Hyperledger Mentorship Project Presentation

August 2021

Hyperledger Iroha + Cactus - Integration

- **Introduction**
  - **Name**: Han Xu
  - **Location:** Urbana, IL
  - **University:** University of Illinois at Urbana-Champaign
  - **Mentor(s):** Peter Somogyvari, Grzegorz Bazior
  - **Hyperledger Project**: Hyperledger Iroha + Cactus - Integration

Hyperledger Iroha + Cactus - Integration

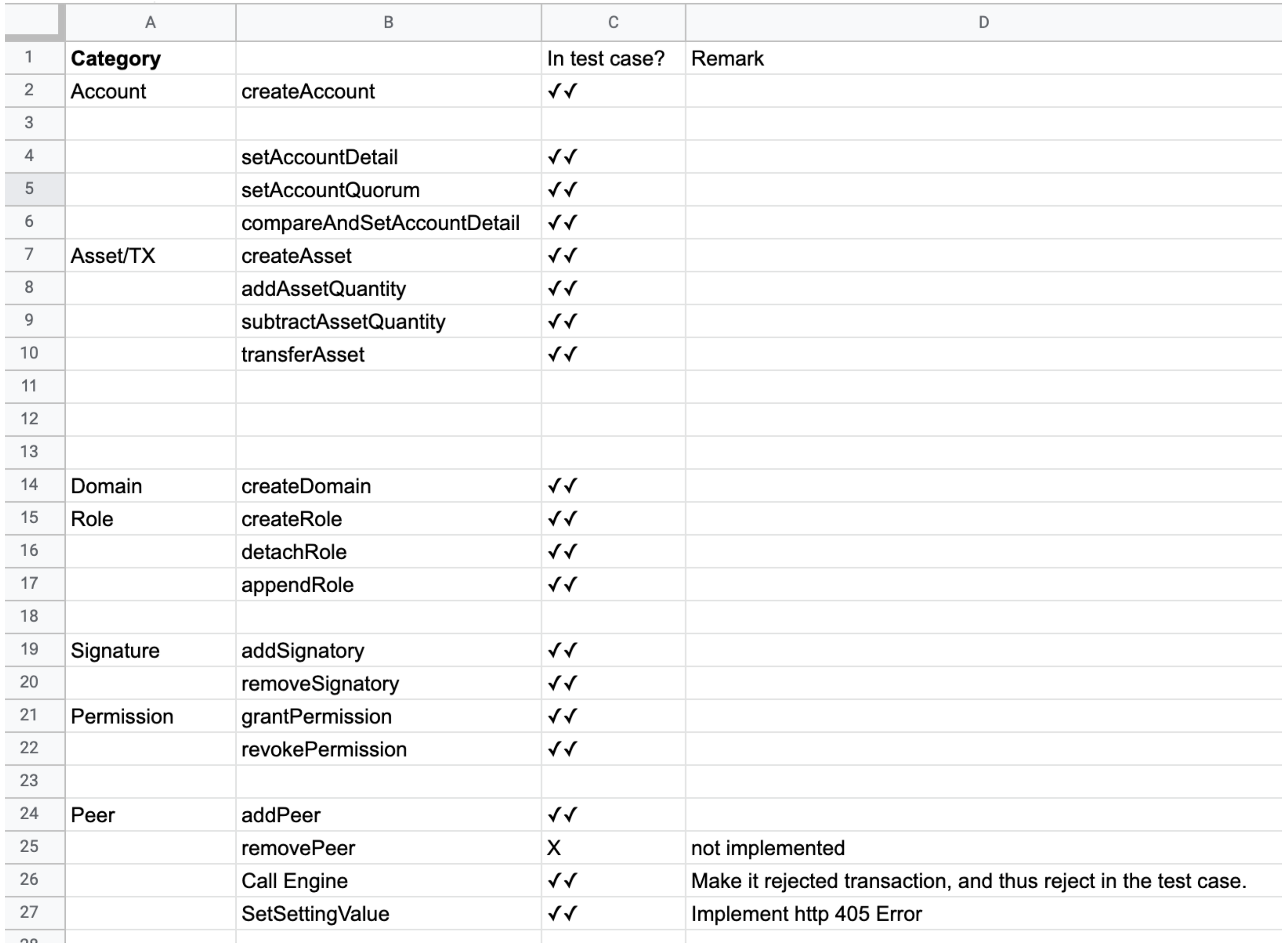
- **Project Description**:
- Cactus is a blockchain decentralized integration tool that allows users to securely integrate different blockchains. It has a pluggable architecture which makes easy to integrate various blockchain by creating plugins. Hence, Cactus can transfer both assets and data between multiple blockchains.
- Iroha (version 1.x) is great with asset management, and has functionality to store data, which makes Cactus and Iroha a perfect fit!
- Technologies used: Typescript, Node.js, Express, Docker

Hyperledger Iroha + Cactus - Integration

- **Project Execution & Accomplishments**:
- Accomplished: the Iroha connector plugin for Cactus project & modified Iroha all-in-one (AIO) dockerfile (and thus docker image) & documented example of integration between two Iroha networks with Cactus
- Not accomplished: examples of Iroha & Fabric integration
- Most proud: was able to support most of Iroha's commands and queries, as well as validate them.
- Most challenging: understanding Cactus's full architecture (i.e., how Cactus works, and how it connects to Iroha)

Hyperledger Iroha + Cactus - Integration

- **List of integrated Iroha commands**

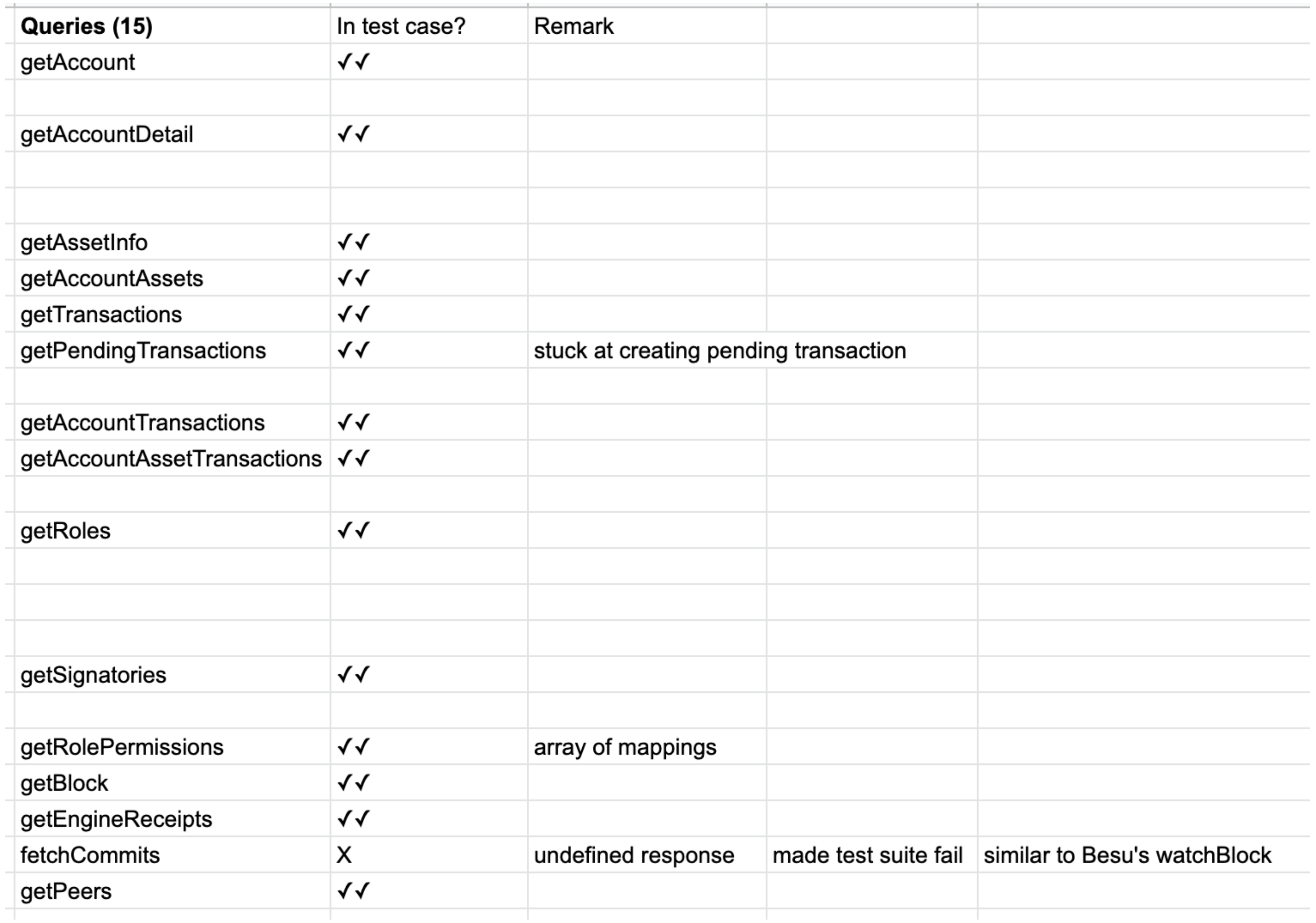


	A	B	C	D
1	**Category**		In test case?	Remark
2	Account	createAccount	✓✓	
3				
4		setAccountDetail	✓✓	
5		setAccountQuorum	✓✓	
6		compareAndSetAccountDetail	✓✓	
7	Asset/TX	createAsset	✓✓	
8		addAssetQuantity	✓✓	
9		subtractAssetQuantity	✓✓	
10		transferAsset	✓✓	
11				
12				
13				
14	Domain	createDomain	✓✓	
15	Role	createRole	✓✓	
16		detachRole	✓✓	
17		appendRole	✓✓	
18				
19	Signature	addSignatory	✓✓	
20		removeSignatory	✓✓	
21	Permission	grantPermission	✓✓	
22		revokePermission	✓✓	
23				
24	Peer	addPeer	✓✓	
25		removePeer	X	not implemented
26		Call Engine	✓✓	Make it rejected transaction, and thus reject in the test case.
27		SetSettingValue	✓✓	Implement http 405 Error

HYPERLEDGER
BLOCKCHAIN TECHNOLOGIES FOR BUSINESS

Hyperledger Iroha + Cactus - Integration

- List of integrated Iroha queries

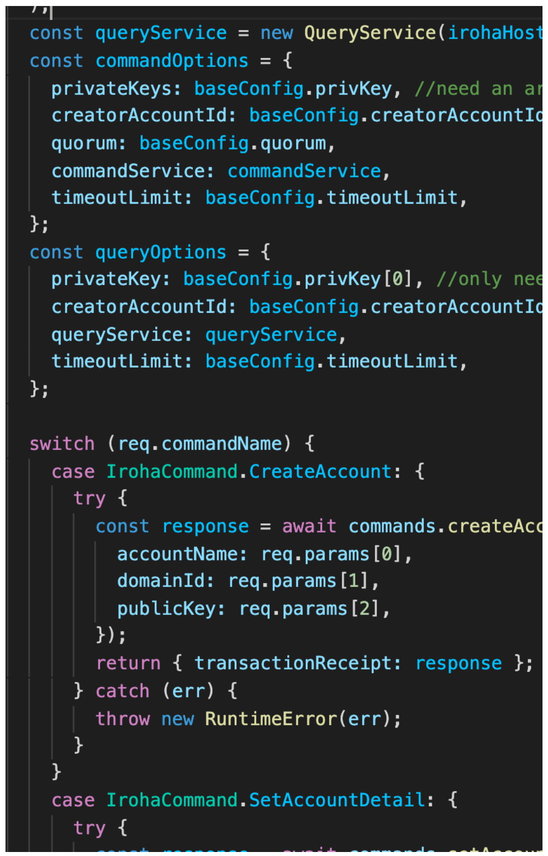


Queries (15)	In test case?	Remark		
getAccount	✓✓			
getAccountDetail	✓✓			
getAssetInfo	✓✓			
getAccountAssets	✓✓			
getTransactions	✓✓			
getPendingTransactions	✓✓	stuck at creating pending transaction		
getAccountTransactions	✓✓			
getAccountAssetTransactions	✓✓			
getRoles	✓✓			
getSignatories	✓✓			
getRolePermissions	✓✓	array of mappings		
getBlock	✓✓			
getEngineReceipts	✓✓			
fetchCommits	X	undefined response	made test suite fail	similar to Besu's watchBlock
getPeers	✓✓			

HYPERLEDGER
BLOCKCHAIN TECHNOLOGIES FOR BUSINESS

Hyperledger Iroha + Cactus - Integration

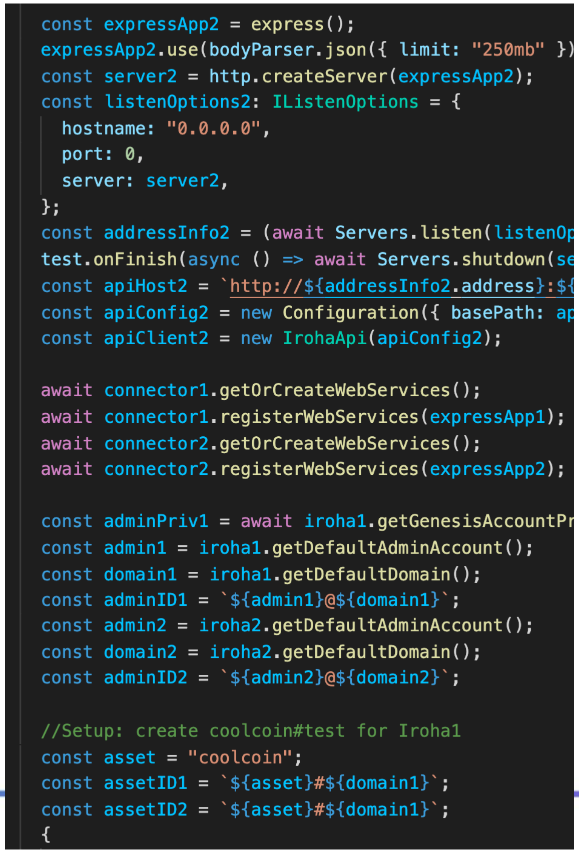
- **Project Output or Results:**
- 1st PR: https://github.com/hyperledger/cactus/pull/1183 (approved)
- 2nd PR: https://github.com/hyperledger/cactus/pull/1169 (merged)



```
});
const queryService = new QueryService(irohaHost
const commandOptions = {
  privateKeys: baseConfig.privKey, //need an ar
  creatorAccountId: baseConfig.creatorAccountIc
  quorum: baseConfig.quorum,
  commandService: commandService,
  timeoutLimit: baseConfig.timeoutLimit,
};
const queryOptions = {
  privateKey: baseConfig.privKey[0], //only nee
  creatorAccountId: baseConfig.creatorAccountIc
  queryService: queryService,
  timeoutLimit: baseConfig.timeoutLimit,
};

switch (req.commandName) {
  case IrohaCommand.CreateAccount: {
    try {
      const response = await commands.createAcc
        accountName: req.params[0],
        domainId: req.params[1],
        publicKey: req.params[2],
      });
      return { transactionReceipt: response };
    } catch (err) {
      throw new RuntimeError(err);
    }
  }
  case IrohaCommand.SetAccountDetail: {
    try {
```

Deal with Iroha commands/queries



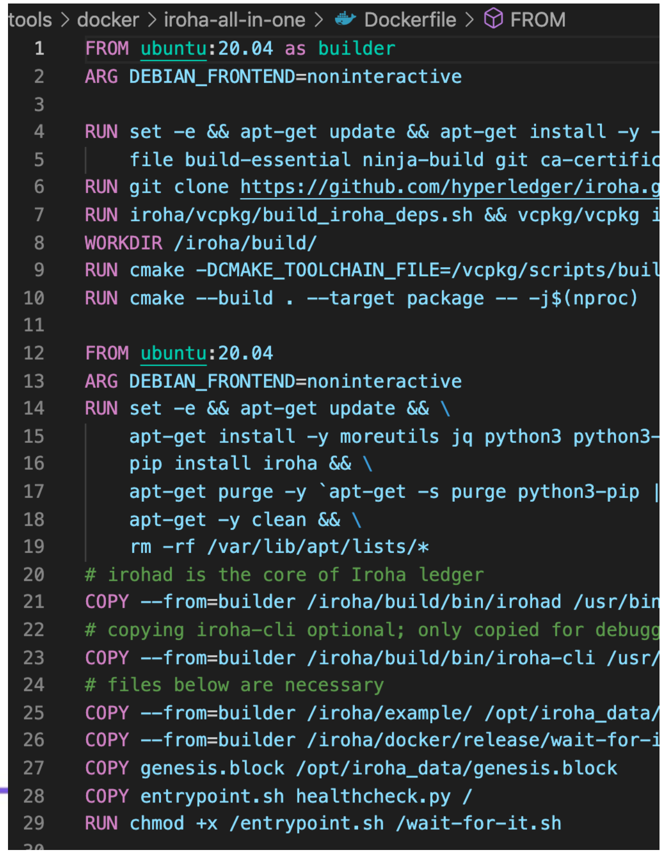
```
const expressApp2 = express();
expressApp2.use(bodyParser.json({ limit: "250mb" })
const server2 = http.createServer(expressApp2);
const listenOptions2: IListenOptions = {
  hostname: "0.0.0.0",
  port: 0,
  server: server2,
};
const addressInfo2 = (await Servers.listen(listenOp
test.onFinish(async () => await Servers.shutdown(se
const apiHost2 = `http://${addressInfo2.address}:${
const apiConfig2 = new Configuration({ basePath: ap
const apiClient2 = new IrohaApi(apiConfig2);

await connector1.getOrCreateWebServices();
await connector1.registerWebServices(expressApp1);
await connector2.getOrCreateWebServices();
await connector2.registerWebServices(expressApp2);

const adminPriv1 = await iroha1.getGenesisAccountPr
const admin1 = iroha1.getDefaultAdminAccount();
const domain1 = iroha1.getDefaultDomain();
const adminID1 = `${admin1}@${domain1}`;
const admin2 = iroha2.getDefaultAdminAccount();
const domain2 = iroha2.getDefaultDomain();
const adminID2 = `${admin2}@${domain2}`;

//Setup: create coolcoin#test for Iroha1
const asset = "coolcoin";
const assetID1 = `${asset}#${domain1}`;
const assetID2 = `${asset}#${domain1}`;
{
```

Iroha Cactus integration example (iroha node 1 transfers to node2)



tools > docker > iroha-all-in-one > Dockerfile > FROM

```
FROM ubuntu:20.04 as builder
ARG DEBIAN_FRONTEND=noninteractive

RUN set -e && apt-get update && apt-get install -y -
    file build-essential ninja-build git ca-certific
RUN git clone https://github.com/hyperledger/iroha.g
RUN iroha/vcpkg/build_iroha_deps.sh && vcpkg/vcpkg i
WORKDIR /iroha/build/
RUN cmake -DCMAKE_TOOLCHAIN_FILE=/vcpkg/scripts/buil
RUN cmake --build . --target package -- -j$(nproc)

FROM ubuntu:20.04
ARG DEBIAN_FRONTEND=noninteractive
RUN set -e && apt-get update && \
    apt-get install -y moreutils jq python3 python3-
    pip install iroha && \
    apt-get purge -y `apt-get -s purge python3-pip |
    apt-get -y clean && \
    rm -rf /var/lib/apt/lists/*
# irohad is the core of Iroha ledger
COPY --from=builder /iroha/build/bin/irohad /usr/bin
# copying iroha-cli optional; only copied for debugg
COPY --from=builder /iroha/build/bin/iroha-cli /usr/
# files below are necessary
COPY --from=builder /iroha/example/ /opt/iroha_data/
COPY --from=builder /iroha/docker/release/wait-for-i
COPY genesis.block /opt/iroha_data/genesis.block
COPY entrypoint.sh healthcheck.py /
RUN chmod +x /entrypoint.sh /wait-for-it.sh
```

Part of the Iroha AIO Dockerfile

Hyperledger Iroha + Cactus - Integration

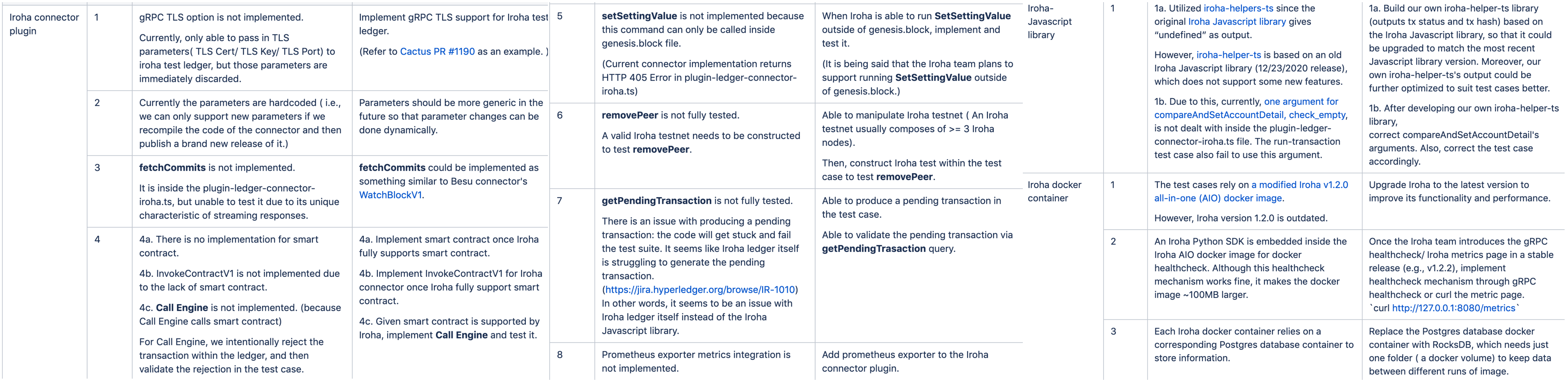
- Live demo of Iroha-Iroha asset transfer

Hyperledger Iroha + Cactus - Integration

- **Recommendations for future work (the top 3):**
- 1. Implement gRPC TLS transmission protocol for Iroha ledger.
- 2. Parameters should be more generic in the future so that parameter changes can be done dynamically.
- 3. Currently, utilized a third-party open-source library, “iroha-helper-ts”. But in the future, build our own “iroha-helper-ts” library (outputs tx status and tx hash) based on the Iroha Javascript library, so that it could be upgraded to the latest Javascript library and optimized.
- For a more detailed list, see: https://wiki.hyperledger.org/display/INTERN/Project+Plan+-+HL+Iroha+and+HL+Cactus+Integration

Hyperledger Iroha + Cactus - Integration

› **Full list of recommendations for future work:**

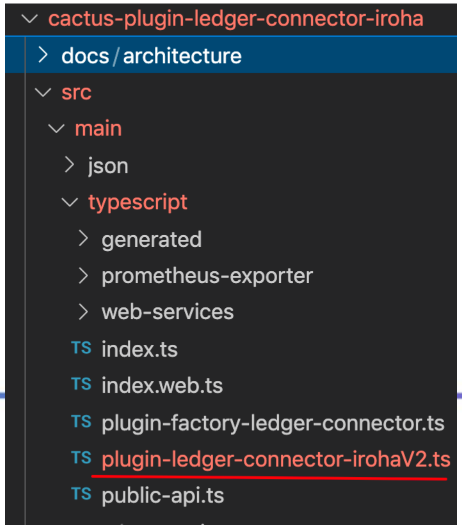


Iroha connector plugin	1	gRPC TLS option is not implemented. Currently, only able to pass in TLS parameters(TLS Cert/ TLS Key/ TLS Port) to iroha test ledger, but those parameters are immediately discarded.	Implement gRPC TLS support for Iroha test ledger. (Refer to Cactus PR #1190 as an example.)
	2	Currently the parameters are hardcoded (i.e., we can only support new parameters if we recompile the code of the connector and then publish a brand new release of it.)	Parameters should be more generic in the future so that parameter changes can be done dynamically.
	3	**fetchCommits** is not implemented. It is inside the plugin-ledger-connector-iroha.ts, but unable to test it due to its unique characteristic of streaming responses.	**fetchCommits** could be implemented as something similar to Besu connector's WatchBlockV1.
	4	4a. There is no implementation for smart contract. 4b. InvokeContractV1 is not implemented due to the lack of smart contract. 4c. **Call Engine** is not implemented. (because Call Engine calls smart contract) For Call Engine, we intentionally reject the transaction within the ledger, and then validate the rejection in the test case.	4a. Implement smart contract once Iroha fully supports smart contract. 4b. Implement InvokeContractV1 for Iroha connector once Iroha fully support smart contract. 4c. Given smart contract is supported by Iroha, implement **Call Engine** and test it.
	5	**setSettingValue** is not implemented because this command can only be called inside genesis.block file. (Current connector implementation returns HTTP 405 Error in plugin-ledger-connector-iroha.ts)	When Iroha is able to run **SetSettingValue** outside of genesis.block, implement and test it. (It is being said that the Iroha team plans to support running **SetSettingValue** outside of genesis.block.)
	6	**removePeer** is not fully tested. A valid Iroha testnet needs to be constructed to test **removePeer**.	Able to manipulate Iroha testnet (An Iroha testnet usually composes of >= 3 Iroha nodes). Then, construct Iroha test within the test case to test **removePeer**.
	7	**getPendingTransaction** is not fully tested. There is an issue with producing a pending transaction: the code will get stuck and fail the test suite. It seems like Iroha ledger itself is struggling to generate the pending transaction. (https://jira.hyperledger.org/browse/IR-1010) In other words, it seems to be an issue with Iroha ledger itself instead of the Iroha Javascript library.	Able to produce a pending transaction in the test case. Able to validate the pending transaction via **getPendingTrasaction** query.
	8	Prometheus exporter metrics integration is not implemented.	Add prometheus exporter to the Iroha connector plugin.
Iroha-Javascript library	1	1a. Utilized iroha-helpers-ts since the original Iroha Javascript library gives "undefined" as output. However, iroha-helper-ts is based on an old Iroha Javascript library (12/23/2020 release), which does not support some new features. 1b. Due to this, currently, one argument for compareAndSetAccountDetail, check_empty, is not dealt with inside the plugin-ledger-connector-iroha.ts file. The run-transaction test case also fail to use this argument.	1a. Build our own iroha-helper-ts library (outputs tx status and tx hash) based on the Iroha Javascript library, so that it could be upgraded to match the most recent Javascript library version. Moreover, our own iroha-helper-ts's output could be further optimized to suit test cases better. 1b. After developing our own iroha-helper-ts library, correct compareAndSetAccountDetail's arguments. Also, correct the test case accordingly.
Iroha docker container	1	The test cases rely on a modified Iroha v1.2.0 all-in-one (AIO) docker image. However, Iroha version 1.2.0 is outdated.	Upgrade Iroha to the latest version to improve its functionality and performance.
	2	An Iroha Python SDK is embedded inside the Iroha AIO docker image for docker healthcheck. Although this healthcheck mechanism works fine, it makes the docker image ~100MB larger.	Once the Iroha team introduces the gRPC healthcheck/ Iroha metrics page in a stable release (e.g., v1.2.2), implement healthcheck mechanism through gRPC healthcheck or curl the metric page. `curl http://127.0.0.1:8080/metrics`
	3	Each Iroha docker container relies on a corresponding Postgres database container to store information.	Replace the Postgres database docker container with RocksDB, which needs just one folder (a docker volume) to keep data between different runs of image.

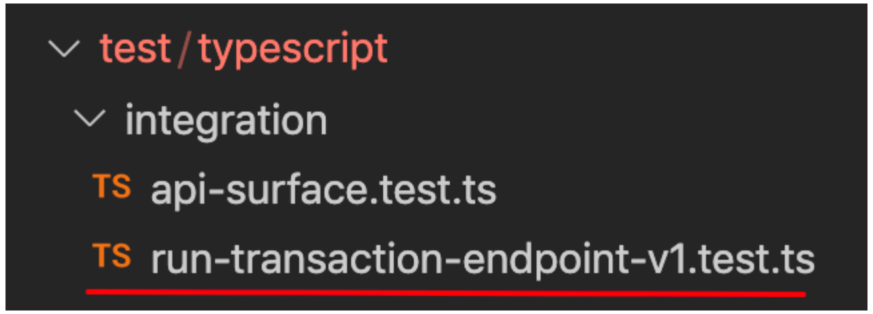
› For a more detailed list, see: https://wiki.hyperledger.org/display/INTERN/Project+Plan+-+HL+Iroha+and+HL+Cactus+Integration

Hyperledger Iroha + Cactus - Integration

- **Usual steps for developing a Cactus connector plugin**
- (in this case: develop an Iroha 2.x plugin for the Cactus project)
- 1. Create a modified Iroha 2.x all-in-one (AIO) docker image
- 2. Utilize Cactus to start the Iroha 2.x AIO docker container
- 3. Use Cactus’s function calls to manipulate Iroha 2.x docker container (i.e., conduct various transactions/ invoke smart contract/ etc) (core file: plugin-ledger-connector-irohaV2.ts)
- 4. Need to modify other files to make the whole connector package compile
- 5. Validate the implementation (step 3) was successful via test cases (e.g., run-transaction-endpoint-v1.test.ts)



```
cactus-plugin-ledger-connector-iroha
  > docs / architecture
  v src
    v main
      > json
      v typescript
        > generated
        > prometheus-exporter
        > web-services
        TS index.ts
        TS index.web.ts
        TS plugin-factory-ledger-connector.ts
        TS plugin-ledger-connector-irohaV2.ts
        TS public-api.ts
```



```
v test / typescript
  v integration
    TS api-surface.test.ts
    TS run-transaction-endpoint-v1.test.ts
```

Hyperledger Iroha + Cactus - Integration

- **Insights Gained:**
- Learnt about the workflow of how an open source project is developed

- **Advice:**
- Have a solid plan in the beginning so that you can follow
- Actively reach out to mentors to seek advice and feedback
- Learning from the open-source community is also very helpful

Hyperledger Iroha + Cactus - Integration

- **Questions**

Hyperledger Iroha + Cactus - Integration

- A big thank to my mentors (Peter and Greg), and also the Cactus and Iroha community.
- Also thank LFX for hosting and sponsoring this mentorship

THANK YOU!