Simulating Hyperledger Networks with Shadow

Final Report : Addition part to Phase 2

(Python Codes and Steps included at the end.)

Ahmet Turkmen

Veysel Karani Pehlivan

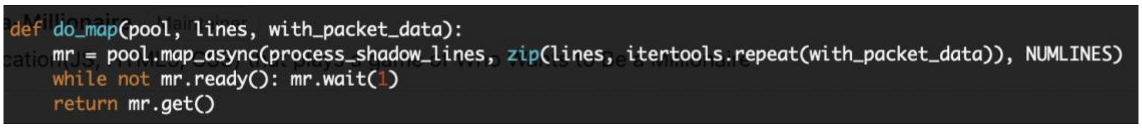
Explanation of Python Scripts

In this section, the python script will be shown and explained briefly. As described in Phase 1, after shadow run with a provided XML file, it creates log files from happened transactions between defined nodes. To extract some important performance metrics of created network, created logs should be parsed in a way that all transactions and their process time exported in structed way.

In parse-shadow: :

This script is basically creates threads with number of processes exist in system and then, it starts them from processing pool. It parses packet data, nodes received and send packet data, bytes total, bytes control header, packets in total for each node. Example python snippets are given below;

Following snippet creates pool which contains processes to run in parallel.

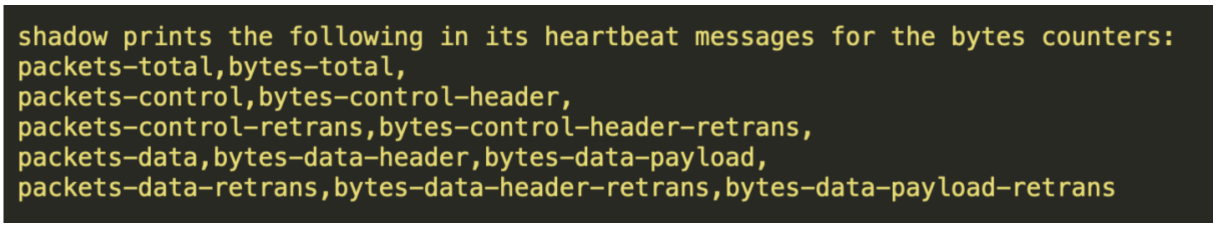


```python
def do_map(pool, lines, with_packet_data):
    mr = pool.map_async(process_shadow_lines, zip(lines, itertools.repeat(with_packet_data)), NUMLINES)
    while not mr.ready(): mr.wait(1)
    return mr.get()
```

Snippet 1 Pooling the processes

Since it is in parallel process, pool mapping them in asynchronous way as shown figure above.

Another example from python script can be given from process_shadow_lines function which basically calculates following information for each node with provided snippets from the script.

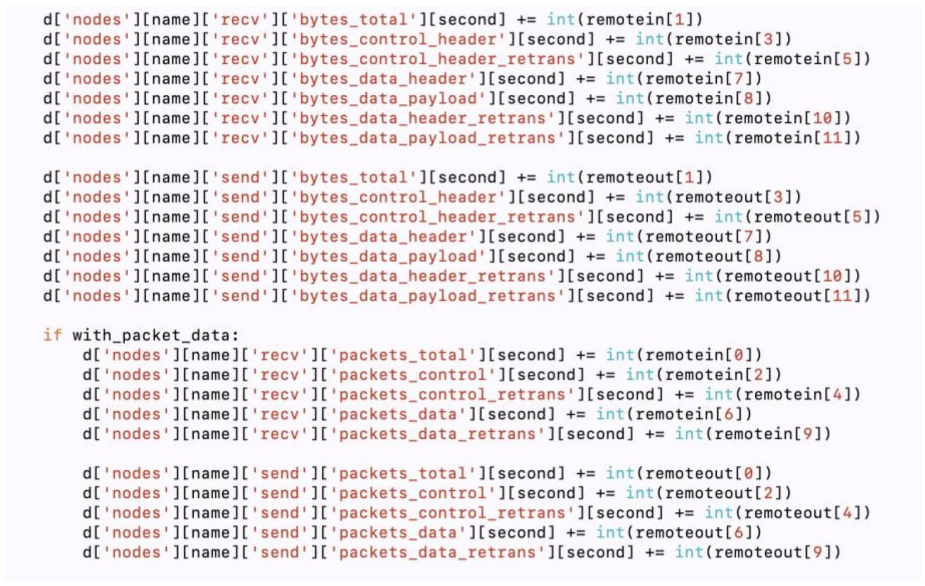


```
shadow prints the following in its heartbeat messages for the bytes counters:
packets-total,bytes-total,
packets-control,bytes-control-header,
packets-control-retrans,bytes-control-header-retrans,
packets-data,bytes-data-header,bytes-data-payload,
packets-data-retrans,bytes-data-header-retrans,bytes-data-payload-retrans
```

Snippet 2 Information which calculated to graph performance metrics

The information represented above is calculated as following in the script.

There are also different function in the python script however, these given parts of it is more crucial than other functions in the script. In following snippet shows that how the information mentioned Snippet 2 calculated, variable d is a dictionary in the script.

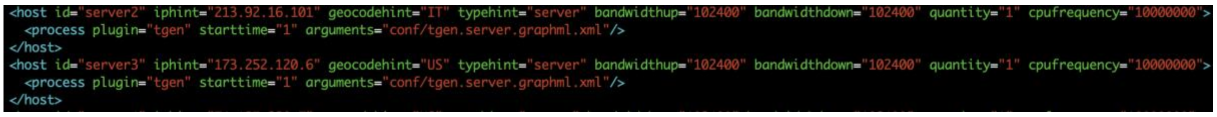


```python
d['nodes'][name]['recv']['bytes_total'][second] += int(remotein[1])
d['nodes'][name]['recv']['bytes_control_header'][second] += int(remotein[3])
d['nodes'][name]['recv']['bytes_control_header_retrans'][second] += int(remotein[5])
d['nodes'][name]['recv']['bytes_data_header'][second] += int(remotein[7])
d['nodes'][name]['recv']['bytes_data_payload'][second] += int(remotein[8])
d['nodes'][name]['recv']['bytes_data_header_retrans'][second] += int(remotein[10])
d['nodes'][name]['recv']['bytes_data_payload_retrans'][second] += int(remotein[11])

d['nodes'][name]['send']['bytes_total'][second] += int(remoteout[1])
d['nodes'][name]['send']['bytes_control_header'][second] += int(remoteout[3])
d['nodes'][name]['send']['bytes_control_header_retrans'][second] += int(remoteout[5])
d['nodes'][name]['send']['bytes_data_header'][second] += int(remoteout[7])
d['nodes'][name]['send']['bytes_data_payload'][second] += int(remoteout[8])
d['nodes'][name]['send']['bytes_data_header_retrans'][second] += int(remoteout[10])
d['nodes'][name]['send']['bytes_data_payload_retrans'][second] += int(remoteout[11])

if with_packet_data:
    d['nodes'][name]['recv']['packets_total'][second] += int(remotein[0])
    d['nodes'][name]['recv']['packets_control'][second] += int(remotein[2])
    d['nodes'][name]['recv']['packets_control_retrans'][second] += int(remotein[4])
    d['nodes'][name]['recv']['packets_data'][second] += int(remotein[6])
    d['nodes'][name]['recv']['packets_data_retrans'][second] += int(remotein[9])

    d['nodes'][name]['send']['packets_total'][second] += int(remoteout[0])
    d['nodes'][name]['send']['packets_control'][second] += int(remoteout[2])
    d['nodes'][name]['send']['packets_control_retrans'][second] += int(remoteout[4])
    d['nodes'][name]['send']['packets_data'][second] += int(remoteout[6])
    d['nodes'][name]['send']['packets_data_retrans'][second] += int(remoteout[9])
```

Snippet 3 How the crucial information is calculated in the script

In parse tgen script:

Another script which is used in phase 1 is that parse tgen script, before explanation of some parts of this script, to give information about t gen might be more appropriate. tGen is a tor network generator which creates tor network and make connections between provided nodes. The nodes are provided in an XML file, defining of nodes can be observed from given snippet below.



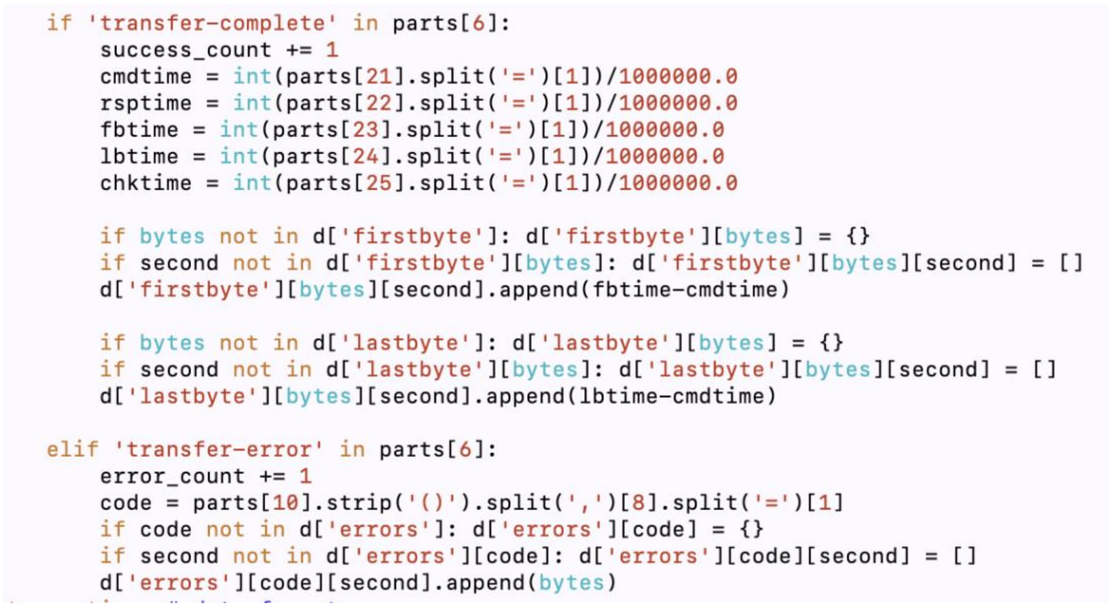
```xml
<host id="server2" iphint="213.92.16.101" geocodehint="IT" typehint="server" bandwidthup="102400" bandwidthdown="102400" quantity="1" cpufrequency="10000000">
  <process plugin="tgen" starttime="1" arguments="conf/tgen.server.graphml.xml"/>
</host>
<host id="server3" iphint="173.252.120.6" geocodehint="US" typehint="server" bandwidthup="102400" bandwidthdown="102400" quantity="1" cpufrequency="10000000">
  <process plugin="tgen" starttime="1" arguments="conf/tgen.server.graphml.xml"/>
</host>
```

Snippet 4 Defining hosts for Tor network

The information about hosts should be given in a xml file and that xml file is forwarded as argument of tgen which handles to create Tor network based on given specification of hosts in XML file.

The script (parse tgen) parses logs which are generated tgen in a json format for plotting performance metrics under different conditions. The python script (parse tgen) also runs in parallel which number of processes exist in the system, which makes it very fast. It emphasis on firstbyte, lastbyte and errors in the generated network by parsing logs which are generated by tgen process.

In process tgen log function of the script, number of errors, number of completed transfers are calculated by given snippet from the script.



```python
if 'transfer-complete' in parts[6]:
    success_count += 1
    cmdtime = int(parts[21].split('=')[1])/1000000.0
    rsptime = int(parts[22].split('=')[1])/1000000.0
    fbtime = int(parts[23].split('=')[1])/1000000.0
    lbtime = int(parts[24].split('=')[1])/1000000.0
    chktime = int(parts[25].split('=')[1])/1000000.0

    if bytes not in d['firstbyte']: d['firstbyte'][bytes] = {}
    if second not in d['firstbyte'][bytes]: d['firstbyte'][bytes][second] = []
    d['firstbyte'][bytes][second].append(fbtime-cmdtime)

    if bytes not in d['lastbyte']: d['lastbyte'][bytes] = {}
    if second not in d['lastbyte'][bytes]: d['lastbyte'][bytes][second] = []
    d['lastbyte'][bytes][second].append(lbtime-cmdtime)

elif 'transfer-error' in parts[6]:
    error_count += 1
    code = parts[10].strip('()').split(',')[8].split('=')[1]
    if code not in d['errors']: d['errors'][code] = {}
    if second not in d['errors'][code]: d['errors'][code][second] = []
    d['errors'][code][second].append(bytes)
```

Snippet 5 Calculates completed transfers and errors in parse tgen python script

The dictionary defined as d in given snippet above is defined as following :

d = {'firstbyte':{}, 'lastbyte':{}, 'errors':{}}.

While the shown calculation in Snippet 5 is running, it keeps dictionary d updated by appending.

In summary, both scripts parse the log files which are generated by shadow to compare and plot performance metrics of simulated environment under shadow. Both of them runs in parallel by adding threads to pool which is specialized library in python to run processes in an asynchronous way.

Both python scripts mentioned above is accessible from following link given below :
https://www.dropbox.com/sh/e88tyqnt77fpsju/AADxVh07cmOr3vCJFgBpm9-pa?dl=0

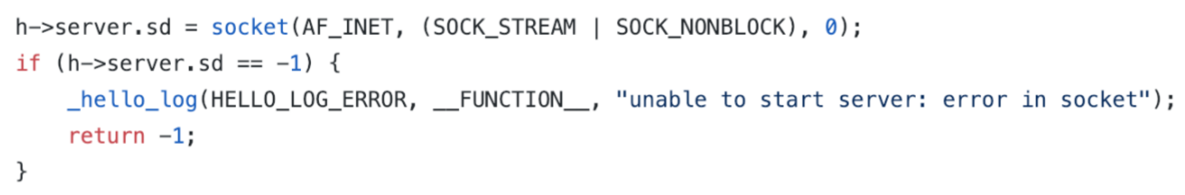
Obstacles and Resolves

While we were working on investigating example plugins for shadow network simulation tool, we noticed that the required knowledge on C and C++ is proficient which we do not have on C or C++. Hence, we started to investigate existing written plugins in C, however, along the way, we faced hundreds of errors while deploying them in Debian server which runs on google cloud services. The shared libraries of C programming language were not preinstalled in Debian and most them which are used in shadow plugins deprecated or the name of installation package is changed. Hence, we consumed a lot of time for configuring libraries and environment of Debian server to run them. Although we successfully configured the Debian server, there was no clear explanation of written plugins in C for this reason we stacked at this point. Main reasons of being stacked at this point can be summarized as follows:

- There is no clear explanation or documentation of written plugins and used methods while writing them.
- Since it is an open source project, there is only small group of people is contributing however they contributed in their internship period.
- Almost three months passed however there is no any development in project by Linux foundation and other contributors.
- Project requires high level of knowledge on C and C++
- Lack of support from other contributors.
- There are tons of bugs and problems on existing plugins such as memory leaks.

- Since it is a low-level programming, it is very hard to manage memory for different versions of Linux distribution.

To understand difficulty level of writing plugins in C, a snippet of an existing plugin is given below.



```
h->server.sd = socket(AF_INET, (SOCK_STREAM | SOCK_NONBLOCK), 0);
if (h->server.sd == -1) {
    _hello_log(HELLO_LOG_ERROR, __FUNCTION__, "unable to start server: error in socket");
    return -1;
}
```

Snippet 6 Piece of code from HelloWorld plugin of Shadow

The code given in Snippet 6 is taken from hello-world plugin of shadow and just an initial piece of code. The total number of lines in hello world plugin of shadow is nearly 500 lines which makes it difficult to understand and implement.

During implementation of Iroha network on Debian server, we successfully generated Iroha framework by creating accounts, transactions under Docker. When we looked at open issues on implementing such a system inside Docker containers, we noticed that someone reported an issue which states that there is a bug on the code which is given at getting starting tutorial page of iroha. Since we successfully run Iroha under Docker and no error is taken by system, we closed that issue by commenting on the statement. Issue is accessible from this link : https://jira.hyperledger.org/browse/IR-133 which is solved by us.

In summary, we consumed a lot of time on configuring systems and fixing bugs which are exists in open source projects which are not supported greatly.

In following sections, implementation of Iroha framework on Debian is explained and implemented using Docker technology. It is required to implement and understand Iroha working principles.

Introduction to Hyperledger Iroha

Hyperledger Iroha is a blockchain platform and library that can be used to build trusted, secure and fast applications. The most important feature of Hyperledger Iroha is that it is a permission based blockchain. For consensus, it uses Byzantine fault tolerant consensus algorithm. Like every blockchain platform, Iroha is free and open source.

Key Features of Iroha

- “*Simple deployment and maintenance*
- *Variety of libraries for developers*
- *Role-based access control*
- *Modular design, driven by command–query separation principle*
- *Assets and identity management*”

Iroha vs. Bitcoin or Ethereum

Bitcoin and Ethereum are permission less ledgers which means to join and access data you do not need permission. They also have native cryptocurrencies. For example, you need to have Bitcoin to interact with Bitcoin network.

Iroha does not have a native cryptocurrency because it was not created for this. It is created for enterprises. Enterprises need permission based blockchain. This is because they do not want to share all information. And, only authorized people can make some changes in the permission-based system.

One important difference between Iroha and Ethereum is that there are prebuilt commands for common actions in the Iroha. This makes developer’s job easier and faster. They do not

need to write smart contracts all the time. On the other hand, developers need to write smart contracts all the time in Ethereum.

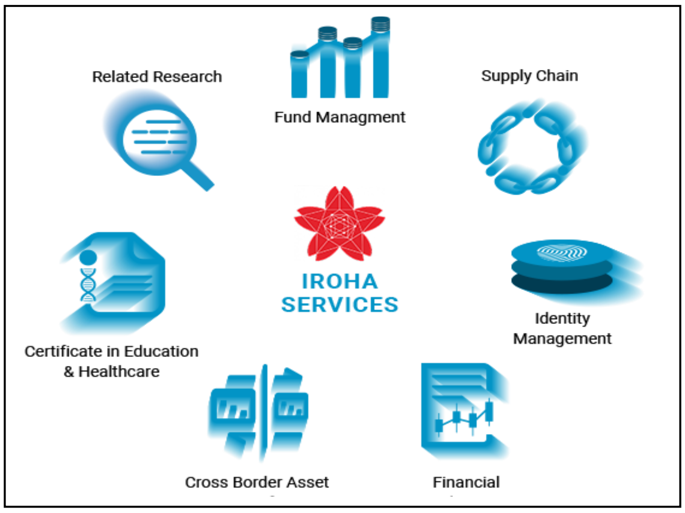
Iroha vs. Other Hyperledger Frameworks

Iroha and other Hyperledger frameworks such as Fabric and Sawtooth differ in terms of consensus algorithms. Iroha uses Byzantine fault tolerant consensus algorithm. Thanks to BFT, transactions are added to ledger with low latency. On the other hand, other

Hyperledger frameworks use probabilistic consensus algorithms such as Nakamoto consensus.

As I mentioned before, Iroha has prebuilt commands. These commands make Iroha better compared to Ethereum and other Hyperledger frameworks. Common tasks like creating digital assets, registering accounts and transferring assets between accounts are easier in Iroha. Also, Iroha is more secure than others.

Use Cases of Iroha



Iroha can be used in different areas such as Certificates in Education, Cross-Border Asset Transfers, Financial Applications and Supply Chain. In this part, I will explain Supply Chain part. Blockchain is almost perfect way for supply chain management. This is because blockchain is decentralized and includes untrusted nodes. For enterprises, they need permission based blockchain for supply chain. This is because there are different roles, and everyone should not join the system. Some big companies like IBM and Walmart uses blockchain for supply chain management. For small companies, Hyperledger Iroha can be solution because it is free to use.

For example, there is food supply chain that includes different actors like farmers, grocery stores and customers. In this system, each tomato is associated with Iroha item. Thanks to blockchain, asset creation and distribution are open for nodes in the network. In this way, people can learn history of the tomato.

Conclusion

How was the project overall?

Overall, project was seminal. Before working on this project, we had no idea about Hyperledger frameworks such as Fabric and Iroha. Thanks to use case scenarios of Iroha, we applied for TUSIAD competition with a blockchain project.

In this project, we are in an international team. Although we both worked on an international project before, working in an international group is still exciting for us. Moreover, the project

is a Linux Foundation project and open source. That makes this project more exciting. We like the idea of open source. And, playing a role in an open source project is an honor for us.

Is international collaboration aid to develop plugins?

Like all projects, this project has some problems too. Since this project is open source, developing process is so slow. Developers work these kinds of project for free, therefore developing process is slow. However, the project is our class project. We must give something but other people in group do not. For these reasons, we could not develop a plugin. We have a mentor in this project, but he did not help about where we should start to

work for developing a plugin. So, we saw that all Linux Foundation projects are not fancy like what we think.

What level of degree is required?

For this project, we must be proficient in Python and at least one of C/C++/Rust according to the project description. However, the problem is that there was no task about Python so far. For plugin, only C can be used. That is a big problem for us. This is because we have some

knowledge, but we are not proficient in C. I think the problem is not about our knowledge. The problem is guidance in this project.

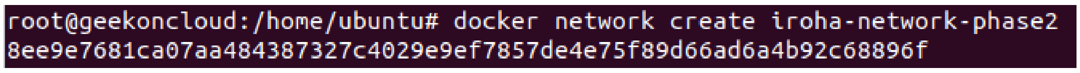
All requested steps and work is done by creating transactions over iroha framework on terminal and managing them using Python script is added to following section of this document.

Final Report *(Addition for Final Report) *(The Python Script can be accessible : https://gist.github.com/ahmetturkmen/01c2423c988f9e58455fd3c7395d0e55)

Creating an Iroha Network on Debian

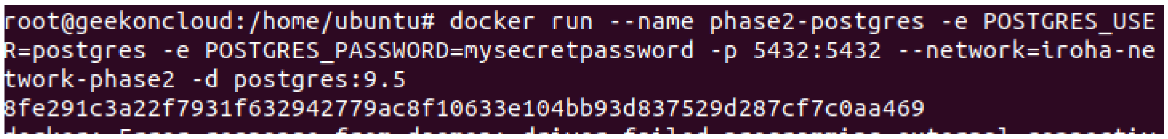
For creating an Iroha network, you need to have Docker installed in your machine.

Firstly, you need to type command below



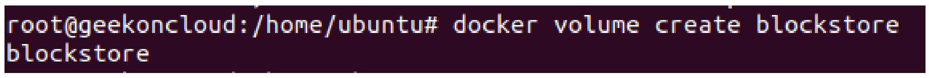
```
root@geekoncloud:/home/ubuntu# docker network create iroha-network-phase2
8ee9e7681ca07aa484387327c4029e9ef7857de4e75f89d66ad6a4b92c68896f
```

To run Iroha, there should be PostgreSQL database. Containers of Iroha and Postgres will run on iroha-network-phase2. So now, we need to run PostgreSQL in container Iroha. PostgreSQL should be attached to iroha-network-phase2.



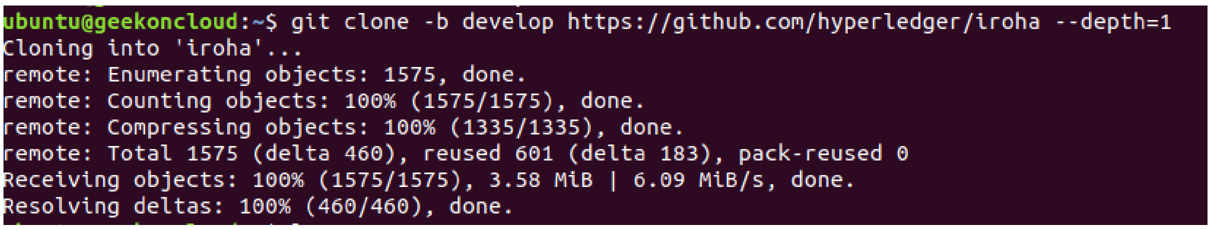
```
root@geekoncloud:/home/ubuntu# docker run --name phase2-postgres -e POSTGRES_USE
R=postgres -e POSTGRES_PASSWORD=mysecretpassword -p 5432:5432 --network=iroha-ne
twork-phase2 -d postgres:9.5
8fe291c3a22f7931f632942779ac8f10633e104bb93d837529d287cf7c0aa469
```

Before running Iroha, there should be determined volume to store blocks of the ledger. By using following command, it will be done.



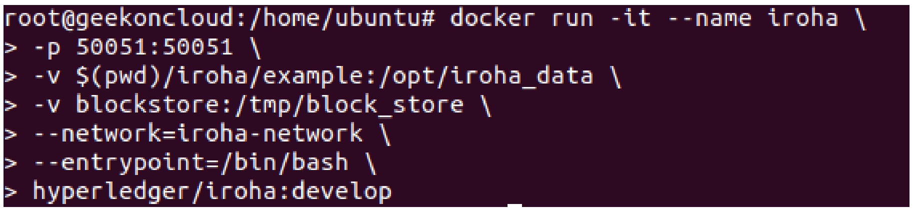
```
root@geekoncloud:/home/ubuntu# docker volume create blockstore
blockstore
```

Then, we need to configure our system by cloning Hyperledger Iroha repository. This repository includes a configuration file, keypairs for users, list of peers and genesis block.



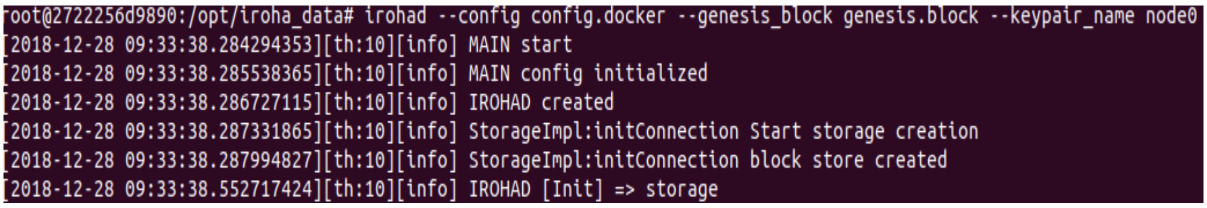
```
ubuntu@geekoncloud:~$ git clone -b develop https://github.com/hyperledger/iroha --depth=1
Cloning into 'iroha'...
remote: Enumerating objects: 1575, done.
remote: Counting objects: 100% (1575/1575), done.
remote: Compressing objects: 100% (1335/1335), done.
remote: Total 1575 (delta 460), reused 601 (delta 183), pack-reused 0
Receiving objects: 100% (1575/1575), 3.58 MiB | 6.09 MiB/s, done.
Resolving deltas: 100% (460/460), done.
```

Now, we are ready to run our container by following command. By using this command, you will enter interactive shell of container of Iroha.



```
root@geekoncloud:/home/ubuntu# docker run -it --name iroha \
> -p 50051:50051 \
> -v $(pwd)/iroha/example:/opt/iroha_data \
> -v blockstore:/tmp/block_store \
> --network=iroha-network \
> --entrypoint=/bin/bash \
> hyperledger/iroha:develop
```

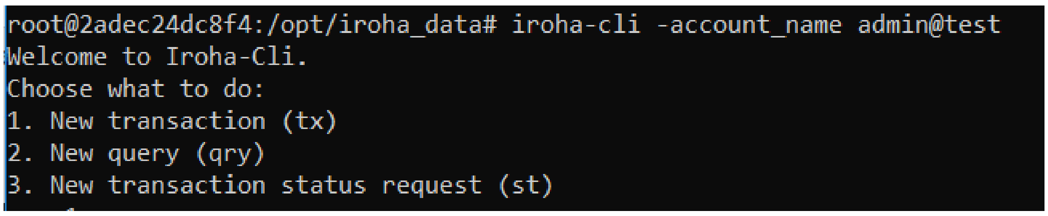
Finally, we will run Iroha by starting Iroha daemon. By using command below, we have a running Iroha node.



```
root@2722256d9890:/opt/iroha_data# irohad --config config.docker --genesis_block genesis.block --keypair_name node0
[2018-12-28 09:33:38.284294353][th:10][info] MAIN start
[2018-12-28 09:33:38.285538365][th:10][info] MAIN config initialized
[2018-12-28 09:33:38.286727115][th:10][info] IROHAD created
[2018-12-28 09:33:38.287331865][th:10][info] StorageImpl:initConnection Start storage creation
[2018-12-28 09:33:38.287994827][th:10][info] StorageImpl:initConnection block store created
[2018-12-28 09:33:38.552717424][th:10][info] IROHAD [Init] => storage
```

We can use Iroha with different ways. We can use Iroha by writing code in programming languages like Java, JavaScript and Python. The other way is using iroha-cli which is a command line tool for interacting with Iroha. By using this tool, we can simply test Iroha by sending some transactions.

For testing, we need to open a new terminal while your Iroha daemon is running. In our terminal, we should type "*docker exec -it iroha /bin/bash".* Then, we should run iroha-cli by typing "*iroha-cli -account_name admin@test*". We will see menu like image below.

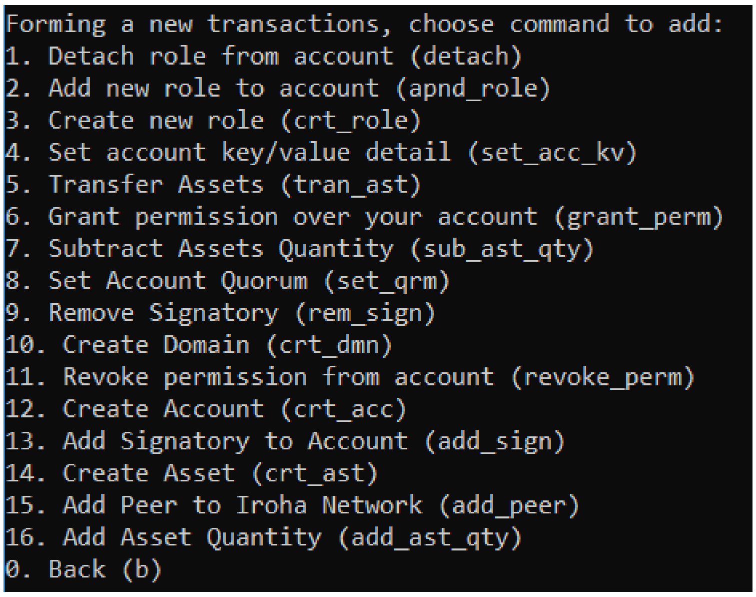


```
root@2adec24dc8f4:/opt/iroha_data# iroha-cli -account_name admin@test
Welcome to Iroha-Cli.
Choose what to do:
1. New transaction (tx)
2. New query (qry)
3. New transaction status request (st)
```

Now, we can make some transactions with Iroha.

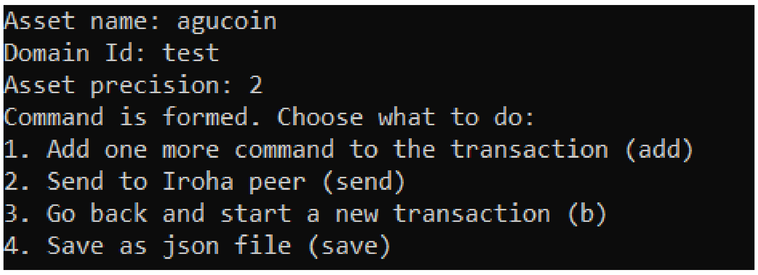
Creating a Transaction

For creating a transaction, there are several steps you need to follow. Firstly, choose “1. New transaction (tx)” option in Iroha-cli interface. We will see the output below.



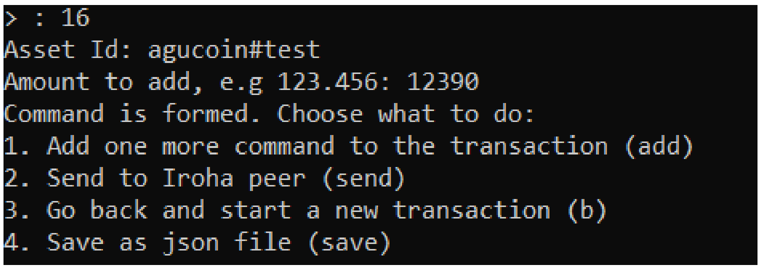
```
Forming a new transactions, choose command to add:
1. Detach role from account (detach)
2. Add new role to account (apnd_role)
3. Create new role (crt_role)
4. Set account key/value detail (set_acc_kv)
5. Transfer Assets (tran_ast)
6. Grant permission over your account (grant_perm)
7. Subtract Assets Quantity (sub_ast_qty)
8. Set Account Quorum (set_qrm)
9. Remove Signatory (rem_sign)
10. Create Domain (crt_dmn)
11. Revoke permission from account (revoke_perm)
12. Create Account (crt_acc)
13. Add Signatory to Account (add_sign)
14. Create Asset (crt_ast)
15. Add Peer to Iroha Network (add_peer)
16. Add Asset Quantity (add_ast_qty)
0. Back (b)
```

Before creating a transaction, we need to create an asset. For this select “14. Create Asset (crt_ast)”. In our example, our asset name is agucoin. We already have a domain name. As we remember, we used “iroha-cli -account_name admin@test” command. In this command, we give “test” for domain name. For precision we give 2.

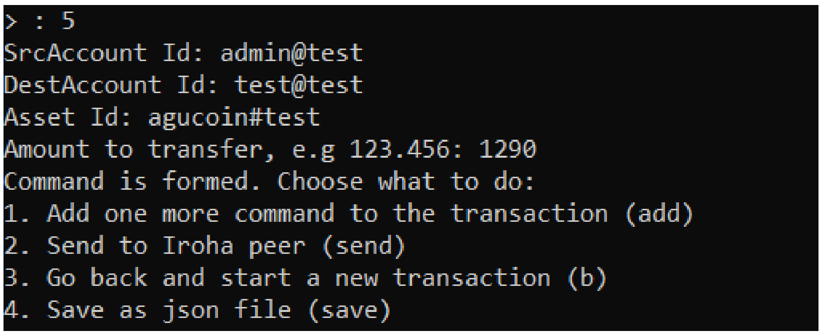


```
Asset name: agucoin
Domain Id: test
Asset precision: 2
Command is formed. Choose what to do:
1. Add one more command to the transaction (add)
2. Send to Iroha peer (send)
3. Go back and start a new transaction (b)
4. Save as json file (save)
```

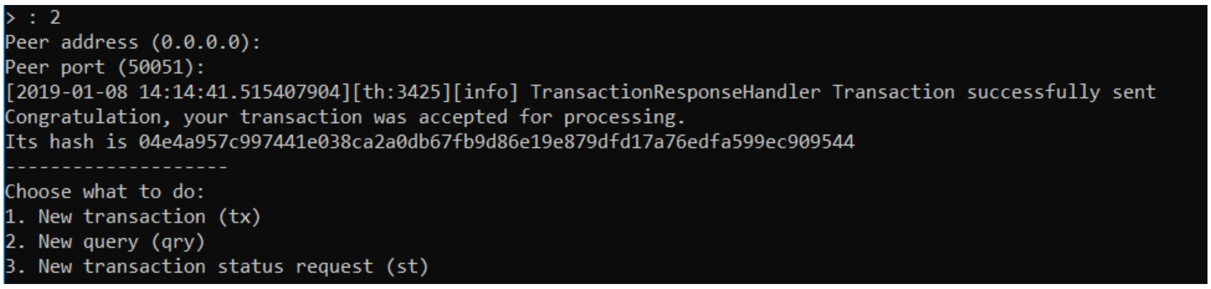
Before sending to Iroha peer, we can add one more command to the transactions. We will add asset quantity to an account. For this, type 16. For asset id, we give “agucoin#test”. For amount to add, I gave 123.90.


```
> : 16
Asset Id: agucoin#test
Amount to add, e.g 123.456: 12390
Command is formed. Choose what to do:
1. Add one more command to the transaction (add)
2. Send to Iroha peer (send)
3. Go back and start a new transaction (b)
4. Save as json file (save)
```

Now I can transfer some money to an account. I will send some money to "test@test". Our account name is admin@test. Like previous step, type 1 to add one more command. Then, select fifth option which is "Transfer Assets (trans_ast)". Src account id is admin@test and destAccount id is test@test. Our asset id is agucoin#test. I gave for 12.90 for amount to send.


```
> : 5
SrcAccount Id: admin@test
DestAccount Id: test@test
Asset Id: agucoin#test
Amount to transfer, e.g 123.456: 1290
Command is formed. Choose what to do:
1. Add one more command to the transaction (add)
2. Send to Iroha peer (send)
3. Go back and start a new transaction (b)
4. Save as json file (save)
```

Now, we are ready to send our transaction to Iroha peer. For doing this, choose "Send to Iroha peer" option. I used default options for peer address and peer port. Congratulations, you made first successful transaction in Iroha network.


```
> : 2
Peer address (0.0.0.0):
Peer port (50051):
[2019-01-08 14:14:41.515407904][th:3425][info] TransactionResponseHandler Transaction successfully sent
Congratulation, your transaction was accepted for processing.
Its hash is 04e4a957c997441e038ca2a0db67fb9d86e19e879dfd17a76edfa599ec909544
--------------------
Choose what to do:
1. New transaction (tx)
2. New query (qry)
3. New transaction status request (st)
```

Since Iroha provides API, given explanations and steps can be obtained using Python.

In following, some snippets from Python script is given and explained below, however if all code is required. It can be accessible from this link:

https://gist.github.com/ahmetturkmen/01c2423c988f9e58455fd3c7395d0e55

Step 1

```
import iroha
import block_pb2
import endpoint_pb2
import endpoint_pb2_grpc
import queries_pb2
import grpc
```

The required libraries should be imported before any process procedure. If there is an error while importing libraries then C & C++ libraries should be installed which are gcc and libso libraries.

Step 2 : Creating Iroha Objects

```
tx_builder = iroha.ModelTransactionBuilder()
query_builder = iroha.ModelQueryBuilder()
crypto = iroha.ModelCrypto()
proto_tx_helper = iroha.ModelProtoTransaction()
proto_query_helper = iroha.ModelProtoQuery()
## Read public and private keys
admin_priv = open("admin@test.priv", "r").read()
admin_pub = open("admin@test.pub", "r").read()
key_pair = crypto.convertFromExisting(admin_pub, admin_priv)
```

In this step, iroha objects should be defined in order to make some transaction over defined objects which uses helpers of Iroha. In this example , created public key is read to give some privileges to users. These objects will be used to create assets, accounts, sending objects, transactions and printing status of transactions.

Step 3: Defining functions to make script more readable, there are three different functions which deals with sending transactions, querying transactions and reporting transaction reporters.

Creating Transaction in Iroha using Pyhton .

```
def send_tx(tx, key_pair):
        tx_blob = proto_tx_helper.signAndAddSignature(tx, key_pair).blob()
        proto_tx = block_pb2.Transaction()
        if sys.version_info[0] == 2:
                tmp = ''.join(map(chr, tx_blob))
        else:
                tmp = bytes(tx_blob)
        proto_tx.ParseFromString(tmp)
        channel = grpc.insecure_channel(IP+':'+port)
        stub = endpoint_pb2_grpc.CommandServiceStub(channel)
        stub.Torii(proto_tx)
```

Querying committed transactions from Iroha by Python.

```
def send_query(query, key_pair):
        query_blob = proto_query_helper.signAndAddSignature(query, key_pair).blob()
        proto_query = queries_pb2.Query()
        if sys.version_info[0] == 2:
                tmp = ''.join(map(chr, query_blob))
        else:
                tmp = bytes(query_blob)
        proto_query.ParseFromString(tmp)
        channel = grpc.insecure_channel(IP+':'+port)
        query_stub = endpoint_pb2_grpc.QueryServiceStub(channel)
        query_response = query_stub.Find(proto_query)
        return query_response
```

These are two different functions which makes transactions and querying transactions' information from Iroha framework.

In same sense, to create domain and assets we can call Python library as follows.

```
tx = tx_builder.creatorAccountId(creator) \
        .createdTime(current_time) \
        .createDomain("domain", "user") \
        .createAsset("coin", "domain", 2).build()
```

After transaction is defined as given above send_tx function should be called.

```
send_tx(tx, key_pair)
print_status(tx)
```

Creating account using Python :

```
user1_kp = crypto.generateKeypair()
tx = tx_builder.creatorAccountId(creator) \
                           .createdTime(current_time) \
                           .createAccount("userone", "domain", user1_kp.publicKey()).build()
send_tx(tx, key_pair)
print_status(tx)
```

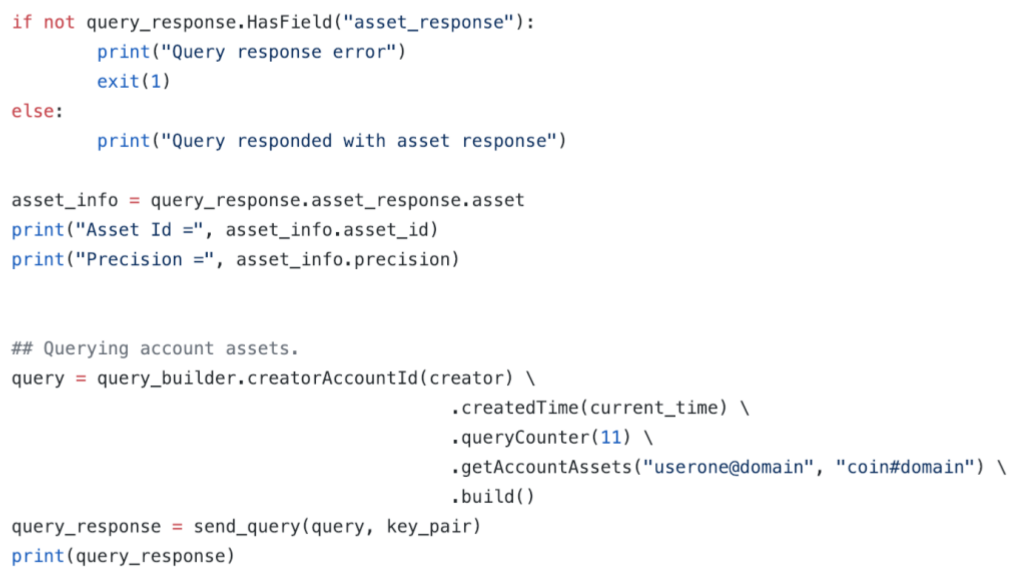
Sending assets using defined Python Iroha object.

```
tx = tx_builder.creatorAccountId(creator) \
                           .createdTime(current_time) \
                           .transferAsset("admin@test", "userone@domain", "coin#domain", "Some message",
                                ↪"2.0").build()
send_tx(tx, key_pair)
print_status(tx)
```

Querying assets information using Python:

```
query = query_builder.creatorAccountId(creator) \
                                         .createdTime(current_time) \
                                         .queryCounter(1) \
                                         .getAssetInfo("coin#domain") \
                                         .build()
query_response = send_query(query, key_pair)
```

Printing in structured way of transaction information & Querying account assets



```
if not query_response.HasField("asset_response"):
        print("Query response error")
        exit(1)
else:
        print("Query responded with asset response")

asset_info = query_response.asset_response.asset
print("Asset Id =", asset_info.asset_id)
print("Precision =", asset_info.precision)


## Querying account assets.
query = query_builder.creatorAccountId(creator) \
                                        .createdTime(current_time) \
                                        .queryCounter(11) \
                                        .getAccountAssets("userone@domain", "coin#domain") \
                                        .build()
query_response = send_query(query, key_pair)
print(query_response)
```

Iroha supports different programming languages, it has several SDKs for nodejs , python and java. So that all of them can be written in different languages.

References

1. Ruta, M., Scioscia, F., Ieva, S., Capurso, G., Pinto, A., & Di Sciascio, E. (2018). A Blockchain Infrastructure for the Semantic Web of Things. In *CEUR Workshop Proceedings*. https://doi.org/10.1016/j.csda.2009.12.007
2. HyperledgerDocs. (2018). Introduction — hyperledger-fabricdocs master documentation. https://doi.org/10.2320/matertrans.47.1898
3. Drescher, D. (2017). *Blockchain basics: A non-technical introduction in 25 steps*. *Blockchain Basics: A Non-Technical Introduction in 25 Steps*. https://doi.org/10.1007/978-1-4842-2604-9
4. IBM. (2018). IBM Blockchain based on Hyperledger Fabric from the Linux Foundation.
5. Dhillon, V., Metcalf, D., & Hooper, M. (2017). The Hyperledger Project. In *Blockchain Enabled Applications*. https://doi.org/10.1007/978-1-4842-3081-7_10