



University of Applied Sciences

Intern : Martijn Lammers

Supervisor(s) : Leen Blom, Edwin Fennema

Developer Guide CCF 1.0.13

3 September 2021

Contents

1	Preface	3
2	Pre-requisites	4
2.1	Foreknowledge	4
2.2	Hardware	4
2.3	Software	4
3	Dictionary	5
4	Installing CCF	6
5	Typescript Applications	7
5.1	Setup	8
5.1.1	Folder hierarchy	8
5.1.2	Configuration files	8
5.2	Packages	8
5.2.1	CCF package	8
5.2.2	Other packages	9
5.3	Create Application	9
5.3.1	Logic	9
5.3.2	Endpoint configuration	10
5.4	Preparing application for use	11
5.4.1	Typescript conversion	11
6	Testing applications	12
6.1	Run local network (Test network)	12
6.2	Using local network	13
6.2.1	Changing application	15
6.2.2	Use application	16
7	Public Network	16
7.1	Generating keys	17
7.2	Nodes	17
7.2.1	Startup node	17
7.2.2	Recover node	17
7.2.3	Add node	18

8	Goverance	18
8.1	Proposals	18
8.1.1	Register and activate members	19
8.1.2	Creating proposals	19
8.1.3	Sending proposals	20
8.1.4	Accepting proposals	20
9	Using the network application	22
9.1	Open network	22
9.2	Client	22

1 Preface

Special thanks to the CCF maintainers for helping me out during my struggles, I would not have gotten through it without them. This document is an accumulation of the findings on how to move through the CCF framework by the Centric intern Martijn Lammers. The document hasn't encapsulated the full extent of the framework and it's highly recommended to also read the [official documentation](#) besides this as they elaborate more than this document might do.

After reading this manual, the reader should be able to create their own multi-node governed network containing distributed ledgers running in encrypted CPU enclaves with custom Javascript applications accessing Key value-stores. The shell scripts given are used to automate the multitude of commands you'd otherwise have to manually type in.

If you've used the manual before or are aware with how to use CCF, setup instantly using:

```
$ git clone https://github.com/martijnlammers/ccfsetup.git
$ bash ~/ccfsetup/ccf_all.sh
```

Afterwards you would only need to change your DNS in the configuration files.

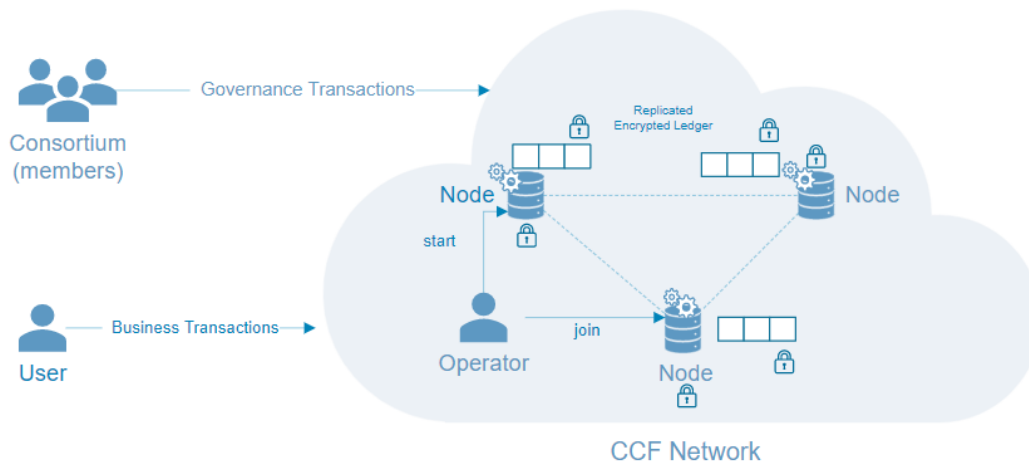


Figure 1: CCF Concept

2 Pre-requisites

2.1 Foreknowledge

- CCF Concepts
- Linux bash commands.
- NodeJS
- JSON
- Typescript
- Python, pip
- Networking basics (HTTPS, Requests, Certificates)

2.2 Hardware

- Intel CPU with SGX support

2.3 Software

- Ubuntu 20.04
- (Optional) Visual Studio Code

3 Dictionary

Request: Web packet send to a server.

Response: A returning web packet received by the server.

Enclave: Encrypted enviornments in your CPU and RAM
inaccessible by the rest of the computer.

CCF: Confidential Consortium Framework, used to create
democratic node network.

Node: Connection points, in our case, computers running
ccf connected to another computer.

Key-value store(KV store): Datastructure storing a value
attached to a key into the enclave.

User: Certificate/keys registered as user in the network.

Member: Certificate/keys registered as member in the network.

Anonymous user: User without registered certificates/keys
in the network.

Endpoint: Javascript file exporting functions that handle
incoming requests.

Ledger: A file tracking the historical changes to the system.
(Transactions, proposals, votes)

4 Installing CCF

Before you can use the framework, you first need to install it.

```
$ git clone https://github.com/mAlyanak/ccfsetup.git
$ bash ./ccfsetup/install_ccf.sh -v 1.0.13
```

Running this script will install the dependencies of CCF and CCF itself. You may use a different version number to install later versions of CCF. This manual is specific for 1.0.13, although .14 and .15 also are very probable to work identical to its predecessor .13. Running the script without flags will default to installing 1.0.13.

5 Typescript Applications

This section of the manual will explain how to create an application, not how to actually use it in the network or test it. The reader can write applications in either C++ or Javascript/Typescript. However there will not be an explanation on how to create C++ applications. The reader can find more information on how to do that [here](#).

Javascript/Typescript applications specifically are bundles. A bundle is a folder structure that contains endpoint files, with your logic and a configuration file, called 'app.json' with the metadata.

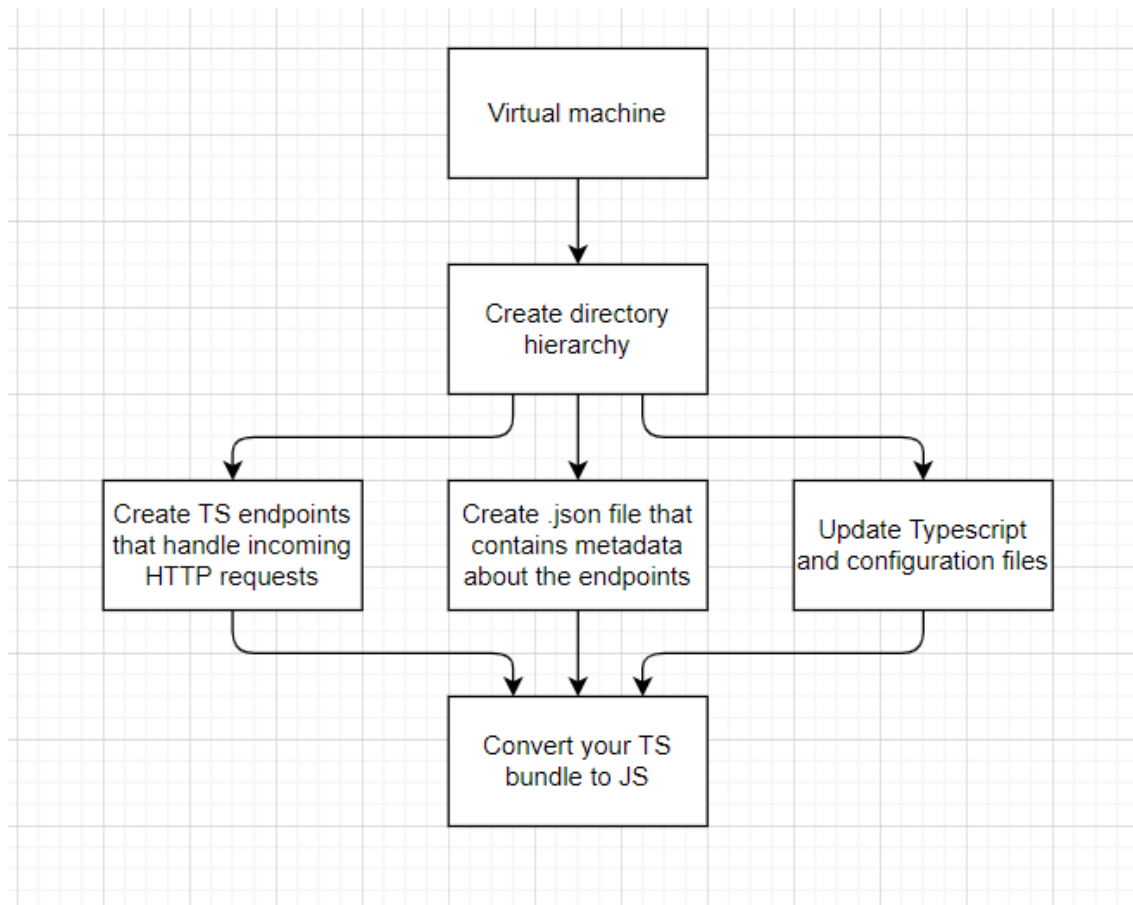


Figure 2: Application building flowchart

5.1 Setup

Run the following scripts to quickly prepare a directory you can work in to create your custom JS/TS app.

5.1.1 Folder hierarchy

```
$ bash ./ccfsetup/ccf_app_setup.sh
```

As you might notice, the script has generated a folder with some files, this is a generic app bundle. 'app.json' will keep the metadata - or configuration - of your endpoints. Find more on how to configure your endpoints [here](#).

Inside the folder 'endpoints' the reader can create endpoints. Endpoints are Javascript files exporting functions. These functions can then later be invoked by users to access the created application, defined by the 'app.json' file.

After creating your endpoints, you need to export your functions in 'all.ts'. This specific file is in harmony with the configuration files that will be used later.

It's important to maintain this structure for now as the configuration files depend on finding them like this, especially if it's the first time of working with CCF. If one would want to change the structure, it would also be necessary to change the configuration files.

5.1.2 Configuration files

```
$ bash ./ccfsetup/ccf_app_config.sh -v 1.0.13
```

When running 'ccf_app_config.sh' you will generate 3 files. These configuration files are necessary for the Typescript to Javascript conversion. 'rollup.config.js' is used to shake off any duplicate modules, 'tsconfig.json' contains your Typescript configuration and 'package.json' keeps track of the dependencies you're using.

5.2 Packages

5.2.1 CCF package

CCF also has a helpful package with various tools the reader can use, by example; cryptography, parsing/converters, key-value store access and more. The full docu-

mentation on the CCF package can be found [here](#). The generated endpoint sample imports this module by standard so you can immediately make use of it.

5.2.2 Other packages

If you want to use other modules, you'd have to change your configuration files. For this, edit 'package.json' and add the desired module to the schema with the right version number.

5.3 Create Application

5.3.1 Logic

As mentioned before, a JS/TS application is actually actually a bundle of files. You have your primary endpoint files and a file containing metadata('app.json'). These endpoint files will contain all of your application logic. These js/ts endpoint files export functions that you can later invoke/call using requests. The functions all would have to receive a request as parameter and return a response, you can however also create seperate functions that can use a different signature. In the folder 'endpoints' under 'src' you can find an example.

5.3.2 Endpoint configuration

After programming your application logic and exporting functions, the next step would be to configure your endpoints. Edit 'app.json', here the reader can see rather self-explanatory fields that need to be fill in for every used endpoint.

'/sample' : This is a REST API url, you can name it anything you'd like. When making requests you'd have to invoke a call to this url.

'post' : HTTP method. POST and GET are most usually used.

'js_module': The Javascript file where your function can be found in.

'js_function': The exported function that needs to be executed.

'forwarding_required': A string indicating whether the endpoint is always forwarded, or whether it is safe to sometimes execute on followers.

'authn_policies': How you authenticate who can make API calls, if this is left blank anyone can make requests to the endpoint.

'mode': What access your function has to the key-value store, can be readwrite, readonly and historical.

'openapi': Used for swagger API documentation. (Can be left blank)

5.4 Preparing application for use

5.4.1 Typescript conversion

Before you can actually use your application, the last step would be to convert your TS to JS.

```
$ cd ~/app_building_dir
$ npm install
$ npm run build
$ cd ~
```

Using 'npm install' you install all your dependencies/modules, 'npm run build' will do the conversion from TS to JS. After running 'npm run build' your converted files can be found in the folder 'dist'. This version is ready for deployment in a sandbox (test environment) or a public network.

6 Testing applications

This section of the manual will explain how you can test your custom applications.

6.1 Run local network (Test network)

The official release of CCF contains a sandbox script to run a local network. In this network, there only is 1 member so that proposals are automatically accepted. You can load your application to the network in order to tests its functionality. Technically you can also use a real network to test your application. If you're at an early stage, this is fine, but obviously you don't want to be continuessly changing the application when you're running a real network after testing phase, so thats where the sandbox comes in.

```
$ sudo ~/ccfrelease/bin/sandbox.sh
```

You have to use sudo to run the sandbox, cause the script needs the root permissions. You might run into a pyparser/cimetric error the first time you run the script, but the second time you run it, it'll be gone. It's important to note that running the sandbox does not make use of an encrypted enclave. If you do want to use these, it's necessary to also add the flag '-enclave-type release'.

When starting the sandbox, it will output what address you can access your node on and in what directory your sample certificates/keys are generated at. You can potentially generate more nodes in the sandbox adding '-n local://127.0.0.1:8000 -n local://127.0.0.2:8000 ...'.

```
mar@lam:~$ sudo ~/ccfrelease/bin/sandbox.sh -n local://127.0.0.1:8000 -n local://127.0.0.2:8000 -n local://127.0.0.3:8000
Setting up Python environment...
No package/app specified. Defaulting to installed JS logging app
Python environment successfully setup
[10:16:47.540] Starting 3 CCF nodes...
[10:16:47.541] Virtual mode enabled
[10:16:53.344] Started CCF network with the following nodes:
[10:16:53.345]   Node [0] = https://127.0.0.1:8000
[10:16:53.346]   Node [1] = https://127.0.0.2:8000
[10:16:53.346]   Node [2] = https://127.0.0.3:8000
[10:16:53.347] You can now issue business transactions to the /home/mar/ccfrelease/bin/../lib/libjs_generic application
[10:16:53.347] Loaded JS application: /home/mar/ccfrelease/bin/../samples/logging/js
[10:16:53.347] Keys and certificates have been copied to the common folder: /home/mar/workspace/sandbox/common
[10:16:53.348] See https://microsoft.github.io/CCF/main/use_apps/issue_commands.html for more information
[10:16:53.348] Press Ctrl+C to shutdown the network
```

Figure 3: ip(s) and Certificates/keys

6.2 Using local network

The sandbox (and the public network) make use of three different role groups: members, users and anonymous users. Members primarily go over the governance in the network. They can send signed requests to the network and offer proposals. After a proposal is made, all of the members get to vote for or against it, if a majority accepts, the proposal gets applied to the network. The local network only can have one member, so proposals are automatically accepted.

Users, as described, mainly use the application and have access to endpoints with user authentication policy only if their certificates are registered in the network. Anonymous users can make requests to any endpoint without authentication policies, they also don't have to be registered in the network and don't need any certificates.

The roles do overlap, meaning a member can also be registered as an user with the same certificates. You can add or remove members/users using proposals. Proposals in a nutshell are signed requests with a json file containing the proposal. More elaboration on how to create and send these follows.

Endpoints that go over the governance by default have member authentication, so that's why only members can make proposals. Members can't make use of the application unless they're also registered as users.

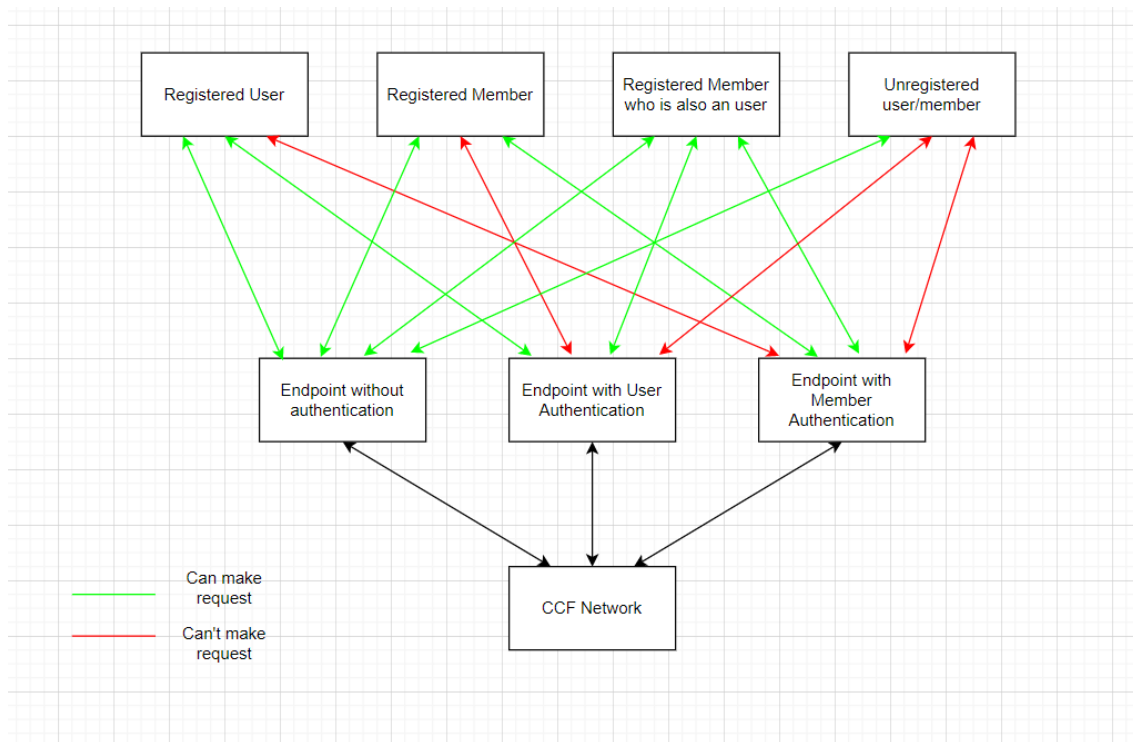


Figure 4: Roles visualized

6.2.1 Changing application

Changing an application in your sandbox (and public network) can be done through proposals. For this, you first need to generate a proposal for your application using CCF's python module. Go into a second terminal while your network is running in the other one.

```
# In a second terminal, go into your workspace
$ cd ~/app_building_dir

# Create proposal for your application
$ python3 -m ccf.proposal_generator set_js_app ~/app_building_dir/dist
```

After you generated your proposal, a member needs to send a signed request to the network. The sandbox automatically generates certificates and keys for one user and a member you can use. You can send signed requests using `scurl.sh`, this is a script given by ccf that will sign curl requests for you.

```
# Copy certificates in your working directory (Ease of access)
$ mkdir ~/app_building_dir/certificates
$ sudo cp ~/workspace/sandbox_common/*.pem ~/app_building_dir/certificates

# Change the permissions of the certificates so you can use them.
$ sudo chmod ugo+rwX ~/app_building_dir/certificates/*.pem

# Send proposal to change application
# Note, the '\ ' character is a command that
# will allow you to continue your command in another line.

$ scurl.sh https://127.0.0.1:8000/gov/proposals \
--cacert ./certificates/networkcert.pem \
--signing-key ./certificates/member0_privk.pem \
--signing-cert ./certificates/member0_cert.pem \
--data-binary @set_js_app_proposal.json

# Instantly changes network application cause you're the only member.
```


6.2.2 Use application

Now that you've changed the application, you can send requests to your own custom endpoints in a local network. This goes practically the same way as sending proposals, just with different certificates, endpoint and data. No certificates in the samples case as we're sending a request to an anonymous endpoint. If you're a user, or anonymous, you don't need to sign your request and you can use standard curl to send your requests.

```
# Send anonymous request
$ curl https://127.0.0.1:8000/app/sample -X POST \
--cacert ./certificates/networkcert.pem \
--data-binary '{"something":"somethingElse"}'

# Your output should say: {"response":"Hello world!"}
# unless you changed the sample application.

# If the endpoint has user authentication,
# you would need to send the request with the certificates
# that are registered as a user like so:
$ curl https://127.0.0.1:8000/app/sample -X POST \
--cacert ./certificates/networkcert.pem \
--key ./user0_privk.pem \
--cert ./user0_cert.pem \
--data-binary '{"something":"somethingElse"}'
```

7 Public Network

```
$ bash ~/ccfsetup/ccf_node_configs.sh
```

After running 'ccf_node_configs.sh', there will be a folder with 3 different configuration files. These 3 configuration files all have a different purpose. 'start_node.ini' is the first node to start up your network. After a main node is created, it's possible to add a new node with 'join_node.ini' to your network. If you want to create extra nodes, you'd have to replicate 'join_node.ini' with different configuration. Lastly, there is 'recover_node.ini', which you will use to recover a single node, in case it for some reason shut off or got an invalid ledger.

7.1 Generating keys

Before you can create your nodes, you need initial member keys and certificates so the network can indentify your member. Nodes need to have at least 1 member before you can start them up. A single member (so with thesame certs/keys) can be used to start up two different nodes.

```
$ cd ~/network/certificates
```

```
# You don't want to be using sh in front of this command as it
# will use dash shell instead of bash shell. You can name your member
# however you want, I kept it generic and called my member 'Member'.
# If you're using a different name, you also have to edit start_node.ini
# and change 'member-info='
$ ../../ccfrelease/bin/keygenerator.sh \
--name Member \
--gen-enc-key
```

7.2 Nodes

7.2.1 Startup node

Finally, you've reached the point where you can startup a single node network! Make sure to check the configuration file(s) before you start, add or recover your node. If you don't have a DNS, you need to send your requests to the public address. After checking your configuration, it's possible to start a network using the following command:

```
$ cd ~
$ sudo ~/ccfrelease/bin/cchost \
--config=$(pwd)/network/configurations/start_node.ini
```

7.2.2 Recover node

Let's assume the node shut down or has too much bad data in the ledger, it would be necessary to recover that node to make it functional again. To do this, the 'recover_node.ini' file will be used.

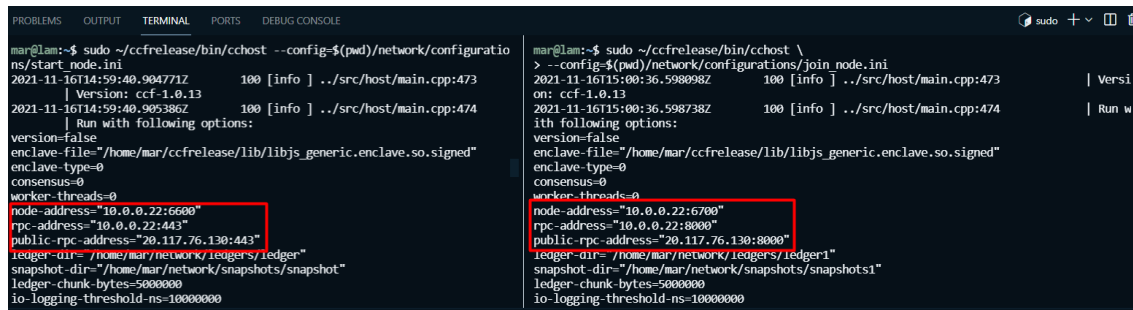
```
$ sudo ~/ccfrelease/bin/cchost \
--config=$(pwd)/network/configurations/recover_node.ini
```

After running the command, the node will be reintroduced to the network.

7.2.3 Add node

In a different VM with a different ip, or terminal with different ports, you can make a node join a network. It is however necessary to have the network certificate of the main node and configure the target address of the network you want to participate in. If you want to create a new node with the samples, inside a second terminal run:

```
$ sudo ~/ccfrelease/bin/cchost \
--config=$(pwd)/network/configurations/join_node.ini
```



```
mar@lam:~$ sudo ~/ccfrelease/bin/cchost --config=$(pwd)/network/configuratio
ns/start_node.ini
2021-11-16T14:59:40.904771Z 100 [info ] ../src/host/main.cpp:473
| Version: ccf-1.0.13
2021-11-16T14:59:40.908386Z 100 [info ] ../src/host/main.cpp:474
| Run with following options:
version=false
enclave-file="/home/mar/ccfrelease/lib/libjs_generic.enclave.so.signed"
enclave-type=0
consensus=0
worker-threads=0
node-address="10.0.0.22:6600"
rpc-address="10.0.0.22:443"
public-rpc-address="20.117.76.130:443"
ledger-dir="/home/mar/network/ledgers/ledger"
snapshot-dir="/home/mar/network/snapshots/snapshot"
ledger-chunk-bytes=5000000
io-logging-threshold-ms=10000000

mar@lam:~$ sudo ~/ccfrelease/bin/cchost \
> --config=$(pwd)/network/configurations/join_node.ini
2021-11-16T15:00:36.598098Z 100 [info ] ../src/host/main.cpp:473
on: ccf-1.0.13
2021-11-16T15:00:36.598738Z 100 [info ] ../src/host/main.cpp:474
ith following options:
version=false
enclave-file="/home/mar/ccfrelease/lib/libjs_generic.enclave.so.signed"
enclave-type=0
consensus=0
worker-threads=0
node-address="10.0.0.22:6700"
rpc-address="10.0.0.22:8000"
public-rpc-address="20.117.76.130:8000"
ledger-dir="/home/mar/network/ledgers/ledger1"
snapshot-dir="/home/mar/network/snapshots/snapshots1"
ledger-chunk-bytes=5000000
io-logging-threshold-ms=10000000
```

Figure 5: Two nodes running in two terminals in one VM

8 Governance

This section explains how to register and activate a member, create a proposal, vote for or against them, and changing the network application as example. Every transaction, proposal and changes are stored in the ledger.

8.1 Proposals

Members go over the governance of the network. The main function of these members is making and voting for proposals. There is a plethora of proposals members can make adding users, members, changing app or constitution and much more. The full list can be outputted by running:

```
$ python3 -m ccf.proposal_generator
```

8.1.1 Register and activate members

If you've followed the rest of the manual, you currently have one registered member. Registered members cannot make proposals yet, firstly, they have to be trusted (or activated) by the network. Assuming you have a running network, in a second terminal you can send your request to activate a member.

```
# Make sure to change https://demo2.uksouth.cloudapp.azure.com
# to your own host address.

# Make sure to use the correct certificates and
# send your requests to a valid url,
# else you're expected to get a curl/SSL error (or even get stuck).

# Activate initial member
$ cd ~
$ STATE_DIG=$(curl \
https://demo2.uksouth.cloudapp.azure.com/gov/ack/update_state_digest \
-X POST \
--cacert ~/network/certificates/network_cert.pem \
--key ~/network/certificates/Member_privk.pem \
--cert ~/network/certificates/Member_cert.pem)

$ ./ccfrelease/bin/scurl.sh \
https://demo2.uksouth.cloudapp.azure.com/gov/ack \
--cacert ~/network/certificates/network_cert.pem \
--signing-key ~/network/certificates/Member_privk.pem \
--signing-cert ~/network/certificates/Member_cert.pem \
--header "Content-Type: application/json" \
--data-binary ${STATE_DIG}
```

If you receive nothing back after the second command, you successfully activated your member. You can now make proposals with this member.

8.1.2 Creating proposals

The next few steps will teach the reader to make and accept proposals. This will be done through the example of proposing a new javascript application.

```
# Create a proposal for you application.
$ cd ~/app_building_dir/
$ python3 -m ccf.proposal_generator set_js_app ./dist/
```

8.1.3 Sending proposals

After creating the proposal, you need to send a signed request with your proposal and authentication to the network.

```
# Make sure to change https://demo2.uksouth.cloudapp.azure.com
# to your own host address.

# Send proposal
$ PROPID=$(scurl.sh https://demo2.uksouth.cloudapp.azure.com/gov/proposals\
--cacert ~/network/certificates/network_cert.pem \
--signing-key ~/network/certificates/Member_privk.pem \
--signing-cert ~/network/certificates/Member_cert.pem \
--data-binary @set_js_app_proposal.json \
-H "content-type: application/json" | jq -r '.proposal_id')
```

This request will give you back a response containing the vote count, proposal id, state of the proposal and votes. You will need the proposal id to vote for the proposal.

8.1.4 Accepting proposals

Normally, after proposing a change, all the members would need to vote for or against it. The python proposal_generator also created a vote ballot besides your proposal. You can edit this. In the case of our sample, it won't be necessary to vote for proposals as there only is one member, causing the proposal to instantly be accepted. In a real situation with more members, you would need to send a proposal vote like so:

```
# Make sure to change https://demo2.uksouth.cloudapp.azure.com
# to your own host address.

# True or false, accept or reject,
# change accordingly in set_js_app_vote_for.json.

# Send vote to proposal
$ scurl.sh https://demo2.uksouth.cloudapp.azure.com/gov/proposals\
```

```
/${PROPID}/ballots \  
--cacert ~/network/certificates/network_cert.pem \  
--signing-key ~/network/certificates/Member_privk.pem \  
--signing-cert ~/network/certificates/Member_cert.pem \  
--data-binary @set_js_app_vote_for.json -H "content-type: application/json"
```

9 Using the network application

This sections explains how users can send request to a ccf app by using a client.

9.1 Open network

Before users can make use of the application, the service first needs to be opened.

```
# Make sure to change https://demo2.uksouth.cloudapp.azure.com
# to your own host address.

# Create proposal to open network
$ python3 -m ccf.proposal_generator transition_service_to_open

# Send proposal
$ PROPID=$(scurl.sh https://demo2.uksouth.cloudapp.azure.com/\
gov/proposals --cacert ~/network/certificates/network_cert.pem \
--signing-key ~/network/certificates/Member_privk.pem \
--signing-cert ~/network/certificates/Member_cert.pem \
--data-binary @transition_service_to_open_proposal.json \
-H "content-type: application/json" | jq -r '.proposal_id')

# Accept proposal
$ scurl.sh https://demo2.uksouth.cloudapp.azure.com/gov/proposals\
/${PROPID}/ballots \
--cacert ~/network/certificates/network_cert.pem \
--signing-key ~/network/certificates/Member_privk.pem \
--signing-cert ~/network/certificates/Member_cert.pem \
--data-binary @transition_service_to_open_vote_for.json \
-H "content-type: application/json"
```

9.2 Client

```
$ mkdir ~/client/
$ cp ~/ccfsetup/client_sample.js ~/client/

# Edit the options in the sample to fit your own system.
```

It's important to note this client will not work unless you change the options. It is only an exemplary template on how you can create your client. CCF uses a standard protocol, HTTP over TLS, so technically you can create a client in any language that supports these protocols. You can use this sample client outside your VM environment as well, however you need to transfer the right certificates to your machine.

```
$ node ~/client/client_sample.js  
# Should return {"response":"Hello world!"}
```