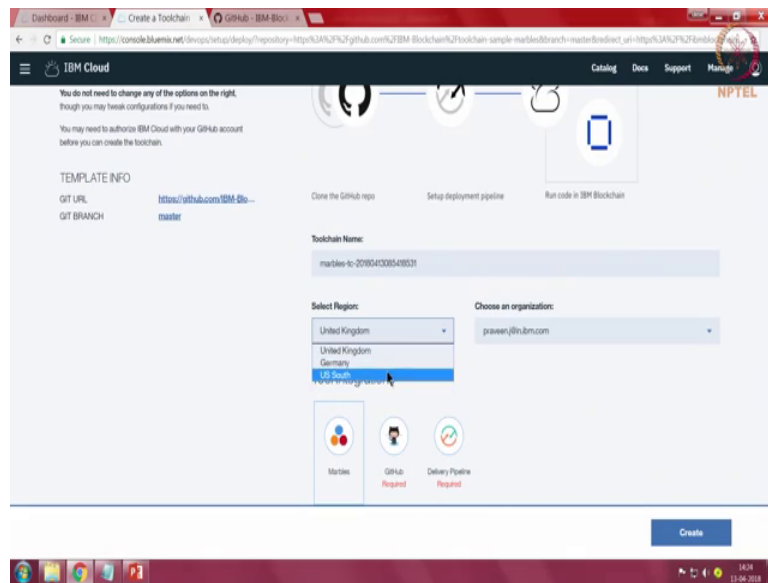
This was just a sample application you can of course, take the same code and deploy it on the second channel that you had right.

(Refer Slide Time: 02:38)



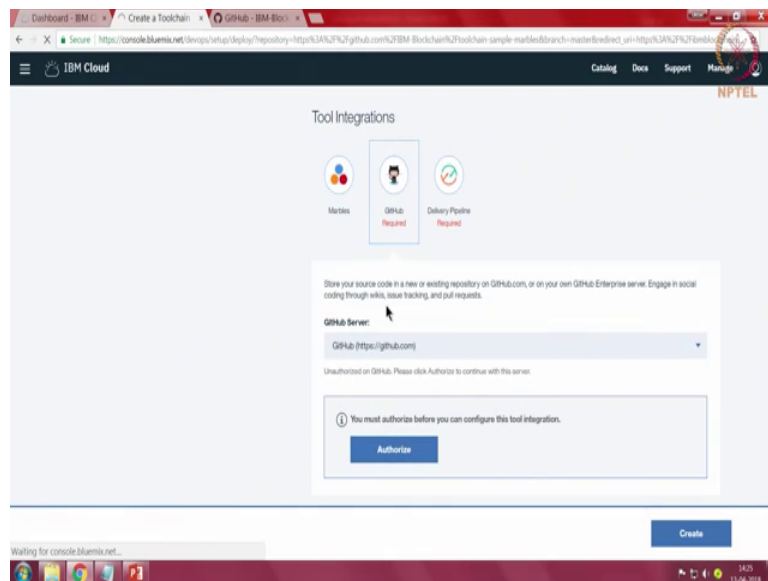
So, what is happening behind the scenes? So, in the behind the scenes, what is going to happen, as we going to clone the Github repo. So, IBM is made this piece of code public you will clone that into your own Github repository. There is a deployment pipeline that will go automatically deploy that code onto IBM cloud. We will deploy this part contracts, on to the channel we have select it and it will also go deploy the application, as the separate run time, right. So, all this is getting going to get deployed for you.

(Refer Slide Time: 03:04)



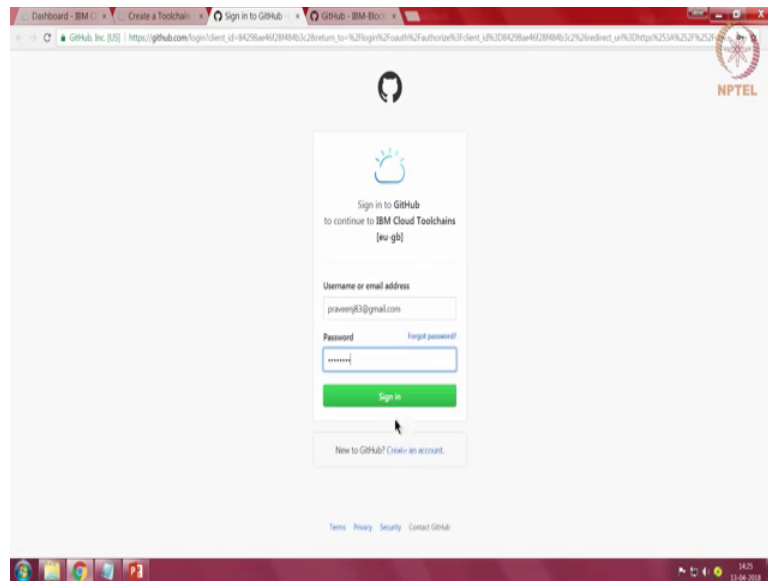
Let us select the regions, it is going to be us south again.

(Refer Slide Time: 03:13)

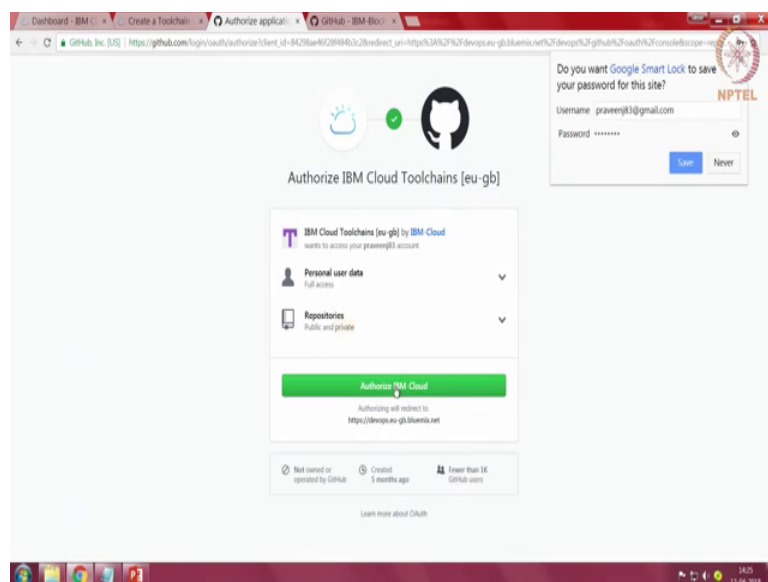


We going to deploy the marbles application Github, I am going give my authorization for my Github.

(Refer Slide Time: 03:16)

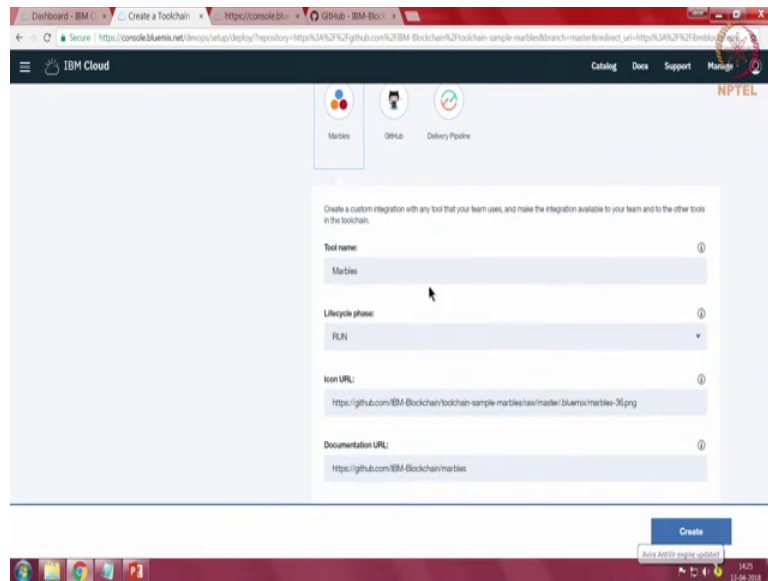


(Refer Slide Time: 03:29)

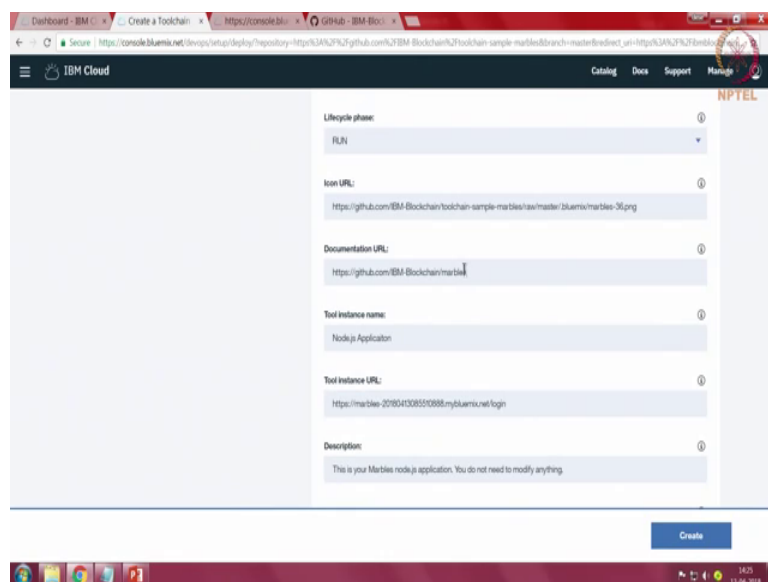


So, let me add that. I will authorize, it to clone into my Github. So, I am going to authorize the IBM cloud ok. So, I think that should get done and be reflected here.

(Refer Slide Time: 03:41)



(Refer Slide Time: 03:46)

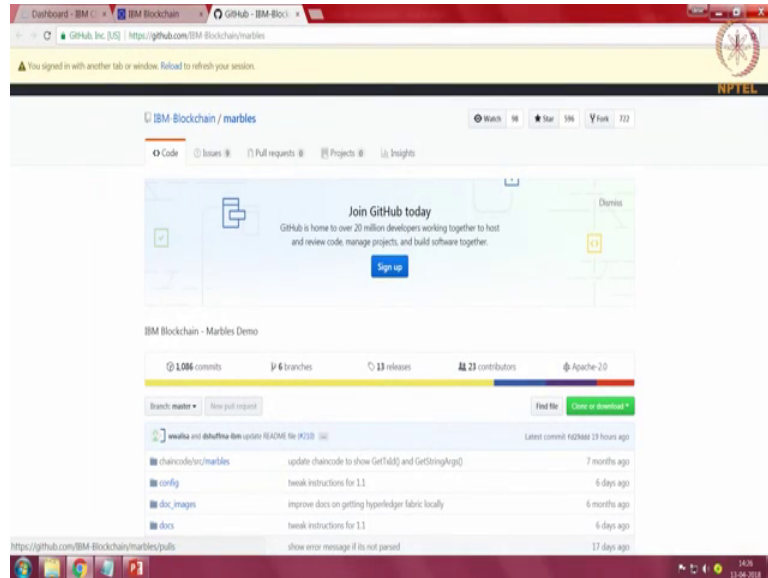


So, now that required has gone away, if you see it. So, if already authorized Github. And all these details you do not have to modify it, but these are some of the configurations there are set up gives you the URLs. It says that this is a Node.js application is being written, few other things. Let us go ahead and create this.

So, this is going to take about maybe five to ten minutes, really depends on your internet speed, because it is going to clone some Docker containers, it is going to do a bunch of things. So, it says 5 to 10 minutes, likely it going to take that much. So, in the meanwhile, what we will do is, while this gets created it also it has how much has

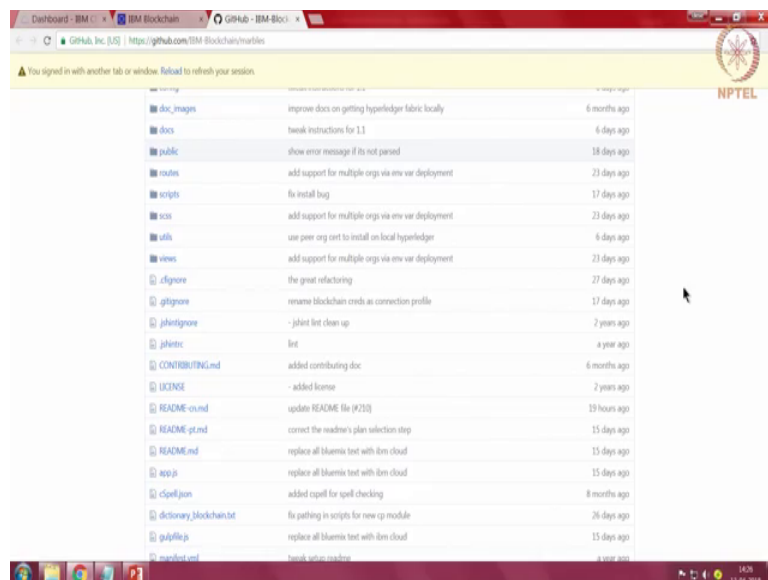
progressed. Let us go and look at the code itself for what the smart contract looks like, ok.

(Refer Slide Time: 04:27)



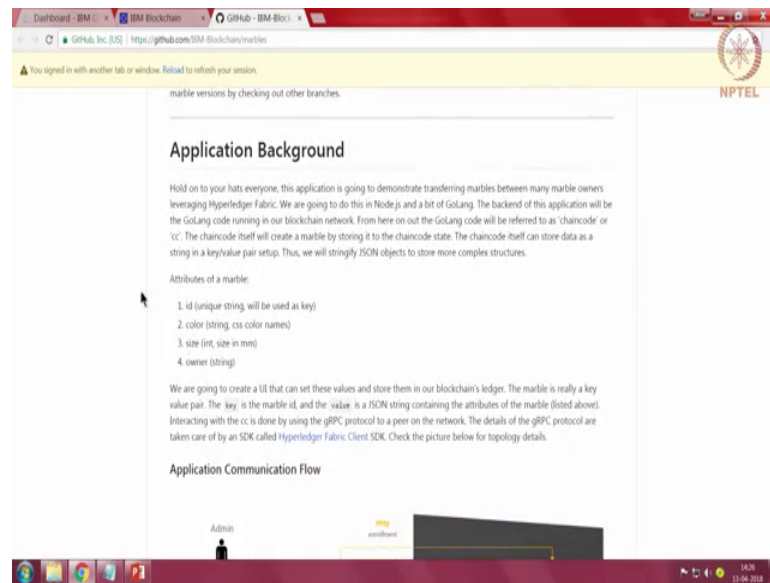
So, this is where IBM is made the code public. So, this is the marbles code. This is again, gone through various iterations.

(Refer Slide Time: 04:38)



It is fairly well written, and is a good way, for you to get a feel what a smart contract looks like.

(Refer Slide Time: 04:48)

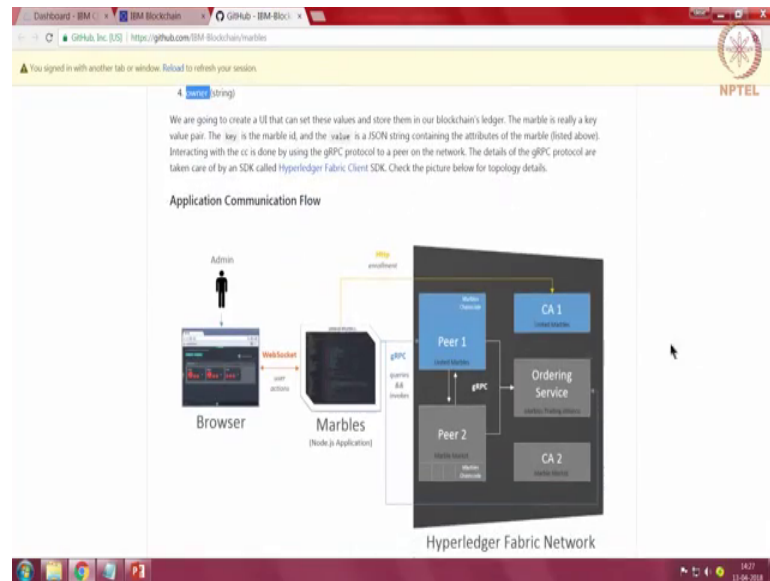


So, what is this actually going to do, right. So, let us look at this; so going to be creating marbles. So, each person can say I have a new marble add that to the network. So, that is one operation. They can transfer the ownership of marbles between people. So, if I own a marble, I can transfer that to somebody else and in that same way in I cannot change the ownership of the marbles that I do not own, right. If I do not own marble I cannot change someone else is marble to be my own.

So, that will be prevented. So, there will be an access control rule that we will implement, and there are some attributes to a marble that a marble has an id, it has a color there will be the multiple colors that are available. It has a size small or large and has an owner, right. So, once the marble is created each time it could change ownership across people and all that will be tracked on blockchain.

You are calling it marbles here, but you can think of any kind of an asset that is getting that is owned and I can, the ownership can transfer amongst people, right.

(Refer Slide Time: 05:53)

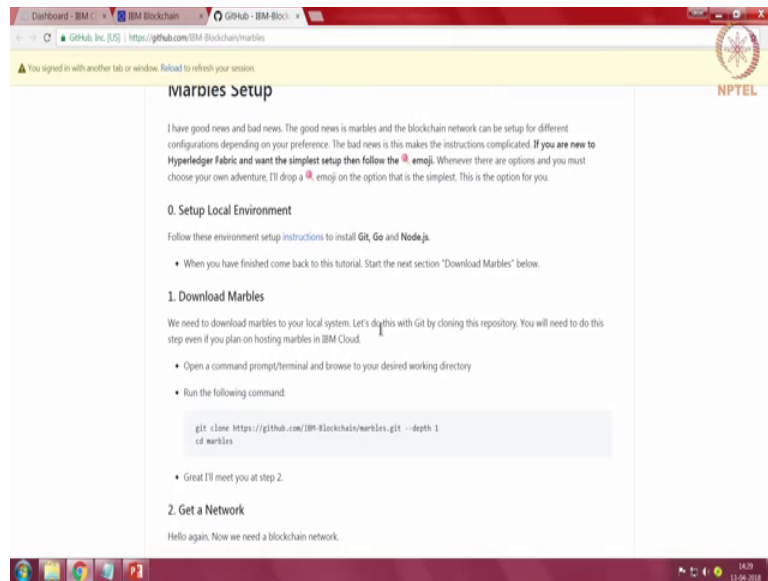


So, what is the architecture look like? So, what we have is there is the marbles application. There is an application is written in Node.js. It has a client which uses the node SDK that fabric provides to connect to the smart contracts on blockchain, right. So, there is, a smart contract, deployed on each of the pairs, this is the marbles smart contract, we will look at the smart contract also and just a little bit.

What is the network look like; we have two peers, this is on the default channel um. This is the ordering service and there are two CAs for that, that each of the peers are connected to like. So, the C actually the peer zone connects to the CA directly. The CA is the one showing identities and you going to use those identities and peer will valid with those identities. And apart from the Node.js application, we also have a browser UI from which we can trigger transactions via the application on to blockchain.

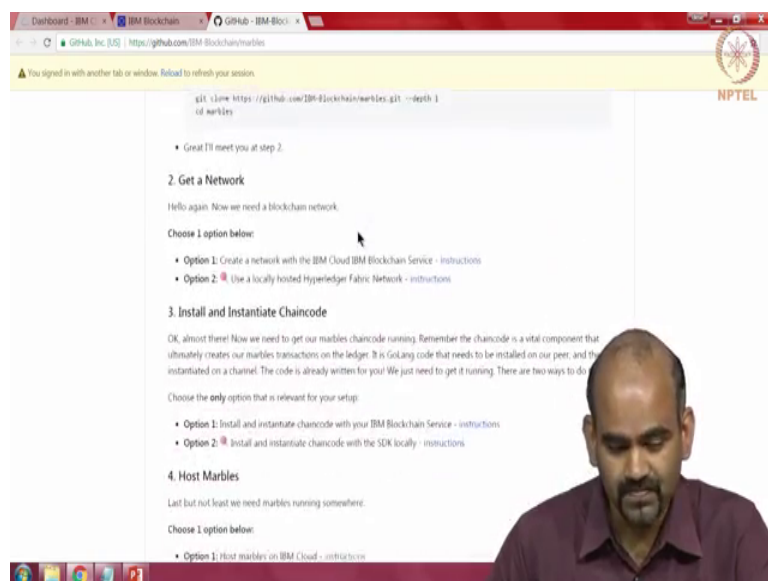
So, there is a web socket connection between the browser and the marbles Node.js application and from the application to the peers there is a gRPC Google RPC connection that uses the fabric SDK to communicate with blockchain. So, that is the overall network architecture. So, this tells you how to setup marbles yourself.

(Refer Slide Time: 07:14)

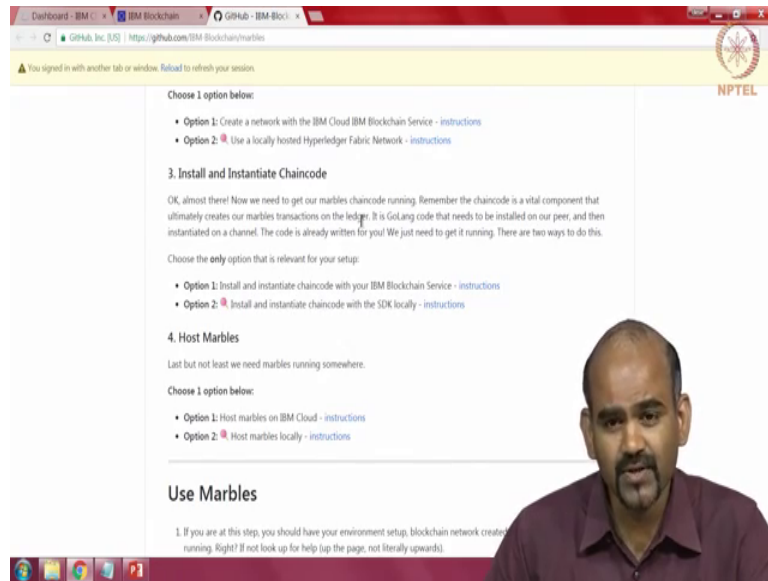


So, this will be the next demo that we have, right. It is if the IBM cloud makes it very simple for you just through a few clicks you can experience all of this, but what does it take to actually do this yourself. So, we are going to this, as the next demo.

(Refer Slide Time: 07:35)



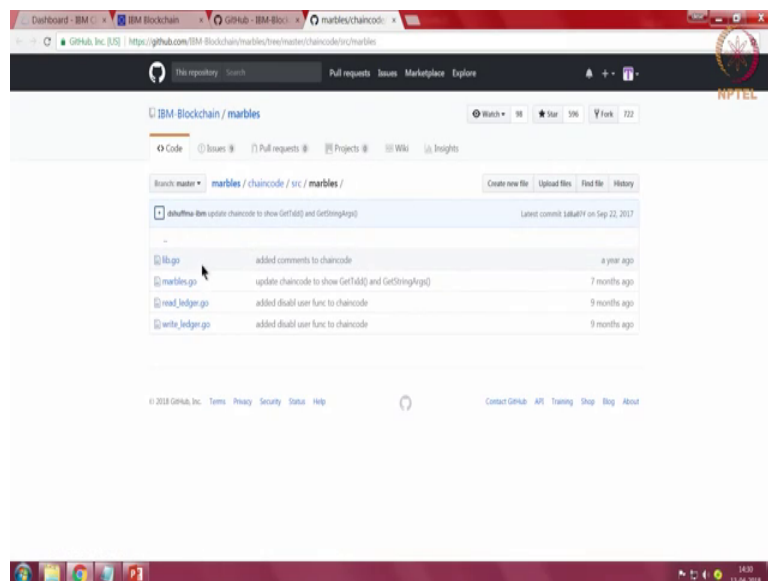
(Refer Slide Time: 07:36)



So hold; hold your horses for that it will take you through all the steps that is right now, in the five to ten minutes whatever is happening. We are going to do that step by step ourselves on virtual machine that or any other system, that you that you have, ok.

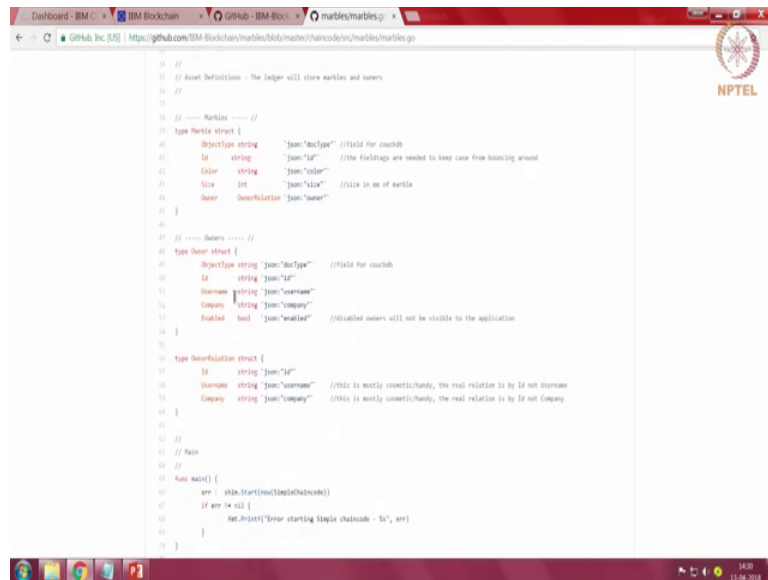
So, this is how the marbles we will draw go to the application or host at itself later, but let us look at little bit of the code itself, right. So, let us start with the, the smart contract code. So, let me open up. So, if you go to Github, there is a folder called chain code source marbles.

(Refer Slide Time: 08:15)



So, I am going to open that in another tab. So, let us go through these parts of this code piece, of code that is in this folder is what is getting deployed as a smart contract on each of the peers on blockchain, right. So, this is part of your install and (Refer Time: 08:25). So, this is the smart contract code that is run in a decentralized part.

(Refer Slide Time: 08:36)

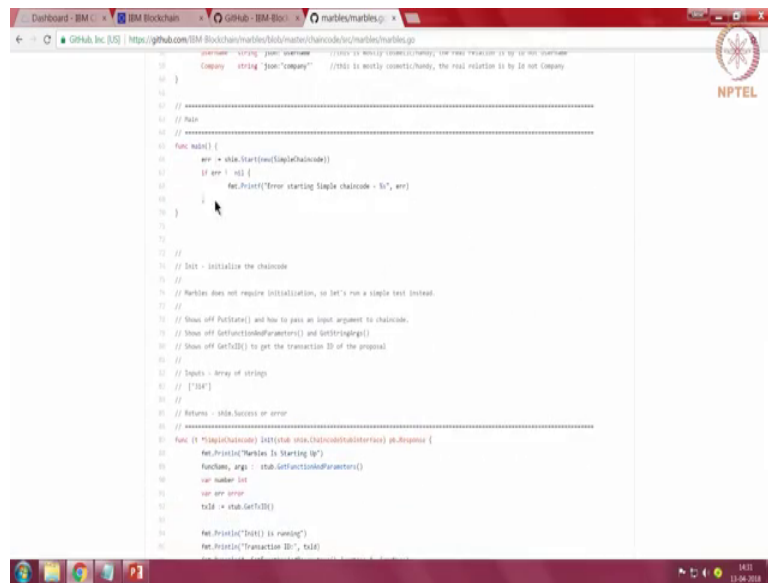


```
34 //
35 // Asset Definitions - The ledger will store marbles and owners
36 //
37 // ----- Marbles ----- //
38 type Marble struct {
39     ObjectType string "json:\"docType\"" //field for couchdb
40     Id string "json:\"id\"" //the fieldtags are needed to keep case from bouncing around
41     Color string "json:\"color\""
42     Size int "json:\"size\"" //size in mm of marble
43     Owner OwnerRelation "json:\"owner\""
44 }
45
46 // ----- Owners ----- //
47 type Owner struct {
48     ObjectType string "json:\"docType\"" //field for couchdb
49     Id string "json:\"id\""
50     Username string "json:\"username\""
51     Company string "json:\"company\""
52     Enabled bool "json:\"enabled\"" //disabled owners will not be visible to the application
53 }
54
55 type OwnerRelation struct {
56     Id string "json:\"id\""
57     Username string "json:\"username\"" //this is mostly cosmetic, the real relation is by Id not Username
58     Company string "json:\"company\"" //this is mostly cosmetic, the real relation is by Id not Company
59 }
60
61 //
62 // Main
63 //
64 func main() {
65     err := chain.StartLedgerChainCode()
66     if err != nil {
67         fmt.Println("Error starting ledger chaincode - %v", err)
68     }
69 }
```

So, let us open marbles dot go, right. What does it provide? It (Refer Slide Time: 08:36) defines, the data structures that we will be using. So, we have a real structure called marble, right. It has an id, it has a color, has a size and an owner, right. Now the owner is actually a relationship, it is actually of type owner relation.

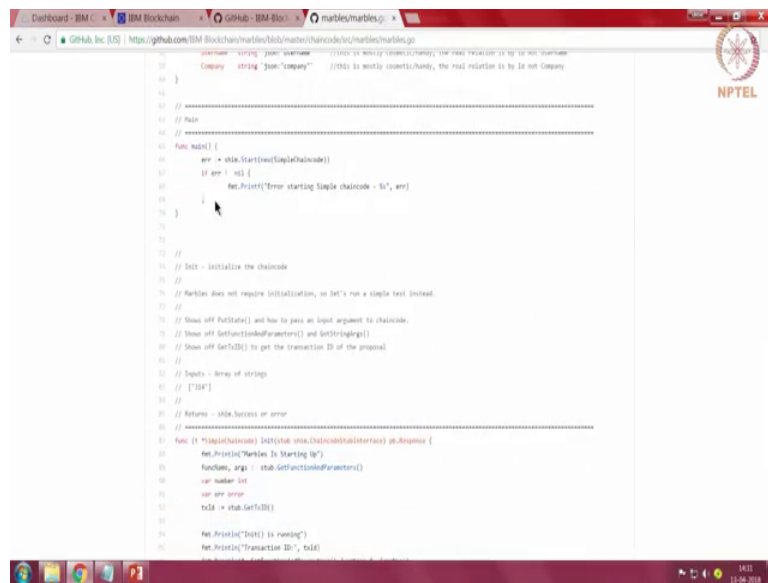
Now, owner relation itself has an id, user name and company, but I think the owner object is, is a little more interesting. So, it has the details about the object type. The main identifier is really this id field.

(Refer Slide Time: 09:11)



```
18     username: string, password: string //This is mostly cosmetic/handy, the real relation is by ID not username
19     company: string "Iron Company" //This is mostly cosmetic/handy, the real relation is by ID not Company
20 }
21 // =====
22 // Main
23 // =====
24 func main() {
25     err := vsh.StartNew(SimpleChaincode)
26     if err != nil {
27         fmt.Printf("Error starting Simple chaincode - %v", err)
28     }
29 }
30 //
31 // Init - initialize the chaincode
32 //
33 // Marbles does not require initialization, so let's run a simple test instead.
34 //
35 // Show off PutState() and how to pass an input argument to chaincode.
36 // Show off GetFunctionParameters() and GetStringArgs()
37 // Show off GetTxID() to get the transaction ID of the proposal.
38 //
39 // Inputs - array of strings
40 // ["ID"]
41 //
42 // Returns - vsh.Success or error
43 // =====
44 func (*SimpleChaincode) Init(vsh.ChaincodeStubInterface) pb.Response {
45     fmt.Println("Marbles is Starting Up")
46     functions, args := stub.GetFunctionParameters()
47     var number int
48     var err error
49     txID := stub.GetTxID()
50     fmt.Println("Init() is running")
51     fmt.Println("Transaction ID:", txID)
52 }
```

(Refer Slide Time: 09:15)



```
18     username: string, password: string //This is mostly cosmetic/handy, the real relation is by ID not username
19     company: string "Iron Company" //This is mostly cosmetic/handy, the real relation is by ID not Company
20 }
21 // =====
22 // Main
23 // =====
24 func main() {
25     err := vsh.StartNew(SimpleChaincode)
26     if err != nil {
27         fmt.Printf("Error starting Simple chaincode - %v", err)
28     }
29 }
30 //
31 // Init - initialize the chaincode
32 //
33 // Marbles does not require initialization, so let's run a simple test instead.
34 //
35 // Show off PutState() and how to pass an input argument to chaincode.
36 // Show off GetFunctionParameters() and GetStringArgs()
37 // Show off GetTxID() to get the transaction ID of the proposal.
38 //
39 // Inputs - array of strings
40 // ["ID"]
41 //
42 // Returns - vsh.Success or error
43 // =====
44 func (*SimpleChaincode) Init(vsh.ChaincodeStubInterface) pb.Response {
45     fmt.Println("Marbles is Starting Up")
46     functions, args := stub.GetFunctionParameters()
47     var number int
48     var err error
49     txID := stub.GetTxID()
50     fmt.Println("Init() is running")
51     fmt.Println("Transaction ID:", txID)
52 }
```

The rest of it is just cosmetic, right. So, there is a main function, but I think the two important functions to look at are Init and below that there is Invoke, right. What is the Init does, is the Init function of a smart contract. So, this is a required function that you need to implement in your smart contract.

(Refer Slide Time: 09:21)

```
Dashboard - IBM C x IBM Blockchain x GitHub - IBM Blo... x marbles:marbles.g...
https://github.com/IBM-Blockchain/marbles/blob/master/chaincode/chaincode.go

18     username string "juan" username //1000 is really expensive, the real relation is by ID not username
19     }
20     Company string "IBM Company" //IBM is mostly correct, handy, the real relation is by ID not Company
21 }
22 // =====
23 // Main
24 // =====
25 func main() {
26     err := vha.StartNewSingleChaincode()
27     if err != nil {
28         fmt.Println("Error starting single chaincode - %v", err)
29     }
30 }
31 // =====
32 //
33 // // Init - initialize the chaincode
34 // //
35 // // Marbles does not require initialization, so let's run a simple test instead.
36 // //
37 // // Show off PutState() and how to pass an input argument to chaincode.
38 // // Show off GetFunctionParameters() and GetStringArgs()
39 // // Show off GetTxID() to get the transaction ID of the proposal.
40 // //
41 // // Inputs - array of strings
42 // // ["ID"]
43 // //
44 // // Returns - vha.Success or error
45 // // =====
46 func (*vha) Init(chaincodeID string, vha vha.ChaincodeStubInterface) pb.Response {
47     fmt.Println("Marbles Is Starting %v")
48     function, args := stub.GetFunctionAndParameters()
49     var number int
50     var err error
51     txID := stub.GetTxID()
52     fmt.Println("Init() is running")
53     fmt.Println("Transaction ID:", txID)
54 }
55 // =====
```

What the Init function does, is it initializes your smart contract with any parameters you need to set, right. So, you can set certain values as initialization in your smart contract and this will get called at the time of instantiation of your chain code on a particular channel.

So, if installed this chain code on a peer and it is not the level of channel in this point, whether you instantiate that chain code for a particular channel saying this I want this copy of a chain code running on this channel. And then, there are a separate state for that chain code for the channel and you will be allow to initialize that state.

(Refer Slide Time: 10:15)

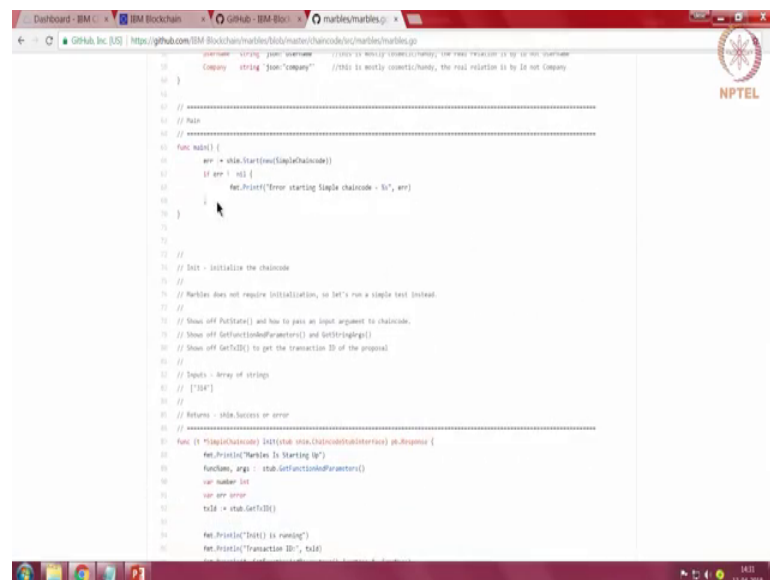
```
Dashboard - IBM C x IBM Blockchain x GitHub - IBM Blo... x marbles:marbles.g...
https://github.com/IBM-Blockchain/marbles/blob/master/chaincode/chaincode.go

56     fmt.Println("Init() is running")
57     fmt.Println("Transaction ID:", txID)
58     function, args := stub.GetFunctionAndParameters()
59     fmt.Println(" GetFunctionAndParameters() args count:", len(args))
60     fmt.Println(" GetFunctionAndParameters() args found:", args)
61 }
62 // =====
63 // expecting 1 arg for instantiate or upgrade
64 if len(args) == 1 {
65     fmt.Println(" GetFunctionAndParameters() arg(0) length", len(args[0]))
66 }
67 // expecting arg(0) to be length 9 for upgrade
68 if len(args[0]) == 9 {
69     fmt.Println(" Oh oh, arg(0) is empty...")
70 } else {
71     fmt.Println(" Great news everyone, arg(0) is not empty")
72 }
73 // convert numeric string to integer
74 number, err := strconv.Atoi(args[0])
75 if err != nil {
76     return vha.Error("Expecting a numeric string argument to Init() for instantiate")
77 }
78 // this is a very simple test. let's write to the ledger and error out on any errors
79 // it's handy to read this right away to verify network is healthy if it wrote the correct value
80 err = stub.PutState("sdTest", []byte(strconv.Itoa(number)))
81 if err != nil {
82     return vha.Error(err.Error()) //sdTest fail
83 }
84 }
85 }
86 // =====
87 // showing the alternative argument vha function
88 fmt := vha.GetStringArgs()
89 fmt.Println(" GetStringArgs() args count:", len(args))
90 fmt.Println(" GetStringArgs() args found:", args)
91 // =====
92 // store compatible marbles application version
```

So, what are we doing in this marble's application, what, what are we initializing; so, here we are going to take one argument just, just for just, just for illustration of how that work. It is going to be a numeric argument. So, we can actually (Refer Time: 10:27) this argument, you can you can send. That is an argument to your chain code at the time of instantiation, and you can just check whether it is, it is actually a numeric argument.

So, this is your way to validate any inputs, provided to your network. So, for instance you can initialize your network with saying, that are going to be three marbles that each of the members are going to own. So, I will create those marbles as, as part initialization for instance. You could do that, but we are not doing that here yet in this, in this example.

(Refer Slide Time: 11:01)



```
18     company string "company" //this is mostly cosmetic/pretty, the real relation is by its not Company
19 }
20 // =====
21 // Main
22 // =====
23 func main() {
24     err := vshim.Start(SingleChaincode)
25     if err != nil {
26         fmt.Println("Error starting single chaincode - %v", err)
27     }
28 }
29 //
30 //
31 // Data - initialize the chaincode
32 //
33 // Marble does not require initialization, so let's run a single test instead.
34 //
35 // Shows off PutState() and how to pass an input argument to chaincode.
36 // Shows off GetFunctionParameters() and GetStringArgs()
37 // Shows off GetTxID() to get the transaction ID of the proposal.
38 //
39 // Inputs - array of strings
40 // ["TXID"]
41 //
42 // Returns - vshim.Success or error
43 // =====
44 func (*SingleChaincode) Init(vshim.ChaincodeStubInterface) pb.Response {
45     fmt.Println("Marbles: In Starting Up")
46     functions, err := vshim.GetFunctionParameters()
47     var number int
48     var err error
49     txID := vshim.GetTxID()
50
51     fmt.Println("Data is wrong")
52     fmt.Println("Transaction ID:", txID)
```

This just goes through that, and all its stores are just, just some key. This is again, as an example of what you can do. So, you can put state, you can add a new key and put some value to it. So, you just storing, saying the UI is, is 4.0.1. So, the initialization steps just allow you to initialize, but let us look at the Invoke, right.

(Refer Slide Time: 11:23)

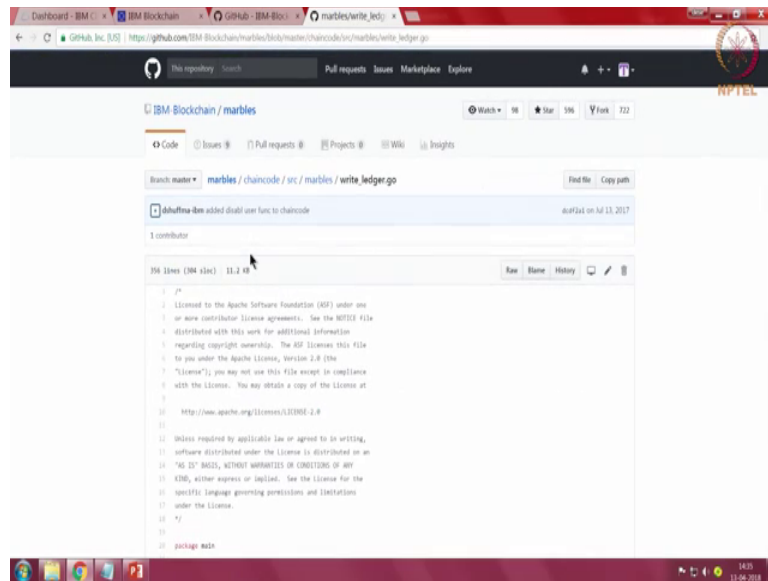
```
134 }
135 //
136 // Invoke - the entry point for invocations
137 //
138 function invoke(chaincode, args) {
139     function getFunctionFromParameters() {
140         let f = args[0];
141         let f = f.toLowerCase();
142         return f;
143     }
144     // Handle different functions
145     let f = getFunctionFromParameters();
146     if (f === "init") { //Initialize the chaincode state, used as reset
147         return init(chaincode, args);
148     } else if (f === "read") { //query: read ledger
149         return read(chaincode, args);
150     } else if (f === "write") { //query: write to ledger
151         return write(chaincode, args);
152     } else if (f === "delete_marble") { //delete a marble from state
153         return delete_marble(chaincode, args);
154     } else if (f === "set_marble") { //create a new marble
155         return set_marble(chaincode, args);
156     } else if (f === "set_owner") { //change owner of a marble
157         return set_owner(chaincode, args);
158     } else if (f === "get_history") { //read a history of a marble
159         return get_history(chaincode, args);
160     } else if (f === "get_marbles_by_range") { //read a bunch of marbles by start and stop id
161         return get_marbles_by_range(chaincode, args);
162     } else if (f === "disable_owner") { //disable a marble owner from appearing on the ID
163         return disable_owner(chaincode, args);
164     }
165     // error out
166     return "Received unknown invoke function name: " + f;
167 }
```

The Invoke is what is going to get called each time your chain code is going to get is you trying to perform a transaction no, you are trying to queries some values stored in the chain code then it is comes through the Invoke function.

So, the peer will be calling there is Invoke function in the chain code and that is how transactions execute, and everything within this Invoke function, whatever is the execution is going around atomically which means that any state you change here. Either all of it will change or none of it will change, right. So here, if you look at some of the some of the functions that are implemented; so implements a simple read and write, but we can do delete marble you can initialize the marble you can set the owner for a particular marble you can read everything. So, tell me all the marbles are (Refer Time: 12:15) all the users, tell me everything. So, that is another query.

You can get history for a particular entity. So, you for a particular marble tell me all the transactions that have happened for it, right. May be chains shown a show five times shall be all those transactions. So, you will you can get go get history as well. You can get marbles by range. So, based on the name of the marble whatever if we identify you have given, give me every all give me all the marbles between range this between this value and this value in the id range, right. So, we can do that, ok.

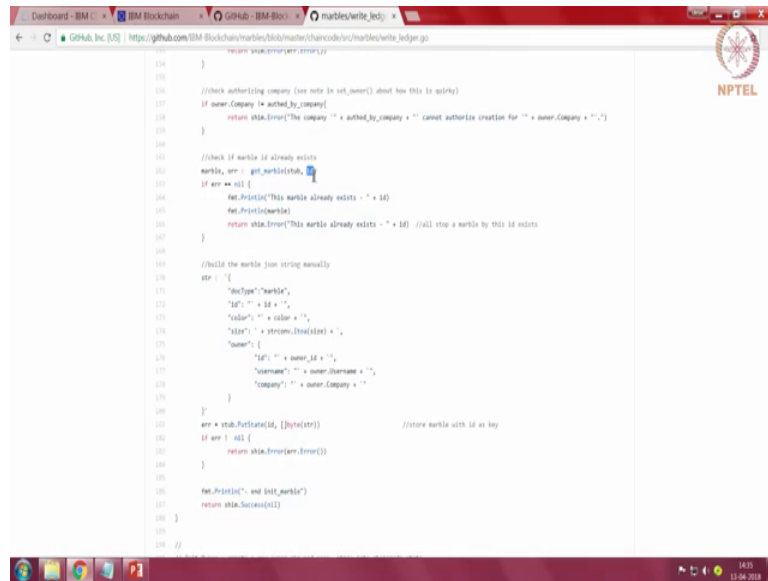
(Refer Slide Time: 13:05)



So, let us go look at some of these examples. So, let's go back up a folder and let us look at write ledger dot go. So, this has all the functions that are modifying the state of the chain code itself. So, these will be committed as transactions on blockchain when, when Invoked, ok.

So, let us take one example I will come back to the delete marble, but let us go to Init marble right. So, what Init marble does this is as it shows as it says in the comments here. It is going to create a new marble with a certain set of parameters. So, it takes a few parameters as inputs. So, this is the args string adding, it expects 5 parameters. So, that is you can do a check on the number of parameters and then, what are what are those parameters right.

(Refer Slide Time: 13:53)



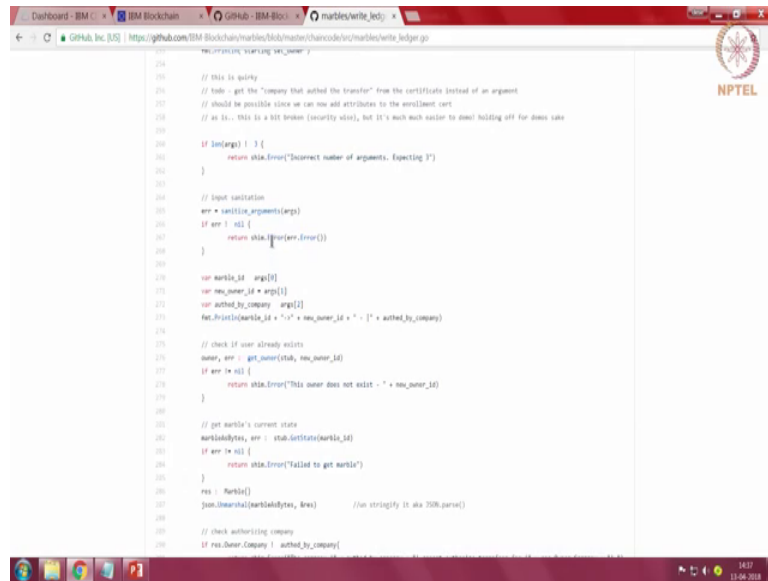
```
154 }
155
156 //check authorizing company (see note in set_name) about how this is going)
157 if owner.Company != author_by_company{
158     return fmt.Errorf("the company '%s' author_by_company '%s' cannot authorize creation for '%s' + owner.Company + "'")
159 }
160
161 //check if marble id already exists
162 marble, err := get_marble(id)
163 if err == nil {
164     fmt.Printf("This marble already exists - " + id)
165     fmt.Printf(marble)
166     return fmt.Errorf("This marble already exists - " + id) //will stop a marble by this id exists
167 }
168
169 //build the marble json string manually
170 str := []
171 "type": "marble",
172 "id": " + id + ",
173 "color": " + color + ",
174 "size": " + stream.ToInt() + ",
175 "owner": {
176     "id": " + owner_id + ",
177     "username": " + owner_username + ",
178     "company": " + owner_company + "
179 }
180 }
181
182 err = stub.PutState(id, []byte(str)) //store marble with id as key
183 if err != nil {
184     return fmt.Errorf(err.Error())
185 }
186
187 fmt.Printf("\n, and get_marble")
188 return fmt.Success()
189 }
190 }
191 }
```

So, it is doing a few checks you are saying the third argument must be a numeric string and. So, on, and then it is also checks whether the id you provided, is actually unique if it is not then the marble already exists and you cannot how can create a marble that already exist. So, you are going to get an error there.

So, all these errors would result in the transaction itself throwing an error and you will get that, as part of the notification back from lock chain. So, here transaction has failed and you will also the error message along with it, ok; so now, all the checks up first. So, we are going to create a new marble. So, we going to set dock type is marble. There is an id, color, size and owner. So, you going to set all of those things, as provided in the user input and you are going to call put state to store that information on to the ledger. So, the put state is actually a call to the blockchain ledger the state database and you are going to store this key value peer. The key is the id and the value are the object representing the marble itself, with all of these parameters, right.

So now, if once the put state is unused stored that on a blockchain. Remember this is just the endorsement phase when this is executing it. After this, all the nodes have to execute it has to go through ordering service has to be validate it and only then, we will get committed into the final state and blockchain, right. So, that is the process.

(Refer Slide Time: 15:26)



```
254 // @param {string} newOwner
255
256 // This is quirky
257 // todo - get the "company that authorized the transfer" from the certificate instead of an argument
258 // should be possible since we can now add attributes to the enrollment cert
259 // as is.. this is a bit broken (security wise), but it's such such easier to debug hitting off for demo sake
260
261 if (args.length !== 3) {
262   return shim.error("Incorrect number of arguments. Expecting 3")
263 }
264
265 // Input sanitation
266 var err = validate_arguments(args)
267 if (err) {
268   return shim.error(err)
269 }
270
271 var marble_id = args[0]
272 var new_owner_id = args[1]
273 var authorized_company = args[2]
274 var hash = sha256(marble_id + "-" + new_owner_id + "-" + authorized_company)
275
276 // check if user already exists
277 var user_err = get_user_by_id(new_owner_id)
278 if (user_err) {
279   return shim.error("This owner does not exist - " + new_owner_id)
280 }
281
282 // get marble's current state
283 var marble_bytes, err = shim.getState(marble_id)
284 if (err) {
285   return shim.error("Failed to get marble")
286 }
287
288 var res = Buffer.from(marble_bytes, 'hex') //on strings it aka JSON.parse()
289
290 // check authorizing company
291 if (res.author_company !== authorized_company)
```

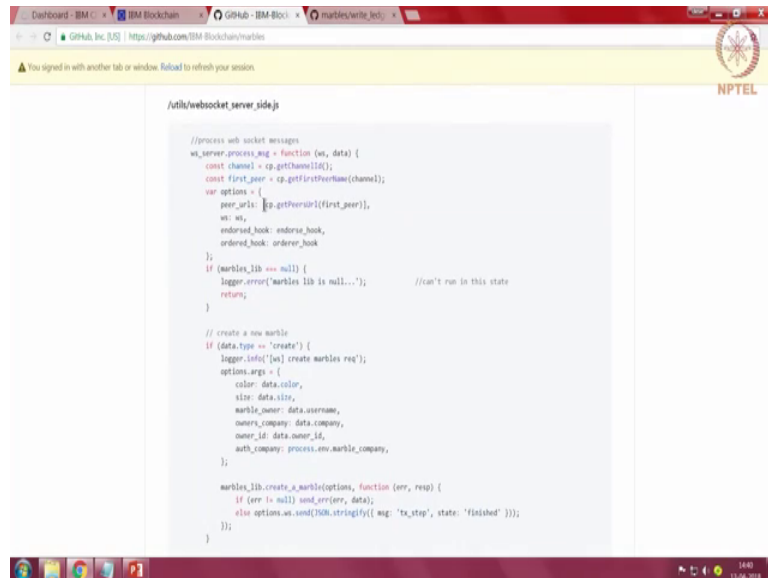
You can also Init owner. So, this is creating a new owner or new possible owner in the blockchain, you can set owner for a particular marble. So, what this we will do is the does the initial sanity checks. It checks if the provided user, the new owner is actually a valid user. So, you it does that. It gets the current state of the marble. It tries to find out who is a current owner. So, it does a get state. So, it goes and looks up the blockchain ledger to find out who owns this marble.

Now, check who is authorizing this right? So, the person who Invoked this transaction is that person the owner of this marble. If you are not owner then you are not going to be, you are not going authorized them to transfer the marbles are they do not own, right. So, you have to own the marble you are trying to transfer. So, unless you do that this cannot be done.

(Refer Slide Time: 16:12)

So, let us go back to our Github. And let us look we will go through the chain codes. What are the other aspects the application? So, the other aspects are the application code themselves.

(Refer Slide Time: 18:03)



```
./utils/websocket_server_side.js

//process web socket messages
ws_server_process_msg = function (ws, data) {
  const channel = cp.getChannelID();
  const first_peer = cp.getFirstPeerName(channel);
  var options = {
    peer_urls: [cp.getPeerID(first_peer)],
    ws: ws,
    endorsed_hook: endorse_hook,
    ordered_hook: orderer_hook
  };
  if (marbles_lib === null) {
    logger.error('marbles lib is null...'); //can't run in this state
    return;
  }

  // create a new marble
  if (data.type == 'create') {
    logger.info('[ws] create marbles req');
    options.args = {
      color: data.color,
      size: data.size,
      marble_owner: data.username,
      owners_company: data.company,
      owner_id: data.owner_id,
      auth_company: process.env.marble_company,
    };

    marbles_lib.create_a_marble(options, function (err, resp) {
      if (err != null) send_err(err, data);
      else options.ws.send(JSON.stringify({ msg: 'tx_step', state: 'finished' }));
    });
  }
}
```

So, let us look at that briefly. So, this all the application code is inside the utils directly and there is a websocket server side dot js. So, this is the server side Node.js application that we talked about and this is the one that is going to interface, with the blockchain use the client SDK use the node, node SDK to call blockchain to perform transactions.

So, what are some other things it does it, it is a channel id it gets the peer URL; so all of these will be configuration parameters for the application and if you if you want to create and marbles. So, this is data type is creating then what I do, do is, is create an object and it will call create a marble, right. It is part of the library is folder. So, you going to create a marble with these, these parameters and that create a marble will be calling the SDK functions.

(Refer Slide Time: 18:57)

```

owner_company: data.company,
owner_id: data.owner_id,
auth_company: process.env.marble_company,
});

marbles_lib.create_marble(options, function (err, resp) {
  if (err != null) send_error(err, data);
  else options.as.send(JSON.stringify({ msg: 'tx_step', state: 'finished' }));
});

// transfer a marble
else if (data.type == 'transfer_marble') {
  logger.info('[ui] transferring res');
  options.args = {
    marble_id: data.id,
    owner_id: data.owner_id,
    auth_company: process.env.marble_company
  };

  marbles_lib.set_marble_owner(options, function (err, resp) {
    if (err != null) send_error(err, data);
    else options.as.send(JSON.stringify({ msg: 'tx_step', state: 'finished' }));
  });
}
...

```

This snippet of `process_msg()` receives all websocket messages (code found in `app.js`). It will detect what type of (websocket) message was sent. In our case, it should detect a `transfer_marble` type. Looking at that code we can see it will setup an `options` variable and then kick off `marbles_lib.set_marble_owner()`. This is the function that will tell the SDK to build the proposal and process the transfer action.

Next let's look at that function.

`/utils/marbles.cc.lib.js`

(Refer Slide Time: 19:02)

build the proposal and process the transfer action.

Next let's look at that function.

`/utils/marbles.cc.lib.js`

```

//-----
// Set Marble Owner
//-----
marbles_chaincode.set_marble_owner = function (options, cb) {
  console.log('');
  logger.info('Setting marble owner...');

  var opts = {
    peer_urls: g_options.peer_urls,
    peer_tx_opts: g_options.peer_tx_opts,
    channel_id: g_options.channel_id,
    chaincode_id: g_options.chaincode_id,
    chaincode_version: g_options.chaincode_version,
    event_urls: g_options.event_urls,
    endorsed_hook: options.endorsed_hook,
    ordered_hook: options.ordered_hook,
    cc_function: 'set_owner',
    cc_args: [
      options.args.marble_id,
      options.args.owner_id,
      options.args.auth_company
    ],
  };

  fvw.invoke_chaincode(errs1108, opts, cb);
};
...

```

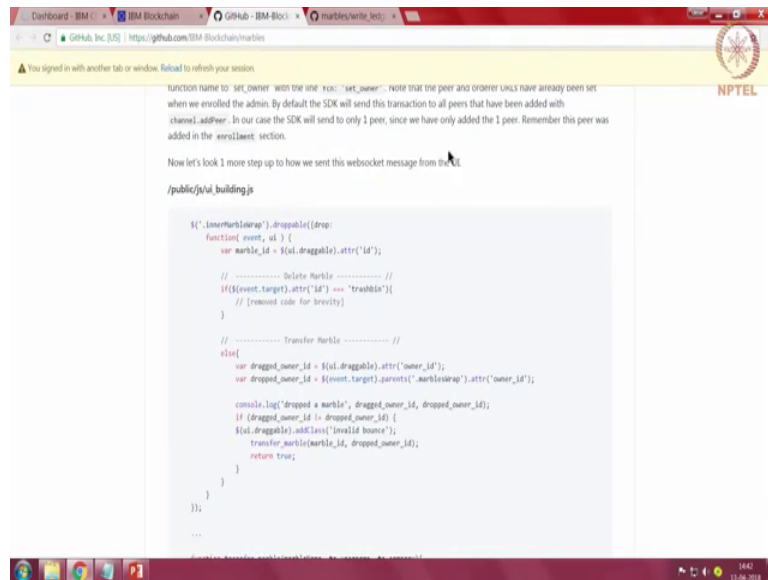
The `set_marble_owner()` function is listed above. The important parts are that it is setting the proposal's invocation function name to `'set_owner'` with the line `cc_function: 'set_owner'`. Note that the peer and channel URLs have already been set

Likewise, you can also transfer a marble, it calls that right. So, this is the library function that I talked about. So, this has functions for set marble owner and will also have something for create marble also, right. So, the set an owner is if you look at it, it has the peer URLs the channel id, this particular chain code id for the for this marbles chain code, all these are already set in earlier path of the code, right.

It what about endorsements you need right and finally, the chain code function to call. So, it is going to call the set owner function this we saw this function previously in the write ledger dot go file and it has the arguments that need to be passed in chain code.

So, it sends all of this information, it calls Invoke chain code. So, this is something in the SDK. So, you go Invoke the chain code with these parameters and that will then call the blockchain will send, send control to the peer to Invoke this particular function on the chain code ok. So, that was the server side application.

(Refer Slide Time: 20:03)



```
function name to 'set_owner' with the line 'set_owner'. Note that the peer and channel IDs have already been set when we enrolled the admin. By default the SDK will send this transaction to all peers that have been added with channel_address. In our case the SDK will send to only 1 peer, since we have only added the 1 peer. Remember this peer was added in the 'enrollPeer' section.
```

Now let's look 1 more step up to how we sent this websocket message from the UI

/public/js/ui_building.js

```
$( '.innerFormArea' ).draggable( {
  function( event, ui ) {
    var marble_id = $(ui.draggable).attr( 'id' );

    // ----- Delete Marble ----- //
    if( $(event.target).attr( 'id' ) === 'trashbin' ){
      // [removed code for brevity]
    }

    // ----- Transfer Marble ----- //
    else{
      var dragged_owner_id = $(ui.draggable).attr( 'owner_id' );
      var dropped_owner_id = $(event.target).parent( '.marblesArea' ).attr( 'owner_id' );

      console.log( 'Dropped a marble', dragged_owner_id, dropped_owner_id );
      if( dragged_owner_id != dropped_owner_id ){
        $(ui.draggable).addClass( 'invalid bounce' );
        transfer_marble( marble_id, dragged_owner_id );
        return true;
      }
    }
  }
});
```

So, apart from that you can also have UI. So, we have, we have we have a UI for this and there is ui building the js and it has the abilities that you will (Refer Time: 20:16) capabilities you will see here. Actually, we able to easily drag and drop marbles as a way of transferring ownership, right; so all those UI capabilities are also available as dot js code. By that is all this is now a step removed from the blockchain itself. So, I will leave you to, to see these things that leisure ok.

(Refer Slide Time: 20:28)

```
$(event.target).attr('id') === 'trashbin')
// (removed code for brevity)
}

// ----- Transfer Marble ----- //
$(el)
var dragged_owner_id = $(el.draggable).attr('owner_id');
var dropped_owner_id = $(event.target).parents('.marbleswrap').attr('owner_id');

console.log('dropped a marble', dragged_owner_id, dropped_owner_id);
if (dragged_owner_id != dropped_owner_id) {
  $(el.draggable).addClass('invalid bounce');
  transfer_marble(marble_id, dragged_owner_id);
  return true;
}
}
}

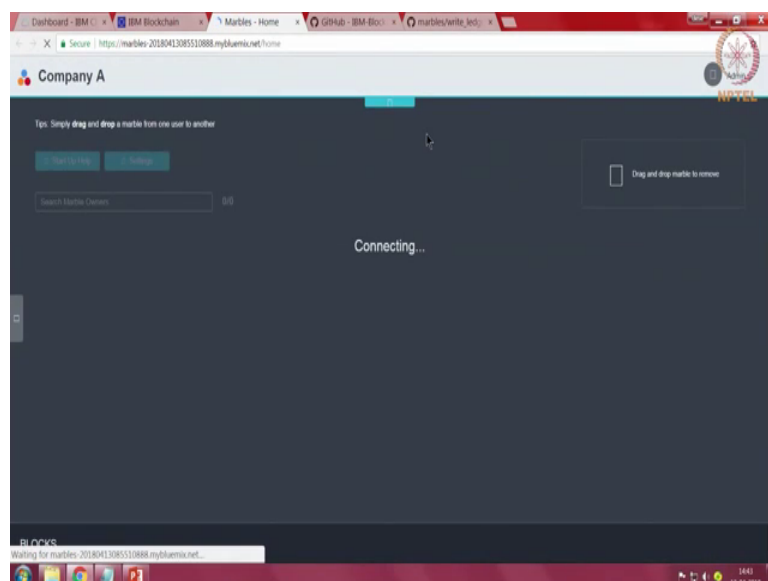
...

function transfer_marble(marbleId, to_username, to_company){
  show_to_step({ state: 'building_proposal' }, function () {
    var obj = {
      type: 'transfer_marble',
      id: marbleId,
      owner_id: to_owner_id,
      v: 1
    };
    console.log('sent + ' sending transfer marble msg', obj);
    ws.send(JSON.stringify(obj));
    refreshHomePage();
  });
}
```

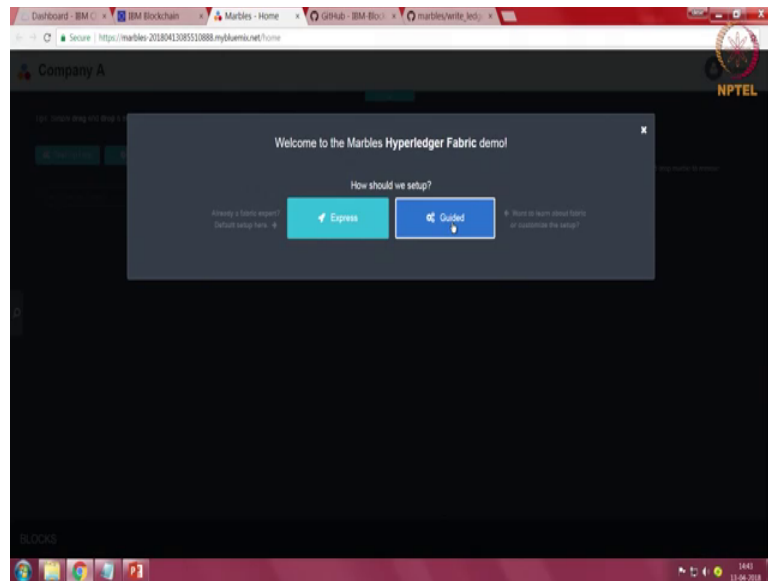
In the first section referencing \$('*.innerMarbleswrap') you can see we used jQuery and jQuery-UI to implement the drag and

So now, let us go back to our UI are R and C, ok. Our marbles application has launched. So, you will see that it is no longer this black and white picture this actually a color picture that tells you when it is, it is ready to launch. So, let us go ahead and launch it.

(Refer Slide Time: 21:01)

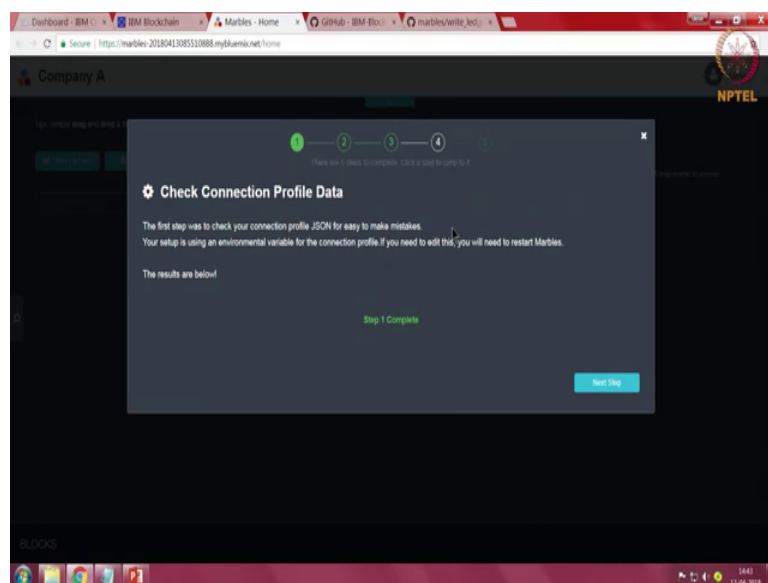


(Refer Slide Time: 21:04)



So, you are going to launch this marbles application. So, this is now been deployed on your default channel, that you had previously. You have two organizations or company A and company B that are part of this demo ok.

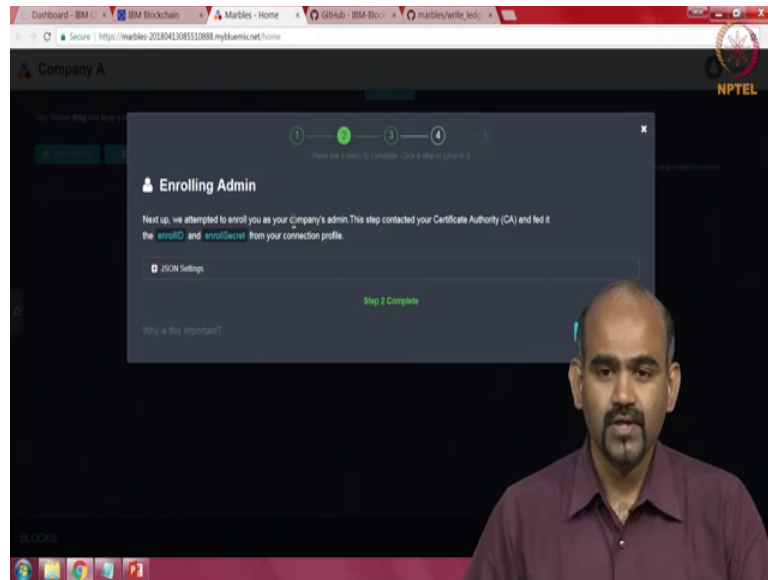
(Refer Slide Time: 21:17)



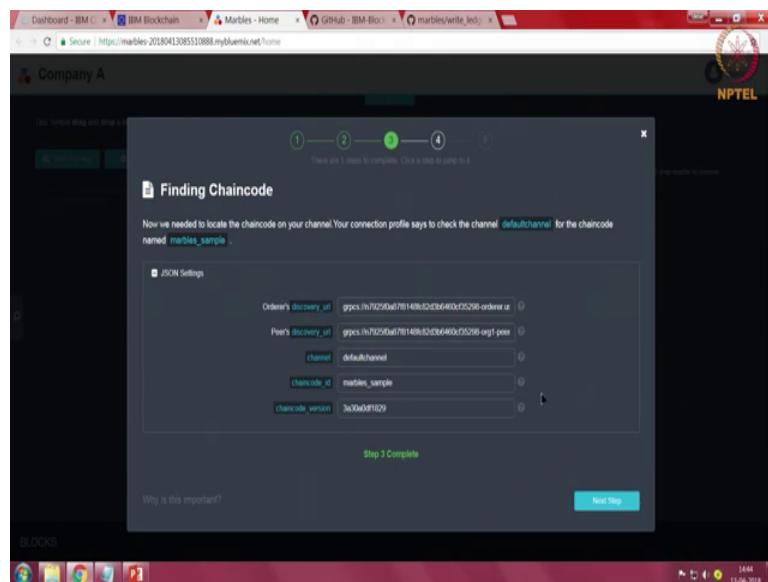
So, let us go through the guided tool, for now. So, what are some of the things that are happening in the back at, right? We kind of gone pass this is just let IBM cloud deal with all these, thing what are some other thing we are doing. One is the connection profile. So, this tells you what is the network, how do you connect to the network, what peer to you need to connect to who i's the orderer what is the policy.

So, all of that is captured, what are the channels I mean all that is caption in your connection profile. So, you need that set. So, as part of your network there will be available in one phase, you can actually go, look at your connection profile.

(Refer Slide Time: 21:52)



(Refer Slide Time: 22:05)

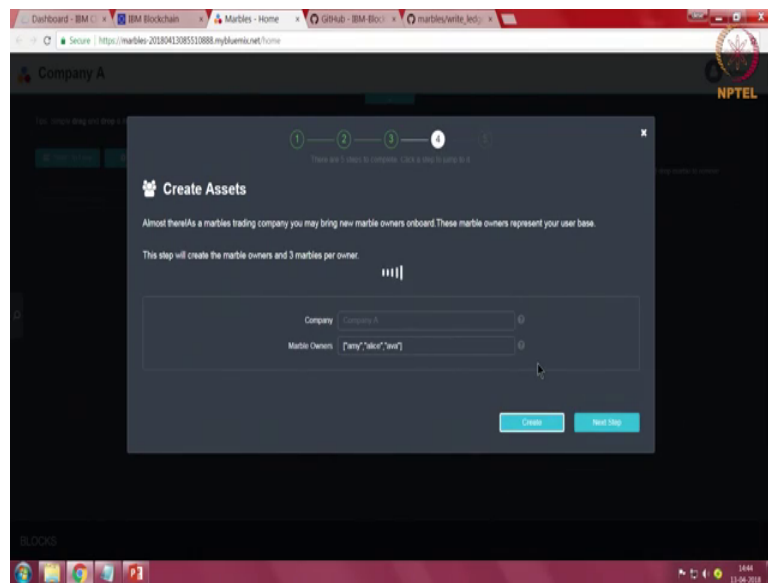


So, next thing is you have to set up identity right. So, you have identities for your admins, for each of your organizations um. So, you have that you need to have that setup. Then you have to get your chain code deploy, that chain code or install that chain code

on each of the peers and instantiate that chain code on the default channel on which you want to run this.

So, that is happening in step three. So, if you then, each of these will also give you the settings that that are being used. So, you can go through the SBM as well, but all this has been done for you in the background.

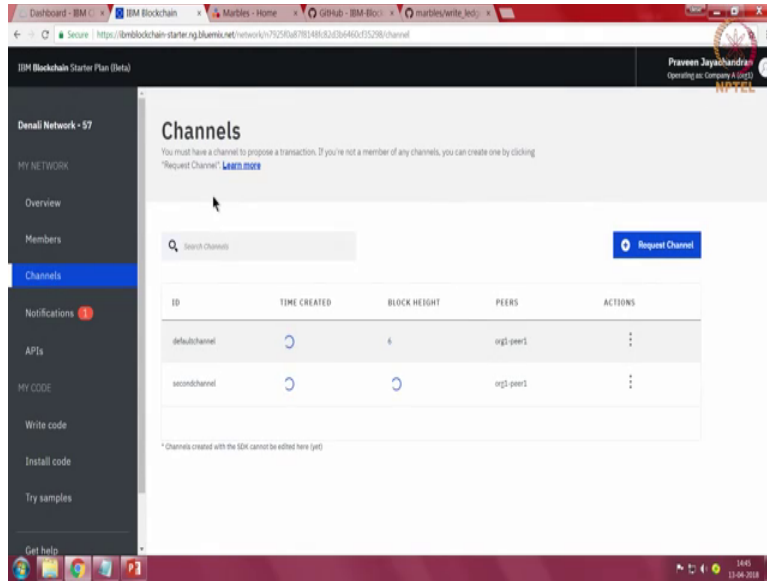
(Refer Slide Time: 22:24)



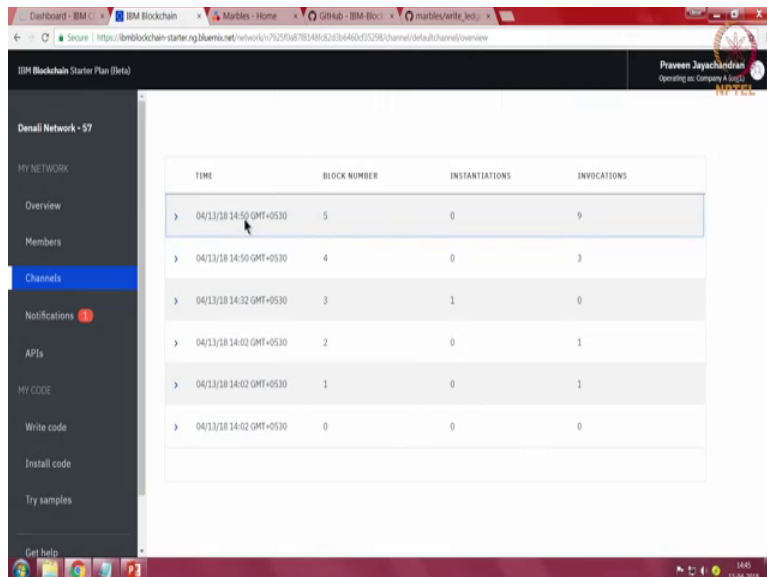
So, it is just telling you what are what are these things that have been done and finally, we are going to be creating some assets. We have three marble users Amy, Alice and Ava and each of them are going to be having three marbles. So, we have already performed some Invoke functions on blockchain to create these owners and create the marbles owned by each of them.

So, let us go ahead and create them. So, that calls this calling the Invoke functions in the background. So, at we will also look at what we can also parallelly look. Let us going to a channel and watch how the block height keeps increasing for the channels.

(Refer Slide Time: 23:04)



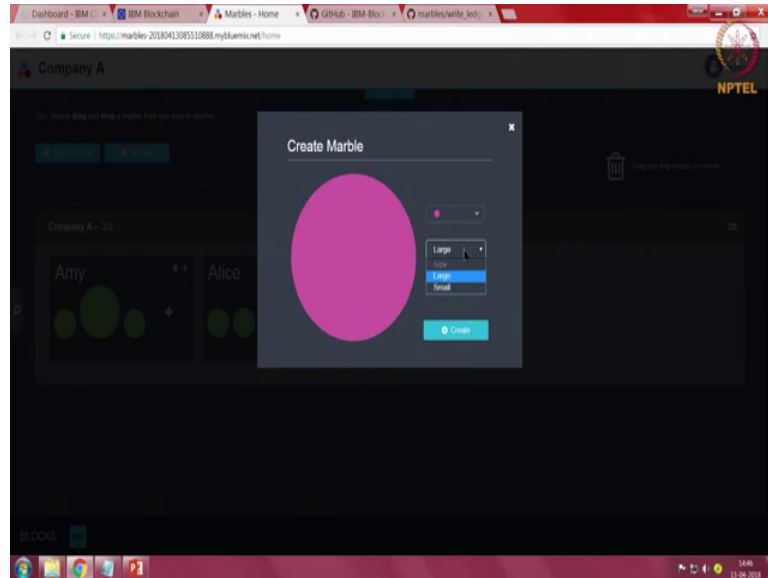
(Refer Slide Time: 23:10)



(Refer Slide Time: 23:12)

they are all have different things. Ava has three large marbles, Alices three small marble and so on. So, for each of these marbles we can easily in this UI go create new marbles.

(Refer Slide Time: 23:59)



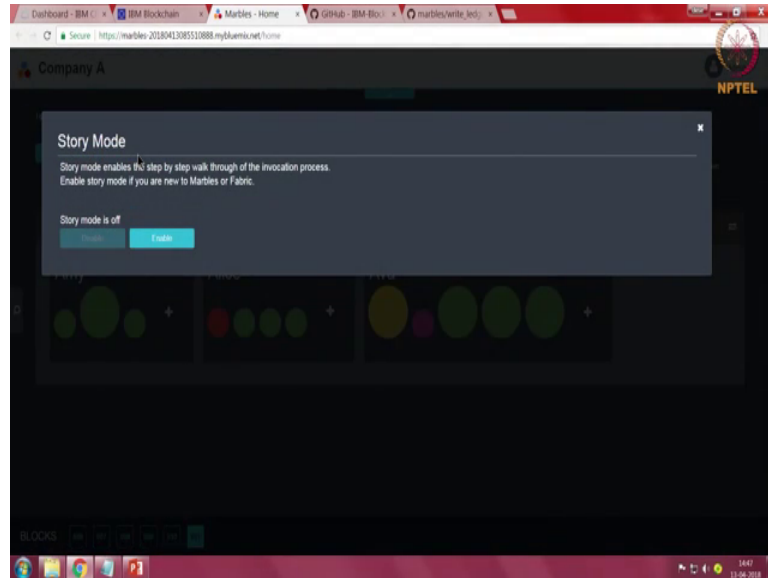
So, let us add a marble for Ava, we can choose a color, let us pick a color and we can say what size it is and let us go create it. So, when I clicked on create its actually going to call our marbles application which in turn calls blockchain to put that information on blockchain that Ava a now has a new marble purple marble of size small right. So, that information would be added to blockchain. If you go back and see our channels, the channel height would have also increased right. So, you should have added one more block here block head, block head 7 ok.

So, let us add another, marble just for the heck of it. A marble is been created to for Amy and here below, here also shows you the blocks. So, it shows you as 006 each block has been added sequentially. So, what we can do now; is we can easily pickup and drag and drop to somebody else in this is really shows a change of ownership, it signifies the change of ownership.

So, the red marble now belongs to Alice and a new block has been committed to the ledger, right. So, we can just do it once more let us say, this is the large marble and Amy gave that marble to Ava correct. So, some of those other things you can play around with later is you can for instance go try and shutdown a peer, bring it back up and it should

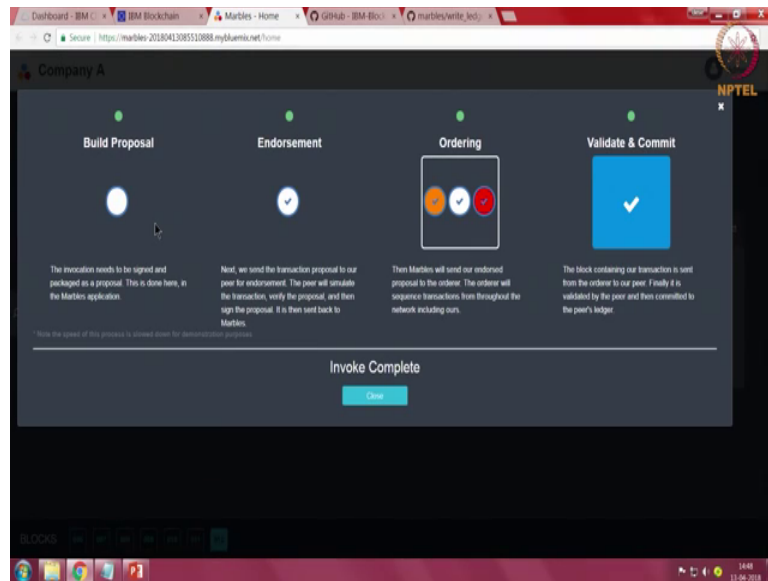
still show the same state of the ledger. Some of those things we can play around with um. So, that shows you some of the fault tolerance aspects.

(Refer Slide Time: 25:52)



Now going to show that here; let us the other aspect here is in the again let me just check back in the settings place, there is a something called story mode. You can actually, go enable it let us enable it. Actually, it is a pretty cool thing tells you, what is happening in the background the story mode is now on, ok. So let us see, let us go through this we are going to create a new marble this is now story, story mode is enabled let us choose green and let us choose small.

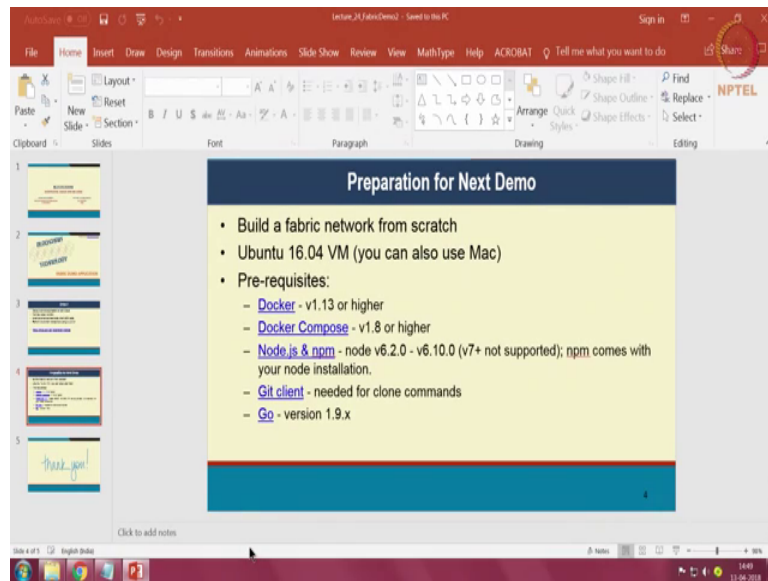
(Refer Slide Time: 26:22)



So, when I say create, it will actually tell you, what is going to happen in the back end. So, for instance the first part of it is you are going to build a proposal a transaction proposal that has this information about what is this new marble who is the new owner t, that information is created in the transaction. He has sent to the peers for endorsement. The peers will execute the smart contract the function that was Invoked. It will then say I agree that this is the output of the transaction and send that endorsement back right. So, that is the transaction response.

Once the client is collected enough endorsements, it can then submit to ordering. So, the ordering service might get such requests from multiple clients. We will order these transactions and I will say these transactions have to be now committed into a block and then each peer. Once they get the block from the order, they will validate these transactions make sure there is no double spending make sure there are no errors and then it will commit the transactions. So, that is the processes. So, it tells you what is really happening in the backend when you do any one of these transactions, ok.

(Refer Slide Time: 27:36)



So, that ends the demo, but before we close this lecture I would like to just prepare for the next demo. So, so far in the last two demos, we had it easy we had the IBM cloud do much of the work in the backend and we just work through what IBM cloud is doing in the backend.

So, in the next time we are going to do it ourselves, right. We are going to do everything from scratch, build a network ourselves. So, for that what I would encourage you to, to do is be prepared with the Ubuntu VM. So, that is got I am going to use in the demo by that you can also use a Mac it will work just as well in a Mac there are instructions out there, where I will show you the point us for it.

And there are few prerequisites, so please install those prerequisites, before you get to the next lecture. So, you need Docker for Docker compose Node js. So, Docker and Docker compose are how, each entry of your network the peers, the orders, the chain codes, they are all going to be deployed as Docker containers. You need Node.js and npm. So, for the some of the application development you need a Git client, and you need Go lank.

So, please install these prerequisites before your next lecture. And, next lecture we will do it all from scratch and you will get a feel for how it, what it looks like to run on network yourself, ok.

With that, thanks a lot. And I will see you soon at the next lecture.