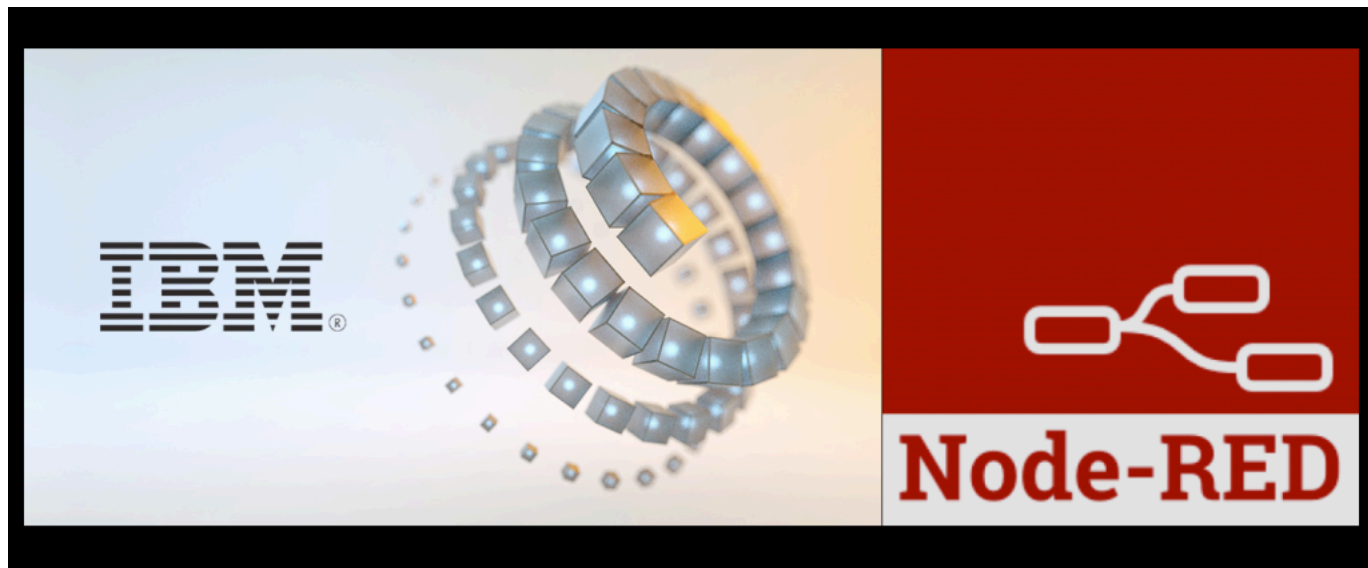Build your own block chain in 15 minutes on Node-RED using Node.js, JavaScript, Cloudant/CouchDB, IoT Platform/MQTT on a free IBM Cloud account…


Note: To do the tutorial you need a free Bluemix (IBM PaaS Cloud) account. You can obtain one http://nopanic.ai/ibmcloud and the raw file (JSON) for this NodeRED flow is here: https://raw.githubusercontent.com/romeokienzler/noderedchain/master/nodered_flow.json

Tutorial Objective

In this exercise, you will generate a small block-chain implementation on your own. All you need is a free IBM Cloud account and some very basic programming skills (JavaScript). After you've completed the tutorial you'll have a complete in-depth view how block-chains work.
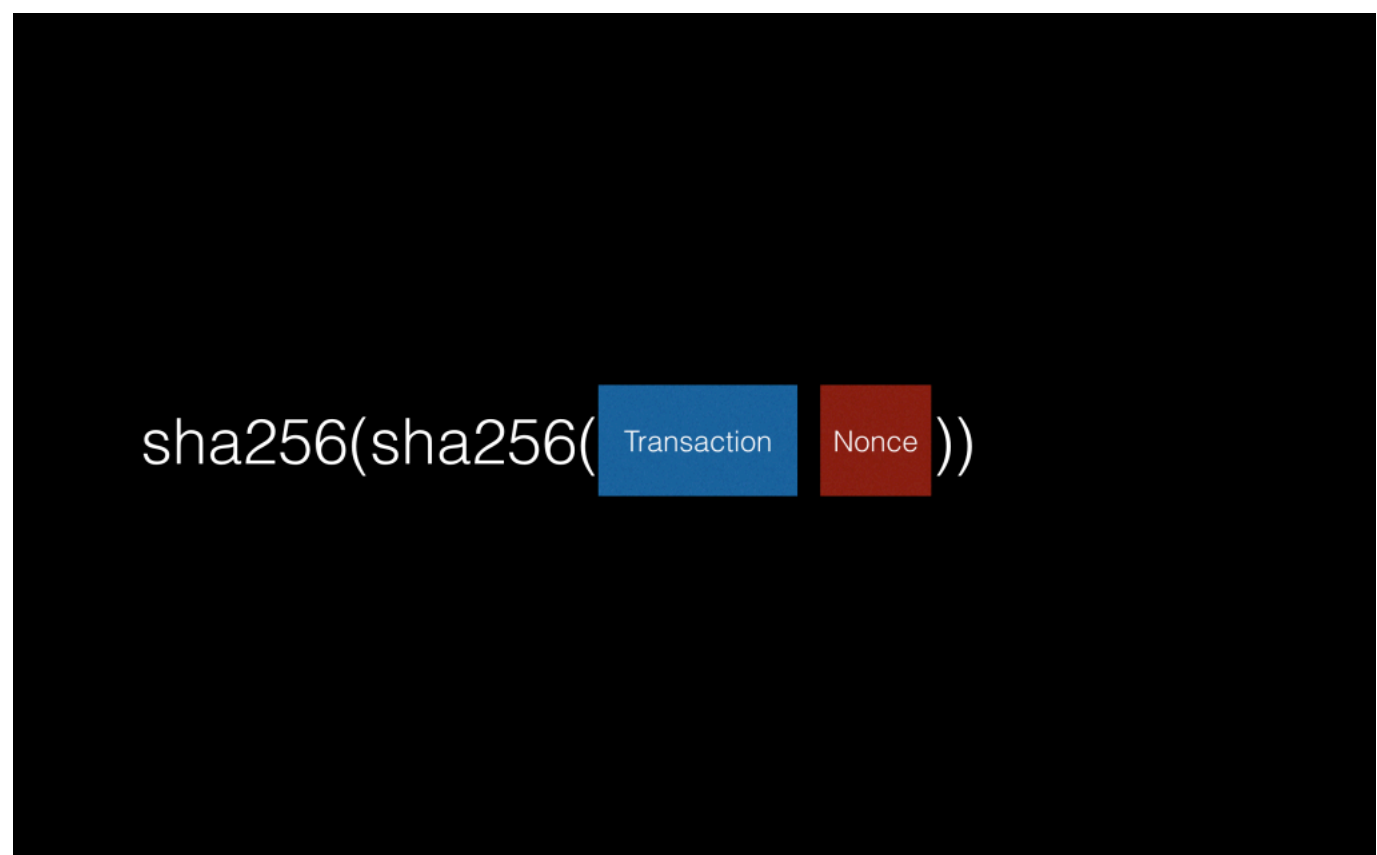
Refresher – How Bitcoin consensus works
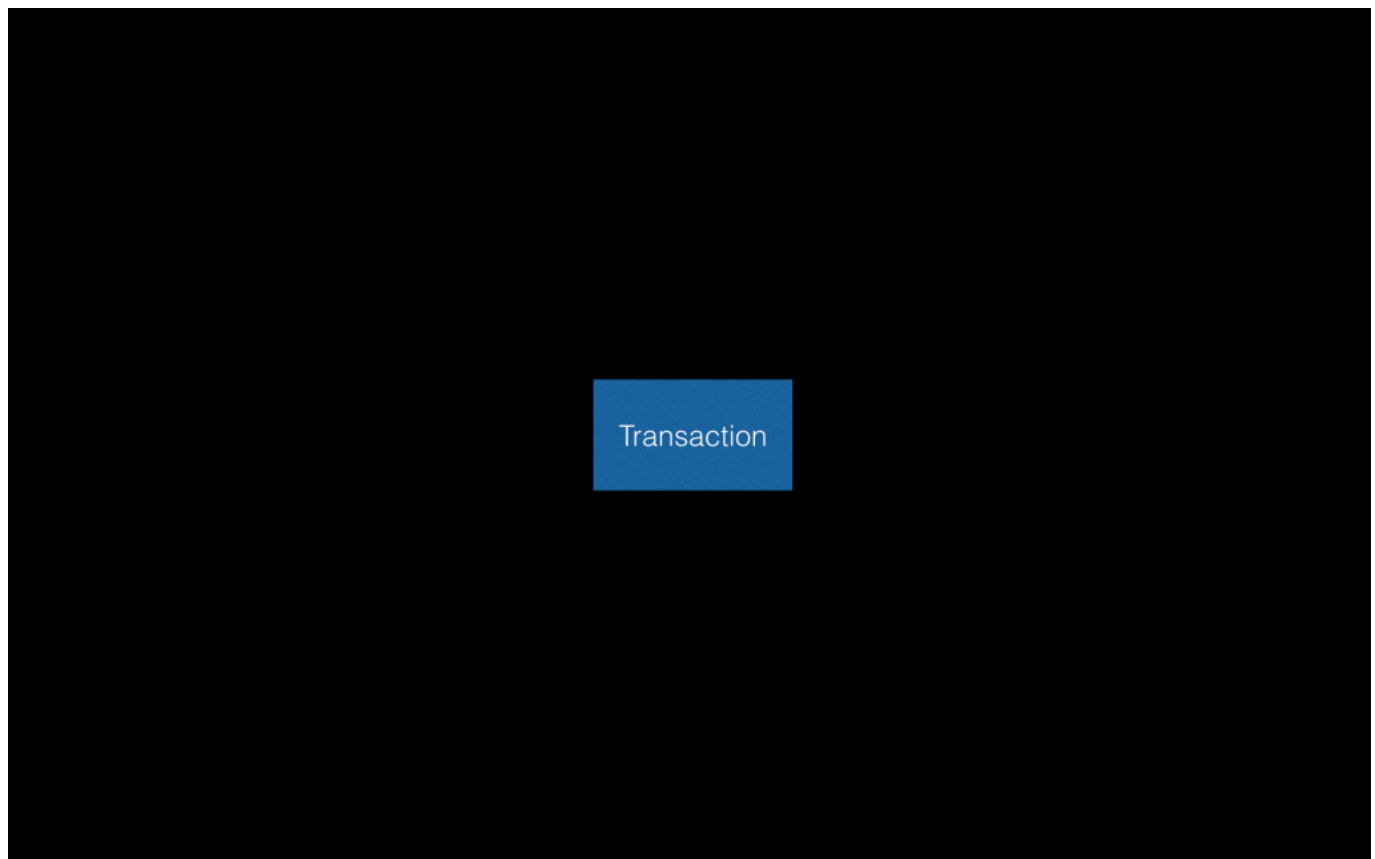
Let's consider a transaction (or any arbitrary string) you want to securely store on a block chain.
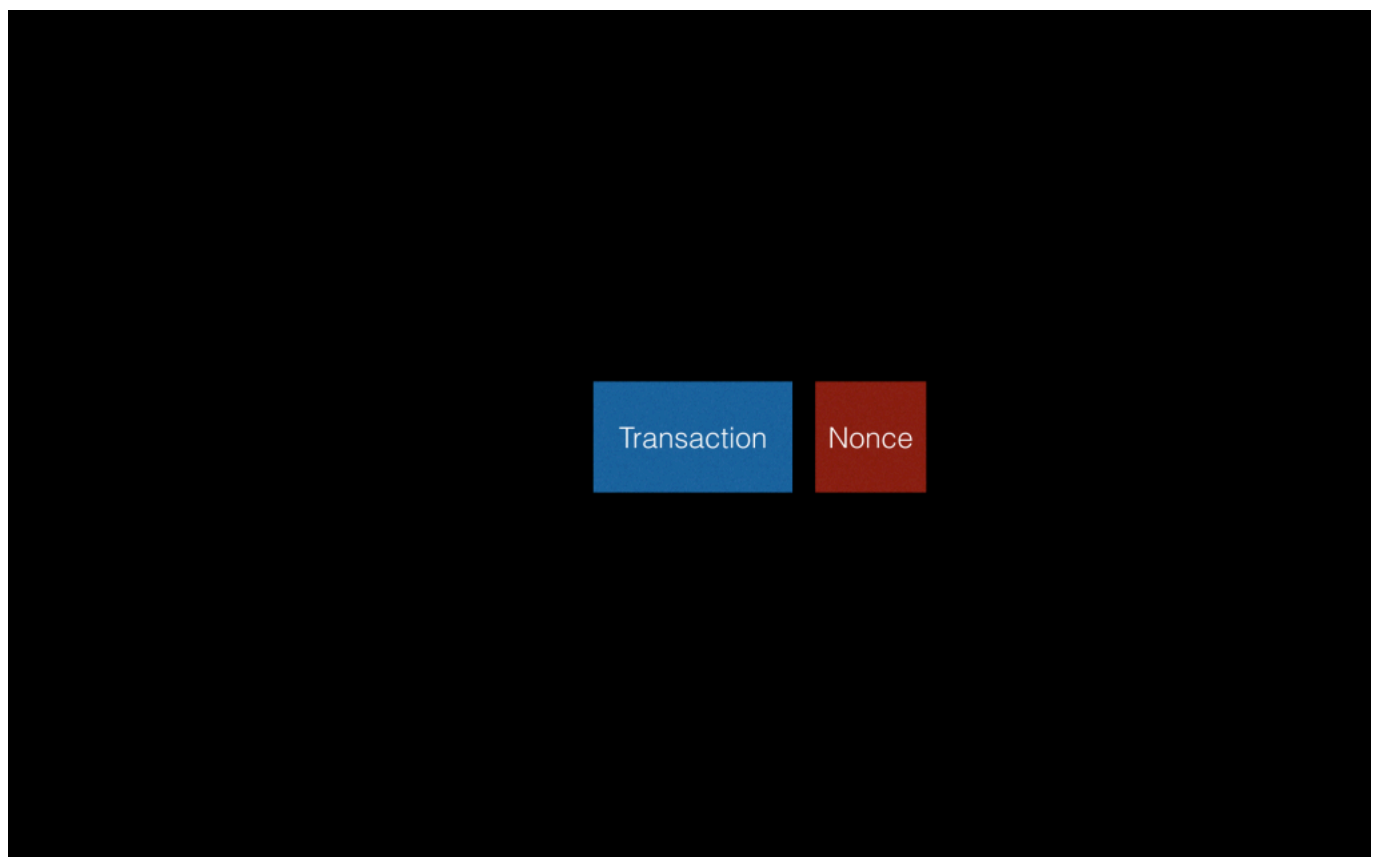
To do this (so that the block chain accepts your transaction) you have to find (compute) a cryptographic token – a nonce – matching your (and only your) transaction (string).

So, in Bitcoin this is done by applying the sha256 hash function twice and…

sha256(sha256(Transaction Nonce))=8375927
sha256(sha256(Transaction Nonce))=9284691
sha256(sha256(Transaction Nonce))=5830274
sha256(sha256(Transaction Nonce))=0002738
sha256(sha256(Transaction Nonce))=1785762
sha256(sha256(Transaction Nonce))=5876235
sha256(sha256(Transaction Nonce))=5839275

…try different nonce values until you find a hash value starting with a specific number of d zeros. In Bitcoin d is called difficulty and is varied to achieve a constant block rate of 10 minutes. In other words, the chance of finding a nonce that generates a hash value for your transaction with d leading zeros decreases with increasing d. Therefore, the computational power needed to find a valid bitcoin block also increases with d. Since the aggregated computational power of all bitcoin peers varies (but continuously increases) parameter d is automatically adjusted to obtain a constant block rate of one block every 10 minutes.

```
Hello, world#0 => fa2881e9b47b8e1535df08f1d6d47b71854aa0706c959a2726fcb964fc90ff15
Hello, world#1 => 795544e740045733b4713381f8e3e47bfff379e059aca1971b711b0ab8b54fb4
Hello, world#2 => d4d47d6600c4b5f6e2aa6936925741758714a5f8dd8d69eda0ccf3a0287a2c0e
Hello, world#3 => 32192c79cd80b64e57808745bbbaafc4aebab6bec25f5df5435a439323930833
...
...
...
Hello, world#69159 => 46201a335a85608d349fec758409160b40c612fdb51a6c91e65d3b4fddb2f06a
Hello, world#69160 => fecc199b038da2d577745fb05ea85227eb823b2489bb8070e2f42f38d60155f3
Hello, world#69161 => 35ff61c959bec6a6c66ff2cc602a84d02823c46e9ca2e70f03f6ea9c9212842b
Hello, world#69162 => 0000c5a9a24161e58868c858fc2700eeabf21a86862cbfa3bbd18a4d63e5b010
```

So, congratulations, you've understood the so called "proof-of-work" algorithm. Since finding a nonce is computationally intensive and validating if a transaction+nonce creates a hash code with d starting zeros can be done in milliseconds you have the asymetric cryptographic property that creating a transaction is complex but validating a transaction is easy.

It is interesting to know that the Bitcoin peer-to-peer network currently has a computing power of 19881622 Petaflops – in contrast the fastest supercomputer in the world only has 93014.6 Teraflops.

Let's create our own hashing algorithm in JavaScript

Since JavaScript out of the box doesn't provide access to the SHA256 hash function we'll implement one on our own as formally described as follows:

$$hash(s) = \left(\sum_{i=1}^{strlen(s)} code(s_i)\right) + hash(s_1..s_{strlen(s)-1})$$

Don't be scared, basically what this algorithm does is to take the ASCII value of each character in the string and sum it up. Afterward the same is done for the same string but with the last character removed – and so on – recursively – until there are no more characters left.

So, this can be easily implemented in JavaScript:

```javascript
String.prototype.hashCode = function() {
    if (this.length === 0) {
        return 0;
    } else {
        return parseInt(this.split('').map(function(char) {
            return char.charCodeAt(0);
        }).reduce(function(current, previous) {
            return previous + current;
        }))+100000;
    }
};
```

Let's turn this hash function into a proof-of-work algorithm

So, we will use this hash function to create a proof-of-work algorithm by just hashing our string together with a nonce and calculating a modulo operation. We change the nonce until we find one leading to zero. In JavaScript this will look something like this:

```javascript
nonce = 0;
while (true) {
    hash = (transaction.hashCode() + nonce) % 25000000;
    if (hash == 0) {
        break;
    }
    nonce++;
}
```

Now just run a little test and see whether it works:

NODEChain Transaction Creation and Submission tool

Transaction:	fertig! :)
Nonce:	24899195

Create Transaction Hash | Submit Transaction

Hashing finished after 139026ms

## Publishing transactions to other peers

Since most block-chains are peer-to-peer networks we will simulating this by creating another HTTPS endpoint accepting transactions and corresponding nonce values, and distributing them to all other peers using a MQTT message broker (in this case provided by the IBM Watson IoT Platform)

The "distributeTransaction" endpoint takes transaction and nonce as parameters and implements this.

Verifying and storing a transaction into Cloudant/CouchDB

We subscribe to published transactions and verify them:
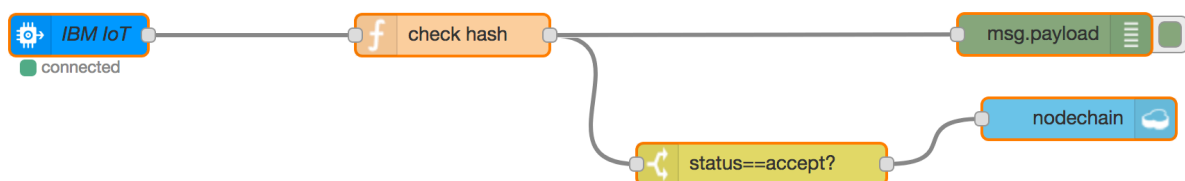
Using the following JavaScript function we'll check whether the transaction is valid or not and add this finding as status code to the "msg" JSON object flowing through our flow:

Edit function node

Delete   Cancel   Done

˅ **node properties**

🏷 Name   check hash   📄˅
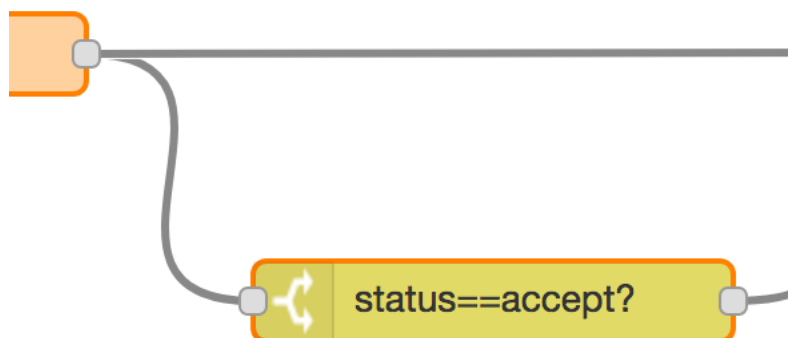
🔧 Function

```
 1▾ String.prototype.hashCode = function() {
 2▾     if (this.length === 0) {
 3          return 0;
 4▾     } else {
 5▾         return parseInt(this.split('').map(function(char) {
 6              return char.charCodeAt(0);
 7▾         }).reduce(function(current, previous) {
 8              return previous + current;
 9▴         }))+100000;
10▴     }
11▴ };
12
i 13  hash = (msg.payload.transaction.hashCode() + parseInt(msg.payload.nonce)) % 25000000
14▾ if (hash === 0) {
15      msg.payload.status="accept";
16▾ } else {
17      msg.payload.status="reject";
18      msg.payload.hash=hash;
19      msg.payload.hashcode=msg.payload.transaction.hashCode();
20▴ }
21  return msg;
```

Now we need to verify the transactions by checking the status code and in case it was accepted we are storing them to a data base.

We split the flow after this function:

On the other branch of the spit we use the switch function to decide whether we want to let the message pass through or not based on the status code:

If accepted, we just store it in Cloudant/CouchDB, here is what's necessary for doing that:

Add a user interface

We add a little user interface in NodeRED, which is straightforward, just add another http endpoint and use the template node to create a HTML single page application:

The template is a little implementing the following:

- A form to input your transaction
- A proof-of-work algorithm using the JavaScript engine of your browser

Pulling all together

Now we have all ingredients we need to implement a block chain consensus protocol. We can generate nonce values from transactions and we can submit, verify and store transactions to a data base. Most of block chains implement a sort of peer-to-peer protocol, but for sake of simplicity we'll omit this step and connect peers together using a MQTT message broker. So now it's time to convince a colleague or friend (or even more) to implement the very same process in his IBM Cloud account. Here are the steps you need to follow:

1. Create a IBM Cloud Account here: http://nopanic.ai/ibmcloud
2. Create a new NodeRED boilerplate in US-South, NOT in UK/DE since the MQTT message broker will run in another tenant:
   https://console.bluemix.net/catalog/starters/internet-of-things-platform-starter
3. After installation finishes open the URL

4. Finish NodeRED installation by setting a password
5. Create a new Flow

6. Import the following flow
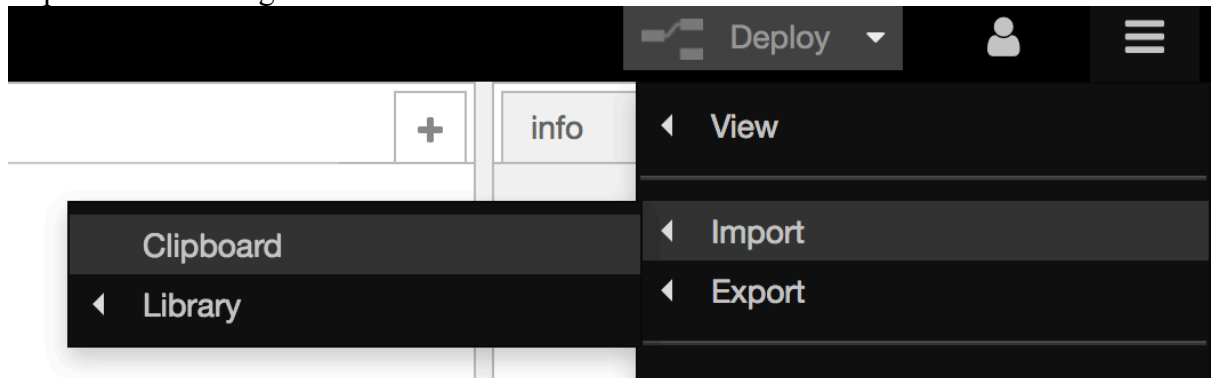
   https://raw.githubusercontent.com/romeokienzler/noderedchain/master/nodered_flow.json
7. Double-Click on the Cloudant-Node (which is labeled as „nodechain") and click on "Done" (this is a workaround)
8. Click on deploy

Now we are done and can use it:
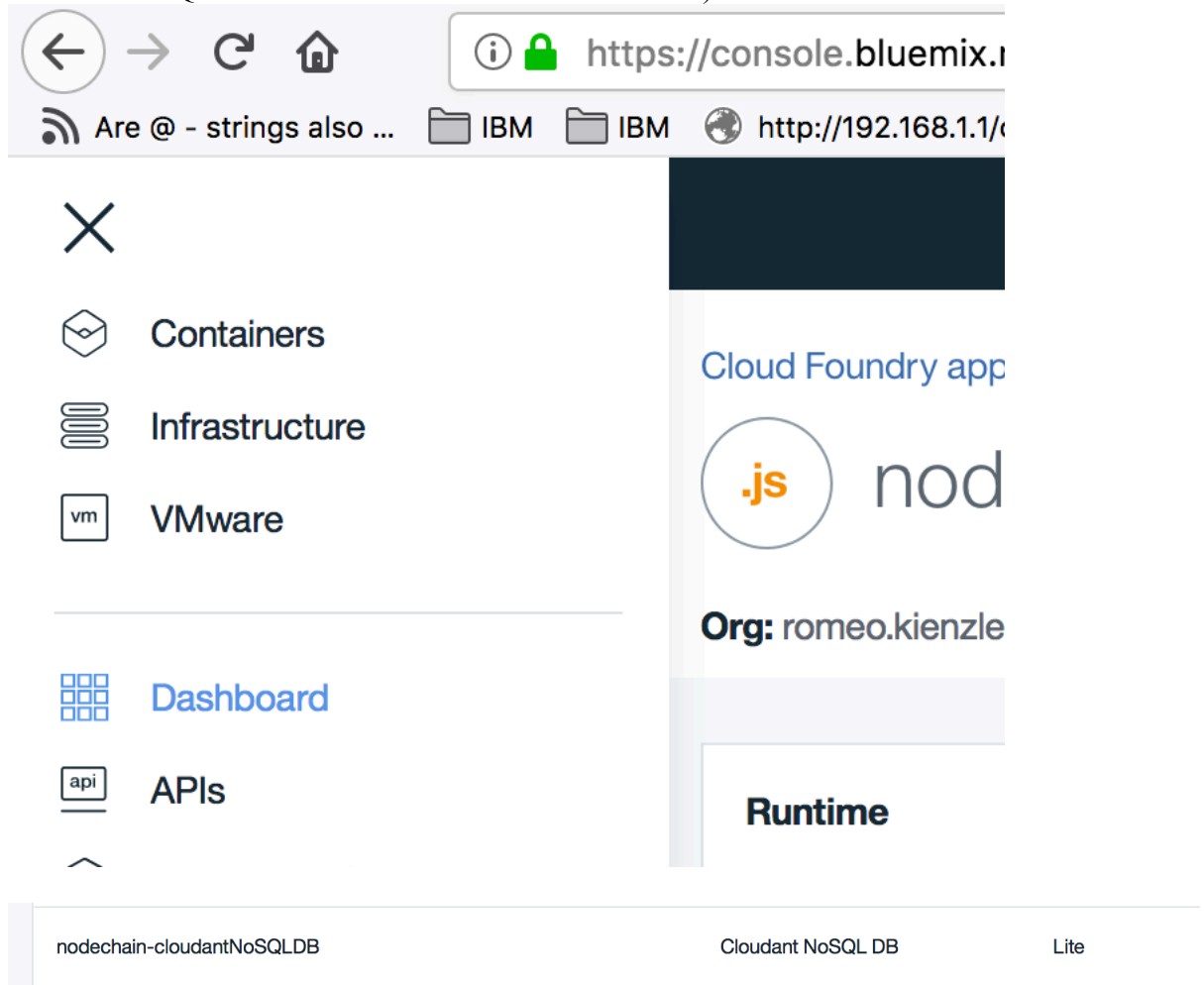
1. Create a transaction
   Let's assume the following is the URL of your application:
   https://noderedchain.mybluemix.net/red

   Please replace it with the following one:
   https://noderedchain.mybluemix.net/makeTransaction
2. Please put your name or any string into the transaction field
3. Click on „Create Transaction Hash", you are now creating a hash in your browser
   which should complete within 60-180 seconds
4. Once the Nonce is found, please click on "Submit Transaction"
5. Open your Cloudant UI – you should now see the transaction in your list and in the
   Cloudant databases of all your peers
   (To open the dashboard go to Bluemix.net, then Dashboard, then click on the
   Cloudant NSQL database instance and then on Launch)

Conclusion

So, as you can see implementing a block chain is very easy – at least for a learning and demo purpose. You should now have completely understood the concept of proof-of-work and distributed consensus.