Navigating the IBM-Blockchain dashboard

After we subscribe to their service, create a blockchain and launch the dashboard, We see Peer logs and chaincode logs below.

The Network Tab:

By default, the network tab will be shown.

The starter network ID is the unique identification given to this network of peers (nodes) and a membership service as well (lite node).

The refresh timer refreshes the logs after it expires. This can be manually changed to an interval anywhere between 0 to 6 minutes. So if blocks invoked are not yet shown on the logs, then calling refresh will update the logs.

In the peer logs menu, the list includes 4 validating peers, along with a membership service that takes care of issuing enrollment and transaction certificates. The routes indicate the protocols used to communicate with them, and currently has options of HTTPS and gRPC. URLs are also given, which can be used to contact them for the necessary operations with our app.

Discovery indicates the number of other peers visible to a particular peer in the network. For example, if Validating Peer 0 says 4 on discovery, then it means it has discovered 4 peers in the network.

The block height indicates the number of blocks in the chain. This gets increased by 1 every time we deploy or invoke an operation. Even if a deploy fails, a block is still added, along with the information of the reason for failure/authentication.

Actions tell whether to manually stop the peer or not.

Coming to the chaincode ID logs,

Each chaincode ID will be a 128 bit unique hashed string to identify each successfully deployed chaincode on the peer. If we are to execute a specific code in the peer, which is made to run on docker, then we identify that using this chaincode ID.

The logs with the 4 options of the peers, can be used to fetch the logs of this chaincode operations on a specific peer chosen. A sample log would be entering a function like getstate() or putstate() in the shim interface, printing all the printf statements in the code and exiting the same.

The Blockchain Tab:

Coming to the blockchain tab, We have a GUI of the logs, with overviews for the number of the blocks(including the genesis block that was created when we subscribed to the blockchain service and created a blockchain) and others like the number of successful deployments or invocations done etc.

Below would be a list of the blocks, with the number and the time it was created, and upon clicking a block we would also get its contents, such as chaincodeID, exact time of creation, UUID(the transaction ID that can be used to get detailed information of the transaction- all invocations and deployments here are modelled as transactions) and the payload as well.

The Demo Chaincode Tab:

The Demo chaincode tab, has 3 sample applications which can be deployed to get a better understanding of the way chaincode is written and how it works using hyperledger fabric. For starters it would be best to deploy the example02 application that subtracts an amount from one user and adds it to another user B, as this example is easy to understand. The marble and commercial paper application have a lot of UI in them, so that is very intuitive and includes animations of transferring assets as well. The Interact link would require the correct chaincodeID among the given set of IDs and a couple of built in actions as well, such as querying and transfer. The ‘show API details’ checkbox

gives detailed information of each communication made with the blockchain, along with the exact JSON sent and received.

The APIs Tab:

The APIs tab includes a wide variety of API endpoints to do various operations on the blockchain.

In the beginning you will be able to choose one among the four peers to interact with, and any deployments/invocations will be internally carried out among the other peers as well, so that each peer is always up to date with its other peers. Since we have the URLs of all the validating peers, we can perhaps employ a load balancer in our app that accordingly directs the requests to other peers when the load gets too high (for example, exceeds a threshold of 5 requests per second).

There is a set of Network Enroll IDs which include a pair of predefined of usernames and secret passwords which will be used to generate the enrolment certificated by the membership peer. This can be edited outside this dashboard, and we can edit the JSON, giving usernames and secret keys of our choice.

Below would be the set of HTTP APIs, with each endpoint offering a specific functionality. Each endpoint has a sample JSON message body that includes the format of the request and the response, along with a couple of lines about its functionality.

1)Block

This gives the block details when fed with the block number in the chain. For example if we give input as 25 and click on 'try it out', it will send a request(GET or POST) to the peer's URL + the specific end points, in this case /chain/blocks and will display the response which would be either the block information or error , on an unsuccessful request.

Among other APIs, the Registrar and the chaincode APIs are the most important.

The Registrar API is used to register a particular user to a peer so as to able to access the blockchain. This would require registering with the Membership Service node, which will validate the sent enroll ID and secret key and then generate a certificate with this user and send a successful response saying username logged in. If a user is already logged in, it will send the same. Note that this certificate is a onetime use only certificate and hence if it is lost when in transit between the membership node and the peer, or is deleted purposefully by the user (using the same Registrar API) then the same credentials cannot be used again and this user will not be able to carry out further operations on the blockchain.

After a user is logged on, he/she would have to deploy the code onto the blockchain network if they are logging in for the first time. For this we would require the chaincode API which has 3 types of function calls, all through the same end point, i.e. /chaincode.

The following fields are expected in the JSON body:

- The version of json rpc, which should be 2.0.
- The method to be invoked in the chaincode, which can be query, deploy or invoke.
- The parameters to be sent to this function , which can be further grouped to:
    - Type, which is the language in which the chaincode is written in, usually it will be 1, which indicates that it is written in GoLang.
    - The chaincodeID, which will be the hash generated after we deploy successfully the code. If we are deploying for the first time, then we will have to give the path to our GitHub code here.
    - The Constructor message, which further includes a set of fields to be sent:
        - The function to be called in the chaincode-note that the previously mentioned method is the function to be called in the shim interface according to hyperledger fabric, and this function here is the function to be called in the chaincode that you have written for your application(maybe written in java or go; it is called from the shim interface).

        - The command line arguments to be sent to this function; it is an array of strings, so you can send multiple arguments.
    - The secure context, which will be our enrollID , simply a layer of protection to check whether the user is allowed to carry out this operation.
- The id, which would be mentioned only if we want a response message. If we do not write this field, we will not receive any response. The id takes a non-negative value as argument.

What happens when we deploy our chaincode?

The code is first taken from our GitHub repository and then a Docker image is installed onto all the peers. This will take some time and that is why deployments are slightly slower. Added to that are the requirements of deploying the same code onto all the peers and interacting with the membership node for transaction certificate authentication (Although this will be issued during the registrar process, it will be verified here to allow this deployment or invocation to be modelled into a transaction, creating a block to the chain. If at all this certificate is lost, then we will not be able to create further blocks.)

Query is only for reading the blocks and will not create any blocks. Invoking would be to write information (as per your app's requirements) and will create a new block. Deployments are only for the first time.

Other Tabs:

The Logs tab would give you the logs of all the nodes in the network, and they would include all the information, including the path from which the code was deployed, and error messages of the chaincode and all the printf statements, the REST APIs used etc. This is in much detail when compared to the logs option in the main dashboard.

The service status would denote the uptime of the network along with other information, and the support tab contains useful links to learn and deploy chaincode.