REST API blockchain KALIMA

# Introduction

In this tutorial we present the procedure to use to communicate with the Kalima blockchain with the help of a REST API. This last one allows us to use some methods REST to simplify the navigation of a user in his web application (PHP, Node.js, Vue.js, etc…) with the blockchain Kalima.

# REST API Docker

To download the rest api docker image:

**docker pull kalimasystems/kalima-rest-api:latest**

# Launch of the API Node.js

Before the launch of the docker image, we need to create a persistent volume.

The command that we need to create a volume persistent is:

docker volume create <volume-name>

To verify the creation of the volume and its location on your PC, we can use this command:

docker inspect inspect <volume-name>

The docker image is now in your PC and the volume is created. To launch the image, you can use this next command:

docker run --publish 8080:8080 --detach --name kalima-rest-api --mount source=<volume-name>,target=/home/rcs/jit --env PORT=8080 --env SERIAL\_ID=<YouSerialId> --env PRIVACHAIN=<privachain-name> docker.registry.kalimadb.com/kalima-rest-api:latest

Possibility of working in https with mode = 1 and passing the crt and key files. To do this, put the .crt and the .key in a certs folder, and move this folder to the docker volume, in the rest folder. To find out where the docker volume is: docker inspect volume-name and see "Mountpoint". Then launch the docker specifying the paths of the certificates:

docker run --publish 8080:8080 --detach --name kalima-rest-api --mount source=<volume-name>,target=/home/rcs/jit --env SERIAL\_ID=<YouSerialId> --env PRIVACHAIN=<privachain-name> --env PORT=8080 --env MODE=1 --env CERTIFICATE\_CRT=/home/rcs/jit/rest/certs/mycert.crt --env CERTIFICATE\_KEY=/home/rcs/jit/rest/certs/mycert.key docker.registry.kalimadb.com/kalima-rest-api:latest

Explication of the command:

**. --publish:** this option asks Docker to transfer the traffic an entering port of the host towards another port of the container. All containers have their own privates ports, so if you want to reach one from the network, you must transfer the traffic with this option. Or else, security system rules will prevent the traffic network from reaching your container, as a security posture by default.

**--detach:** ask docker to execute this container in the background

**. --name:** specifies a name for this container.

**. --mount:** use a persistent volume**.** This volume will be used by /home/rcs/jit of docker image.

**. --env:** this optionlets us pass environment variables to the container who launches the image. Up to 6 variables should be passed to the container:

* PORT 🡺 Listening port of the API REST. Use the same port that you use in publish (if you set –publish 7070:80, so set PORT=80)
* SERIAL\_ID 🡺 Your serial id (for authorize the API REST Node on Kalima Blockchain)
* PRIVACHAIN 🡺 The name of the privachain you want to connect to
* MODE 🡺 Set to 1 if you want to use https. 0 (or not set) if you want to use http.
* CERTIFICATE\_CRT and CERTIFICATE\_KEY 🡺 The path of certificate files (only with https)

To verify that the container is launch et created, the next command can be used:

docker ps

To look at the logs of the container, we can use this next command:

docker logs kalima-rest-api –f

1. Communication with the Kalima blockchain

The authorization header must be included to authenticate with the REST API, you need a token (Please contact Kalima to get one) to be able to authenticate like this:

curl -H "Authorization: aea4004084bb9f383803b0b51537b2f0"

**The authorization part coming soon.**

After every request, there are 3 responses:

. 200: success of the request (with a body or none)

. 405: request not found

. 407: not authorized (coming soon)

. 406: error in the request (with a body)

Example of 406 error:

{“error”:”address is null”}

# All GET commands

## GET a transaction

*Get a specific transaction by key and address.*

**Request:**

curl '<http://localhost:8080/transaction?address=/sites&key=StAubin>'

**Params:**

* address:indicate the address of the transaction you want
* keys: the key of the transaction you want inside this address

**Response:**

{

"address":"/sites",

"key":"StAubin",

"globalSequence":2402,

"sequence":8,

"previous":7

"body":"eyJpZCI6IlN0QXViaW4iLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEJ5IjoiIiwidXBkYXRlZEJ5IjoiIiwibmFtZSI6IlBvc3RlIHNvdXJjZSBkZSBTYWludCBBdWJpbiIsIm5vVmVyc2lvbiI6IiIsImRlc2NyaXB0aW9uIjoiIiwicHJvcHMiOnt9LCJ0eXBlIjoiIiwic3RhdHVzIjoiIiwib3JnYW5pc2F0aW9uSWQiOiIiLCJ0aHVtYm5haWxVcmwiOiJodHRwOi8vMzcuNTkuMTA4LjE2OjQ1ODcvU3RBdWJpbi8iLCJncmFwaGljc0lkIjoiIiwiem9uZXNJZHMiOlsiR3JvdW5kX0Zsb29yIiwiVW5kZXJmbG9vcl8xIl0sImxvY2F0aW9uSWQiOiJTdEF1YmluIn0=",

"hash":"elXyAtw01KH9EZ+wYXJup/UpP8Ea2m/0HOj3xfnLusw=",

"props":"ZGF0ZT0yMjEwMDUKaXA9LzkwLjQ4LjEwNi4xNjI6NjQzNDEKdGltZT0xNTU0NDYKdHRsPS0xCg=="

}

**Response fields:**

* address: address of the transaction
* key: key of the transaction
* globalSequence: the unique sequence of the transaction in the blockchain
* sequence: the unique sequence of the transaction within this address
* previous: sequence of the previous transaction within this address
* body: Payload of the transaction (base 64 encoded)
* hash: Hash of the transaction

## GET addresses list

*Get the list of addresses on which the is authorized*

**Request:**

curl '[http://localhost:8080/cache/list](http://localhost:9090/cache/list)'

**Response:**

"addresses": [

"/StAubin/Plan/layers",

"/Aubervilliers/Schema/layers",

"/Grigny/Plan/layers",

"/Aubervilliers/Plan/drawings",

"/Montjay/users",

"/Saclay/devices",

"/Essonnes/Plan/drawings",

"/Montjay/zones",

"/Morigny/users",

"/Morigny/zones",

"/Essonnes/Schema/layers",

"/Juine/Plan/drawings",

Etc...

]

}

**Response fields:**

* addresses: array that list all addresses

## GET keys addresses

*Get the keys of all transactions within a cache (cache is current values for an address)*

**Request:**

curl “http://localhost:8080/cache/keys?address=/sites”

**Params:**

* address:the address

**Response:**

{

"keys": [

"Aubervilliers",

"Epinay",

"Essonnes",

"Grigny",

"Juine",

"Montjay",

"Morigny",

"Saclay",

"StAubin"

]

}

**Response fields:**

keys: array that list all keys within the cache of this address

## GET Cache Info

*Get some information about a cache*

**Request:**

curl “[http://localhost:8080/cache?address=/sites](http://localhost:9090/cache?address=/sites&key=StAubin)”

**Params:**

* address:The address of the cache whose info you want

**Response:**

{

"size":9,

"sequenceMax":8,

"sequenceMin":0,

"address":"/sites",

}

**Response fileds:**

* size: number of transactions in this cache
* sequenceMax: higher sequence in this cache
* sequenceMin: smaller sequence in this cache
* address: address corresponding to this cache

## GET after sequence

*Get n transactions that happens after another transaction*

**Request:**

curl “http://localhost:8080/cache/after?address=/Essonnes/devices&n=2&seq=2”

**Params:**

* address:The address of your search
* n: the number of transactions you want
* seq: The start sequence. It means you will get in response n transactions that fallow this sequence if any

**Response:**

{

"messages":[

{

"date":"221005",

"time":"155444",

"address":"/Essonnes/devices",

"key":"000425191801DBA8",

"globalSequence":1521,

"sequence":3,

"previous":2,

"body":"",

"hash":"Z0mQiX8qxAw0M6AauaaMLLs1GgRs4w4HsS5px+47iNs=",

"props":"ZGF0ZT0yMjEwMDUKaXA9LzkwLjQ4LjEwNi4xNjI6NjQzNDEKdGltZT0xNTU0NDQKdHRsPS0xCg=="

},

{

…

}

]

}

**Response fields:**

A json array of “messages”. For each message there is:

* address: address of the transaction
* key: key of the transaction
* body: Payload of the transaction (base 64 encoded)
* globalSequence: the unique sequence of the transaction in the blockchain
* sequence: the unique sequence of the transaction within this address
* previous: sequence of the previous transaction within this address
* date: transaction date (YYMMdd format)
* time: transaction time (HHmmss format)
* hash: Hash of the transaction

## GET before sequence

*Get n transactions that happens befor another transaction*

**Request:**

curl “http://localhost:8080/cache/after?address=/Essonnes/devices&n=2&seq=2”

**Params:**

* address:The address of your search
* n: the number of transactions you want
* seq: The start sequence. It means you will get in response n transactions that is before this sequence if any

**Response:**

{

"messages":[

{

"date":"221005",

"time":"155444",

"address":"/Essonnes/devices",

"key":"000425191801DBA4",

"globalSequence":1518,

"sequence":0,

"previous":-1,

"body":"",

"hash":"eUwHUbPWJmjSA+pQgETjrIx5w41K+QHfWu+9DhlVK/w=",

"props":"ZGF0ZT0yMjEwMDUKaXA9LzkwLjQ4LjEwNi4xNjI6NjQzNDEKdGltZT0xNTU0NDQKdHRsPS0xCg=="

},

{

…

}

]

}

**Response fields:**

A json array of “messages”. For each message there is:

* address: address of the transaction
* key: key of the transaction
* body: Payload of the transaction (base 64 encoded)
* globalSequence: the unique sequence of the transaction in the blockchain
* sequence: the unique sequence of the transaction within this address
* previous: sequence of the previous transaction within this address
* date: transaction date (YYMMdd format)
* time: transaction time (HHmmss format)
* hash: Hash of the transaction

# All POST command

## POST a transaction

Create a new transaction In the blockchain

**Request:**

curl -d “{"address":"/demo/alarms", "key":"test", "body":"dGVzdA==", "ttl":"-1"}'”-H "Content-Type: application/json" -X POST http://localhost:8080/transaction

**Params:**

* address:The address where you want to put a new transaction
* keys: The key of the transaction (if key already exist, current value in cache will be updated)
* body: The payload of the transaction, can be byte array, string json, or whatever you want (base64 encoded)
* ttl: Time to live. Set a timer for the transaction. It means that if you set ttl to 3, the current value will be deleted in cache after 3 seconds (So there will be your transaction first, and then after 3 seconds another transaction with empty payload). Set to -1 to never delete the current value

**Response:**

http status 200 or error

## Move a transaction

*Send a transaction from an address to another*

**Request:**

curl -d “{"fromAddress":"/demo/alarms", "toAddress":"/demo/temperature", "key":"test"}” -H "Content-Type: application/json" -X POST http://localhost:8080/transaction/move

**Params:**

* fromAddress:to specify the initial address of the transaction
* toAddress: to specify the final address of the transaction

**Response:**

http status 200 or error

# All DELETE commands

## DELETE a transaction

*Remove a current value (send a new transaction with empty payload)*

*If key is null, so you can remove all current values of an address*

**Request:**

curl -X DELETE “http://localhost:8080/transaction?address=/demo/alarms&key=test”

**Params:**

* address:The addressof the current value you want to delete
* keys: The key of the current value you want to delete (Optional: If null delete the entire cache)

**Response:**

http status 200 or error

# All LEDGER commands

With these commands you can retrieve all transactions of the blockchain, not only values in cache.

## User Token

Some of requests to ledger needs a user token. Some of requests returns a limit number of message, even if the params of the request asks for more. Fore example, if we want to retrieve transactions on the address /x between the sequence 10 and the sequence 10010, the request will return only 20 transactions, because return 10000 transaction at once would be too long and too greedy. However, after using such request, it’s possible to navigate in our previous search with requests like nextSearch, previousSearch, firstSearch and lastSearch. The user token is used in that case to find the last transactions sent.

## By Sequence

Retrieve a specific transaction with address and the sequence

**Request:**

curl “[http://localhost:8080/ledger/bySequence?address=/addresses&seq=1](http://localhost:9090/ledger/bySequence?address=/addresses&seq=1)”

**Params:**

* address : Address of the transaction that we want to retrieve
* seq : Sequence of the transaction that we want to retrieve

**Response:**

[{ "address":"/addresses",

"key":"test",

"globalSequence":48,

"sequence":1,

"previous":0,

"body":"dGVzdDI=",

"hash":"mGzuNLtnwfkfW04lXOG7o0TZWpuGY7608RLy8sRRygY=",

"props":"ZGF0ZT0yMjExMDgKaXA9LzE5Mi4xNjguMS4xMjo2MDI2MAp0aW1lPTA3MDA0Mwp0dGw9LTEK",

"date":"221108",

"time":"070043"}]

**Response fields:**

* address: address of the transaction
* key: key of the transaction
* body: Payload of the transaction (base 64 encoded)
* globalSequence: the unique sequence of the transaction in the blockchain
* sequence: the unique sequence of the transaction within this address
* previous: sequence of the previous transaction within this address
* date: transaction date (YYMMdd format)
* time: transaction time (HHmmss format)
* hash: Hash of the transaction

## Between sequence

Retrieves a transaction list between two sequences on a given address

**Request:**

Curl “http://localhost:8080/ledger/betweenSequences?address=/addresses&fromSeq=0&toSeq=254&pageSize=3&userToken=test”

**Params:**

* address: Address of the transaction that we want to retrieve
* fromSeq: sequence of the first transaction you want
* toSeq: Sequence of the last transaction you want
* pageSize: Size of the « page ». The number of returned transactions can’t be higher that this number. Usefull to display all transactions of the blockchain in a pager for example
* userToken : (return to the section userToken)

**Response:**

[

{

"address":"/addresses",

"key":"test",

"globalSequence":47,

"sequence":0,

"previous":-1,

"body":"dGVzdDE=",

"hash":"ermcFDS5yDZiJwQO3qrTcoPeP5ybY6Tb+u6vSEsGPeA=",

"props":"ZGF0ZT0yMjExMDgKaXA9LzE5Mi4xNjguMS4xMjo2MDI2MAp0aW1lPTA3MDA0Mgp0dGw9LTEK",

"date":"221108",

"time":"070042"

},

{

…

},

{

…

}

]

**Reponse fields:**

A json array of messages with fields:

* address: address of the transaction
* key: key of the transaction
* body: Payload of the transaction (base 64 encoded)
* globalSequence: the unique sequence of the transaction in the blockchain
* sequence: the unique sequence of the transaction within this address
* previous: sequence of the previous transaction within this address
* date: transaction date (YYMMdd format)
* time: transaction time (HHmmss format)
* hash: Hash of the transaction

## From last

Retrieves the last n transactions (the most recent transactions) for a specific address

**Request:**

curl 'http://localhost:8080/ledger/fromLast?address=/addresses&n=2'

**Params:**

* address: Address of the transaction that we want to retrieve
* n: to choose a specific number of transactions

**Response:**

[

{

"address":"/addresses",

"key":"test",

"globalSequence":48,

"sequence":1,

"previous":0,

"body":"dGVzdDI=",

"hash":"mGzuNLtnwfkfW04lXOG7o0TZWpuGY7608RLy8sRRygY=",

"props":"ZGF0ZT0yMjExMDgKaXA9LzE5Mi4xNjguMS4xMjo2MDI2MAp0aW1lPTA3MDA0Mwp0dGw9LTEK",

"date":"221108",

"time":"070043"

},

{

…

}

]

**Reponse fields:**

A json array of messages with fields:

* address: address of the transaction
* key: key of the transaction
* body: Payload of the transaction (base 64 encoded)
* globalSequence: the unique sequence of the transaction in the blockchain
* sequence: the unique sequence of the transaction within this address
* previous: sequence of the previous transaction within this address
* date: transaction date (YYMMdd format)
* time: transaction time (HHmmss format)

## Next search

Return the next page of the previous research. For exemple if pageSize = 10 and that we have previously researched with /ledger/betweenSequences transactions at this address /addresses between the sequence 0 and 254 (so we get sequences 0 to 9), this request return transactions whose they sequences go from 10 to 19

**Request:**

curl “http://localhost:8080/ledger/nextSearch?address=/addresses&pageSize=3&userToken=test”

**Params:**

* address: Address of the transaction that we want to retrieve
* pageSize: Size of the « page ». The number of returned transactions can’t be higher that this number
* userToken : (return to the section userToken)

**Response:**

[

{

"address":"/addresses",

"key":"test",

"globalSequence":57,

"sequence":10,

"previous":9,

"body":"dGVzdA==",

"hash":"a/4yLl+7YCkAextkjb3nm+RZy1DMfV/Bq17oWDkkr7k=",

"props":"ZGF0ZT0yMjExMTAKaXA9LzE5Mi4xNjguMS4xMjo2NDgwMQp0aW1lPTAyNDgxMQp0dGw9LTEK",

"date":"221110",

"time":"024811"

},

{

…

},

…

]

**Reponse fields:**

A json array of messages with fields:

* address: address of the transaction
* key: key of the transaction
* body: Payload of the transaction (base 64 encoded)
* globalSequence: the unique sequence of the transaction in the blockchain
* sequence: the unique sequence of the transaction within this address
* previous: sequence of the previous transaction within this address
* date: transaction date (YYMMdd format)
* time: transaction time (HHmmss format)

## Previous search

Return the previous page of the previous research

**Request:**

curl “http://localhost:8080/ledger/previousSearch?address=/addresses&pageSize=3&userToken=test”

**Params:**

* address: Address of the transaction that we want to retrieve
* pageSize: Size of the « page ». The number of returned transactions can’t be higher that this number
* userToken : (return to the section userToken)

**Response:**

[

{

"address":"/addresses",

"key":"test",

"globalSequence":47,

"sequence":0,

"previous":-1,

"body":"dGVzdDE=",

"hash":"ermcFDS5yDZiJwQO3qrTcoPeP5ybY6Tb+u6vSEsGPeA=",

"props":"ZGF0ZT0yMjExMDgKaXA9LzE5Mi4xNjguMS4xMjo2MDI2MAp0aW1lPTA3MDA0Mgp0dGw9LTEK",

"date":"221108",

"time":"070042"

},

{

…

},

…

]

**Reponse fields:**

A json array of messages with fields:

* address: address of the transaction
* key: key of the transaction
* body: Payload of the transaction (base 64 encoded)
* globalSequence: the unique sequence of the transaction in the blockchain
* sequence: the unique sequence of the transaction within this address
* previous: sequence of the previous transaction within this address
* date: transaction date (YYMMdd format)
* time: transaction time (HHmmss format)

## First search

Return the first page of the previous research

**Request:**

curl “http://localhost:8080/ledger/firstSearch?address=/addresses&pageSize=3&userToken=test”

**Params:**

* address: Address of the transaction that we want to retrieve
* pageSize: Size of the « page ». The number of returned transactions can’t be higher that this number
* userToken : (return to the section userToken)

**Response:**

[ {

"address":"/addresses",

"key":"test",

"globalSequence":47,

"sequence":0,

"previous":-1,

"body":"dGVzdDE=",

"hash":"ermcFDS5yDZiJwQO3qrTcoPeP5ybY6Tb+u6vSEsGPeA=",

"props":"ZGF0ZT0yMjExMDgKaXA9LzE5Mi4xNjguMS4xMjo2MDI2MAp0aW1lPTA3MDA0Mgp0dGw9LTEK",

"date":"221108",

"time":"070042"

},

{

…

}, …

]

**Reponse fields:**

A json array of messages with fields:

* address: address of the transaction
* key: key of the transaction
* body: Payload of the transaction (base 64 encoded)
* globalSequence: the unique sequence of the transaction in the blockchain
* sequence: the unique sequence of the transaction within this address
* previous: sequence of the previous transaction within this address
* date: transaction date (YYMMdd format)
* time: transaction time (HHmmss format)

## Last search

Return the last page of the previous research

**Request:**

curl “http://localhost:8080/ledger/lastSearch?address=/addresses&pageSize=3&userToken=test”

**Params:**

* address: Address of the transaction that we want to retrieve
* pageSize: Size of the « page ». The number of returned transactions can’t be higher that this number
* userToken : (return to the section userToken)

**Response:**

[

{

"address":"/addresses",

"key":"test",

"globalSequence":98,

"sequence":51,

"previous":50,

"body":"dGVzdA==",

"hash":"tUAUNOStJVdjxygWVQ9fq9rSSgsgz+3OGRzfFq8EQuM=",

"props":"ZGF0ZT0yMjExMTAKaXA9LzE5Mi4xNjguMS4xMjo2NDgwMQp0aW1lPTAyNDgyMAp0dGw9LTEK",

"date":"221110",

"time":"024820"

},

{

…

},

{

…

}

]

**Reponse fields:**

A json array of messages with fields:

* address: address of the transaction
* key: key of the transaction
* body: Payload of the transaction (base 64 encoded)
* globalSequence: the unique sequence of the transaction in the blockchain
* sequence: the unique sequence of the transaction within this address
* previous: sequence of the previous transaction within this address
* date: transaction date (YYMMdd format)
* time: transaction time (HHmmss format)