



Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes – RABAT

Filière : Génie Logiciel

Rapport SSI : ADAMANT – Decentralized Blockchain Messenger

Réalisée par :

EL ARFAOUI Ikrame
EL AZHAR Asmaa
LAHNINE Fatima-Ezzahra
OUDAOUD El Mehdi
RAMI Anass

Encadré par :

Dr. D. El Majdoubi



Introduction

Vous avez sûrement entendu parler des crypto-monnaies, une monnaie numérique sécurisée par la cryptographie. En fait, les crypto-monnaies reposent la plupart du temps sur des réseaux décentralisés basés sur la technologie Blockchain.

La Blockchain ou chaîne de blocs est une technologie qui permet de stocker et de transmettre des informations sans mécanisme de contrôle. Techniquement parlant, il s'agit d'une base de données distribuée. Les informations envoyées par les utilisateurs et les liens internes de la base de données sont régulièrement vérifiés et regroupés en blocs pour former une chaîne. L'ensemble est protégé par cryptographie. Étant une base de données, la blockchain maintient une liste d'enregistrements **pour empêcher la falsification ou la modification par les nœuds de stockage** ; par conséquent, il s'agit d'un registre de sécurité distribué pour toutes les transactions exécutées depuis le démarrage du système distribué.

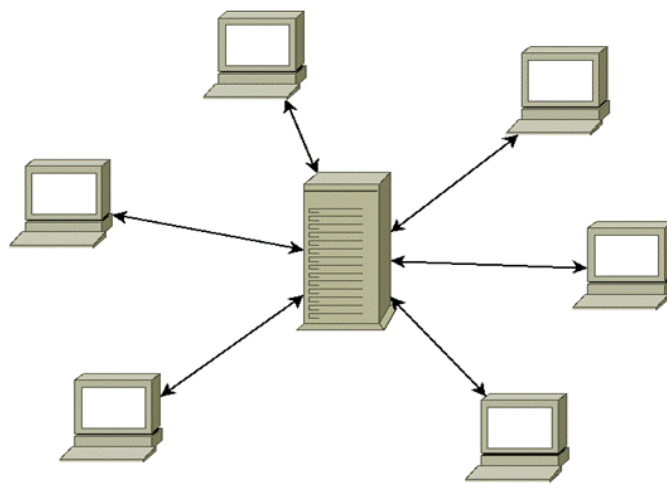
Il ne s'agit pas seulement d'un hommage à la tendance, mais d'une optimisation des activités et d'une réduction des coûts. De grands fabricants de logiciels, tels qu'Amazon et IBM, proposent déjà leurs services pour mettre en œuvre la blockchain pour les entreprises. Dans ce cadre, ADAMANT a montré en pratique comment fonctionne un messenger blockchain. Contrairement au réseau public, toutes les données d'ADAMANT Business sont stockées sur les nœuds de l'entreprise. La propriété des données est particulièrement importante pour l'entreprise.

Ainsi on utilisera Docker pour déployer notre propre boîte de messagerie ADAMANT et on essayera de l'exploiter.

1. Centralized vs Decentralized Applications

2.1. Systèmes centralisés

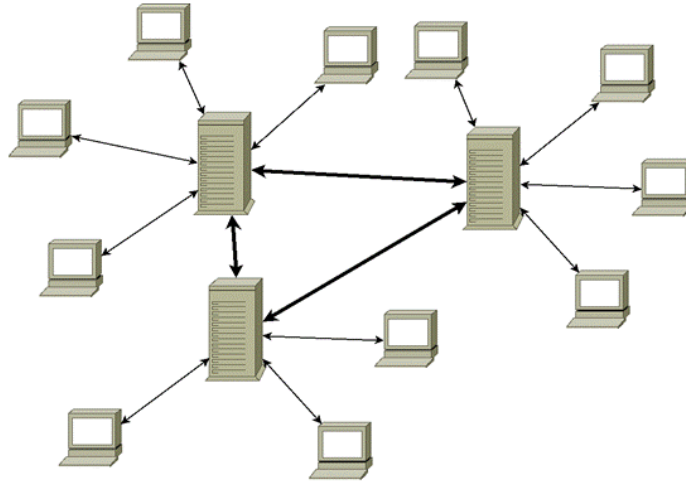
La majorité des applications qu'on utilise quotidiennement sont des applications centralisées. Google est un exemple d'un tel système, même Facebook. En tant que plate-forme centralisée, de tels systèmes nécessitent que les données passent par un seul endroit. Ce système est idéal pour suivre et stocker des informations. Un hub central gère le flux de données, c'est-à-dire que vous ne pouvez pas envoyer ou recevoir d'informations sans qu'elles ne passent par un point unique. C'est la raison pour laquelle Facebook est considéré comme le roi de la collecte de données, des milliards de personnes interagissent sur ce média et d'autres sites de médias sociaux comme Instagram et WhatsApp appartenant à Facebook au quotidien. Toutes ces données deviennent extrêmement faciles à stocker car il s'agit d'un système centralisé.



2.2. Systèmes décentralisés

Un système décentralisé peut être décrit comme un système qui a plusieurs points qui traitent l'information. Il n'y a aucune partie centrale qui se charge de gérer l'ensemble du système, mais plutôt l'interaction peer-to-peer (P2P) qui dirige le réseau. Ainsi, par rapport à un réseau centralisé, un système décentralisé permet une plus grande confidentialité des données. Il est beaucoup plus difficile de suivre

l'information dans un tel système, car l'information passe par une variété de points de validation.



2. Pourquoi ADAMANT Messenger

ADAMANT messenger est une plateforme fiable pour les communications d'entreprise, grâce aux avantages qu'elle garantit par rapport aux messageries P2P classiques.

- Un clic pour créer un compte - pas de numéro de téléphone ni d'e-mail ;
Les applications ADAMANT n'ont pas accès aux contacts et à l'emplacement.
- Pas de connexions directes, toutes les données passent par les nœuds distribués. Pas d'accès aux adresses IP des utilisateurs.
- Tous les messages sont chiffrés de bout en bout à l'aide de curve25519xsalsa20poly1305. Pas de surprise ici, mais l'avantage d'ADAMANT est un code source complètement ouvert.
- La possibilité d'attaque MITM est exclue — chaque message est une transaction et est signé par Ed25519 EdDSA ;

- Un message va à un bloc. Il n'y a aucun moyen de changer la séquence et les timestamps blocs, et donc l'ordre des messages.
- "Je ne l'ai pas dit" ne fonctionnera pas avec les messages stockés dans une blockchain.
- Aucune structure centrale qui effectue des vérifications d'authenticité des messages. Ceci est contrôlé par le système hôte distribué basé sur le consensus et appartient aux utilisateurs.
- Pas de censure — impossible de bloquer des comptes et de supprimer des messages.
- La possibilité d'obtenir toutes vos boîtes de dialogue à partir de n'importe quel appareil à tout moment est la possibilité de ne pas stocker du tout les boîtes de dialogue localement.
- Confirmation de livraison du message. Pas à l'appareil de l'utilisateur, mais au réseau. En fait, il s'agit d'une confirmation de la capacité du destinataire à lire votre message. Il s'agit d'une fonctionnalité utile pour l'envoi de notifications critiques.
- La blockchain permet également une intégration étroite avec les crypto-monnaies, et la possibilité d'envoyer des jetons dans les chats.

4. Fonctionnement de Blockchain Messenger:

Dans une blockchain, les tokens sont transférés par des transactions. Pour les messages, ils possèdent un type spécial de transaction qui exige de passer par les étapes qui suivent.

4.1. Chiffrement des messages :

Le message est chiffré avec la clé publique du destinataire et la clé privée de l'expéditeur. Le compte de destinataire doit être initialisé, en d'autres termes, avoir au moins une transaction, pour pouvoir prendre la clé publique du réseau.

ADAMANT crypte les messages avec l'algorithme Curve25519 (NaCl Box). Comme le compte contient des clés Ed25519, pour former une boîte, les clés doivent d'abord être converties en Curve25519 Diffie-Hellman.

4.2. Mise du texte chiffré dans des transactions :

En générale, les transactions se structure selon la forme suivante :

```
{
  "id": "15161295239237781653",
  "height": 7585271,
  "blockId": "16391508373936326027",
  "type": 8,
  "block_timestamp": 45182260,
  "timestamp": 45182254,
  "senderPublicKey": "bd39cc708499ae91b937083463fce5e0668c2b37e78df28f69d132fce51d49ed",
  "senderId": "U16023712506749300952",
  "recipientId": "U17653312780572073341",
  "recipientPublicKey": "23d27f616e304ef2046a60b762683b8dabebe0d8fc26e5ecdb1d5f3d291dbe21",
  "amount": 204921300000000,
  "fee": 50000000,
  "signature": "3c8e551f60fedb81e52835c69e8b158eb1b8b3c89a04d3df5adc0d99017ffbc06a7b16ad76d519f",
  "signatures": [],
  "confirmations": 3660548,
  "asset": {}
}
```

Pour une transaction de message, le plus important est l'**asset** : on doit y mettre le message dans l'objet de chat avec cette structure :

message - le message crypté

own_message - non-corps (nonce)

type - le type de message

Les messages sont également typés. Essentiellement, le paramètre type indique comment interpréter le message. Il est possible d'envoyer uniquement du texte, ou d'envoyer un objet avec un contenu intéressant à l'intérieur - par exemple, ADAMANT effectue des transferts de crypto-monnaies dans les chats de cette manière.

Le résultat d'une telle transaction est sous la forme suivante :

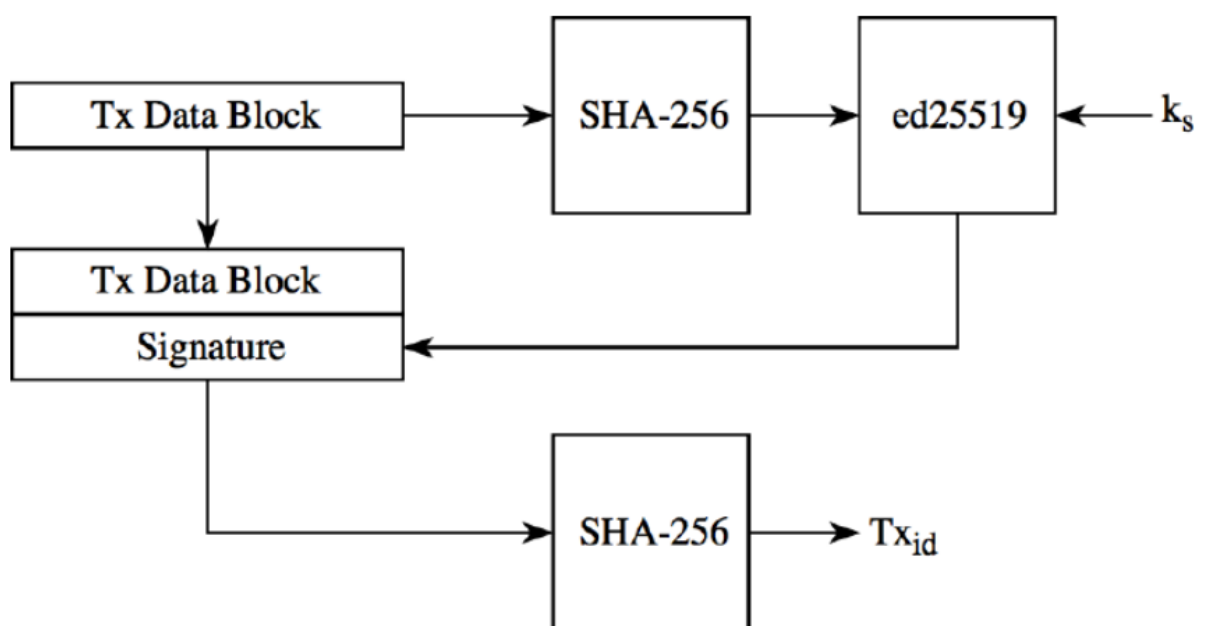
```

{
  "transaction": {
    "type": 8,
    "amount": 0,
    "senderId": "U12499126640447739963",
    "senderPublicKey": "e9cafb1e7b403c4cf247c94f73ee4cada367fcc130cb3888219a0ba0633230b6",
    "asset": {
      "chat": {
        "message": "cb682accceef92d7cddaaddb787d1184ab5428",
        "own_message": "e7d8f90ddf7d70efe359c3e4ecfb5ed3802297b248eacbd6",
        "type": 1
      }
    }
  },
  "recipientId": "U15677078342684640219",
  "timestamp": 63228087,
  "signature": "тут будет подпись"
}

```

4.3. Signature de la transaction :

Pour assurer l'authenticité de l'expéditeur et du destinataire, l'heure d'envoi et le contenu du message, la transaction est signée. La signature numérique permet de vérifier l'authenticité de la transaction à l'aide d'une clé publique ; une clé privée n'est pas nécessaire pour cela, mais la signature elle-même est effectuée par la clé privée.



On peut voir sur le diagramme que nous hachons d'abord la transaction avec SHA-256, puis que la signature est obtenue en utilisant Ed25519 (EdDSA) ; l'identifiant de la transaction fait partie du hachage SHA-256.

4.4. L'envoi de la transaction à un noeud :

Comme le réseau est décentralisé, n'importe lequel des nœuds avec une API ouverte fera l'affaire. On effectue une requête POST pour l'api/transactions comme suit :

```
curl 'api/transactions' -X POST \  
  
-d 'TX_DATA'
```

Dans la réponse, on reçoit l'identifiant de la transaction :

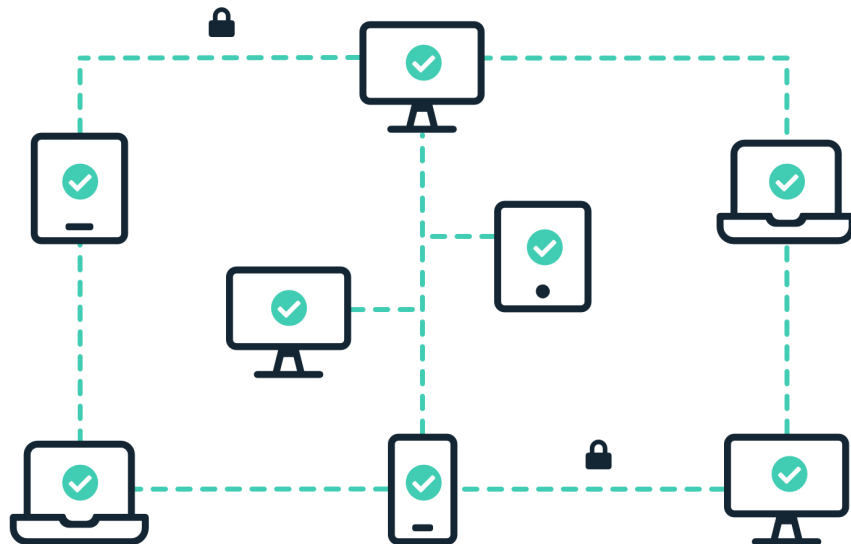
```
{  
  
  "success": true,  
  
  "nodeTimestamp": 63228852,  
  
  "transactionId": "6146865104403680934"  
}
```

4.5. Validation de la transaction :

Un système distribué de nœuds basé sur le consensus détermine la fiabilité d'une transaction de messages. De la part de qui et à destination de qui, quand, si le message a été remplacé par un autre, et si l'heure d'envoi a été bien renseignée. Il s'agit d'un avantage très important de la blockchain : aucune structure centrale n'est responsable des vérifications, et la séquence des messages et leur contenu ne peuvent pas être trafiqués.

D'abord, un nœud vérifie la fiabilité, puis l'envoie aux autres - si la plupart disent que tout est en ordre, la transaction sera incluse dans le bloc suivant - on parle alors de consensus.

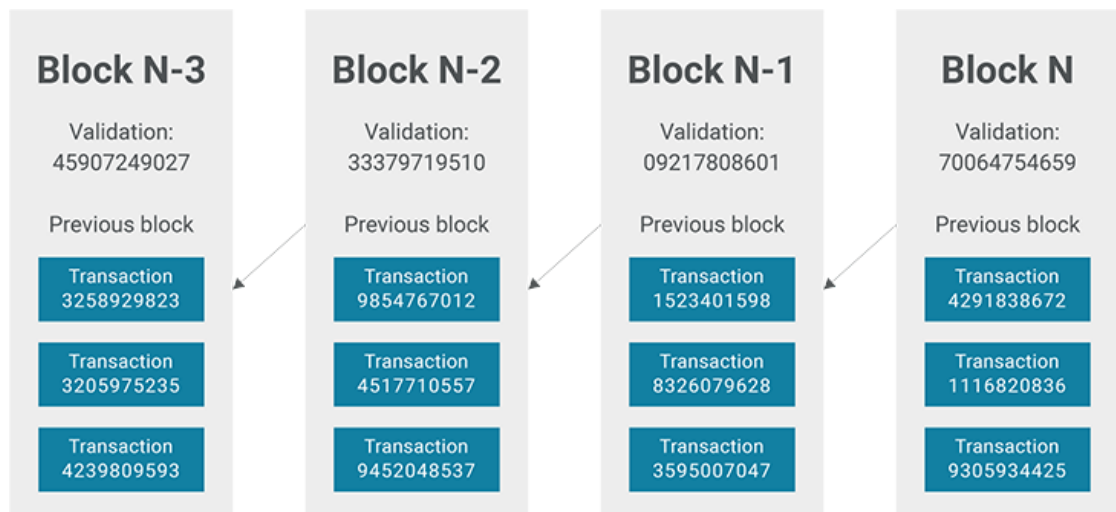
Consensus protocol



4.6. Inclusion de la transaction message dans un bloc:

Si un consensus est atteint, la transaction du message sera placée dans le bloc suivant avec les autres transactions validées.

Les blocs sont strictement séquencés, et chaque bloc suivant est formé sur la base des hashages des blocs précédents.



L'essentiel est que le message est également inclus dans cette séquence et ne peut être réaménagé. Si plusieurs messages entrent dans le bloc, leur ordre sera déterminé par le timestamp des messages.

4.7. Lecture du message:

L'application de messagerie récupère les transactions de la blockchain qui sont envoyées au destinataire. Pour réaliser cette opération, il faut créer le point de terminaison api/chatrooms.

Toutes les transactions sont accessibles à tous, il est possible de recevoir n'importe quel message crypté. Mais seul le destinataire peut le décrypter avec sa clé privée et la clé publique de l'expéditeur.

Comme les messages sont délivrés en à peu près 5 secondes de cette manière (temps de formation d'un nouveau bloc), il existe des connexions socket client-nœud et nœud-nœud. Lorsqu'un nœud reçoit une nouvelle transaction, il vérifie sa validité et la transfère aux autres nœuds. La transaction est disponible pour les clients messagers avant même le consensus et l'inclusion dans un bloc. Par conséquent, les messages sont délivrés instantanément, de la même manière que les autres systèmes de messagerie.

4.8. Emulation d'une messagerie en temps réel:

Dans certains cas, le client doit obtenir de nouvelles transactions instantanément, ce qui permet aux utilisateurs de bénéficier de performances supérieures. Il est très important pour les applications de messagerie de fournir une expérience fluide pour le chat. De cette façon, les nœuds fournissent des connexions socket.

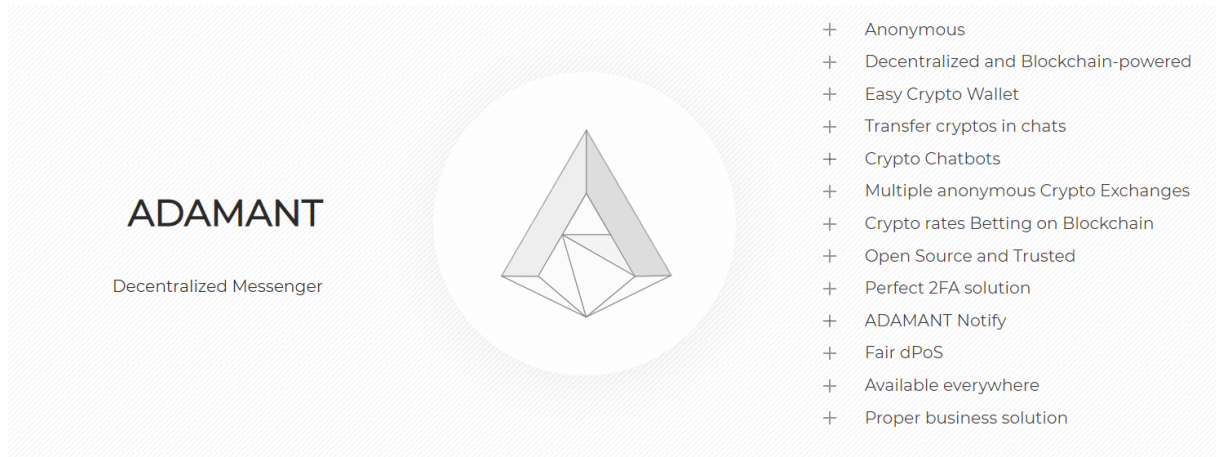
Les clients peuvent écouter le socket pour être informés de toutes les transactions, y compris celles qui ne sont pas encore confirmées. Les nouvelles transactions non confirmées n'ont pas de block_timestamp. La plupart des transactions récupérées par le socket sont nouvelles et non confirmées car elles viennent d'apparaître dans le réseau ADAMANT, et ne sont pas encore déposées dans la blockchain. Il est important de comprendre ce fait pour traiter correctement les transactions de transfert (montant > 0), vérifier si la transaction a été confirmée dans le réseau, et compter les confirmations. Une transaction non confirmée ne peut jamais être confirmée, elle peut être rejetée par le consensus des nœuds.

5. Setup du projet

5.1. Outils utilisé

ADAMANT :

Adamant est la plateforme de messagerie blockchain open source qu'on va utiliser :



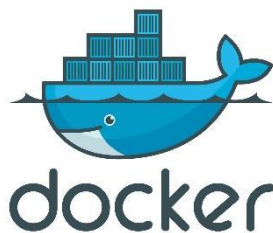
Vous pouvez voir le source code du projet ici : [Adamant-im/adamant: ADAMANT Blockchain Node \(github.com\)](https://github.com/Adamant-im/adamant)

Et Voici une version PWA de la plateforme messagerie ADAMANT :

code source : <https://github.com/Adamant-im/adamant-im>

plateforme : <https://msg.adamant.im/>

Docker :



Docker est un outil qui peut emballer une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur. C'est aussi l'outil que nous allons utiliser pour déployer la plateforme adamant localement.

5.2. Installation des outils

Prérequis :

- 4 GB RAM
- 50 GB espace disque libre (plus ou moins, en fonction de la hauteur du bloc actuel).

● Installation de Docker :

Pour Windows, le fichier Docker for Windows Installer.exe est disponible. Il se télécharge généralement dans notre dossier Téléchargements, ou on peut l'exécuter à partir de la barre des téléchargements récents au bas de votre navigateur Web.

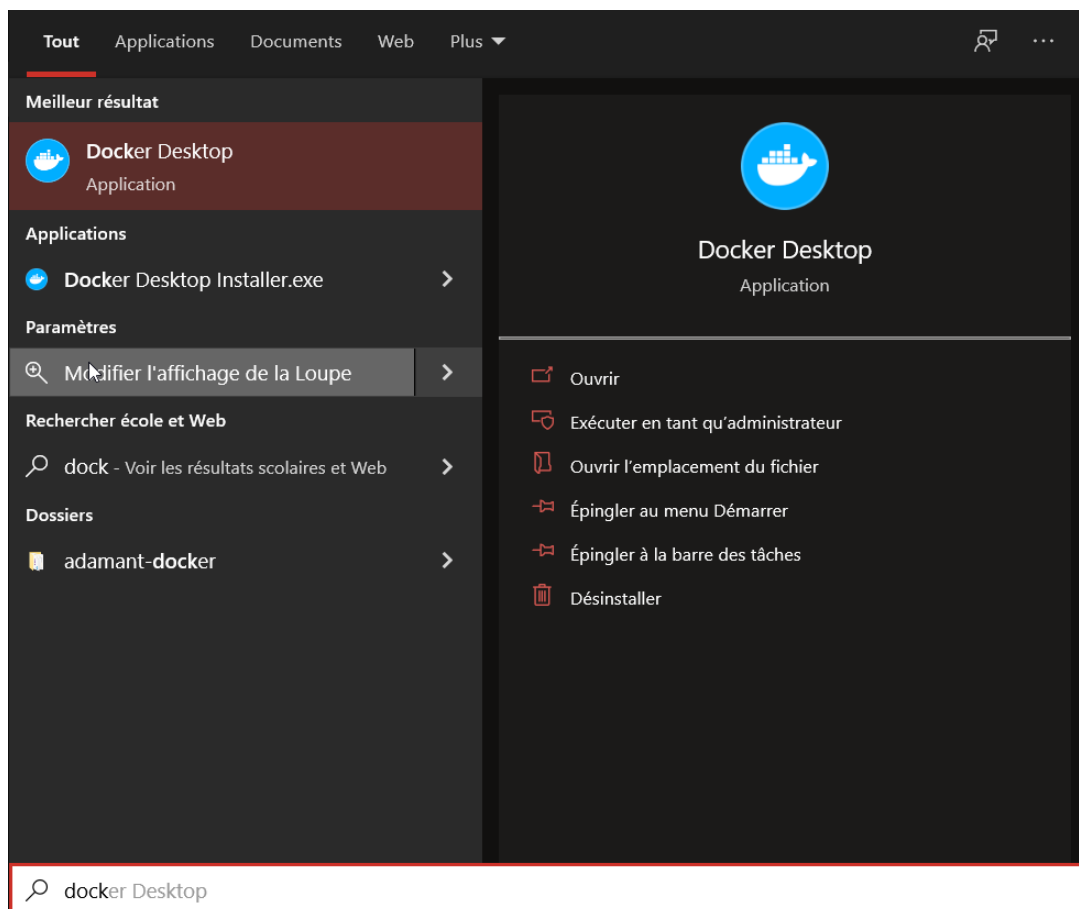
Double-clique sur Docker Desktop Installer.exe pour exécuter le programme d'installation.

En suivant l'assistant d'installation pour accepter la licence, autorise l'installation et procède à l'installation.

Au cours du processus d'installation, il nous est demandé d'autoriser Docker.app avec notre mot de passe système. Un accès privilégié est nécessaire pour installer les composants réseau, les liens vers les applications Docker et gérer les machines virtuelles Hyper-V (VM).

On clique sur Terminer dans la boîte de dialogue de fin d'installation.

Docker ne démarre pas automatiquement après l'installation. Pour le lancer, on recherche Docker, sélectionnez Docker Desktop dans les résultats de la recherche, puis cliquez dessus.



Lorsque la baleine dans la barre d'état reste stable, Docker est opérationnel et accessible depuis n'importe quelle console.



- **Installation du nœud ADAMANT**

Nous devons cloner ce repository git pour pouvoir installer et exécuter ADAMANT node : <https://github.com/Adamant-im/adamant-docker> (pour la version 0.4)

```
PS C:\Users\Fatima Ezzahra> git clone https://github.com/Adamant-im/adamant-docker
>>
Cloning into 'adamant-docker'...
remote: Enumerating objects: 64, done.
Receiving objects: 31% (20/64) | 15.70 KiB | 243.00 KiB/s, done.
Receiving objects: 100% (64/64), 15.70 KiB | 243.00 KiB/s, done.
Resolving deltas: 100% (32/32), done.
PS C:\Users\Fatima Ezzahra> cd adamant-docker
```

5.3. Exécution du nœud ADAMANT

1) Téléchargez toutes les images docker nécessaires

Dans le répertoire contenant le répertoire cloné, exécutez la commande suivante :

```
docker-compose pull
```

```
D:\Daoud\Progs for fun\Projex\Ensias Projects\SSI project\adamant-docker>docker-compose pull
Pulling db ... done
Pulling adamant-node ... done
```

2) Exécutez ensuite le service de base de données

On exécute la commande :

```
docker-compose up -d db
```

Puis la commande :

```
docker-compose logs
```

Pour vérifier si le conteneur a démarré sans aucun problème.

```

C:\Windows\System32\cmd.exe
D:\Daoud\Progs for fun\Projex\Ensias Projects\SSI project\adamant-docker>docker-compose logs
Attaching to adamant-docker_db_1
db_1 | The files belonging to this database system will be owned by user "postgres".
db_1 | This user must also own the server process.
db_1 |
db_1 | The database cluster will be initialized with locale "en_US.utf8".
db_1 | The default database encoding has accordingly been set to "UTF8".
db_1 | The default text search configuration will be set to "english".
db_1 |
db_1 | Data page checksums are disabled.
db_1 |
db_1 | fixing permissions on existing directory /var/lib/postgresql/data ... ok
db_1 | creating subdirectories ... ok
db_1 | selecting default max_connections ... 100
db_1 | selecting default shared_buffers ... 128MB
db_1 | selecting dynamic shared memory implementation ... posix
db_1 | creating configuration files ... ok
db_1 | running bootstrap script ... ok
db_1 | performing post-bootstrap initialization ... ok
db_1 | syncing data to disk ... ok
db_1 |
db_1 | Success. You can now start the database server using:
db_1 |
db_1 |     pg_ctl -D /var/lib/postgresql/data -l logfile start
db_1 |
db_1 | WARNING: enabling "trust" authentication for local connections
db_1 | You can change this by editing pg_hba.conf or using the option -A, or
db_1 | --auth-local and --auth-host, the next time you run initdb.
db_1 | LOG: could not bind IPv6 socket: Cannot assign requested address
db_1 | HINT: Is another postmaster already running on port 5432? If not, wait a few seconds and retry.
db_1 | LOG: database system was shut down at 2021-12-14 10:06:40 UTC
db_1 | waiting for server to start...LOG: MultiXact member wraparound protections are now enabled
db_1 | LOG: autovacuum launcher started
db_1 | LOG: database system is ready to accept connections
db_1 | done
db_1 | server started
db_1 | CREATE DATABASE
db_1 |
db_1 | CREATE ROLE
db_1 |
db_1 | /usr/local/bin/docker-entrypoint.sh: ignoring /docker-entrypoint-initdb.d/*
db_1 |
db_1 | waiting for server to shut down...LOG: received fast shutdown request
db_1 | .LOG: aborting any active transactions
db_1 | LOG: autovacuum launcher shutting down
db_1 | LOG: shutting down
db_1 | LOG: database system is shut down

```

3) Exécutez ensuite adamant-node :

```
docker-compose up -d adamant-node
```

```

D:\Daoud\Progs for fun\Projex\Ensias Projects\SSI project\adamant-docker>docker-compose up -d adamant-node
adamant-docker_db_1 is up-to-date
Creating adamant-docker_adamant-node_1 ... done

```

Encore une fois pour vérifier si tout marche bien :

```
docker-compose logs
```

```

db_1 | LOG: database system is ready to accept connections
adamant-node_1 | NOTICE: view "blocks_list" does not exist, skipping
adamant-node_1 | NOTICE: view "full_blocks_list" does not exist, skipping
adamant-node_1 | NOTICE: view "trs_list" does not exist, skipping
adamant-node_1 | NOTICE: trigger "protect_mem_accounts" for relation "mem_accounts" does not exist, skipping
adamant-node_1 | NOTICE: function revert_mem_account() does not exist, skipping
adamant-node_1 | NOTICE: ALTER TABLE / ADD CONSTRAINT USING INDEX will rename index "peers_unique" to "address_unique"
adamant-node_1 | NOTICE: function rounds_fees_init() does not exist, skipping
adamant-node_1 | NOTICE: trigger "rounds_fees_delete" for relation "blocks" does not exist, skipping
adamant-node_1 | NOTICE: function round_fees_delete() does not exist, skipping
adamant-node_1 | NOTICE: trigger "rounds_fees_insert" for relation "blocks" does not exist, skipping
adamant-node_1 | NOTICE: function round_fees_insert() does not exist, skipping
adamant-node_1 | NOTICE: calculating fees for rounds, please wait...
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - server
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - accounts
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - transactions
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - blocks
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - signatures
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - transport
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Releasing enqueued broadcasts
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Queue empty
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - loader
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - system
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - peers
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - delegates
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - rounds
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - multisignatures
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - dapps
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - chats
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - states
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - crypto
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - sql
adamant-node_1 | [dbg] 2021-12-14 10:22:14 Loading module - cache
adamant-node_1 | [inf] 2021-12-14 10:22:14 Secu started: 0.0.0:36666
adamant-node_1 | [inf] 2021-12-14 10:22:14 Modules ready and launched
adamant-node_1 | [inf] 2021-12-14 10:22:14 Blocks 1
adamant-node_1 | NOTICE: relation "mem_accounts" already exists, skipping
adamant-node_1 | NOTICE: relation "mem_accounts_balance" already exists, skipping
adamant-node_1 | NOTICE: relation "mem_round" already exists, skipping
adamant-node_1 | NOTICE: relation "mem_round_address" already exists, skipping
adamant-node_1 | NOTICE: relation "mem_round_round" already exists, skipping
adamant-node_1 | NOTICE: relation "mem_accounts2delegates" already exists, skipping
adamant-node_1 | NOTICE: relation "mem_accounts2delegates_accountId" already exists, skipping
adamant-node_1 | NOTICE: relation "mem_accounts2u_delegates" already exists, skipping
adamant-node_1 | NOTICE: relation "mem_accounts2u_delegates_accountId" already exists, skipping
adamant-node_1 | NOTICE: relation "mem_accounts2multisignatures" already exists, skipping
adamant-node_1 | NOTICE: relation "mem_accounts2multisignatures_accountId" already exists, skipping
adamant-node_1 | NOTICE: relation "mem_accounts2u_multisignatures" already exists, skipping

```

Maintenant on peut redémarrer en utilisant :

```

docker-compose stop
docker-compose start

```








































On peut voir nos conteneurs sur docker desktop



4) Validation de la connexion du nœud à la blockchain ADAMANT:

Allez sur <https://explorer.adamant.im/networkMonitor> à partir de n'importe quel ordinateur.

Trouvez votre nœud dans la liste des nœuds par son adresse IP. Veuillez noter qu'il peut s'écouler plusieurs minutes avant que votre nœud n'apparaisse dans la liste.

IP Address	Port	Hostname ▼	Country	Status	Version	Platform	Height
104.192.4.103	36666	swap.firulais.io			0.6.5		25324279
162.55.32.80	36666	static.80.32.55.162.clients.your-server.de			0.6.0		25324281
5.161.53.79	36666	static.79.53.161.5.clients.your-server.de			0.6.5		25324281
5.161.53.74	36666	static.74.53.161.5.clients.your-server.de			0.6.5		25324281
157.90.121.31	36666	static.31.121.90.157.clients.your-server.de			0.6.5		25324281
95.217.160.248	36666	static.248.160.217.95.clients.your-server.de			0.6.0		25324279
159.69.41.238	36666	static.238.41.69.159.clients.your-server.de			0.6.0		25324281
157.90.229.236	36666	static.236.229.90.157.clients.your-server.de			0.6.0		25324281
135.181.3.218	36666	static.218.3.181.135.clients.your-server.de			0.6.0		25324277
23.88.114.213	36666	static.213.114.88.23.clients.your-server.de			0.6.5		25324281
78.47.205.206	36666	static.206.205.47.78.clients.your-server.de			0.6.0		25324281
65.21.156.2	36666	static.2.156.21.65.clients.your-server.de			0.6.5		25324281
23.88.59.178	36666	static.178.59.88.23.clients.your-server.de			0.6.5		25324281

Voici notre nœud :

196.121.75.230	36666	196.121.75.230.unknown			0.4.0		1
----------------	-------	------------------------	---	--	-------	---	---

```

D:\Daoud\Progs for fun\Projex\Ensias Projects\SSI project\adamant-docker>docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
7aca9c2d57c1   adamantim/adamant-node:latest      "/bin/bash ./starter..." 13 minutes ago Up 9 minutes  0.0.0.0:36666->36666/tcp, :::36666->36666
ebdd1c92e773   adamant-docker_adamant-node_1     "docker-entrypoint.s..." 29 minutes ago Up 9 minutes  5432/tcp
eaffbc44dbe3   adamant-docker_db_1               "docker-entrypoint.s..." 44 hours ago   Up 8 hours    0.0.0.0:5466->5432/tcp, :::5466->5432/tcp

```

```

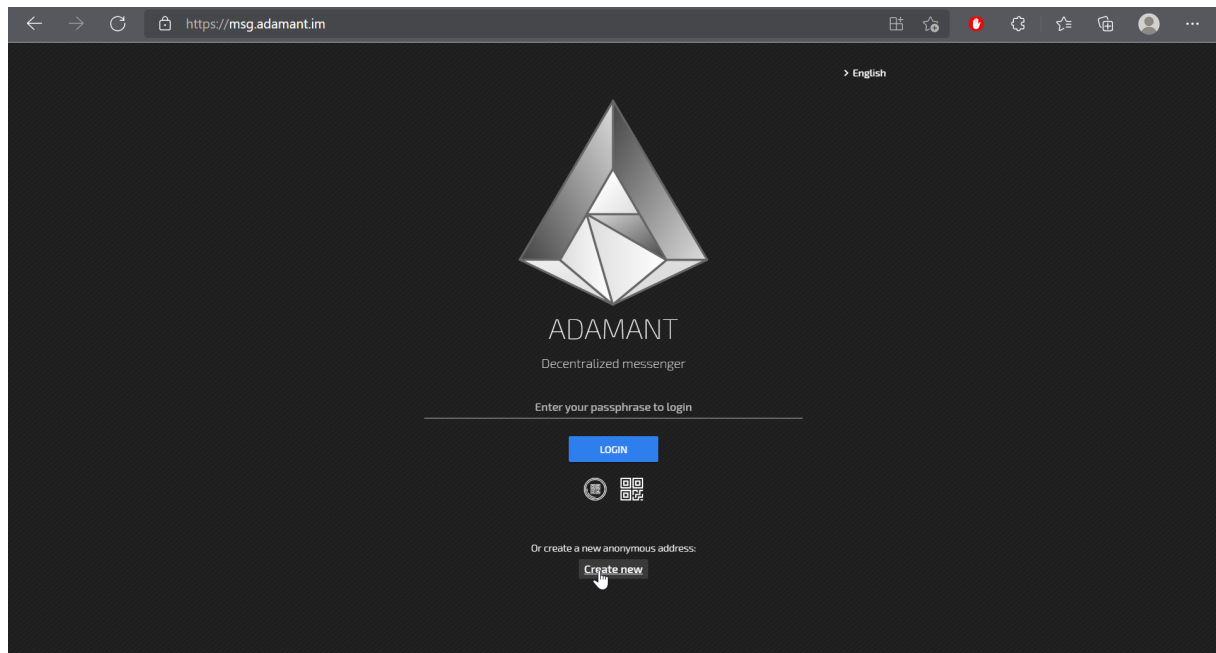
Windows PowerShell
PS C:\Users\HP> docker exec 7aca9c2d57c1 curl -k -X GET http://localhost:36666/api/blocks/getHeight
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
100    57    100    57      0      0    57      0  0:00:01 --:--:--  0:00:01   850
{"success":true,"nodeTimestamp":135106700,"height":41332}
PS C:\Users\HP>

```

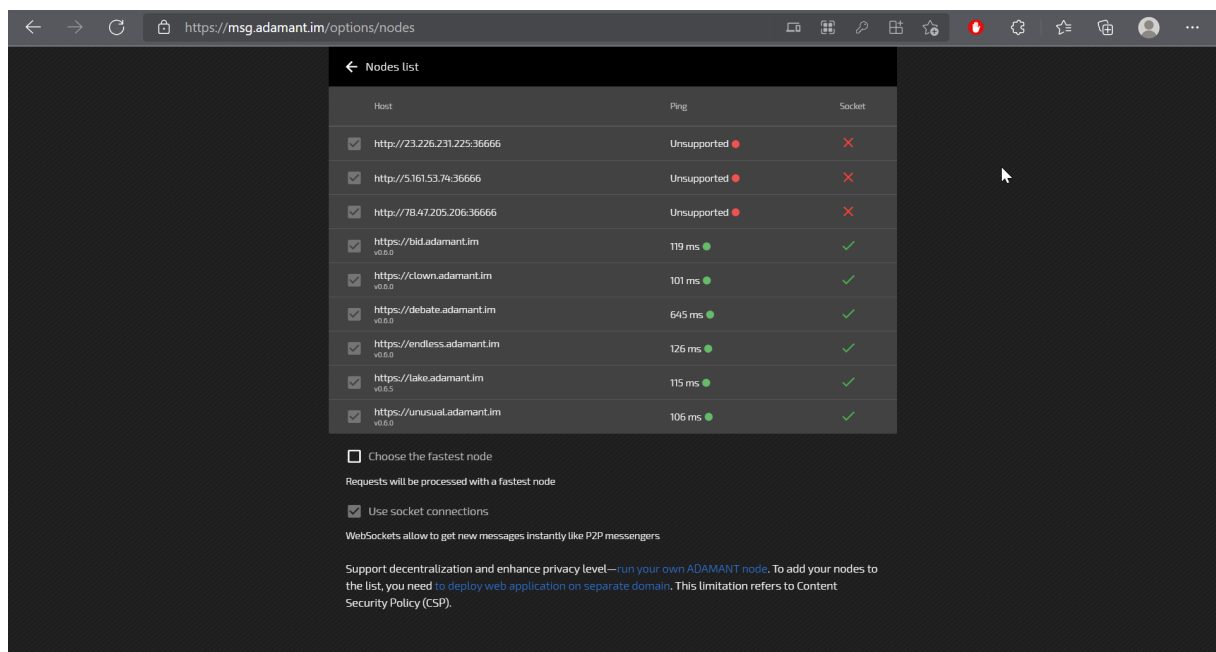
Donc le nœud c'est installer correctement et notre IP est exposé.

L'API publique permet aux applications de messagerie ADAMANT de se connecter au nœud pour envoyer et récupérer des messages. Après avoir activé l'API publique, vous pouvez choisir un nœud dans la section "Paramètres" de l'application de messagerie, ou utiliser le nœud pour vous connecter à partir d'autres services.

Tout d'abord on crée un compte :

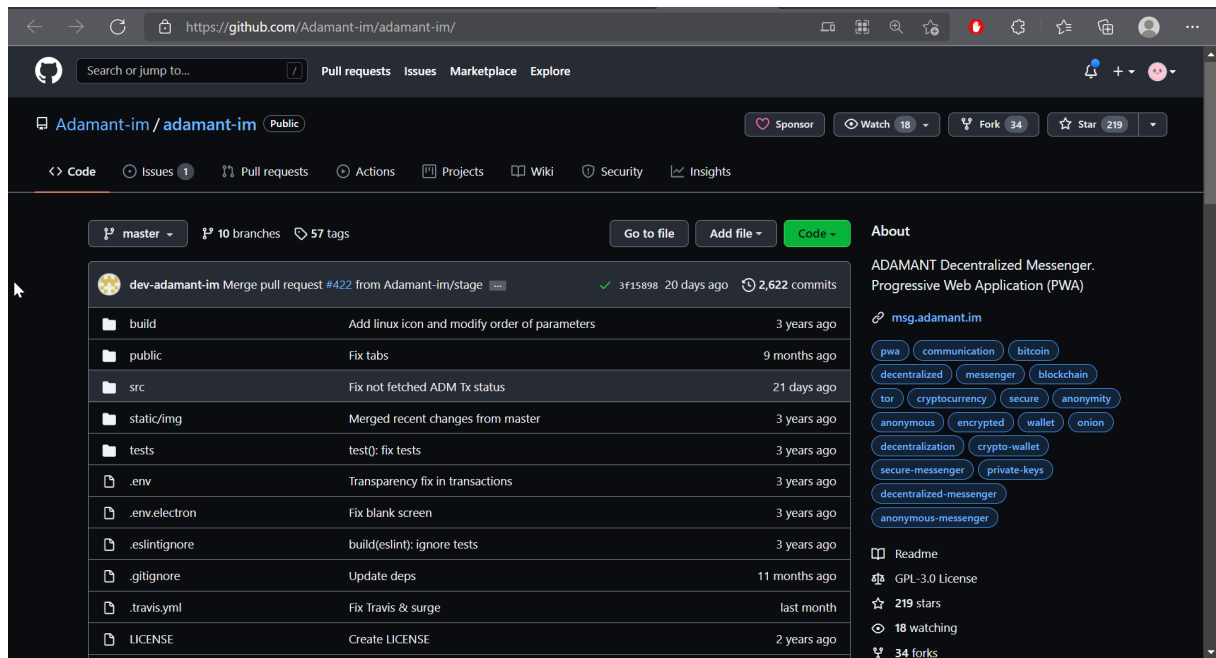


Puis on bascule vers Options>Nodes :



On doit donc passer à l'étape suivante et déployer l'application web sur un domain(localhost) séparé.

[Adamant-im/adamant-im: ADAMANT Decentralized Messenger. Progressive Web Application \(PWA\) \(github.com\)](https://github.com/adamant-im/adamant-im)



Problème Persiste puisque notre application web ne support apparemment que des noeuds de version 0.6+ donc on essaye de créer notre propre DockerFile monter l'image et voir ce que ça donne:

Fichier Docker pour héberger notre Noeud:

```
#version 0.6.5

FROM ubuntu:latest

ENV DEBIAN_FRONTEND noninteractive

ENV TERM linux

RUN sed -i 's/^mesg n$/tty -s \&\& mesg n/g' /root/.profile

RUN apt-get update && apt-get install -y --no-install-recommends apt-utils

RUN apt-get update && apt-get install -y git bash locales

RUN apt-get install -y mc git curl python build-essential curl automake autoconf libtool redis-server wget autotools-dev libsodium-dev

RUN curl -sL https://deb.nodesource.com/setup_16.x | bash -

RUN apt-get install -y nodejs

RUN apt-get purge -y postgres*
```

```
RUN sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/ zesty-pgdg main"
> /etc/apt/sources.list.d/pgdg.list'

RUN wget -q https://www.postgresql.org/media/keys/ACCC4CF8.asc -O - | apt-key
add -

RUN apt-get install -y libpq5

RUN apt-get install -y libpq-dev

RUN apt-get install -y python3-dev

# postgresql postgresql-contrib

RUN npm install -g pm2

#RUN export PATH=$PATH:/usr/pgsql-9.1/bin

RUN useradd adamant -s /bin/bash -m

RUN echo "adamant:password" | chpasswd

RUN echo "%adamant ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers

USER adamant

WORKDIR /home/adamant

ADD https://api.github.com/repos/adamant-im/adamant/git/refs/heads/master
version.json

RUN git clone https://github.com/Adamant-im/adamant

WORKDIR adamant

RUN npm install

# ceci sont les commande qui figure dans starter.sh

# RUN sed -i 's/"public": false,/"public": true,/g' config.json

# RUN sed 's/"host": "localhost"/"host": "'$DATABASE_HOST'"/g' config.json >
config.json

# RUN sed 's/"user": ""/"user": "'$DATABASE_USER'"/g' config.json >
config.json

# RUN sed 's/"password": "password"/"password": "'$DATABASE_PASSWORD'"/g'
config.json > config.json
```

```
# RUN sed 's/"secu_main"/"$DATABASE_NAME"/g' config.json > config.json

# RUN sed 's/"consoleLogLevel": "none"/"consoleLogLevel": "'$LOG_LEVEL'"/g'
config.json > config.json

#pour que notre noeud ne vérifie pas toutes les transactions ayant lieu sur la
blockchain (ce qui peut prendre des jours) adamant nous propose une base de
donnée pour éviter ceci.

RUN wget https://explorer.adamant.im/db_backup.sql.gz

RUN gunzip db_backup.sql.gz

RUN psql adamant_main < db_backup.sql

RUN rm db_backup.sql


EXPOSE 36666

COPY starter.sh /home/adamant/adamant

CMD ["/bin/bash", "./starter.sh"]
```

et voici notre fichier docker-compose.yml

```
version: '3'

services:

  adamant-node:

    restart: always

    build: .

    depends_on:

      - "db"

    ports:

      - "36666:36666"

    environment:

      - DATABASE_HOST=db

      - DATABASE_NAME=adamant_main

      - DATABASE_USER=adamant_main
```

```
- DATABASE_PASSWORD=password  
- LOG_LEVEL=debug
```

```
links:
```

```
- db
```

```
db:
```

```
restart: always
```

```
image: postgres:latest
```

```
environment:
```

```
- POSTGRES_USER=adamant_main  
- POSTGRES_PASSWORD=password
```

Comme on a dit, il vaut mieux importer l'image de la blockchain, qui permet de gagner du temps lors de la synchronisation des nœuds, mais vous devez faire entièrement confiance à l'image. Si vous sautez cette étape, votre nœud vérifiera chaque transaction, ce qui prend du temps (jusqu'à plusieurs jours).

```
=> CACHED [22/27] RUN npm install 0.0s  
=> [23/27] RUN wget https://explorer.adamant.im/db_backup.sql.gz 303.8s  
=> => # 406650K ..... 11% 2.20M 38m51s  
=> => # 406700K ..... 11% 1.32M 38m51s  
=> => # 406750K ..... 11% 2.62M 38m51s  
=> => # 406800K ..... 11% 2.63M 38m51s  
=> => # 406850K ..... 11% 1.67M 38m51s  
=> => # 406900K .....
```

voici l'image fournie par Adamant pour faire le déploiement pour des versions antérieure :

```
version: '3'
```

```
services:
```

```
adamant-node:
```

```
restart: always
```

```
image: adamantim/adamant-node:latest
```

```
depends_on:
  - "db"

ports:
  - "36666:36666"

environment:
  - DATABASE_HOST=db
  - DATABASE_NAME=adamant_main
  - DATABASE_USER=adamant_main
  - DATABASE_PASSWORD=password
  - LOG_LEVEL=debug

links:
  - db

db:
  restart: always
  image: postgres:9.6.3
  environment:
    - POSTGRES_USER=adamant_main
    - POSTGRES_PASSWORD=password
```

On peut aussi déployer le front-end sur un autre conteneur en montant le fichier dockerfile suivant :

```
FROM ubuntu:latest

ENV DEBIAN_FRONTEND noninteractive

ENV TERM linux
```

```

RUN sed -i 's/^mesg n$/tty -s \&\& mesg n/g' /root/.profile

#? initial machine setup

RUN apt-get update && apt-get install -y --no-install-recommends apt-utils

RUN apt-get update && apt-get install -y git bash locales

#? installing node js

RUN curl -sL https://deb.nodesource.com/setup_16.x | bash -

RUN apt-get install -y nodejs

RUN useradd adamant -s /bin/bash -m

RUN echo "adamant:password" | chpasswd

RUN echo "%adamant ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers

USER adamant

WORKDIR /home/adamant

ADD https://api.github.com/repos/adamant-im/adamant-im/git/refs/heads/master
version.json

RUN git clone https://github.com/Adamant-im/adamant-im

WORKDIR /adamant-im

RUN npm install --global yarn

RUN yarn install

RUN yarn run serve

EXPOSE 8080

CMD "/bin/bash"

```

exemplaire :

compte1 : organ ritual thing powder senior ahead palm luxury reject wire disagree
goose

compte2: weasel gravity front virus abuse attack clay ask miracle marine junk coin

Les webSockets sont activés par défaut dans la version 0.6 d'Adamant. Si ce n'est pas le cas, pour activer le nœud pour les sockets, il faut consulter la section wsClient du fichier config.json avec enable = true:

```
"wsClient": {  
  "portWS": 36668,  
  "enabled": true  
}
```

Pour vérifier si le nœud offre des connexions socket, on demande le statut :

<https://endless.adamant.im/api/node/status>

Les nœuds émettent des transactions de tous types avec l'événement newTrans, et l'application cliente doit les filtrer en fonction de ses besoins (par exemple, quelle adresse ADM, quel destinataire ou expéditeur, quel type de transaction). Pour les connexions socket, les nœuds stockent le pool de transactions pendant 60 secondes.

Le socket ne doit pas être la seule connexion, mais un complément aux demandes REST. En effet, la connexion socket peut être perdue pendant un certain temps, et récupérer les transactions par REST de temps en temps rend une application beaucoup plus fiable. Le socket permet néanmoins d'envoyer des requêtes REST moins souvent. Lorsqu'une nouvelle transaction est reçue par socket, elle doit être vérifiée par une requête REST pour connaître son statut.

Pour écouter les nouvelles transactions par socket, il suffit de s'abonner aux événements de socket du nœud. Voici un exemple :

```
connect (socketAddress) {  
  this.connection = io(`wss://${socketAddress}:36668`, { reconnection:  
false, timeout: 5000 })  
  
  this.connection.on('connect', () => {  
    this.currentSocketAddress = socketAddress  
    // Subscribe to transactions for specific ADM address  
    this.connection.emit('address', this.adamantAddress)  
  })  
  
  this.connection.on('disconnect', reason => {  
    if (reason === 'ping timeout' || reason === 'io server disconnect') {  
      console.warn('[Socket] Disconnected. Reason:', reason)  
    }  
  })  
}
```



```

    this.connection.on('connect_error', (err) => {
        console.warn('[Socket] connect_error', err)
    })

    // `newTrans` event emits when node finds new transaction for subscribed
    ADM address
    this.connection.on('newTrans', transaction => {
        // Filter necessary transaction types
        if ((transaction.recipientId === this.admAddress) && (transaction.type
=== 0 || transaction.type === 8)) {
            console.info(`[Socket] New incoming socket transaction received:
${transaction.id}`);
            this.onNewMessage(transaction);
        }
    });
}

```

```
[Socket] Connecting to wss://lake.adamant.im..
```

```
[Socket] Connected to wss://lake.adamant.im and subscribed to transactions of U4553056564765152121
```

Conclusion

La preuve est faite qu'un messenger blockchain peut exister. Auparavant, il n'y a eu qu'une seule tentative en 2012, Bitmessage, qui a échoué en raison du long délai de livraison des messages, de la charge du processeur et du manque d'applications mobiles.

Le scepticisme récent est lié au fait que le messenger blockchain est en avance sur son temps : les gens ne sont pas prêts à assumer la responsabilité de leur compte, la propriété des informations personnelles n'est pas encore à la mode et les technologies actuelles ne permettent pas d'atteindre des vitesses élevées sur la blockchain. Si ce n'est pas ADAMANT, des analogues plus avancés apparaîtront à l'avenir.