Desplegar el servicio de Mailbox

Para poder hacer uso del Mailbox como punto de intercambio de mensajes no es necesario desplegar un nuevo servicio, ya que existe el mailbox oficial gestionado por LACChain en la siguiente dirección: https://mailbox.lacchain.net.

Sin embargo, si se desea utilizar como un servicio auto gestionado y poder controlar las claves y servicios de autenticación, es posible hacerlo desde el repositorio oficial (https://github.com/lacchain/mailbox).

El código esta desarrollado en NodeJS version 14.4 y utiliza el motor de base de datos Redis version 4.0, por lo cual es necesario instalar dichos programas.

A continuación se muestran los pasos para poder desplegar el Mailbox de manera local directo desde el código fuente.

1. Para clonar el repositorio se debe ejecutar el siguiente comando:

```
$ git clone https://github.com/lacchain/mailbox
```

El resultado del comando debería descargar la carpeta del código fuente del Mailbox, como se muestra en la siguiente imagen:

```
Cloning into 'mailbox'...
remote: Enumerating objects: 92, done.
remote: Counting objects: 100% (92/92), done.
remote: Compressing objects: 100% (65/65), done.
remote: Total 92 (delta 34), reused 78 (delta 20), pack-reused 0
Receiving objects: 100% (92/92), 478.40 KiB | 591.00 KiB/s, done.
Resolving deltas: 100% (34/34), done.
```

2. Una vez clonado el repositorio, procedemos a instalar la version 14.4 de NodeJS. Para instalar esta versión se recomienda utilizar el gestor de versiones de NVM (https://github.com/nvm-sh/nvm), ejecutando el siguiente comando:

```
➜  ~ nvm install 14.4
v14.4.0 is already installed.
Now using node v14.4.0 (npm v6.14.5)
➜  ~ nvm use 14.4
Now using node v14.4.0 (npm v6.14.5)
```

3. Después de cambiar a la version 14.4 de NodeJS, vamos a instalar las dependencias del proyecto entrando a la carpeta descargada por git (cd mailbox) y haciendo uso del siguiente comando:

```
$ npm install
```

El comando descargara e instalará las dependencias necesarias para la ejecución del Mailbox.

```
> keccak@3.0.1 install /Users/sergio/mailbox/node_modules/keccak
> node-gyp-build || exit 0


> secp256k1@4.0.2 install /Users/sergio/mailbox/node_modules/secp256k1
> node-gyp-build || exit 0

npm WARN mailbox@1.0.0 No repository field.
npm WARN mailbox@1.0.0 license should be a valid SPDX license expression

added 287 packages from 290 contributors and audited 287 packages in 75.074s

51 packages are looking for funding
  run `npm fund` for details

found 7 vulnerabilities (6 moderate, 1 high)
  run `npm audit fix` to fix them, or `npm audit` for details
```

4. Antes de poder ejecutar el Mailbox es necesario configurar algunas variables de entorno para el correcto funcionamiento del sistema:

   - DID_RESOLVER_URL: La dirección del servicio de resolución universal de DIDs (default: https://resolver.lacchain.net/)
   - DID: El DID asociado al Mailbox (es necesario generar uno nuevo)
   - ENCRYPTION_PUBLIC_KEY: La clave pública de encriptación asociada al DID del Mailbox (es necesario generar y asociar una clave al DID)
   - ENCRYPTION_PRIVATE_KEY: La clave privada de encriptación asociada al DID del Mailbox (es necesario generar y asociar una clave al DID)
   - REDIS_HOST: La dirección del servidor de Redis (default: localhost)
   - REDIS_PORT: El puerto del servidor de Redis (default: 6379)
   - MEDIATOR_PORT: El puerto de la API expuesta por el Mailbox

Como podemos observar el Mailbox necesita de un DID y un par de claves de encriptación, para ello existe un utilidad dentro del repositorio que ayuda a generarlo automáticamente mediante el siguiente comando:

```
$ node --experimental-modules --experimental-json-modules src/did.js
```

El comando anterior genera un par de claves de encriptacion (usando el algoritmo de sodium) y la agrega como un Método de Verificación (keyAgreement) a un nuevo DID, dando como resultado lo siguiente:

```
(node:89315) ExperimentalWarning: Importing JSON modules is an experimental feature. This feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
(node:89315) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the Buffer.allo
c(), Buffer.allocUnsafe(), or Buffer.from() methods instead.
DID:  did:lac:main:0x9330ec9948dc422c2d6cc2e0e436b5beb922061c
DID Private Key:  2612da35b1bf9865c6b11fa7f5fdb4f15779bd4a41656aaf02061856adb5edf4
Encryption Public Key:  76f75459c5164945d74822e3cee288f54030742e5a2eefd470255d19d735f0a3
Encryption Private Key:  e54a65bdec35c66f41c5e9a59da7411f978e3e3cbbcba5dddad72b77f72c11ed76f75459c5164945d74822e3cee288f54030742e5
a2eefd470255d19d735f0a3
```

De la ejecución del comando anterior se pueden asignar las siguientes variables de entorno:

- DID: did:lac:main:0x9330ec9948dc422c2d6cc2e0e436b5beb922061c
- ENCRYPTION_PUBLIC_KEY: 76f75459c5164945d74822e3cee288f54030742e5a2eefd470255d19d735f0a3

- ENCRYPTION_PRIVATE_KEY: e54a65bdec35c66f41c5e9a59da7411f978e3e3cbbcba5dddad72b77f72c11ed76f75459c5164945d74822e3cee288f54030742e5a2eefd470255d19d735f0a3

Finalmente, procedemos a iniciar el servicio de Mailbox mediante el siguiente comando:

```
$ MEDIATOR_PORT=8080 DID=did:lac:main:0x9330ec9948dc422c2d6cc2e0e436b5beb922061c
ENCRYPTION_PUBLIC_KEY=76f75459c5164945d74822e3cee288f54030742e5a2eefd470255d19d735f0a3
ENCRYPTION_PRIVATE_KEY=e54a65bdec35c66f41c5e9a59da7411f978e3e3cbbcba5dddad72b77f72c11ed76f75459c5164945d74822e3cee288f54
030742e5a2eefd470255d19d735f0a3 node --experimental-modules --experimental-json-modules src/app.js
```

Si todo ha salido bien, se debería ver el siguiente resultado en la terminal, indicando que el servicio de Mailbox esta corriendo por el puerto 8080:

```
➜  mailbox git:(master) ✗ MEDIATOR_PORT=8080 DID=did:lac:main:0x9330ec9948dc422c2d6cc2e0e436b5beb922061c ENCRYPTION_PUBLIC_KEY=76f
75459c5164945d74822e3cee288f54030742e5a2eefd470255d19d735f0a3 ENCRYPTION_PRIVATE_KEY=e54a65bdec35c66f41c5e9a59da7411f978e3e3cbbcba
5dddad72b77f72c11ed76f75459c5164945d74822e3cee288f54030742e5a2eefd470255d19d735f0a3 node --experimental-modules --experimental-jso
n-modules src/app.js
LACChain ID | API Server v2.0 HTTP port 8080
```

Desplegar el Mailbox usando Docker

Alternativamente, es posible desplegar el Mailbox usando la imagen de Docker oficial en la siguiente url: ghcr.io/lacchain/mailbox:latest

Para desplegarlo el servicio usando Docker con las mismas variables de entorno, se puede usar los siguientes comandos:

```
$ docker pull ghcr.io/lacchain/mailbox:latest

$ docker run -p 8080:8080 \
-e MEDIATOR_PORT=8080 \
-e DID=did:lac:main:0x9330ec9948dc422c2d6cc2e0e436b5beb922061c \
-e ENCRYPTION_PUBLIC_KEY=76f75459c5164945d74822e3cee288f54030742e5a2eefd470255d19d735f0a3 \
-e
ENCRYPTION_PRIVATE_KEY=e54a65bdec35c66f41c5e9a59da7411f978e3e3cbbcba5dddad72b77f72c11ed76f75459c5164945d74822e3cee288f54
030742e5a2eefd470255d19d735f0a3 \
ghcr.io/lacchain/mailbox:latest
```

Autenticación al Mailbox

Para poder interactuar con la API expuesta por el Mailbox es necesario realizar un proceso de autenticación basado en el estándar DID Connect (https://github.com/KayTrust/did-connect). En el siguiente fragmento de código se muestra un ejemplo de como generar un token de acceso:

```
import didJWT from "did-jwt";
import moment from "moment";

const MAILBOX_DID = 'did:lac:main:0x9330ec9948dc422c2d6cc2e0e436b5beb922061c';

const user = {
        did: 'did:lac:main:0xf3beac30c498d9e26865f34fcaa57dbb935b0d74',
        privateKey: '278a5de700e29faae8e40e366ec5012b5ec63d36ec77e8a2417154cc1d25383f'
};

didJWT.createJWT(
        { sub: user.did, aud: MAILBOX_DID, exp: moment().add( 1, 'minutes' ).valueOf() },
        { issuer: user.did, signer: didJWT.ES256KSigner( user.privateKey ) },
        { alg: 'ES256K' }
).then( console.log );
```

Para ejecutar el código anterior hay que ejecutar el siguiente comando dentro de la carpeta raíz del repositorio (suponiendo que el archivo se llama auth.js):

```
$ node --experimental-modules auth.js
```

El resultado de la ejecución nos dará el JWT generado para autenticarse al Mailbox, como se muestra en la siguiente imagen:

```
eyJhbGciOiJFUzI1NksiLCJ0eXAiOiJKV1QifQ.eyJpYXQiOjE2Mzk1MzI3MDcsImV4cCI6MTYzOTUzMjc2NzIxNSwic3ViIjoiZGlkOmxhYzptYWluOjB4ZjNiZWFjMzBjNDk4ZDllMjY4NjVmMzRmY2FhNTdkYmI5MzViMGQ3NCIsImF1ZCI6ImRpZDpsYWM6bWFpbjoweDkzMzBlYzk5NDhkYzQyMmMyZDZjYzJlMGU0MzZiNWJlYjkyMjA2MWMiLCJpc3MiOiJkaWQ6bGFjOm1haW46MHhmM2JlYWMzMGM0OThkOWUyNjg2NWYzNGZjYWE1N2RiYjkzNWIwZDc0In0.vNBphcNHpVR6L6yP2g938EnkbhJLLvcIuWSb6W1vAKMZZwbH7NH9ZCJYbfALTqaCq1nflYiyiNIlyBO2ZYaLUg
```

Envio de una Credencial Verificable

Una vez generado el token de acceso al Mailbox, se debe incluir como una cabecera HTTP (Authorization) en cada Request al servicio. El servicio de Mailbox permite enviar cualquier tipo de mensajes, sin embargo, en el siguiente fragmento de código se muestra como enviar una Credencial Verificable (VC):

```
import didJWT from "did-jwt";
import moment from "moment";
import { didCommService } from "./src/services/index.js";
import axios from "axios";

const MAILBOX_DID = 'did:lac:main:0x9330ec9948dc422c2d6cc2e0e436b5beb922061c';
const alice = {
        did: 'did:lac:main:0x4094ea372e16fdfa13168a1c32555c30f098de56',
        privateKey: '4a1ee35bc72b5ce4f49b57d0318a622963a5191b7ad460572c343f31ebaf2e9c',
        keyPair: {
                publicKey: 'd442f5164b0a8eca6eab64afcaca86af3b3df62591dc1dbaa3f90843c9bdc2b2',
                privateKey:
'a2176c619dc06fa60bb499873fc49b3938f552e4191d9de3f0025d6fee3dfdead442f5164b0a8eca6eab64afcaca86af3b3df62591dc1dbaa3f90843c9bdc2b2'
        }
}
const bob_did = 'did:lac:main:0xbd278b861a6ca02b5a8048f87cd70c56f4a1b621';

async function send() {
        const token = await didJWT.createJWT(
                { sub: alice.did, aud: MAILBOX_DID, exp: moment().add( 1, 'minutes' ).valueOf() },
                { issuer: alice.did, signer: didJWT.ES256KSigner( alice.privateKey ) },
                { alg: 'ES256K' }
        );
        const encryptedToBob = await didCommService.encrypt( {
                '@context': 'https://www.w3.org/2018/credentials/v1',
                id: '73bde252-cb3e-44ab-94f9-eba6a8a2f28d',
                type: 'VerifiableCredential',
                issuer: alice.did,
                issuanceDate: '2021-12-12T07:17:34.479Z',
                expirationDate: '2022-12-12T07:17:34.479Z',
                credentialSubject: {
                        id: bob_did,
                        data: 'anything'
                }
        }, alice.keyPair, bob_did, true );
        const envelope = {
                "type": "https://didcomm.org/routing/2.0/forward",
                "to": [MAILBOX_DID],
                "expires_time": 1516385931,
                "body": {
                        "next": bob_did,
                        "payloads~attach": [ encryptedToBob ]
                }
        }
        const encryptedToMailbox = await didCommService.encrypt( envelope, alice.keyPair, MAILBOX_DID, true );
        const result = await axios.post( 'http://localhost:8080/vc', encryptedToMailbox, { headers: { token } } );

        console.log( result.status );
}

send();
```

Para ejecutar el código anterior hay que ejecutar el siguiente comando dentro de la carpeta raíz del repositorio (suponiendo que el archivo se llama send.js):

```
$ node --experimental-modules send.js
```

El resultado de la ejecución nos dará el código HTTP **200** en caso de que sea haya enviado correctamente o **500** en caso de un error.

```
200
```

Recepción de Credenciales Verificables

Para obtener las credenciales enviadas a un DID específico, es necesario contar con la clave privada del DID y la clave de encriptación asociada que fue usada para cifrar el mensaje. El proceso es similar al envío de credenciales, primero se genera el token de autenticación y después se consulta la ruta /vc de la API por el método GET, como se muestra en el siguiente fragmento de código:

```
import didJWT from "did-jwt";
import moment from "moment";
import { didCommService } from "./src/services/index.js";
import axios from "axios";

const MAILBOX_DID = 'did:lac:main:0x9330ec9948dc422c2d6cc2e0e436b5beb922061c';

const bob = {
        did: 'did:lac:main:0xbd278b861a6ca02b5a8048f87cd70c56f4a1b621',
        privateKey: '50dfe6a632dbc66e03eec8a3e9971a236847cd506783b17e1927273281999db1',
        keyPair: {
                publicKey: '4c9ddabbb67119640a6f3423422b1fbc6fd676cda71640ae1f9e474973dc6be8',
                privateKey:
'4f34a9f691848ce5f1e6cf7dd828b8646714a3c71605679ca48e7a5433dc1e4f4c9ddabbb67119640a6f3423422b1fbc6fd676cda71640ae1f9e474
973dc6be8'
        }
}

async function receive() {
        const token = await didJWT.createJWT(
                { sub: bob.did, aud: MAILBOX_DID, exp: moment().add( 1, 'days' ).valueOf() },
                { issuer: bob.did, signer: didJWT.ES256KSigner( bob.privateKey ) },
                { alg: 'ES256K' }
        );

        const result = await axios.get( 'http://localhost:8080/vc', { headers: { token } } );
        const decrypted = await didCommService.decrypt( result.data[0], bob.keyPair );

        const vc = JSON.parse( decrypted.message );
        console.log( vc );
}

receive();
```

Para ejecutar el código anterior hay que ejecutar el siguiente comando dentro de la carpeta raíz del repositorio (suponiendo que el archivo se llama receive.js):

```
$ node --experimental-modules receive.js
```

El resultado de la ejecución nos mostrará la Credencial Verificable enviada por Alice, como se muestra en la siguiente imagen:

```
{
  '@context': 'https://www.w3.org/2018/credentials/v1',
  id: '73bde252-cb3e-44ab-94f9-eba6a8a2f28d',
  type: 'VerifiableCredential',
  issuer: 'did:lac:main:0x4094ea372e16fdfa13168a1c32555c30f098de56',
  issuanceDate: '2021-12-12T07:17:34.479Z',
  expirationDate: '2022-12-12T07:17:34.479Z',
  credentialSubject: {
    id: 'did:lac:main:0xbd278b861a6ca02b5a8048f87cd70c56f4a1b621',
    data: 'anything'
  }
}
```

Borrar los mensajes del Mailbox

El servicio de intercambio de mensajes Mailbox fue diseñado y funciona de manera similar al servicio de correo POP3, con la única diferencia que los mensajes se almacenan de manera indefinida. Es por ello que para poder limpiar el buzón de mensajes de un DID hay que invocar a la API /vc por el método DELETE para borrar todos los mensajes, como se muestra en el siguiente fragmento de código:

```
import didJWT from "did-jwt";
import moment from "moment";
import axios from "axios";

const MAILBOX_DID = 'did:lac:main:0x9330ec9948dc422c2d6cc2e0e436b5beb922061c';

const bob = {
        did: 'did:lac:main:0xbd278b861a6ca02b5a8048f87cd70c56f4a1b621',
        privateKey: '50dfe6a632dbc66e03eec8a3e9971a236847cd506783b17e1927273281999db1'
}

async function remove() {
        const token = await didJWT.createJWT(
                { sub: bob.did, aud: MAILBOX_DID, exp: moment().add( 1, 'days' ).valueOf() },
                { issuer: bob.did, signer: didJWT.ES256KSigner( bob.privateKey ) },
                { alg: 'ES256K' }
        );

        const result = await axios.delete( 'http://localhost:8080/vc', { headers: { token } } );

        console.log( result.status );
}

remove();
```

El resultado de la ejecución nos dará el código HTTP 200 en caso de que sea hayan eliminado todos los mensajes correctamente, o 500 en caso de algún error.

```
200
```