API REST blockchain KALIMA

I. Introduction

Dans ce tutoriel, on présente la procédure à utiliser afin de communiquer avec la blockchain Kalima à l'aide d'un API REST. Ce dernier permet d'utiliser des appels REST afin de permettre à un utilisateur d'interfacer facilement son application web (php, nodejs, vuejs, …) avec la blockchain Kalima.

II. Registre Docker Privé de Kalima System

L'API REST de Kalima est fourni sous forme d'une image docker qui est placée dans un private docker registry propre à Kalima Systems. Ce registre est sécurisé en https et nécessite une authentification afin de récupérer l'image. Le lien utilisé pour accéder à ce registre docker privé est : https://docker.registry.kalimadb.com

La commande suivante permet de s'authentifier auprès du registre docker privé de Kalima :
sudo docker login docker.registry.kalimadb.com

Afin d'avoir l'identifiant et le mot de passe de connexion, merci de consulter l'espace commun partagé sur notre outil nextcloud.

Une fois la l'authentification est faite auprès du registre docker privé, il faut récupérer l'image docker en utilisant la commande suivante :
sudo docker pull docker.registry.kalimadb.com/kalima-rest-api:latest

Afin de vérifier que l'image est bien récupérée, la commande suivante permet de lister les images docker sur votre machine :
sudo docker images

III. Lancement de l'API nodejs

Avant de lancer l'image docker, il faut créer un volume persistent.

La commande à utiliser pour créer un volume persistent est :

sudo docker volume create volume-name

Pour vérifier la création du volume et son emplacement sur votre machine, on peut utiliser la commande suivante :

sudo docker inspect volume-name

L'image docker est maintenant présente sur votre machine et le volume est créé. Afin de lancer l'image vous pouvez utiliser la commande suivante :

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

Explication de la commande :

- **--publish** : cette option demande à Docker de transférer le trafic entrant sur le port 9090 de l'hôte vers le port 9090 du conteneur. Les conteneurs ont leur propre ensemble privé de ports, donc si vous voulez en atteindre un depuis le réseau, vous devez y transférer le trafic de cette manière. Sinon, les règles de pare-feu empêcheront tout le trafic réseau d'atteindre votre conteneur, comme posture de sécurité par défaut. L'api REST ecoute sur le port 9090 dans le container donc cette valeur ne peut pas être changée. Par contre le port du trafic entrant (port utilisé par votre machine) peut être personnalisé en fonction de l'utilisateur.
- **--detach** : demande à Docker d'exécuter ce conteneur en tache de fond.
- **--name** : spécifie un nom avec lequel vous pouvez faire référence à votre conteneur.
- **--mount** : Utiliser un volume persistent dont le nom est **volume-name**. Ce volume sera utilisé par **/home/rcs/jit** de l'image docker.
- **--env** : Cette option permet de passer des variables d'environnement au containeur qui lance l'image. Deux variables doivent être passé au containeur : « NOTARIES_LIST » passe la liste des notary nodes de la blockchain Kalima et « SERIAL_ID » passe l'id qui permet d'avoir la permission de recevoir les données de Kalima.

Afin de vérifier que le containeur est bien lancé et créé, la commande suivante peut être utilisé :

sudo docker ps

Afin de regarder les logs du containeur, on peut utiliser la commande suivante :

sudo docker logs kalima-rest-api -f

IV. Communication avec la blockchain Kalima

Dans cette partie on présente les différents requêtes utilisés pour communiquer avec l'API REST de la blockchain Kalima. Pour chaque requête on explique la réponse attendue ainsi qu'un exemple de la requête avec la commande curl.

1. Récupérer la liste des cachePaths autorisé sur la blockchain.
    a. Requête : **/?action=getTableInfos**
    b. Réponse : **["cachePath1","cachePath2", "cachePath3", ...]**
    c. Remarques :
        i. La repose est un tableau de string qui contient la liste des cachePaths séparés avec une virgule.
        ii. L'utilisateur reçoit uniquement la liste des cachePaths auquel il est autorisé en fonction du serialID utilisé lors du lancement du containeur.
    d. Exemple : **curl 'http://localhost:9090/?action=getTableInfos'**

2. Récupérer le memCache d'un cachePath
    a. Requête : **/?action=PAGE&cachePath=xx&maxSeq=yy&nbMessages=zz&lastClickDirection=1&head=1&responseType=kk**
    b. Explication de la requête :
        - cachePath : nom du cachePath.
        - maxSeq : à partir de quelle séquence le résultat doit commencer. Notez que si maxSeq = - 1, le résultat sera la première page du memCache. Et si maxSeq = -2, le résultat sera la dernière page du memCache.
        - nbMessages : nombre maximal de messages que l'utilisateur souhaite recevoir.
        - head : Si la valeur est 1 alors l'ordre décroissant dans le memcache est utilisé. Si la valeur est 0 alors l'ordre croissant dans le memcache est utilisé.
        - lastClickDirection : Si la valeur est 1 alors le resultat reçu sera reçu de la plus grande séquence à la plus petite séquence. Si la valeur est 0 alors le résultat reçu sera reçu de la plus petite séquence à la plus grande séquence.
        - responseType : le format de donnée choisi pour un cachePath. Ca peut être un json, un string ou un byteArray.
    c. Réponse: **[["key1","sequence1","body1"],["key2","sequence2","body2"],...]**

d. Remarques :
- Chaque record a une clé unique, une séquence et un body.
- Le champs « body » est encodé en Base64 afin de supporter divers types de données : json, string et byteArray
- Le retour de cette requête est une liste de records. Chaque record contient un string représentant la clé, un string représentant la séquence et un string base64 représentant le body.

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

3. Récupérer un record bien précis d'un cachePath :
   a. Requête : /**?action=getByKey&cachePath=xx&key=yy**
   b. Explication de la requête :
   - cachePath : nom du cachePath.
   - key : clé du record qu'on souhaite récupérer.

   c. Réponse: **["key","sequence","body"]**
   d. Remarques :
   - Cette Requête permet de récupérer un record bien précis du cachePath identifié par la clé.
   - Le body est encodé en base64.

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

4. Détecter un changement au niveau d'un cachePath :
   a. Requête : **/?action=FIL&cachePath=xx&minKey=yy&maxKey=zz**
   b. Explication de la requête :
   - cachePath : nom du cachePath.
   - minKey : minimum sequence value
   - maxKey : maximum sequence value

   c. Réponse: {"kvSize":xx,"sign":yy}
   d. Remarques :
   - Cette Requête permet à l'utilisateur connaitre si un changement a eu lieu au niveau du cachePath.
   - L'utilisateur peut choisir de détecter un changement entre deux clés dans le memCache
   - Si l'attribut «kvSize » change => un ajout ou un effacement de records a eu lieu dans le cachePath. Alors le « kvSize » permet de connaitre le nombre de records figurant dans un memCache
   - Si l'attribut « sign » change => un record est edité ou bien un record a été ajouté/effacé entre les deux records ayant comme clé « minSeq » et « maxSeq » spécifié dans la requête envoyée.
   - Si minSeq=-1 et maxSeq=-1 => vérifier le changement au niveau de tout le cachePath.

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

5. Ajouter un record dans un cachePath :
   a. Requête :
   /?action=addTarget&cacheId=xx&body=yy&key=zz&ttl=kk&responseType=aa
   b. Explication de la requete :
      - cachePath : nom du cachePath.
      - body : la valeur du body de record qu'on ajoute
      - key : clé du record qu'on ajoute dans le memCache
      - ttl : time to live.
      - responseType : type du body choisi pour le record qu'on ajoute. Ca peut être un json, un string ou un byteArray
   c. Réponse: Si le retour est **"[\"NOK\"]"** => erreur lors de l'ajout du record. Sinon, le record est bien ajouté dans le cachePath.
   d. Remarques :
      - Le body doit être encodé en Base64. On utilise ce type d'encodage pour pouvoir transporter les divers types de données supportés. Pour tester l'encodage et le décodage Base64, voila quelques exemples :
        https://www.base64decode.org/
        https://cryptii.com/pipes/base64-to-hex
      - Dans un memCache, il faut ajouter des records qui possèdent le même type de body.
      - Si la valeur du champs ttl est -1 => le record persiste et ne sera pas effacé après un certains temps. Si la valeur est > 0 alors le record sera effacé après « X » secondes. (« X » est la valeur spécifiée dans la requête)
      - Afin d'ajouter un record, c'est une requête **POST**.
      - Afin d'éditer un record, il suffit d'utiliser cette même requête et spécifier la clé du record que vous souhaitez modifier.
      - Si le body est de type json, les champs du json peuvent avoir les valeurs suivantes :
        - 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. Exemples :

Ajouter la valeur "Hello It is working" dans un cachePath de 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/

Ajouter la valeur "00010203" dans un cachePath de 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/

Ajouter un json dans un cachePath de type json :
curl --data "/?action=addTarget" --data "cachePath=/test/json" --data "body=eyJrZXkiOiJ0ZXN0RG9ja2VyIiwidHRsIjotMSwiZmllbGQxIjoidHJ1ZSIsImZpZWxkMiI6InRlc3REb2NrZXIiLCJmaWVsZDMiOiIxMDAuMCIsImZpZWxkNCI6IjIwIiwiZmllbGQ1IjpbeyJrZXkiOiJrZXkxIiwidmFsIjoidmFsMSJ9LHsia2V5Ijoia2V5MiIsInZhbCI6InZhbDIifV0sImZpZWxkNiI6WyJ2YWwxIiwidmFsMiIsInZhbDMiXSwiZmllbGQ3IjoiTm92IDA1LCAyMDIwIDEyOjE3OjAwIEFNIn0=" --data "key=testDocker" --data "ttl=-1" --data "responseType=json" http://localhost:9090/

La valeur du json est :
{"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. Effacer un record d'un memCache :
   a. Requête : **/?action=deleteTarget&cachePath=xx&key=yy**
   b. Explication de la requête :
      - cachePath : nom du cachePath.
      - key : clé du record qu'on souhaite effacer
   c. Réponse : Si le retour est **"[\"NOK\"]"** => erreur lors de l'effacement du record. Sinon, le record est bien effacé.
   d. Remarques :
      - Si la clé qu'on souhaite effacer n'existe pas => le résultat « NOK » est reçu.
   e. Exemple :
   curl 'http://localhost:9090/?action=deleteTarget&cachePath=/test/string&key=testDocker'

7. Récupérer l'historique d'un cachePath : Tri par sequence
    a. Requête :
    /?nodeAction=betweenSequences&cachePath=xx&fromSequence=yy&toSequen ce=zz&userId=kk
    b. Explication de la requête :
        - cachePath : nom du cachePath
        - fromSequence : minimum sequence value
        - toSequence : maximum sequence value
        - userId : l'utilisateur qui fait la recherche
    c. Réponse:
    [["date1","time1","sequence1","previousSequence1","key1","body1","hash1 "],["date2","time2","sequence2","previousSequence2","key2","body2","has h2"],...]
    d. Remarques :
        - Chaque record dans l'historique possède une date bien précise pour identifier quand l'opération a eu lieu.
        - Chaque record possède un numéro de séquence, une clé et un body. Le body est encodé en Base64.
        - Plusieurs utilisateurs peuvent réaliser une recherche dans l'historique. Afin de faire la différence entre la recherche de chaque utilisateur, le champs « userId » dans la requête est utilisée.
        - « fromSequence » doit être plus petite que « toSequence ».
        - Par défaut, cette requête retourne les 20 premiers records du résultat. Afin de visualiser le reste du résultat, les requêtes « firstSearch », « lastSearch », « nextSearch » et « previousSearch » sont utilisés. Ces requêtes sont utilisées dans la suite de ce document.
    e. Exemple :
    curl 'http://localhost:9000/?nodeAction=betweenSequences&cachePath=/test/st ring&fromSequence=0&toSequence=19&userId=antony'

8. Récupérer l'historique d'un cachePath : Tri par date
    a. Requête :
    /?nodeAction=betweenDates&cachePath=xx&fromDate=yy&toDate=zz&userI d=kk
    b. Explication de la requete :
        - cachePath : nom du cachePath
        - fromDate : starting date
        - toDate : end date
        - userId : l'utilisateur qui fait la recherche
    c. Réponse:
    [["date1","time1","sequence1","previousSequence1","key1","body1","hash1 "],["date2","time2","sequence2","previousSequence2","key2","body2","has h2"],...]

d. Remarques :

- Chaque record dans l'historique possède une date bien précise pour identifier quand l'opération a eu lieu.
- Chaque record possède un numéro de séquence, une clé et un body. Le body est encodé en Base64.
- Plusieurs utilisateurs peuvent réaliser une recherche dans l'historique. Afin de faire la différence entre la recherche de chaque utilisateur, le champs « userId » dans la requête est utilisée.
- « fromDate » doit être plus petite que « toDate ».
- Par défaut, cette requête retourne les 20 premiers records du résultat. Afin de visualiser le reste du résultat, les requêtes « firstSearch », « lastSearch », « nextSearch » et « previousSearch » sont utilisés. Ces requêtes sont utilisées dans la suite de ce document.
- Le format de la date à saisir dans les champs « fromDate » et « toDate » est **AAMMDD_HHmmss**. Example : 200708_140000 => 8 juillet 2020, 2 pm

e. Exemple :

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

9. Récupérer l'historique d'un cachePath (type json) : Tri par un ou plusieurs critere
    a. Requête :
    /?nodeAction=criteria&cachePath=xx&i1=v1&i2=v2&...&userId=kalima
    b. Explication de la requête :
        - cachePath : nom du cachePath
        - i1 : nom du champs du premier critère.
        - v1 : valeur du premier critère.
        - i2 : nom du champs du deuxième critère.
        - v2 : valeur du deuxième critère
        - …..
        - userId : l'utilisateur qui fait la recherche
    c. Réponse:
    [["date1","time1","sequence1","previousSequence1","key1","body1","hash1"],["date2","time2","sequence2","previousSequence2","key2","body2","hash2"],...]
    d. Remarques :
        - Cette requête est uniquement utilisée pour les cachePaths de type **json**.
        - Chaque record dans l'historique possède une date bien précise pour identifier quand l'opération a eu lieu.
        - Chaque record possède un numéro de séquence, une clé et un body. Le body est encodé en Base64.
        - Plusieurs utilisateurs peuvent réaliser une recherche dans l'historique. Afin de faire la différence entre la recherche de chaque utilisateur, le champs « userId » dans la requête est utilisée.
        - Par défaut, cette requête retourne les 20 premiers records du résultat. Afin de visualiser le reste du résultat, les requêtes « firstSearch », « lastSearch », « nextSearch » et « previousSearch » sont utilisés. Ces requêtes sont utilisées dans la suite de ce document.
        - On peut ajouter autant qu'on veut de critères à cette requête. Chaque critère doit correspondre à un champs de l'objet json utilisé dans le cachePath.
        - Au minimum il faut avoir un critère pour cette requête.
        - La valeur du critère supporte les expressions régulières pour définir un modèle de recherche. Par exemple, renvoyez toutes les valeurs d'un critère commençant par la lettre A : **A.***
    e. Exemple :
    curl 'http://localhost:9090/?nodeAction=criteria&cachePath=/test/json&field2=test.*&userId=antony

10. Récupérer l'historique d'un cachePath (type string ou byteArray) : Regex
    a. Requête : **/?nodeAction=regex&cachePath=xx&type=yy&regex=zz&userId=dd**
    b. Explication de la requête :
        - cachePath : nom du cachePath
        - type : **string** ou **bytearray**
        - regex : expression regulier
        - userId : l'utilisateur qui fait la recherche
    c. Réponse:
    [["date1","time1","sequence1","previousSequence1","key1","body1","hash1"],["date2","time2","sequence2","previousSequence2","key2","body2","hash2"],...]
    d. Remarques :
        - Cette requête est uniquement utilisée pour les cachePaths de type **string** ou **bytearray**.
        - Chaque record dans l'historique possède une date bien précise pour identifier quand l'opération a eu lieu.
        - Chaque record possède un numéro de séquence, une clé et un body. Le body est encodé en Base64.
        - Plusieurs utilisateurs peuvent réaliser une recherche dans l'historique. Afin de faire la différence entre la recherche de chaque utilisateur, le champs « userId » dans la requête est utilisée.
        - Par défaut, cette requête retourne les 20 premiers records du résultat. Afin de visualiser le reste du résultat, les requêtes « firstSearch », « lastSearch », « nextSearch » et « previousSearch » sont utilisés. Ces requêtes sont utilisées dans la suite de ce document.
        - La valeur du regex supporte les expressions régulières pour définir un modèle de recherche. Par exemple :
            - Renvoyez toutes les valeurs d'un string commençant par la lettre A : **A.***
            - Renvoyer la liste des records byteArray qui se terminent par la suite de byte 0001 : **.*0001**
    e. Exemple :
    Recherche dans cachePath de type byteArray des records qui terminent par le byte **0x10**:
    curl 'http://localhost:9000/?nodeAction=regex&cachePath=/test/bytearray&type=bytearray&regex=.*10&userId=antony'

    Recherche dans cachePath de type string des records qui contiennent la valeur **hello** :
    curl 'http://localhost:9000/?nodeAction=regex&cachePath=/test/string&type=string &regex=.*hello.*&userId=antony'

11. Récupérer l'historique d'un cachePath : nextSearch
    a. Requête : **/?nodeAction=nextSearch&cachePath=xx&size=yy&userId=zz**
    b. Explication de la requête :
        - cachePath : nom du cachePath
        - size : nombre de records qu'on souhaite recevoir en plus du résultat déjà initié (tri par sequence, tri par date, tri par regex ou tri par critere)
        - userId : l'utilisateur qui fait la recherche
    c. Réponse:
    [["date1","time1","sequence1","previousSequence1","key1","body1","hash1"],["date2","time2","sequence2","previousSequence2","key2","body2","hash2"],...]
    d. Remarques :
        - Cette requête peut être appelé uniquement après avoir fait une recherche dans l'historique : tri par séquence, tri par date, tri par regex ou tri par critère
        - Si on arrive à la fin du résultat, cette requête renvoie de nouveau les derniers records.
    e. Exemple :
    curl 'http://localhost:9000/?nodeAction=nextSearch&cachePath=/test/string&size=20&userId=antony'

12. Récupérer l'historique d'un cachePath : previousSearch
    a. Requête : **/?nodeAction=previousSearch&cachePath=xx&size=yy&userId=zz**
    b. Explication de la requête :
        - cachePath : nom du cachePath
        - size : nombre de records qu'on souhaite recevoir du résultat déjà initié (tri par sequence, tri par date, tri par regex ou tri par critère)
        - userId : l'utilisateur qui fait la recherche
    c. Réponse:
    [["date1","time1","sequence1","previousSequence1","key1","body1","hash1"],["date2","time2","sequence2","previousSequence2","key2","body2","hash2"],...]
    d. Remarques :
        - Cette requête peut être appelé uniquement après avoir fait une recherche dans l'historique : tri par séquence, tri par date, tri par regex ou tri par critère
        - Si on arrive au début du résultat, cette requête renvoie de nouveau les premiers **yy** records.
    e. Exemple :
    curl 'http://localhost:9000/?nodeAction=previousSearch&cachePath=/test/string&size=20&userId=antony'

13. Récupérer l'historique d'un cachePath : firstSearch
    a. Requête : **/?nodeAction=firstSearch&cachePath=xx&size=yy&userId=zz**
    b. Explication de la requete :
        - cachePath : nom du cachePath
        - size : nombre de records qu'on souhaite recevoir du résultat déjà initié (tri par séquence, tri par date, tri par regex ou tri par critère)
        - userId : l'utilisateur qui fait la recherche
    c. Réponse:
    [["date1","time1","sequence1","previousSequence1","key1","body1","hash1"],["date2","time2","sequence2","previousSequence2","key2","body2","hash2"],...]
    d. Remarques :
        - Cette requête peut être appelé uniquement après avoir fait une recherche dans l'historique : tri par séquence, tri par date, tri par regex ou tri par critère
        - Cette requête retourne les premiers **yy** records du résultat.
    e. Exemple :
    curl 'http://localhost:9000/?nodeAction=firstSearch&cachePath=/test/string&size=20&userId=antony'

14. Récupérer l'historique d'un cachePath : lastSearch
    a. Requête : **/?nodeAction=lastSearch&cachePath=xx&size=yy&userId=zz**
    b. Explication de la requête :
        - cachePath : nom du cachePath
        - size : nombre de records qu'on souhaite recevoir du résultat déjà initié (tri par séquence, tri par date, tri par regex ou tri par critère)
        - userId : l'utilisateur qui fait la recherche
    c. Réponse:
    [["date1","time1","sequence1","previousSequence1","key1","body1","hash1"],["date2","time2","sequence2","previousSequence2","key2","body2","hash2"],...]
    d. Remarques :
        - Cette requête peut être appelé uniquement après avoir fait une recherche dans l'historique : tri par séquence, tri par date, tri par regex ou tri par critère
    e. Exemple :
    curl 'http://localhost:9000/?nodeAction=lastSearch&cachePath=/test/string&size=20&userId=antony'

15. Vérification du hash d'un ou de plusieurs cachePath :
    a. Requête : **/?nodeAction=checkHashs&cacheList=xx**
    b. Explication de la requête :
        - cacheList : Liste des cachePaths séparé par une virgule
    c. Réponse: **[{"cachePath1":value1,"cachePath2":value2,...}]**
    d. Remarques :
        - La valeur reçue pour chaque cachePath peut être soit « true » soit « false ».
    e. Exemple :
    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}]**

V. Communication événementielle avec la blockchain Kalima

Pour développer des applications Web qui nécessitent la réception des données en temps réel de la blockchain Kalima, vous pouvez utiliser une solution basée sur le « Server-Sent Events ». Il s'agit d'une technologie basée sur HTTP qui vous permet de se connecter au serveur et de recevoir les mises à jour.

Contrairement aux solutions basées sur l'utilisation de « polling », l'avantage de cette solution est qu'elle vous permettra de recevoir les données d'une façon asynchrone de la blockchain Kalima.

Remarques :

➢ Cette partie concerne uniquement la réception des données de la blockchain Kalima.

➢ Dans la suite de ce paragraphe nous utiliserons le terme « client web » pour désigner « application web ».

Le concept de mode événementiel à utiliser pour recevoir les données en temps réel de la blockchain Kalima est :

- Le client web considère NodeJS comme un serveur,
- Chaque client web qui souhaite recevoir des données de la blockchain Kalima doit s'abonner à « /events » de l'url configuré pour NodeJS,

- A chaque mise à jour des données au niveau de la blockchain, NodeJS envoi cette mise à jour à tous les clients abonnés sur « /events ».