API REST blockchain KALIMA

I. Introduction

In this tutorial, Kalima REST API is presented in order to communicate with the Kalima blockchain. This API give the possibility to use REST calls to allow a user to easily interface his web application (php, nodejs, vuejs, ...) with the Kalima blockchain.

II. Kalima System Private Docker Registry

Kalima's REST API is provided as a docker image which is placed in a private docker registry specific to Kalima Systems. This registry is https secured and requires authentication in order to retrieve the image. The link used to access this private docker registry is: https://docker.registry.kalimadb.com

The following command is used to authenticate with Kalima's private docker registry:

sudo docker login docker.registry.kalimadb.com

In order to have the login ID and password, kindly access the repository that we shared with you on nextcloud.

Once the authentication is done with the private docker registry, you must retrieve the docker image using the following command:

sudo docker pull docker.registry.kalimadb.com/kalima-rest-api:latest

In order to verify that the image is successfully pulled, the following command lists the docker images on your machine:

sudo docker images

III. REST API usage

Before launching the docker image, you need to create a persistent volume. The command to use to create a persistent volume is:

sudo docker volume create volume-name

To check the creation of the volume and its location on your machine, you can use the following command:

sudo docker inspect volume-name

The docker image is now present on your machine and the volume is created. In order to launch the image you can use the following command:

sudo docker run --publish 9090:9090 --detach --name kalima-rest-api --mount source=volume-name,target=/home/rcs/jit --env SERIAL_ID=testDockerImage --env NOTARIES_LIST="167.86.124.188:7070,62.171.130.233:7070,62.171.131.157:7070,144.91.108.243:7070" docker.registry.kalimadb.com/kalima-rest-api:latest

You can find below an explication of this command :

- **--publish** : this option instructs Docker to forward incoming traffic on host port 9090 to container port 9090. Containers have their own private set of ports, so if you want to reach one from the network, you have to forward traffic. Otherwise, firewall rules will prevent all network traffic from reaching your container by default. The REST API is listening on port 9090 in the container so this value cannot be changed. On the other hand, the port for incoming traffic (port used by your machine) can be customized by the user.
- **--detach** : asks Docker to run this container in the background.
- **--name** : specifies a name which you can refer to your container.
- **--mount** : Use a persistent volume whose name is **volume-name**. This volume will be used by **/home/rcs/jit** from the docker image.

- **--env** : This option allows you to pass environment variables to the container that launches the image. Two variables must be passed to the container: "NOTARIES_LIST" passes the list of notary nodes of the Kalima blockchain and "SERIAL_ID" passes the id which give permission to receive data from the blockchain Kalima.

In order to verify that the container is launched and created, the following command can be used:

sudo docker ps

In order to look at the logs of the container, we can use the following command:

sudo docker logs kalima-rest-api -f

Ⅳ. Communication with Kalima blockchain

In this section, the different requests used to communicate with the REST API of the Kalima blockchain are presented. For each request, the expected response is explained as well as an example of the request with the curl command.

1. Get List of authorized cachePaths from the blockchain Kalima.
    a. Request : **/?action=getTableInfos**
    b. Response : **[“cachePath1”,”cachePath2”, “cachePath3”, …]**
    c. Remarks :
        i. The response is an array of string that contains the list of cachePaths separated with a comma.
        ii. The user only receives the list of cachePaths to which he is authorized according to the serialID used when launching the container.
    d. Example : **curl 'http://localhost:9090/?action=getTableInfos'**

2. Get MemCache of a cachePath

   a. Request :

   /?action=PAGE&cachePath=xx&maxSeq=yy&nbMessages=zz&lastCli ckDirection=1&head=1&responseType=kk

   b. Request Explanation :

   - cachePath : name of the cachePath.
   - maxSeq : from which sequence the result should start. Note that if maxSeq = - 1, the result will be the first page of the memCache. And if maxSeq = -2, the result will be the last page of the memCache.
   - nbMessages : maximum number of messages the user wants to receive.
   - head : If the value is 1 then the descending order of the memcache is used. If the value is 0 then the ascending order of the memcache is used.
   - lastClickDirection : If the value is 1 then the received result will be received from the highest sequence to the lowest sequence. If the value is 0 then the received result will be received from the lowest sequence to the highest sequence.
   - responseType : the data format chosen for a cachePath. It can be a json, a string or a byteArray.

   c. Response : **[["key1","sequence1","body1"],["key2","sequence2","body2"],...]**

   d. Remarks :

   - Each record has a unique key, a sequence and a body.
   - The "body" field is Base64 encoded in order to support various types of data: json, string and byteArray.
   - The response of this query is a list of records. Each record contains a string representing the key, a string representing the sequence and a base64 string representing the body.

   e. Example :

   curl 'http://localhost:9090/?action=PAGE&cachePath=/sites&maxSeq=-1&nbMessages=20&lastClickDirection=1&head=1&responseType=json

3. Get a specific record from a cachePath :
    a. Request : /**?action=getByKey&cachePath=xx&key=yy**
    b. Request Explanation :
        - cachePath : name of the cachePath.
        - key : Key of the record that we need to get.
    c. Response : **["key","sequence","body"]**
    d. Remarks :
        - This Request is used to retrieve a specific record from the cachePath identified by its key.
        - The « body » field is base64 encoded.
    e. Example :

       curl

       'http://localhost:9090/?action=getByKey&cachePath=/test/string&key=key1'

4. Detect a change (Add, edit,delete) on a specific cachePath :
    a. Request : **/?action=FIL&cachePath=xx&minKey=yy&maxKey=zz**
    b. Request explanation :
        - cachePath : name of the cachePath.
        - minKey : minimum sequence value
        - maxKey : maximum sequence value
    c. Response: **{"kvSize":xx,"sign":yy}**
    d. Remarks :
        - This Request allows the user to know if a change has taken place on a cachePath.
        - The user can choose to detect a change between two keys in a memCache
        - If the "kvSize" attribute changes => an addition or deletion of records has taken place in the cachePath. Then the "kvSize" allows to know the number of records appearing in a memCache.

- If the "sign" attribute changes => a record is edited or a record has been added / deleted between the two records with the key "minSeq" and "MaxSeq" specified in the request sent.
- If minSeq = -1 and maxSeq = -1 => check the change for the entire cachePath.

e. Example :

curl 'http://localhost:9090/?action=FIL&cachePath=/sites&minKey=-1&maxKey=-1'

5. Add a record to a cachePath :
   a. Request :

   /?action=addTarget&cacheId=xx&body=yy&key=zz&ttl=kk&responseType=aa

   b. Request explanation :
   - cachePath : name of the cachePath.
   - body : value of the record that we need to add.
   - key : Key of the record that we need to add.
   - ttl : time to live in the memcache.
   - responseType : Type of the body that we are adding to the record. It can be a json, a string or a byteArray.

   c. Response: If the return is "[\"NOK\"]" => error while adding record. If not, the record is added to the cachePath.

   d. Remarks :
   - The body must be encoded in Base64. This type of encoding is used to be able to transport the various types of data supported. To test Base64 encoding and decoding, here are some examples: https://www.base64decode.org/ https://cryptii.com/pipes/base64-to-hex

- If the value of the ttl field is -1 => the record persists and will not be deleted after a certain time. If the value is > 0 then the record will be erased after "X" seconds. ("X" is the value specified in the query).
- In order to add a record, it is a POST request.
- In order to edit a record, all you have to do is use this same query and specify the key of the record you want to modify.
- If the body is of type json, the fields of the json can have the following values:
  - boolean => **"nomChamps":true** ou **"nomChamps":false**
  - String => **"nomChamps":"helloooo"**
  - Float => **"nomChamps":9.5**
  - Integer => **"nomChamps":10**
  - HashMap : **"nomChamps":{"key1":"val1","key2":"val2",…}**
  - Array : **"nomChamps":["val1","val2","val3"]**
  - Date : **"nomChamps":"Oct 8, 2019 5:24:10 PM"**

e. Examples :

Add the value “Hello It is working” in a cachePath of type string:

curl --data "/?action=addTarget" --data "cachePath=/test/string" --data "body=SGVsbG8gSXQgaXMgd29ya2luZw==" --data "key=testDocker" --data "ttl=-1" --data "responseType=string" http://localhost:9090/

Add the value “00010203” in a cachePath of type byteArray:

curl --data "/?action=addTarget" --data "cachePath=/test/bytearray" --data "body=AAECAw==" --data "key=testDocker" --data "ttl=-1" --data "responseType=byteArray" http://localhost:9090/

Add a json in a cachePath of type json:

curl --data "/?action=addTarget" --data "cachePath=/test/json" --data "body=eyJrZXkiOiJ0ZXN0RG9ja2VyIiwidHRsIjotMSwiZmllbGQxIjoidHJ1ZSIsImZpZWxkMiI6InRlc3REb2NrZXIiLCJmaWVsZDMiOiIxMDAuMCIsImZpZWxkNCI6IjIwIiwiZmllbGQ1IjpbeyJrZXkiOiJrZXkxIiwidmFsIjoidmFsMSJ9LHsia2V5Ijoia2V5MiIsInZhbCI6InZhbDIifV0sImZpZWxkNiI6WyJ2YWwxIiwidmFsMiIsInZhbDMiXSwiZmllbGQ3IjoiTm92IDA1LCAyMDIwIDEyOjE3OjAwIEFNI

n0=" --data "key=testDocker" --data "ttl=-1" --data "responseType=json" http://localhost:9090/

The value of the json that we sent is :

{"key":"testDocker","ttl":-1,"field1":"true","field2":"testDocker","field3":"100.0","field4":"20","field5":[{"key":"key1","val":"val1"},{"key":"key2","val":"val2"}],"field6":["val1","val2","val3"],"field7":"Nov 05, 2020 12:17:00 AM"

6. Delete a record of a memCache :
    a. Request : **/?action=deleteTarget&cachePath=xx&key=yy**
    b. Request explanation :
        - cachePath : name of the cachePath.
        - key : Key of the record that we want to delete.
    c. Response : If the return is "[\"NOK\"]" => error while deleting the record. Otherwise, the record is erased.
    d. Remarks :
        - If the key you want to delete does not exist => the result "NOK" is received.
    e. Example :

    curl 'http://localhost:9090/?action=deleteTarget&cachePath=/test/string&key=testDocker'

7. Get transactions history of a cachePath : Sort by sequence
    a. Request :

    /?nodeAction=betweenSequences&cachePath=xx&fromSequence=yy&toSequence=zz&userId=kk

b. Request explanation :

- cachePath : name of the cachePath.
- fromSequence : minimum sequence value.
- toSequence : maximum sequence value.
- userId : Id of the user that do the search.

c. Response:

[["date1","time1","sequence1","previousSequence1","key1","body1","hash1"],["date2","time2","sequence2","previousSequence2","key2","body2","hash2"],…]

d. Remarks :

- Each record in history has a specific date to identify when the operation took place.
- Each record has a sequence number, a key and a body. The body is Base64 encoded.
- Multiple users can search simultaneously in the history. In order to differentiate between the search for each user, the "userId" field in the query is used.
- « fromSequence » value should be less then the « toSequence » value.
- By default, this query returns the first 20 records of the result. In order to view the rest of the result, the queries “firstSearch”, “lastSearch”, “nextSearch” and “previousSearch” are used. These queries are explained later in this document.

e. Example :

curl 'http://localhost:9000/?nodeAction=betweenSequences&cachePath=/test/string&fromSequence=0&toSequence=19&userId=antony'

8. Get transactions history of a cachePath : Sort by date

a. Request :

/?nodeAction=betweenDates&cachePath=xx&fromDate=yy&toDate=zz&userId=kk

b. Request explanation :

- cachePath : name of the cachePath
- fromDate : starting date
- toDate : end date
- userId : Id of the user that do the search.

c. Response:

[["date1","time1","sequence1","previousSequence1","key1","body1","hash1"],["date2","time2","sequence2","previousSequence2","key2","body2","hash2"],…]

d. Remarks :

- Each record in history has a specific date to identify when the operation took place.
- Each record has a sequence number, a key and a body. The body is Base64 encoded.
- Multiple users can search simultaneously in the history. In order to differentiate between the search for each user, the "userId" field in the query is used.
- « fromDate » value should be less then the « toDate » value.
- By default, this query returns the first 20 records of the result. In order to view the rest of the result, the queries “firstSearch”, “lastSearch”, “nextSearch” and “previousSearch” are used. These queries are explained later in this document.
- The format of the date to be entered in the “fromDate” and “toDate” fields is **AAMMDD_HHmmss**. Example: 200708_140000 => July 8, 2020, 2 pm

e. Exemple :

curl 'http://localhost:9090/?nodeAction=betweenDates&cachePath=/test/string&fromDate=201104_120000&toDate=201104_230000&userId=antony'

9. Get transactions history of a cachePath (type json) : Sort by one or more criteria

   a. Request :

   /?nodeAction=criteria&cachePath=xx&i1=v1&i2=v2&…&userId=kalima

   b. Request explanation :

   - cachePath : name of the cachePath
   - i1 : field name of the first criteria.
   - v1 : value of the first criteria.
   - i2 : field name of the second criteria.
   - v2 : value of the first criteria.
   - userId : Id of the user that do the search.

   c. Response:

   [["date1","time1","sequence1","previousSequence1","key1","body1","hash1"],["date2","time2","sequence2","previousSequence2","key2","body2","hash2"],…]

   d. Remarks :

   - This request is only used for cachePaths of type json.
   - Each record in history has a specific date to identify when the operation took place.
   - Each record has a sequence number, a key and a body. The body is Base64 encoded.
   - Multiple users can search simultaneously in the history. In order to differentiate between the search for each user, the "userId" field in the query is used.
   - By default, this query returns the first 20 records of the result. In order to view the rest of the result, the queries “firstSearch”, “lastSearch”, “nextSearch” and “previousSearch” are used. These queries are explained later in this document.
   - You can add as many criteria as you want to this query. Each criteria must correspond to a field of the json object used in the cachePath.
   - At least there must be one criterion for this query.

- The value of the criteria supports regular expressions to define a search pattern. For example, return all values of a criteria starting with the letter A: **A.***

e. Example :

curl 'http://localhost:9090/?nodeAction=criteria&cachePath=/test/json&field2=test.*&userId=antony

10. Get transactions history of a cachePath (type string ou byteArray) : Regex

a. Request : **/?nodeAction=regex&cachePath=xx&type=yy&regex=zz&userId=dd**

b. Request explanation :

- cachePath : name of the cachePath.
- type : **string** or **bytearray.**
- regex : regular expression.
- userId : Id of the user that do the search.

c. Response:

[["date1","time1","sequence1","previousSequence1","key1","body1","hash1"],["date2","time2","sequence2","previousSequence2","key2","body2","hash2"],…]

d. Remarks :

- This request is only used for cachePaths of type **string** or **bytearray**.
- Each record in history has a specific date to identify when the operation took place.
- Each record has a sequence number, a key and a body. The body is Base64 encoded.
- Multiple users can search simultaneously in the history. In order to differentiate between the search for each user, the "userId" field in the query is used.
- By default, this query returns the first 20 records of the result. In order to view the rest of the result, the queries “firstSearch”, “lastSearch”, “nextSearch” and “previousSearch” are used. These queries are explained later in this document.

- The regex value supports regular expressions to define a search pattern. For example :
  - Return all values of a string starting with the letter A: **A.***
  - Return the list of byteArray records which ends with 0001: **.*0001**

e. Example :

Search in cachePath of type byteArray for records ending with the byte **0x10**:

curl 'http://localhost:9000/?nodeAction=regex&cachePath=/test/bytearray&type=bytearray&regex=.*10&userId=antony'

Search in cachePath of type string for records that contain the value **Hello**:

curl 'http://localhost:9000/?nodeAction=regex&cachePath=/test/string&type=string &regex=.*hello.*&userId=antony'

11. Get transactions history of a cachePath : nextSearch

a. Request : **/?nodeAction=nextSearch&cachePath=xx&size=yy&userId=zz**

b. Request explanation :

- cachePath : name of the cachePath.
- size: number of records you want to receive in addition to the result already initiated (sort by sequence, sort by date, sort by regex or sort by criteria).
- userId : Id of the user that do the search.

c. Response:

[["date1","time1","sequence1","previousSequence1","key1","body1","hash1"],["date2","time2","sequence2","previousSequence2","key2","body2","hash2"],…]

d. Remarks :

- This query can be called only after doing a search in the history: sort by sequence, sort by date, sort by regex or sort by criteria.
- If we reach the end of the result, this query returns the last records again.

e. Example :

curl 'http://localhost:9000/?nodeAction=nextSearch&cachePath=/test/string&size=20&userId=antony'

12. Get transactions history of a cachePath : previousSearch

a. Request : **/?nodeAction=previousSearch&cachePath=xx&size=yy&userId=zz**

b. Request explanation :

- cachePath : name of the cachePath.
- size : number of records you want to receive in addition to the result already initiated (sort by sequence, sort by date, sort by regex or sort by criteria).
- userId : Id of the user that do the search.

c. Response:

[["date1","time1","sequence1","previousSequence1","key1","body1","hash1"],["date2","time2","sequence2","previousSequence2","key2","body2","hash2"],…]

d. Remarks :

- This query can be called only after doing a search in the history: sort by sequence, sort by date, sort by regex or sort by criteria.
- If we get to the start of the result, this query returns the first **yy** records again.

e. Example :

curl 'http://localhost:9000/?nodeAction=previousSearch&cachePath=/test/string&size=20&userId=antony'

13. Get transactions history of a cachePath: firstSearch

a. Request : **/?nodeAction=firstSearch&cachePath=xx&size=yy&userId=zz**

b. Request explanation :

- cachePath : name of the cachePath.
- size : number of records you want to receive in addition to the result already initiated (sort by sequence, sort by date, sort by regex or sort by criteria).
- userId : Id of the user that do the search.

c. Response :

[["date1","time1","sequence1","previousSequence1","key1","body1","hash1"],["date2","time2","sequence2","previousSequence2","key2","body2","hash2"],…]

d. Remarks :

- This query can be called only after doing a search in the history: sort by sequence, sort by date, sort by regex or sort by criteria.
- This query returns the first **yy** records of the result.

e. Example :

curl 'http://localhost:9000/?nodeAction=firstSearch&cachePath=/test/string&size=20&userId=antony'

14. Get transactions history of a cachePath : lastSearch

a. Request : **/?nodeAction=lastSearch&cachePath=xx&size=yy&userId=zz**

b. Request explanation :

- cachePath : name of the cachePath.
- size : number of records you want to receive in addition to the result already initiated (sort by sequence, sort by date, sort by regex or sort by criteria).
- userId : Id of the user that do the search.

c. Response:

[["date1","time1","sequence1","previousSequence1","key1","body1","hash1"],["date2","time2","sequence2","previousSequence2","key2","body2","hash2"],…]

d. Remarks :

- This query can be called only after doing a search in the history: sort by sequence, sort by date, sort by regex or sort by criteria.

e. Example :

curl 'http://localhost:9000/?nodeAction=lastSearch&cachePath=/test/string&size=20&userId=antony'

15. Checking the hash of one or more cachePaths :

a. Request : **/?nodeAction=checkHashs&cacheList=xx**

b. Request explanation :

- cacheList : list of cachePaths separated with a comma.

c. Response: **[{"cachePath1":value1,"cachePath2":value2,…}]**

d. Remarks :

- The value received for each cachePath can be either "true" or "false".

e. Example :

curl 'http://localhost:9000/?nodeAction=checkHashs&cacheList=/sites'

Response : **[{"/sites":true}]**

curl 'http://localhost:9000/?nodeAction=checkHashs&cacheList=/sites,/addresses'

Response : **[{"/sites":true,"addresses":true}]**