**Adding an Org to Channel**

Reconfiguration is something that SDKs will hopefully expose in the future. However, for v1 this was not realistic to be completed. That is why configtxlator exists, to give a language neutral way to help SDK users to perform reconfiguration. Ideally, the application is storing the most up to date copy of the organization definition somewhere for each org, which can be used in the application to construct the change request.

The standard usage is expected to be:

1. SDK retrieves latest config
2. configtxlator produces human readable version of config
3. User or application edits the config
4. configtxlator is used to compute config update representation of changes to the config
5. SDK submits signs and submits config

Currently, we are able to compute config update from sdk , however the last step is recently updated as a part of sdk and will be included in v1.0.1. We are working on consuming the updated methods. You may expect fabric-sdk-java update in next build.

At the moment, the reconfiguration flow to add an org for v1 requires:

1. Generating a genesis block containing the crypto material for your org in a similar manner as we did for configuration of 3 orgs network.

* Make changes in configtx.yaml for 1 org and generate certificates and genesis.block. Follow steps in

<https://github.com/infor-cloud/supply-chain-insights#configurations>

* Run configtxlator git bash using command in :

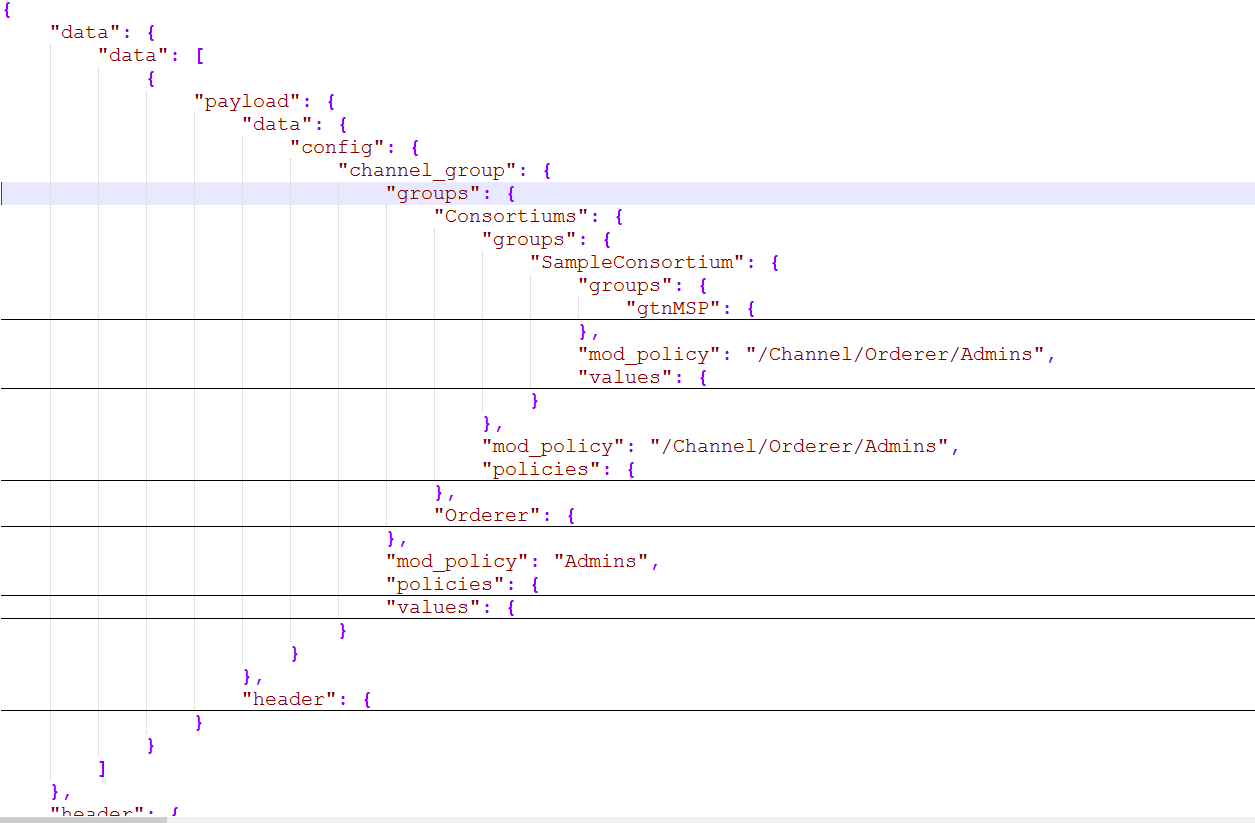
configtxlator start

* Decode the above generated genesis.block using command :

curl **-**X POST **--**data**-**binary @genesis.block http:**//**127.0**.**0.1:7059**/**protolator**/**decode**/**common**.**Block **>** **genesis\_block.json**

This is the new org definition that we need to update to the channel genesis block.

Sample genesis block for 1 org (decoded) :



1. Then to add an organization, simply follow the standard configtxlator reconfiguration flow as outlined in

<http://hyperledger-fabric.readthedocs.io/en/latest/configtxlator.html#reconfiguration-example> , but use the JSON you saved to define the new org. Follow [*UpdateChannelIIT.java*](https://github.com/hyperledger/fabric-sdk-java/blob/master/src/test/java/org/hyperledger/fabric/sdkintegration/UpdateChannelIT.java)for Java code. The channel tx file and orderer is created with 2 orgs – Elemica and D&B. GTN is to be added to it.

Process : The code explained below is Test code with Java v1.0.0 and for purpose of giving an idea about the process, for latest code refer –[*UpdateChannelIIT.java*](https://github.com/hyperledger/fabric-sdk-java/blob/master/src/test/java/org/hyperledger/fabric/sdkintegration/UpdateChannelIT.java)in java sdk v1.0.1

* **Get genesis block of channel** –

@Path("/getConfig")

@POST

**public** **void** getConfigBlock() **throws** IOException{

**byte**[] block = hlconnection.getBlock().toByteArray();

decode("test",block,"common.Block",**true**);

}

Where getBlock() returns a first configuration block. In v1.01 of Java sdk , we can get configuration bytes directly by

final byte[] channelConfigurationBytes = fooChannel.getChannelConfigurationBytes();

* **Decode the channel block –**

To decode a Block , you have to submit a POST request to configtxlator and send genesis block as data. The function below creates a *filename.json* which has **config** section of block.

**private** String decode(String fileName,**byte**[] contentToBeDecoded, String protoNameOfMessage , **boolean** getConfigFromJson){

String url = "http://127.0.0.1:7059/protolator/decode/"+protoNameOfMessage;

JsonNode node = **null**;

**try** {

HttpURLConnection httpcon = (HttpURLConnection) ((**new** URL(url).openConnection()));

httpcon.setDoOutput(**true**);

httpcon.setRequestProperty("Content-Type", "application/json");

httpcon.setRequestProperty("Accept", "application/json");

httpcon.setRequestMethod("POST");

httpcon.connect();

OutputStream os = httpcon.getOutputStream();

os.write(contentToBeDecoded);

os.close();

**int** responseCode = httpcon.getResponseCode();

System.***out***.println("\nSending 'POST' request to URL : " + url);

System.***out***.println("Response Code : " + responseCode);

ObjectMapper mapper = **new** ObjectMapper();

InputStream inputStream=httpcon.getInputStream();

node = mapper.readTree(inputStream);

**try** (FileWriter file = **new** FileWriter("scripts\\"+fileName+".json")) {

**if**(getConfigFromJson){

file.write(node.get("data").get("data").get(0).get("payload").get("data").get("config").toString());

}

**else**

{

file.write(node.toString());

}

System.***out***.println("Successfully Copied JSON Object to File...");

**if** (httpcon.getResponseCode() == HttpURLConnection.***HTTP\_OK***){

System.***out***.println("OK");

}

**else**

{

System.***out***.println("NOT SO OK");

}

}

}

**catch** (Exception e)

{

System.***out***.println(e.getMessage());

}

**return** node.toString();

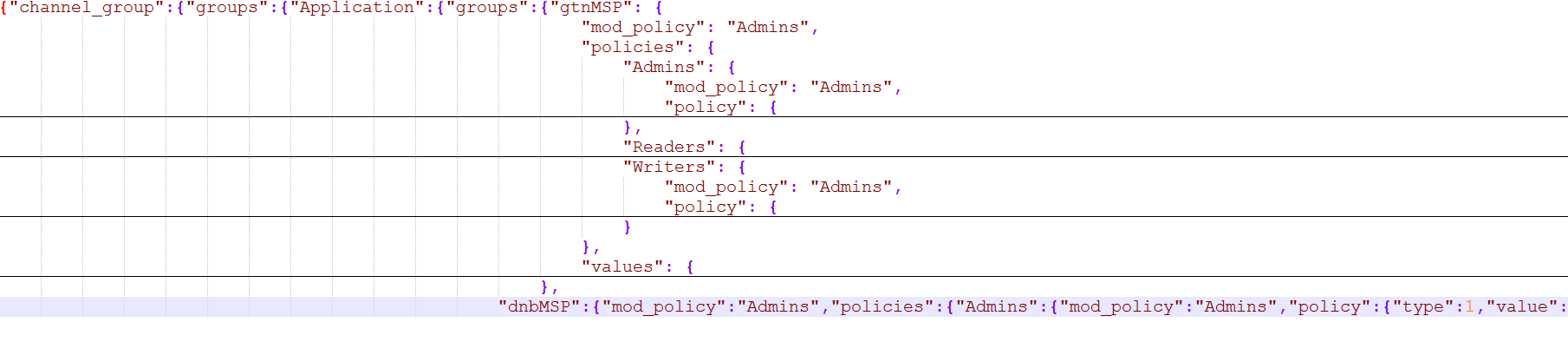
}

* **Edit Decoded Json**

The decoded JSON (*filename.json*) looks similar to what we created of single org. It contains information in similar structure but of multiple orgs. Here is the sample decoded genesis block of 2 orgs (extracted config section) :

{"channel\_group":{"groups":{"Application":{"groups":{"dnbMSP":{"mod\_policy":"Admins","policies":{"Admins":{"mod\_policy":"Admins","policy":{"type":1,"value":{"……………………………,"elemicaMSP":{"mod\_policy":"Admins","policies":{"Admins":{"mod\_policy":"Admins","……………}}}}

Open it in editor and add configuration for gtnMSP as obtained in decoded JSON of genesis block of 1 org. :



Save it as updated\_*filename.json*.

* **Encode and UpdateConfig**

Re-encode both the original config, and the updated config into proto by sending a POST request to configtxlator with filename.json and updated\_filename.json as data. Get byte array of the encoded original and updated config and send them to *configtxlator* service to compute the ***config update*** which transitions between the two.

At this point, the computed config update is now prepared. Then we decode ConfigUpdate and wrap it with an enveloper and  encode it back into the proto form of a full fledged config transaction.

@Path("/encodeConfig")

@POST

**public** **void** setConfigBlock() **throws** IOException{

**try** {

**byte**[]original=encode("scripts\\test","common.Config");

**byte**[]updated=encode("scripts\\updated\_test","common.Config");

updateConfigs(original, updated);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**private** **byte**[] encode(String fileName,String protoNameOfMessage) **throws** IOException{

String url = "http://127.0.0.1:7059/protolator/encode/"+ protoNameOfMessage;

HttpURLConnection httpcon = (HttpURLConnection) ((**new** URL(url).openConnection()));

httpcon.setDoOutput(**true**);

httpcon.setRequestProperty("Content-Type", "application/json");

httpcon.setRequestProperty("Accept", "application/json");

httpcon.setRequestMethod("POST");

httpcon.connect();

**byte**[] content = (Files.*readAllBytes*(Paths.*get*(fileName+".json")));

OutputStream os = httpcon.getOutputStream();

os.write(content);

os.close();

**int** responseCode = httpcon.getResponseCode();

System.***out***.println("\nSending 'POST' request to URL : " + url);

System.***out***.println("Response Code : " + responseCode);

InputStream is=httpcon.getInputStream();

**byte**[] inputStreamInBytes=IOUtils.*toByteArray*(is);

is.close();

**return** inputStreamInBytes;

}

**private** **void** updateConfigs(**byte**[]original,**byte**[]updated) **throws** ClientProtocolException, IOException {

ByteArrayBody originalBody = **new** ByteArrayBody(original,"myfile.txt");

ByteArrayBody updatedBody = **new** ByteArrayBody(updated,"test2.txt");

HttpClient client = HttpClientBuilder.*create*().build();

HttpPost post = **new** HttpPost("http://127.0.0.1:7059/configtxlator/compute/update-from-configs");

MultipartEntityBuilder builder = MultipartEntityBuilder.*create*();

builder.setMode(HttpMultipartMode.***BROWSER\_COMPATIBLE***);

builder.addPart("original", originalBody);

builder.addPart("updated", updatedBody);

HttpEntity entity = builder.build();

post.setEntity(entity);

HttpResponse response = client.execute(post);

InputStream is = response.getEntity().getContent();

**byte**[] content=IOUtils.*toByteArray*(is);

String node=decode("config\_update",content,"common.ConfigUpdate",**false**);

wrapJsonInEnvelope(node,"twoOrgs");

encode("scripts\\config\_update\_as\_envelope","common.Envelope");

}

**private** **void** wrapJsonInEnvelope(String json,String channelID){

String envelopeJson="{\"payload\":{\"header\":{\"channel\_header\":{\"channel\_id\":\""+channelID+"\", \"type\":2}},\"data\":{\"config\_update\":"+json+"}}}";

**try** (FileWriter file = **new** FileWriter("scripts\\config\_update\_as\_envelope.json")) {

file.write(envelopeJson);

} **catch** (IOException e) {

System.***out***.println(e.getMessage());

}

}

* **Submit the config update transaction to ordering to perform a config update.**

This part is not tested. Refer –[*UpdateChannelIIT.java*](https://github.com/hyperledger/fabric-sdk-java/blob/master/src/test/java/org/hyperledger/fabric/sdkintegration/UpdateChannelIT.java)for its implementation.