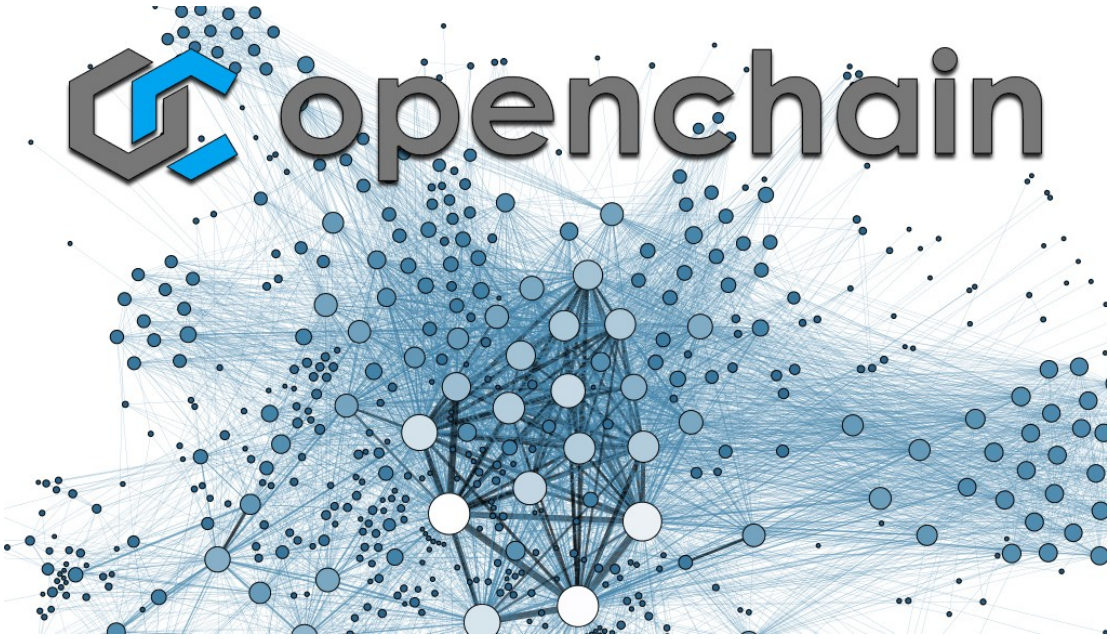
Dokumentasi Openchain

Flavien Charlon

05 September 2017

www.openchain.site

Dokumentasi gambaran cara kerja *framework* Openchain.



Founder & CEO dari Coinprism, Flavien Charlon kepada Bitcoin.com,

“Fungsi utama dari Openchain adalah bertindak sebagai suatu ledger kriptografi, seperti halnya Bitcoin Blockchain. Tetapi, ketika Bitcoin Blockchain digunakan untuk melacak kepemilikan Bitcoin, yang merupakan suatu aset[kripto] terdesentralisasi, Openchain digunakan untuk melacak kepemilikan aset di dunia nyata.”

Charlon adalah mantan software engineer Microsoft yang mendefinisikan suatu tambahan yang berbeda ke wilayah ledger terdistribusi dengan koin berwarna dan Teknologi Aset Terbuka.

Bab 1

Tinjauan

Tinjauan Openchain

Apa itu Openchain?

Openchain adalah suatu teknologi ledger open source yang terdistribusi. Ia cocok untuk organisasi yang ingin menerbitkan dan mengelola aset digital dalam cara yang dapat diukur, aman dan kokoh.

Fitur-fitur Openchain adalah sebagai berikut:

1. Konfirmasi transaksi yang seketika.
2. Tidak ada biaya *mining*.
3. Skalabilitas yang sangat tinggi.
4. Diamankan melalui tandatangan digital.
5. *Kekal*: Dapat ditambatkan *anchor* di dalam Bitcoin Blockchain untuk mendapat manfaat dari *Proof of Work*-nya yang tidak dapat dirubah.
6. Menetapkan *Alias* kepada para pengguna sebagai ganti dari menggunakan *base-58 address*.
7. Tingkat kendali yang beragam:
   - ledger terbuka yang siapa saja dapat bergabung tanpa harus disetujui.
   - ledger *closed-loop* di mana partisipan harus disetujui oleh administrator.
   - Campuran dari dua di atas di mana pengguna yang disetujui menikmati lebih banyak hak ketimbang pengguna yang tidak disetujui.
8. *Hierarchical account system* yang membolehkan untuk mengatur izin-izin pada tingkatan apa pun.

9. Transaksi yang transparan dan dapat diaudit.
10. Dapat menangani *private key* yang hilang atau dicuri tanpa kerugian apa pun bagi para *end user*.
11. Kemampuan untuk menyuruh bermacam *instance* Openchain untuk saling mereplikasi satu sama lain.

Pertanyaan yang Sering Ditanyakan

Apakah Openchain itu block chain?

Openchain berada di bawah payung teknologi Blockchain. Namun, jika kita mengambil istilah "*block chain*" secara literal, Openchain bukanlah "block chain", tetapi sepupu dekatnya. Block chain adalah struktur data yang mengorder *block* dari transaksi dan menautkan block itu secara kriptografi melalui *hashing*.

Openchain tidak menggunakan konsep block tersebut. Transaksi secara langsung diikat melalalui *chain* satu sama lain, dan tidak lagi dikelompokkan di dalam blok. Keharusan untuk mengelompokkan transaksi di dalam block mengharuskan terjadinya *delay* di dalamnya. Bahkan jika sebagian sistem melakukan pengelolaan untuk mengurangi waktu pembuatan block menjadi hanya beberapa detik, beberapa detik termasuk lama untuk penerapan yang peka terhadap delay, seperti perdagangan. Di dalam Openchain, transaksi diikatkan kepada *chain* segera setelah dikirimkan ke jaringan. Sebagai hasilnya, Openchain mampu menawarkan konfirmasi *real-time*.

Ini berarti bahwa istilah yang lebih tepat untuk Openchain adalah "*transaction chain*" ketimbang "*block chain*".

Apakah Openchain adalah *sidechain*?

Adalah mungkin untuk menggunakan *pegging module* yang akan bertindak sebagai *bridge* antara Blockchain (seperti Bitcoin) dan *instance* Openchain. Ketika Bitcoin dikirim kepada alamat spesifik, *proxy* untuk koin itu akan diciptakan pada *instance* Openchain. Kemudian, token *proxy* ini dapat diperoleh kembali untuk membuka Bitcoin pada *chain* utama. Pengaturan ini menciptakan peg 2 arah antara Bitcoin dan *instance* Openchain. Dalam skenario itu, *instance* Openchain berlaku sebagai *sidechain*.

Module pegging adalah opsional, dan suatu *instance* tidak perlu dijadikan sebagai *sidechain* jika itu tidak diperlukan.

Apakah Openchain mendukung *multi-signature*?

Multi-signature didukung. Izin diungkapkan menggunakan suatu daftar dari *public key*, dan sejumlah dari *signature* yang diperlukan. Jika Anda menyediakan 3 *public key*, dan memerlukan 2 signature, Anda memiliki 2-dari-3 akun *multi-signature*. Bacalah mengenai *dynamic permissions* untuk mempelajari lebih jauh mengenai hal itu.

Bab 2

Memulai

Memulai dengan Wallet

Server Openchain menyingkapkan suatu HTTP API publik, yang dapat dipanggil oleh program apa pun yang mampu membuat panggilan HTTP. Untuk mengemas semua operasi tersebut di dalam suatu *user interface* yang *user-friendly*, kami juga menyediakan program *client*: Openchain Wallet. Openchain Wallet tersebut adalah suatu interface berbasis web yang *open source.*

Terhubung kepada suatu server

Wallet adalah suatu aplikasi *client side* yang berjalan di browser, dan mampu terhubung kepada *endpoint* Openchain. Ia dapat terhubung kepada endpoint dan menarik informasi serta mengirimkan transaksi kepada *instance* Openchain. Pertama kali Anda menggunakannya, Anda perlu terhubung kepada satu *endpoint*.

Men-*deploy* Docker Server Openchain

Server Openchain adalah lintas platform dan dapat di-*deploy* sebagai suatu aplikasi DNX pada **Windows**, **OS X** dan **Linux**. Namun, untuk menyederhanakan pengelolaan *dependency* dan menghomogenisasi pen-*deploy*-an dari Openchain, kami mempaketkannya sebagai suatu *image* Docker.

Pada beberapa virtual hosting, docker telah tersedia dalam sistim operasi yang dapat diakses di marketplace.

Menginstall Server Openchain

Kloning-lah repositori openchain/docker dari GitHub, dan kopi-lah file konfigurasinya dari *template* yang disediakan.

```
git clone https://github.com/openchain/docker.git openchain
cd openchain
cp templates/docker-compose-direct.yml docker-compose.yml
mkdir data
cp templates/config.json data/config.json
```

Sekarang, edit-lah file konfigurasi (`data/config.json`):

```
nano data/config.json
```

Aturlah setting *instance_seed*-nya kepada suatu string acak yang (*non-empty*) .

```
[…]
// Define transaction validation parameters
"validator_mode": {
// Required: A random string used to generate the chain namespace
"instance_seed": "",
"validator": {
[…]
```

Note: Secara default, server Openchain akan berjalan pada port 8080. Anda dapat meng-edit `docker-compose.yml` jika Anda ingin menjalankannya pada port yang tidak *default*.

Sekarang Anda dapat men-*start* server tersebut:

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

Ini akan men-*start* server Openchain pada *background*.

Sekarang Anda memiliki suatu server yang *running*, Anda dapat terhubung kepada server tersebut dengan suatu *client*.

Mengkonfigurasi key admin

Gunakan *client* tersebut untuk men-*generate* suatu *seed*, dan ambillah itu menjadi suatu *address*. Setelah Anda memiliki suatu *address*, Anda dapat menggunakannya sebagai suatu *address* admin pada *instance* server Anda. Untuk melakukan itu, update-lah `data/config.json` dan tambahkanlah *address* tersebut kepada *list* `admin_addresses`:

```
// ...
"admin_addresses": [
"<your_address_here>"
],
// …
```

Tip: Konfigurasilah *info record* pada instance baru Anda. *Info record* tersebut digunakan oleh *client* yang terhubung kepada *instance* tersebut untuk menerima informasi tambahan mengenai *instance* yang sedang mereka hubungi tersebut.

Meng-kontrol server

Untuk me-*restart* server tersebut, gunakan:

```
docker-compose restart
```

Untuk men-stop-nya, gunakan:

```
docker-compose stop
```

Bab 3

General

Stream transaksi

Server Openchain menyingkap suatu *endpoint* websocket (`/stream`) yang disebut stream transaksi. Stream transaksi menyediakan *stream* langsung dari transaksi segera setelah transaksi tersebut di-komit ke dalam ledger.

Node validator

Node Server Openchain Server dapat berfungsi di dalam dua mode yang berbeda: mode validator dan mode observer.

Di dalam mode validator, node tersebut menerima transaksi dan memvalidasi transaksi-transaksi itu. Aturan yang membuat suatu transaksi valid atau invalid, dapat disesuaikan. Aturan dapat di-definisikan oleh administrator dari node validator, dan merupakan suatu kombinasi dari aturan izin implisit, dan eksplisit.

Ketika suatu transaksi dianggap valid, transaksi tersebut di-komit ke dalam ledger tersebut.

Node *Observer*

Node *observer* terhubung kepada suatu node *upstream*, dan men-*download* semua transaksi secara *real time* menggunakan stream transaksi. Node validator selalu merupakan node yang paling upstream. Ketika ia memverifikasi suatu transaksi, transaksi tersebut mengalirkannya kepada para observer-nya. Semua observernya

seharusnya memiliki salinan yang pasti sama dari keadaan yang dibuat oleh node verfikasi.

Konfigurasi

Untuk mengkonfigurasi suatu node untuk berada di dalam mode observer, seksi `observer_mode` perlu ada di dalam file konfigurasi, dan `upstream_url` harus di-set kepada URL *root* dari node upstream tersebut.

Menambatkan *Anchor* dan integritas ledger

Openchain dapat dikekalkan dengan meng-komit suatu *hash* dari keseluruhan ledger (hash kumulatif) ke dalam suatu Blockchain yang *non-reversible* semacam Bitcoin.

Note: Di dalam versi saat ini, satu-satunya mode meng-anchor yang tersedia adalah mode blockchain, berdasarkan pada blockchain Bitcoin. Mode meng-anchor yang berbeda akan tersedia di masa depan, seperti meng-anchor di dalam suatu repositori sentral.

Dengan mode meng-anchor kepada Bitcoin, satu transaksi di-komit di dalam setiap block Bitcoin, dan memuat hash kumulatif di saat itu.

Dengan melakukan ini, bahkan jika Openchain sedang memproses ribuan transaksi per detik, hanya satu transaksi yang dikirimkan kepada blockchain Bitcoin setiap 10 menit. Ada keuntungan ganda kepada pendekatan ini:

- Ireversibilitas dari ledger Openchain dijamin oleh para *miner* Bitcoin, karena itu Openchain menikmati tingkat ireversibilitas yang sama seperti halnya Bitcoin itu sendiri.

- Pada resolusi maksimum (satu anchor per *block*), tidak lebih dari 4320 transaksi per bulan (dalam rata-rata) akan di-komit ke dalam blockchain, yang akan mengenakan ongkos sebesar $10 per bulan (Oktober 2015), tanpa menghiraukan jumlah transaksi yang diproses.
- Resolusi tersebut dapat disetel untuk lebih jauh mengurangi ongkos tersebut.
- Openchain dapat memproses ribuan transaksi per detik namun tetap sangat efisien ongkos.

"*Node observer*" yang mereplikasi seluruh transaksi yang terverifikasi secara lokal memiliki kemampuan untuk meng-komputasi versi mereka sendiri dari hash kumulatif tersebut dan membandingkannya kepada anchor-nya di dalam blockchain Bitcoin.

Mengkalkulasi hash kumulatif

Hash kumulatif di-update setiap kali suatu transaksi baru ditambahkan kepada ledger tersebut.

Hash kumulatif pada *height* yang diberikan dikalkulasikan menggunakan hash kumulatif yang sebelumnya dan hash dari transaksi baru tersebut ditambahkan kepada ledger tersebut:

```
cumulative_hash = SHA256( SHA256( previous_cumulative_hash + new_transaction_hash ) )
```

- `previous_cumulative_hash` (32 byte) adalah hash kumulatif pada height sebelumnya. Pada height 0 (ketika ledger tersebut tidak memiliki transaksi), suatu *buffer* 32 byte yang diisi dengan nol, digunakan.
- `new_transaction_hash` (32 byte) adalah hash ganda SHA-256 dari transaksi acak yang ditambahkan kepada ledger tersebut.

Kedua nilai di-*concat* untuk membentuk suatu *array* 64 byte, kemudian di-hash menggunakan SHA-256 ganda.

Format anchor Blockchain

Anchor Blockchain disimpan di dalam blockchain tersebut menggunakan suatu operator `OP_RETURN`, yang diikuti dengan suatu `pushdata` yang memuat anchor tersebut.

```
OP_RETURN <anchor (42 bytes)>
```

Anchor tersebut disusun dalam cara berikut:

```
0x4f 0x43 <transaction count (8 bytes)> <cumulative hash (32 bytes)>
```

- Dua byte pertama mengindikasikan bahwa output tersebut mewakili suatu anchor Openchain.
- `transaction count` adalah jumlah dari transaksi yang diwakili oleh hash kumulatif (*height*-nya). Itu adalah suatu integer *unsigned* 64 bit, yang di-*encode* di dalam *big endian*.
- Hash kumulatif adalah hash kumulatif full (256 bit) sebagaimana dikalkulasi di dalam seksi sebelumnya.

Konfigurasi Server Openchain

Konfigurasi dari server Openchain ditangani melalui suatu file JSON yang dinamai `config.json`. File tersebut disimpan di bawah folder data.

Dimungkinkan untuk mengesampingkan suatu nilai konfigurasi melalui *variable environment*. Nama dari variable tersebut hendaknya adalah *concatenation* dari seluruh komponen dari *path* tersebut, dipisahkan oleh karakter `:`. Misal: `validator_mode:validator:allow_third_party_assets`.

config.json

Berikut ini adalah file default:

```
{
      "enable_transaction_stream": true,
      "storage": {
      "provider": "SQLite",
      "path": "ledger.db"
},
// Define transaction validation parameters
"validator_mode": {
      // Required: A random string used to generate the chain
namespace
      "instance_seed": "",
      "validator": {
      "provider": "PermissionBased",
      // Enable /p2pkh/<address>/ accounts
      "allow_p2pkh_accounts": true,
      // Enable /asset/p2pkh/<address>/ accounts
      "allow_third_party_assets": true,
      // Base-58 addresses that must have admin rights
      "admin_addresses": [
      ],
      "version_byte": 76
      }
},
// Uncomment this and comment the "validator_mode" section to enable
observer mode
// "observer_mode": {
//
"upstream_url": ""
// },
"anchoring": {
      "provider": "Blockchain",
      // The key used to publish anchors in the Blockchain
      "key": "",
      "bitcoin_api_url": "https://testnet.api.coinprism.com/v1/",
      "network_byte": 111,
      "fees": 5000,
      "storage": {
      "provider": "SQLite",
      "path": "anchors.db"
      }
   }
}
```

Seksi Root

• `enable_transaction_stream`: Boolean mengindikasikan apakah websocket stream transaksi hendak dibolehkan atau tidak pada *instance* ini.

Seksi `storage`

`provider` mendefinisikan *engine storage* mana yang digunakan. Dua nilai *built-in* adalah `SQLite` and `MSSQL`.

engine storage `SQLite`

Jika *provider storage* di-set kepada `SQLite`, *chain* tersebut disimpan secara lokal menggunakan `SQLite`. Jika demikian, setting berikut digunakan:

• `path`: Path dari database `Sqlite`, *relative* bagi folder `wwwroot/App_Data`. Path *absolute* juga dibolehkan, tetapi, pastikan user di bawah mana proses DNX berjalan memiliki *write access* kepada file tersebut.

Engine storage `MSSQL`

Jika provider storage di-set kepada `MSSQL`, *chain* tersebut disimpan menggunakan Microsoft SQL Server. Jika demikian, setting berikut digunakan:

- `connection_string`: string koneksi kepada database `SQL Server`.

Note: *Engine storage third party* dapat dibangun dan digunakan oleh Openchain. Setting provider digunakan untuk mengidentifikasi pada saat dijalankan, *engine storage* mana yang seharusnya di-*instantiate*.

Seksi `validator_mode` dan seksi `observer_mode`

Dua seksi ini secara mutual eksklusif. Bergantung pada apakah *instance* tersebut di-set-*up* di dalam mode validator atau mode observer, salah satu dari seksi `validator_mode` atau seksi `observer_mode` harus ada.

Dalam hal mode validator:

- `validator_mode:instance_seed`: Suatu string acak yang hendaknya unik bagi *instance* tersebut. String tersebut di-hash untuk memperoleh suatu *namespace specific* bagi *instance* tersebut.
- `validator_mode:validator:provider`: Type dari validasi yang dilakukan oleh instance Openchain ketika transaksi di-submit. Satu-satunya nilai yang didukung saat ini adalah `PermissionBased`, `PermitAll` dan `DenyAll`.
  - `PermitAll` mengindikasikan bahwa seluruh transaksi adalah valid, tanpa menghiraukan siapa yang menandatangani transaksi itu. Ini digunakan terutama untuk pengetesan.
  - `DenyAll` mengindikasikan bahwa seluruh tranksaksi adalah invalid, tanpa menghiraukan siapa yang menandatangani transaksi itu. Ini digunakan untuk men-set *chain* di dalam mode *read-only*.
  - Lihat pada halaman berikutnya untuk rincian lebih banyak mengenai aturan implisit dari mode `PermissionBased`. Setting konfigurasi yang relevan dengan mode `PermissionBased` adalah sebagai berikut:
    * `validator_mode:validator:allow_p2pkh_accounts`: Boolean yang mengindikasikan apakah akun-akun *P2PKH* (`/p2pkh/<address>/`) di-*enable*-kan.
    * `validator_mode:validator:allow_third_party_assets`: Boolean yang mengindikasikan apakah akun-akun *third party issuance* (`/asset/p2pkh/<address>/`) di-*enable*-kan.
    * `validator_mode:validator:admin_addresses`: *List* dari string yang mewakili semua address dengan hak admin.

    * `validator_mode:validator:version_byte`: Versi byte yang digunakan ketika mewakili suatu public key menggunakan perwakilan *address* Bitcoin-nya.

Dalam hal sebagai mode observer:

- `observer_mode:upstream_url`: URL endpoint dari *instance upstream* yang terhubung. Transaksi akan direplikasi menggunakan endpoint ini.

Men-*setting* info *instance* pada suatu *instance* baru

Record info ledger menyingkap informasi-meta mengenai ledger itu sendiri. Ia digunakan oleh klien yang terhubung kepada *instance* tersebut untuk memperoleh informasi seperti nama dari *instance*, serta syarat dan ketentuan yang terkait.

Setelah Anda men-deploy suatu *instance* baru, sebaiknya buatlah record info. Ini dapat dilakukan dari interface web.

1. Pertama, ikuti langkah-langkah berikut untuk terhubung kepada *instance* tersebut dan *log in*. Pastikan Anda log in dengan satu *seed* yang memiliki akses admin pada *instance* ini karena record infonya hanya dapat dimodifikasi oleh seorang administrator.
2. Pergilah kepada tab ***Advanced*** dan klik ***Edit Ledger Info*** pada bagian sebelah kiri. Layar akan menunjukkan pada Anda suatu form yang akan membolehkan Anda untuk meng-edit nama ledger tersebut dan bidang lain yang disimpan di dalam record `info`.

```
www.openchain.site
```

Penting: Pastikan bahwa URL Root Validator di-set kepada nilai yang sama sebagaimana *setting* `root_url` di dalam file konfigurasi.

Meng-*upgrade* server Openchain

Untuk meng-upgrade suatu *deployment* Openchain melalui Docker, jalankan perintah berikut:

```
git reset --hard
git pull
cp templates/docker-compose-direct.yml docker-compose.yml
docker-compose build
docker-compose restart
```

Note: Jika versi baru yang Anda upgrade menyertakan suatu perubahan skema file konfigurasi, jangan lupa untuk meng-update file konfigurasi sebelum me-restart Openchain.

Men-deploy Openchain di dalam suatu lingkungan produksi

Di dalam produksi, direkomendasikan untuk mem-*proxy* server Openchain dibelakang suatu server *reverse proxy* seperti Nginx. Arsitektur ini membolehkan sejumlah kemungkinan:

- Mengekspos Openchain melalui SSL/TLS
- Meng-host beragam *instance* server Openchain pada port yang sama
- Merubah path URL di mana server Openchain diekspos
- Me-route *request* kepada instance Openchain yang berbeda tergantung pada nama *host* yang digunakan

Dokumen ini menjelaskan sedikit langkah yang perlu untuk mengekspos Openchain melalui Nginx.

Meng-install Docker

Rujuklah pada dokumentasi dasar pen-*deploy*-an Docker untuk mencari tahu bagaimana cara meng-*install* Docker dan Docker Compose.

Tariklah image Docker melalui Docker Compose

Kloninglah repositori `openchain/docker` dari GitHub, dan salinlah file konfigurasi dari template yang disediakan.

```
git clone https://github.com/openchain/docker.git openchain
cd openchain
cp templates/docker-compose-proxy.yml docker-compose.yml
cp templates/nginx.conf nginx/nginx.conf
mkdir data
cp templates/config.json data/config.json
```

Editlah file konfigurasi (`data/config.json`) seperti digambarkan di dalam dokumentasi dasar pen-*deploy*-an Docker.
Sekarang Anda dapat men-start server:

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

Note: Secara default, Nginx akan berjalan pada port 80.

Troubleshooting

Error "The namespace used in the transaction is invalid"

Anda bisa jadi menerima pesan error ini ketika men-submit suatu transaksi. Anda akan mendapatkan error ini jika setelan `root_url` di dalam konfigurasi tidak cocok

dengan *namespace* yang di-set oleh *client* di dalam transaksi tersebut. Client akan selalu menggunakan URL yang mereka hubungi *namespace*-nya.

Ini memastikan bahwa suatu transaksi hanya valid untuk satu instance spesifik dari Openchain, dan tidak mungkin menggunakan kembali suatu transaksi yang ditandatangani pada dua ledger yang berbeda.

Solusi

Untuk menyelesaikan ini, pastikan setelan URL di dalam file konfigurasi (`validator_mode:root_url`) cocok dengan URL yang client gunakan untuk terhubung kepada instance Openchain Anda. Semua komponen dari URL tersebut harus cocok:

• Skema semisal: `http://endpoint.com/` vs `https://endpoint.com/`
• Nama host semisal: `http://127.0.0.1/` vs `http://localhost/`
• Port semisal: `http://endpoint.com:80/` vs `http://endpoint.com/`
• Path semisal: `http://endpoint.com/path/` vs `http://endpoint.com/`

Penting: Pastikan Anda tidak melupakan *trailing slash*, karena client akan selalu menyertakannya di dalam namespace tersebut. Misal: `https://endpoint.com/` sebagai ganti dari `https://endpoint.com`.

Bab 4

API Publik

Struktur data Openchain

Openchain bergantung pada beberapa struktur data untuk komunikasi antara *client* dan server. Struktur data ini adalah suatu bagian *key* dari API Openchain.

Struktur data ini di-serial-kan dan di-deserial-kan menggunakan Protocol Buffers.

Skema

Skema penuhnya adalah sebagai berikut:

```
syntax = "proto3";

package Openchain;

message RecordValue {
      bytes data = 1;
}
message Record {
      bytes key = 1;
      RecordValue value = 2;
      bytes version = 3;
}
message Mutation {
      bytes namespace = 1;
      repeated Record records = 2;
      bytes metadata = 3;
}
message Transaction {
      bytes mutation = 1;
      int64 timestamp = 2;
      bytes transaction_metadata = 3;
}
```

Note: Skema tersebut menggunakan versi 3 dari Protocol Buffers.

Record

Suatu objek record mewakili maksud untuk merubah nilai dari suatu record di dalam penyimpanan data. Key-nya, nilai dan versi dari suatu *record* bisa saja *string byte* apa pun.

```
message Record {
      bytes key = 1;
      RecordValue value = 2;
      bytes version = 3;
}
```

- `key`: Suatu nilai yang secara unik mengidentifikasi record yang hendak dimodifikasi tersebut.
- `value`: Nilai baru yang record tersebut seharusnya miliki setelah *update*. Jika itu tidak disebutkan, versi record diperiksa, tetapi tidak ada update yang dibuat kepada nilai tersebut.
- `version`: Versi terakhir dari record yang sedang di-*update* tersebut. Setiap modifikasi dari record tersebut akan menyebabkan versi tersebut berubah. Jika versi yang disebutkan tidak cocok dengan versi yang sesungguhnya di dalam penyimpanan data, maka update tersebut gagal.

Suatu record yang tidak pernah di-set memiliki suatu nilai dan versi yang keduanya sama kepada suatu *empty byte string*.

Record *check-only*

Jika suatu objek record memiliki suatu bidang nilai kosong, objek record tersebut disebut suatu record *check-only*, dan tidak menyebabkan suatu mutasi kepada record tersebut. Tetapi ia mengekspresikan *requirement*-nya bahwa record tersebut (sebagaimana diwakili oleh bidang key tersebut) harus memiliki versi yang di-spesifikasikan di dalam bidang versi tersebut dari objek record-nya. Jika versinya tidak cocok, keseluruhan mutasi tersebut gagal untuk diterapkan.

Ini menyediakan suatu cara untuk memastikan bahwa suatu record yang diberikan belum dimodifikasi di antara saat transaksi diciptakan dan saat ia divalidasi, bahkan jika record tersebut tidak harus dimodifikasi.

Mutasi

Suatu mutasi adalah satu set record yang secara *atomic* merubah keadaan dari data tersebut. Set record tersebut secara khas di-*generate* oleh suatu *client*, ditandatangani, kemudian dikirim kepada validator-nya bersama dengan tandatangannya.

```
message Mutation {
      bytes namespace = 1;
      repeated Record records = 2;
      bytes metadata = 3;
}
```

- `namespace`: namespace di bawah mana record tersebut hidup. Secara general, setiap *instance* Openchain memiliki *namespac*e-nya sendiri.
- `records`: suatu set dari record-record untuk dimodifikai secara *atomic* oleh mutasi ini. Setiap record diidentifikasi oleh key nya. Versi dari setiap record di dalam mutasi tersebut harus cocok dengan versi yang ada saat itu. Jika ada versi apa pun yang tidak cocok, maka keseluruhan mutasi gagal untuk diterapkan. Record dengan nilai yang tidak disebutkan tidak menyebabkan update, tetapi versi mereka tetap harus cocok agar mutasi tersebut berhasil.
- `metadata`: metadata suka-suka untuk disimpan di dalam mutasi tersebut.

Versi dari semua record yang di-update setelah suatu mutasi menjadi hash dari mutasi tersebut.

Transaksi

Suatu transaksi adalah suatu *wapper* di sekitar suatu mutasi.

```
message Transaction {
      bytes mutation = 1;
      int64 timestamp = 2;
      bytes transaction_metadata = 3;
}
```

- `mutation`: Mutasi yang diterapkan oleh transaksi tersebut. Ia diwakili oleh suatu string *byte* tetapi di-deserialisasi menurut skema dari Mutasi tersebut.
- `timestamp`: Suatu timestamp untuk transaksi tersebut.
- `transaction_metadata`: Metadata suka-suka untuk disimpan di dalam mutasi tersebut. Ini akan secara khas memuat suatu tandatangan digital dari mutasi tersebut oleh pihak-pihak yang diminta.

Struktur ledger

Pada *core*-nya, suatu ledger Openchain adalah suatu *key-value store*, yang diwakili oleh record-record. Pada level *data store*, *key* record dapat berupa string byte apa saja yang dikehendaki, namun ledger Openchain mengharapkan suatu struktur yang didefinisikan dengan baik untuk key record tersebut.

Key record

Key record adalah string yang di-*encode* UTF8. Key tersebut distruktur di dalam tiga bagian, dipisahkan oleh colon (:).

1. Path record: Suatu path di dalam *account hierarchy* yang mengindikasikan di mana record tersebut ditempatkan.
2. Tipe record: Suatu nilai yang mengindikasikan type dari record tersebut.
3. Nama record: Nama dari record tersebut.

Kombinasi dari tiga nilai ini secara unik mengindentifikasikan suatu record.

Contoh 1

```
/p2pkh/mfiCwNxuFYMtb5ytCacgzDAineD2GNCnYo/:ACC:/asset/p2pkh/
‹ → n15g8F3sVLufwvPmmX7tYPWrGGbGSbcaEB/
```

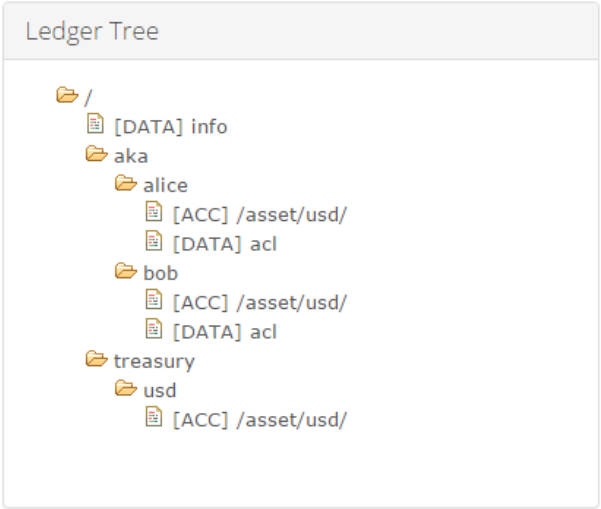
Path-nya adalah `/p2pkh/mfiCwNxuFYMtb5ytCacgzDAineD2GNCnYo/`, tipe record-nya adalah ACC dan nama record-nya adalah `/asset/p2pkh/n15g8F3sVLufwvPmmX7tYPWrGGbGSbcaEB/`.

Contoh 2

```
/:DATA:info
```

Path-nya adalah `/` (root path), tipe record-nya adalah `DATA` dan nama record-nya adalah `info`.

Hirarki akun



Akun diidentifikasi oleh suatu path.

Path akun

Sintaks untuk suatu path akun mengikuti sejumlah aturan:
• Path akun dimulai dengan karakter `/`.
• Path akun diakhiri dengan karakter `/`.
• Seksi dari suatu path akun dipisahkan dengan karakter `/`.
• Seksi dari suatu path akun hanya boleh memuat karakter alfanumerik dan karakter dari set berikut: `$-_.+!*'(),`.

Tipe record

Ada dua tipe record yang valid pada versi Openchain ini.

ACC record

ACC record digunakan untuk mewakili suatu *balance* untuk suatu tipe aset yang diberikan. Nama dari record tersebut haruslah suatu path yang mewakili tipe aset. Nilainya haruslah suatu *64-bits signed integer* yang di-*encode* di dalam *big endian*. Nilai tersebut mewakili *balance* saat itu untuk akun yang diberikan dan tipe aset yang diberikan.

DATA record

DATA record digunakan untuk menyimpan data teks suka-suka. Nama record dapat berupa string UTF-8 mana saja yang valid. Ia dapat digunakan untuk menyimpan hal-hal seperti metadata aset, *symbolic links* di dalam sistim akunting tersebut, izin-izin,

atau potongan penting apa pun dari sembarang data lain yang perlu diamankan secara kriptografi.

Pemanggilan `method`

Server Openchain meng-ekspos suatu HTTP API yang dapat digunakan untuk berinteraksi dengan data-nya. URL dari suatu operasi disusun dari base URL dari *endpoint*-nya, dan meng-*concat*-nya dengan path relatif dari operasi yang sedang dipanggil.

Misal, jika base URL-nya adalah `https://www.openchain.org/endpoint/`, untuk memanggil operasi `/record`-nya (meng-query suatu record), URL full-nya hendaknya `https://www.openchain.org/endpoint/record`.

Men-submit suatu transaksi (`/submit`)

Submit-lah suatu transaksi untuk validasi.
Method: POST

Input

Input adalah suatu dokumen JSON yang dilewatkan sebagai bagian dari isi tanggapan tersebut. Format dari dokumen JSON adalah sebagai berikut:

```
{
      "mutation": "<string>",
      "signatures": [
{
      "pub_key": "<string>",
      "signature": "<string>"
      }
   ]
}
```

Deskripsi dari payload tersebut:

- `mutation`: Mutasi *hex-encoded*. Mutasi tersebut diserialkan menggunakan skema Mutation Protocol Buffers.
- `signatures`: Suatu array dari dokumen dengan dua *property*, `pub_key` dan `signature`.
  - `pub_key`: public key yang di-*hex-encoded* yang digunakan untuk me-*sign*.
  - `signature`: tandatangan yang di-*hex-encoded* dari *hash* dari mutasi tersebut.

Proses penandatanganan

Untuk menghasilkan suatu tandatangan:

1. Serialkan mutasi tersebut menggunakan skema Mutation Protocol Buffers.
2. *Hash*-lah string *byte* mutasi tersebut menggunakan SHA256 ganda.
3. Tanda-tanganilah mutasi tersebut dengan private key yang relevan menggunakan `Secp256k1`. Public key yang cocok harus di-*submit* bersama dengan tandatangan tersebut.

Penting: Anda harus men-*submit* string byte yang eksak seperti diperoleh setelah langkah 1. Jika itu dirubah, hash-nya tidak akan cocok dan tandatangan tersebut kemudian menjadi invalid.

Output

Output adalah suatu dokumen JSON yang dilewatkan sebagai bagian dari isi tanggapan.

```
{
      "transaction_hash": "<string>"
}
```

*Fiel*d `transaction_hash` memuat hash yang di-*hex-encoded* dari transaksi penuhnya.

Meng-query suatu record (`/record`)

Query-lah nilai dan versi dari suatu record yang diberikan key-nya.
Method: GET

Input

Input dilewatkan melalui string *query* sebagaimana parameter URL yang di-*encode*.

`Key`	Key yang di-*hex-encode*d dari record yang di-*query*

Output

Output adalah suatu dokumen JSON yang dilewatkan sebagai bagian dari isi tanggapannya. Format dari dokumen JSON adalah sebagai berikut:

```
{
"key": "<string>",
"value": "<string>",
"version": "<string>"
}
```

Field-nya adalah sebagai berikut:

- `key`: Key yang di-*hex-encoded* dari record tersebut.
- `value`: Nilai yang di-*hex-encoded* dari record tersebut.
- `version`: Versi yang di-*hex-encoded* dari record tersebut.

Transaction stream (`/stream`)

Method: GET
Endpoint ini adalah suatu endpoint WebSocket. Ia dapat digunakan untuk menerima semua transaksi yang baru saja dikonfirmasi secara *real-time*.

Input

Input dilewatkan melalui string query sebagai parameter URL yang di-*encode*.

`From`	(**optional**) hash yang di-*hex-encoded* dari transaksi terakhir yang darinya hendak di-*resume*. Jika dihilangkan, ia akan mulai dari transaksi pertama.

Output

Output adalah suatu stream biner WebSocket.
Setiap pesan di dalam stream tersebut adalah transaksi yang diserialkan.

Me-*retrieve* info *chain* (`/info`)

Mendapatkan informasi mengenai instance Openchain.
Method: GET

Input

Metode ini tidak memiliki parameter input.

Output

Output adalah suatu *array* JSON yang dilewatkan sebagai bagian dari isi tanggapan. Format dari array JSON adalah sebagai berikut:

```
{
	"namespace": "<string>"
}
```

`namespace` adalah perwakilan *hex* dari `namespace` yang diharapkan di dalam transaksi yang di-*submit* kepada instance Openchain.

Meng-*query* suatu akun (`/query/account`)

Meng-query semua record `ACC` pada path yang diberikan (secara *non-recursive*).
Method: GET

Input

Input dilewatkan melalui *string* query sebagai parameter URL yang di-*encode*.

`account`	Path untuk di-*query*

Output

Output adalah suatu array JSON yang dilewatkan sebagai bagian dari isi tanggapan. Format dari array JSON adalah sebagai berikut:

```
[
	{
		"account": "<string>",
		"asset": "<string>",
		"balance": "<string>",
		"version": "<string>"
	}
]
```

Isian dari masing-masing item dari array tersebut adalah sebagai berikut:

- `account`: Path dari record tersebut.
- `value`: ID asset dari record tersebut (nama record).
- `balance`: balance untuk ID asset tersebut pada path itu.
- `version`: Versi yang di-*hex-encoded* dari record tersebut.

Meng-query suatu transaksi (`/query/transaction`)

Mendapatkan kembali suatu transaksi yang diberi hash dari mutasi-nya.
Method: GET

Inputs

Inputs dilewatkan melalui string query sebagai parameter URL yang di-*encode*.

`mutation_hash`	Has yang di-*hex-encoded* dari mutasi yang diwakili oleh transaksi tersebut.
`format`	Format output (`raw` atau `json`).

Output

Output adalah suatu dokumen JSON yang dilewatkan sebagai bagian dari isi tanggapan. Format dari dokumen JSON tersebut bergantung pada argumen format:

1. format output `raw` (default):

```
{
"raw": "<string>"
}
```

Properti `raw` memuat transaksi yang diserialkan.

2. format output `json`

```
{
      "transaction_hash": "<string>",
      "mutation_hash": "<string>",
      "mutation": {
      "namespace": "<string>",
      "records": [
            {
                  "key": "<string>",
                  "value": "<string>",
                  "version": "<string>"
            }
      ]
},
      "timestamp": "<string>",
      "transaction_metadata": "<string>"
}
```

Meng-query suatu versi spesifik dari suatu record (`/query/recordversion`)

Mendapatkan kembali suatu versi spesifik dari suatu record.

Method: GET

Input

Input dilewatkan melalui string query sebagai parameter URL yang di-*encode*.

`key`	Key record yang di-*hex-encode*.

Output

Output adalah suatu dokumen JSON yang dilewatkan sebagai bagian dari *isi tanggapan*. Format dari dokumen JSON adalah sebagai berikut:

```
{
      "key": "<string>",
      "value": "<string>",
      "version": "<string>"
}
```

Isiannya adalah sebagai berikut:

- `key`: Key yang di-*hex-encoded* dari record tersebut.
- `value`: Nilai yang di-*hex-encoded* dari record tersebut.
- `version`: Versi yang di-*hex-encoded* dari record tersebut.

Jika versi record tersebut tidak ada, HTTP code 404 akan dikembalikan oleh server tersebut.

Meng-query semua mutasi yang telah dipengaruhi oleh suatu record (`/query/recordmutations`)

Mendapatkan kembali semua mutasi yang telah mempengaruhi suatu record yang diberikan. **Method**: GET

Inputs

Inputs dilewatkan melalui string query sebagai parameter URL yang di-*encode*.

`key`	key dari record yang mutasi-nya diperoleh kembali.

Output

Output adalah suatu dokumen JSON yang dilewatkan sebagai bagian dari isi tanggapan. Format dari dokumen JSON adalah sebagai berikut:

```
[
	{
		"mutation_hash": "<string>"
	}
]
```

Output adalah suatu list yang mewakili semua hash mutasi dari mutasi yang telah mempengaruhi `key` yang diwakili oleh argumen `key`.

Record Query di dalam suatu akun dan sub-akun-nya (`/query/subaccounts`)

Mendapatkan kembali semua record di bawah suatu path yang diberikan (termasuk sub-path). **Method**: GET

Inputs

Inputs dilewatkan melalui string query sebagai parameter URL yang di-*encode*.

`account`	Path yang sedang di-*query*

Output

Output adalah suatu dokumen JSON yang dilewatkan sebagai bagian dari *isi tanggapan*.

Format dari dokumen JSON adalah sebagai berikut:

```
[
	{
		"key": "<string>",
		"value": "<string>",
		"version": "<string>"
	}
]
```

Isiannya adalah sebagai berikut:

- `key`: key yang di-*hex-encoded* dari record tersebut.
- `value`: nilai yang di-*hex-encoded* dari record tersebut.
- `version`: versi yang di-*hex-encoded* dari record tersebut.

Meng-Query semua record dengan tipe dan nama yang diberikan (`/query/recordsbyname`)

Mendapatkan kembali semua record dengan tipe dan nama yang diberikan

Method: GET

Inputs

Inputs dilewatkan melalui string query sebagai parameter URL yang di-encode.

`name`	Name dari record yang sedang di-query.
`type`	Tipe dari record yang sedang di-query.

Output

Output adalah suatu dokumen JSON yang dilewatkan sebagai bagian dari *isi tanggapan*. Format dari dokumen JSON-nya adalah sebagai berikut:

```
[
      {
            "key": "<string>",
            "value": "<string>",
            "version": "<string>"
      }
]
```

Isiannya adalah sebagai berikut:

- `key`: Key yang di-*hex-encoded* dari record tersebut.
- `value`: Nilai yang di *hex-encoded* dari record tersebut.
- `version`: Versi yang di-*hex-encoded* dari record tersebut.

Bab 5

Aturan ledger

Aturan ledger Default

Aturan Global

Suatu transaksi dibuat dari bermacam mutasi record. Mutasi record `ACC` adalah subjek bagi suatu aturan yang *balancing*. Aturan yang *balancing* bekerja sebagai berikut:

1. Untuk setiap record `ACC`, delta antara *balance* yang sebelumnya dan *balance* baru yang diajukan, dikalkulasikan.
2. Jumlah dari semua delta **per asset type** dikalkulasikan.
3. Untuk setiap asset type, jumlahnya harus sama dengan nol.

Ini menjamin setiap penciptaan aset dan penghancurannya, direkam melalui suatu akun di dalam system. Tetapi ini bermakna bahwa setidaknya satu akun harus mampu memiliki suatu *negative balance*. Biasanya, suatu akun khusus digunakan untuk melakukan yang seperti itu, dan kemampuan untuk menciptakan suatu *negative balance* pada suatu akun menghendaki izin khusus.

Tip: Akun penerbit aset pihak ketiga dibolehkan untuk memiliki *negative balances*.

Alias (/aka/<name>/)

Openchain memiliki kemampuan untuk mendefinisikan alias untuk akun, ini menyederhanakan pengalaman user karena user tidak perlu lagi mengingat suatu string karakter acak base-58.

Untuk melakukan itu, *client* hendaknya memahami sintaks berikut sebagai suatu path akun yang valid: `@<name>`, dan mengubahnya secara internal menjadi `/aka/<name>/`.

Contoh

Jika seorang user ingin mengirim dana kepada akun berikut:

```
@bank
```

Aplikasi *client* hendaknya mengubahnya secara internal menjadi:

```
/aka/bank/
```

Goto records (goto)

Goto records adalah record DATA khusus yang menginstruksikan kepada aplikasi *client* untuk menggunakan suatu akun yang berbeda.

Goto records harus memiliki nama khusus `goto`.

Ketika suatu aplikasi client mengirim dana kepada suatu path, pertama ia harus mencari suatu record DATA yang bernama `goto`. Apabila ada, aplikasi client harus menggunakan path yang didefinisikan sebagai nilai dari *instead* record.

Contoh

Jika seorang user ingin mengirim dana kepada akun berikut:

```
/account/alpha/
```

Pertama client harus memeriksa keberadaan dari suatu record dengan key berikut:

```
/account/alpha/:DATA:goto
```

Jika record tersebut tidak ada, maka tidak terjadi apa-apa dan dana dikirim kepada `/account/alpha/`. Jika record tersebut ada, asumsikan nilainya adalah:

```
/account/beta/
```

Kemudian dana sebagai gantinya dikirim kepada `/account/beta/`.

Note: Dimungkinkan dan direkomendasikan untuk alasan keamanan bahwa aplikasi *client* menggunakan suatu *check-only record* dengan record goto untuk memastikan nilai dari record goto masih valid dan belum berubah ketika transaksi tersebut divalidasi.

Asset definition record (`asdef`)

Penting untuk dapat mengasosiasikan informasi dengan suatu tipe aset sehingga user memiliki ekspektasi yang benar mengenai itu.

Asset definition record dapat digunakan untuk me-*record* informasi ini. Asset definition record adalah suatu record DATA dengan nama khusus `asdef`. Sebagai tambahan, ia harus ditempatkan di bawah path yang sama seperti tempat di mana aset itu dilampirkan.

Contoh

Guna mengasosiakan informasi dengan aset yang diwakili oleh path `/asset/gold/`, record berikut harus di-set:

```
/asset/gold/:DATA:asdef
```

Nilai dari record tersebut adalah suatu string UTF-8 yang mewakili suatu dokumen JSON dengan skema berikut:

```
{
      name: '<string>',
      name_short: '<string>',
      icon_url: '<string>'
}
```

Definisi dari isian-isian di atas adalah sebagai berikut:

- `name`: Nama lengkap dari aset tersebut (misal: `U.S. Dollar, Gold Ounce`).
- `name_short`: Nama pendek dari aset tersebut. Ini digunakan untuk mendenominasi jumlah (misal: `USD, XAU`)
- `icon_url`: URL kepada suatu ikon yang mewakili aset tersebut.

Ledger info record (info)

Setiap instance Openchain dapat menyimpan suatu record DATA yang dinamai info pada path root (`/`). Dengan kata lain, key record seharusnya adalah `/:DATA:info`.

Record info mengekspos *meta-information* mengenai ledger itu sendiri. Nilainya haruslah suatu dokumen JSON dengan skema berikut:

```
{
      name: '<string>',
      validator_url: '<string>',
      tos: '<string>',
      webpage_url: '<string>'
}
```

Definisi dari isian-isian di atas adalah sebagai berikut:

- `name`: Nama dari instance Openchain.
- `validator_url`: URL dari validator utama untuk instance Openchain ini.
- `tos`: Syarat dan ketentuan dari instance Openchain tersebut.
- `webpage_url`: Suatu link kepada konten yang *user-readable* di mana user dapat mendapatkan informasi lebih banyak mengenai instance Openchain ini.

Akun Pay-To-Pubkey-Hash (`/p2pkh/<address>/`)

Akun Pay-To-Pubkey-Hash adalah akun khusus dengan izin implisit. Menandatangani suatu dana transaksi belanja dari akun ini sub-akun apa pun menghendaki private key yang cocok dengan `<address>`.

Ini secara otomatis bekerja dengan akun apa pun dari format tersebut, di mana `<address>` adalah suatu base-58 address yang valid.

Note: `<address>` adalah suatu base-58 address yang dikonstruksi di dalam cara yang sama dengan suatu *address* Bitcoin untuk private dan public key-nya.

Akun penerbit aset pihak ketiga (`/asset/p2pkh/<address>/`)

Akun penerbit aset pihak ketiga (third-party asset issuance account) adalah akun khusus dengan izin implisit. Pemilik dari private key yang cocok dengan `<address>` dapat menandatangani transaksi dana belanja dari akun ini. Dana harus menjadi tipe aset `/asset/p2pkh/<address>`. Juga, address ini di-otorisasi untuk memiliki suatu *negative balance*. Artinya dimungkinkan untuk menggunakan address ini sebagai sumber penerbit dari tipe aset `/asset/p2pkh/<address>`.

Ini secara otomatis bekerja dengan akun apa pun dari format tersebut, di mana `<address>` adalah suatu base-58 address yang valid.

Note: `<address>` adalah suatu base-58 address yang dikonstruksi di dalam cara yang sama dengan suatu address Bitcoin untuk private dan public key-nya.

Dynamic permissions

Openchain mendukung suatu *layout* izin implisit melalui akun P2PKH (`/p2pkh/<address>/`) dan akun penerbit pihak ketiga (`/asset/p2pkh/<address>/`). Juga dimungkinkan untuk secara dinamis mendefinisikan izin dengan men-submit transaksi yang merubah record khusus: record `acl`.

Access Control Lists

Izin diterapkan kepada suatu path khusus. Untuk menerapkan suatu *access control list* kepada suatu path, set-lah record `acl` di bawah path itu. Ia haruslah suatu record DATA. Nilainya adalah suatu file JSON.

Misal, ketika mencoba untuk men-set izin bagi path `/users/alice/`, record berikut harus di-set: `/users/alice/:DATA:acl`.

Skema

Skema dari file JSON yang record tersebut miliki adalah sebagai berikut:

```
[
	{
		"subjects": [
			{
				"addresses": [ "<string>" ],
				"required": <integer>
		}
		],
		"recursive": <boolean>,
		"record_name": "<string>",
		"record_name_matching": "<record-matching-type>",
		"permissions": {
			"account_negative": "<permission>",
			"account_spend": "<permission>",
			"account_modify": "<permission>",
			"account_create": "<permission>",
			"data_modify": "<permission>"
		}
	}
]
```

Konten adalah suatu array yang memuat semua izin yang dapat diterapkan. Ketika record `acl` tidak ada, ini sama dengan memiliki empty array.

Makna dari isian di dalam suatu objek izin adalah sebagai berikut:

- `subjects`: Suatu array dari subjek di mana objek izin diterapkan.
  - `addresses`: Suatu *array* dari string yang mewakili address di mana tandatangan diharapkan.
  - `required`: Jumlah dari tandatangan yang diperlukan dari array address. Apabila array address memuat 3 address, dan `required` di-set kepada 2, itu bermakna bahwa agar izin dapat diterapkan, setidaknya 2 tanda-tangan dari 3 address yang di-spesifikasikan harus ada. Ini dikenal sebagai suatu skema multi-signature `n-of-m`.

- `recursive`: (Default: true) Suatu boolean yang mengindikasikan apakah izin diterapkan secara rekursif kepada sub-akun.

Note: Dengan rekursi, izin di tingkat yang lebih rendah mengesampingkan izin di tingkat yang lebih tinggi.

- `record_name`: (Default: empty string) Pola untuk digunakan bagi nama record yang cocok.
- `record_name_matching`: (Default: `Prefix`) Tipe dari nama record yang cocok untuk digunakan. Ada dua yang mungkin:
  - `Exact` bermakna bahwa nama record harus sama pasti dengan nilai dari isian `record_name` agar izin dapat diterapkan.
  - `Prefix` bermakna bahwa nama record harus dimulai dengan nilai dari isian `record_name` agar izin dapat diterapkan. Menggunakan Prefix dengan suatu `record_name` yang empty bermakna bahwa izin tersebut diterapkan kepada semua record.

Petunjuk: Nama record dari suatu record `ACC` adalah path aset-nya.

• `permissions`: Memuat izin yang diterapkan jika objek izin ini cocok. Arti dari bermacam izin dijelaskan di dalam seksi berikutnya. Nilai harus di-set kepada `Permit` agar izin dikabulkan, atau `Deny` agar izin ditolak. Jika ia tidak di-set, nilai yang diwarisi digunakan.

Permissions

account_negative

Izin ini mengindikasikan hak untuk mempengaruhi balance dari record `ACC`, baik itu meningkatkannya (menerima dana) dan menurunkannya (mengirim dana) tanpa restriksi pada balance final. Jika izin ini dikabulkan, balance record `ACC` dapat dibuat negatif. Izin ini secara khas dikabulkan bagi user untuk membolehkan penerbitan aset.

account_spend

Izin ini mengindikasikan hak untuk mempengaruhi balance dari record `ACC`, baik itu meningkatkannya (menerima dana) dan menurunkannya (mengirim dana) dengan restriksi bahwa balance final harus tetap positif atau nol.

account_modify

Izin ini diperlukan untuk mempengaruhi balance dari record ACC yang telah dirubah sebelumnya (versi record adalah non-empty).

account_create

Izin ini diperlukan untuk mempengaruhi balance dari record ACC yang tidak pernah dirubah sebelumnya (versi record adalah empty).

Note: Seorang user hanya dapat mengirim dana ke suatu akun jika dia memiliki hak `account_negative` atau `account_spend` plus hak `account_modify` atau `account_create`. Mengirim kepada suatu akun memerlukan `account_modify` atau `account_create` pada akun tujuan. Suatu ledger closed loop dapat diciptakan dengan menolak `account_modify` dan `account_create` secara default, dan secara selektif mengabulkan izin-izin tersebut bagi beberapa akun. Dengan melakukan ini, hanya akun yang diizinkan yang dapat menerima dana.

---

data_modify

Izin ini diperlukan untuk merubah suatu record DATA.

How to: Mengkonfigurasi suatu ledger untuk menjadi closed-loop

Institusi dan perusahaan finansial yang membiarkan user mereka untuk men-transfer nilai seringkali berurusan dengan regulasi yang menghendaki institusi dan perusahaan itu untuk “mengenal para pelanggan mereka” (KYC).

Dimungkinkan menggunakan Openchain dalam konfigurasi ini dengan sedikit saja upaya. Seksi ini menggambarkan langkah-langkah yang perlu.
Tujuan dari penjelasan ini adalah mengkonfigurasi Openchain sehingga:

1. User melalui suatu proses registrasi eksternal di mana mereka menyerahkan identitas mereka untuk di-verifikasi oleh perusahaan yang meng-administrasi ledger tersebut, dan menautkan identitas mereka dengan suatu public key.
2. Saat public key cocok dengan user yang terdaftar, dapat digunakan untuk mengirim dana.
3. Dana hanya dapat dikirim kepada user yang teregistrasi.

Dengan cara ini, dana hanya dapat beredar di antara user yang “dikenal”.

Konfigurasi awal

Instance Openchain harus dikonfigurasi dengan akun P2PKH dan akun penerbit pihak ketiga di-*disable*.

Setting `validator_mode:validator:allow_p2pkh_accounts` dan `validator_mode:validator:allow_third_party_assets` keduanya harus di-set kepada `false` untuk mencapai ini. Lihat bagian ini untuk rincian lebih lanjut.

Dengan konfigurasi ini, secara default, user tidak memiliki hak, sementara administrator memiliki semua hak. Tidak mungkin bagi user biasa untuk mengirim atau menerima token.

Proses *Onboarding*

Langkah kedua adalah membangun suatu *workflow* onboarding bagi user. Misal, ini dapat berupa suatu aplikasi *mobile* di mana user menciptakan suatu username dan password, memasukkan alamat email dan men-*submi*t suatu bukti identitas (foto dari kartu identitasnya).

Sebagai bagian dari proses tersebut, suatu private key di-*generate* dan disimpan pada perangkat user. Public key yang cocok dikirim bersama dengan potongan informasi lain. Bagian ini dapat secara keseluruhan tidak nampak bagi user.

Menciptakan hak akses

Pada saat perusahaan telah memvalidasi identitas user tersebut, perusahaan itu dapat menciptakan suatu akun pada Openchain untuk user tersebut, dan menautkan username dengan public key-nya.

Alias didasarkan pada suatu path khusus (`/aka/<alias>/`). Asumsikan bahwa username dari user tersebut adalah alice, kita perlu untuk:

1. Membolehkan user lain untuk mengirim dana kepada `/aka/alice/` (dan sub-akun).
2. Membolehkan public key-nya Alice untuk digunakan membelanjakan dana pada `/aka/alice/` (dan sub-akun).

Ini dapat dicapai dengan menciptakan suatu record `acl` di bawah `/aka/alice/`.

Tip: Lihatlah dokumentasi mengenai dynamic permissions untuk rincian lebih.

Record `/aka/alice/:DATA:acl` harus diciptakan dan di-set kepada:

```
[
 {
   "subjects": [ { "addresses": [ ], "required": 0 } ],
   "permissions": { "account_modify": "Permit", "account_create": "Permit" }
 },
 {
   "subjects": [ { "addresses": [ "<alices-address>" ], "required": 1 } ],
   "permissions": { "account_spend": "Permit" }
 }
]
```

Penting: Karena hanya seorang administrator yang memiliki hak untuk merubah record ini, mutasi yang menciptakan record ini harus ditandatangani menggunakan suatu key administrator.

Address Alice adalah perwakilan base-58 dari hash dari public key-nya. Address tersebut dikonstruksi dengan cara yang sama dengan suatu address Bitcoin.

Dengan men-*tweak* access control list tersebut, dimungkinkan untuk:

1. Menangani bermacam perangkat (dengan key yang berbeda) per user.
2. Menerapkan skema multisignature, misalnya untuk akun-akun yang digunakan bersama.

Menyalurkan kredit kepada akun user

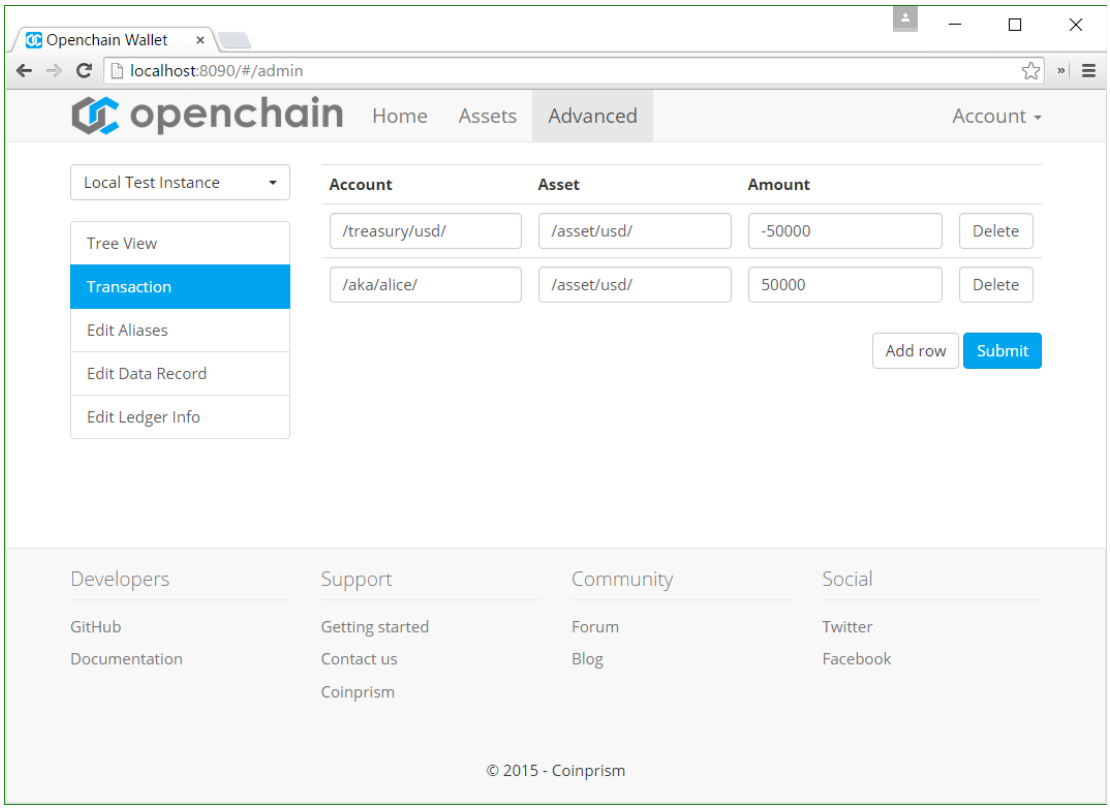
Sekarang user tersebut memiliki suatu akun dia dapat menggunakannya, dia ingin mengisi akun itu dengan dana. Ada banyak konfigurasi yang dimungkinkan untuk ini:

- Suatu perbendaharaan awal diciptakan oleh perusahaan tersebut dan kredit dikirim dari perbendaharaan tersebut.
- Token diterbitkan secara dinamis kapan pun user membeli token itu melalui suatu metode pembayaran eksternal.

Asumsikan yang berikut:

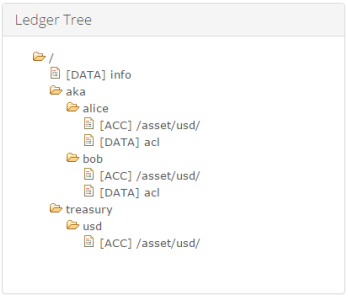
- Path aset untuk token tersebut adalah `/asset/usd/` (ini dapat secara suka-suka dipilih).
- Token tersebut secara dinamis diterbitkan dari akun `/treasury/usd/`.

Suatu transaksi pendanaan akan mengambil bentuk suatu transaksi pengiriman X unit dari aset tersebut `/asset/usd/` dari akun `/treasury/usd/` kepada akun `/aka/alice/`.



Transaksi tersebut hendaknya ditandatangani oleh seorang administrator dan hanya seorang administrator yang memiliki akses kepada `/treasury/usd/`. Balance pada `/treasury/usd/` akan menjadi negatif, dan merefleksikan jumlah total dari token-token yang telah diterbitkan pada ledger. Lagi, administrator dibolehkan untuk membuat balance tersebut, negatif.

Tampilan final ledger dalam tampilan menurun seharusnya terlihat sebagai berikut:



Hilangnya address dan pencurian private key

Terkadang tidak dapat terhindarkan, beberapa user kehilangan perangkat yang di situ private key mereka disimpan.

Ketika ini terjadi, mereka hendaknya melaporkan itu kepada perusahaan yang mengelola instance Openchain. Perusahaan tersebut pertama-tama melakukan pemeriksaan identitas, kemudian meminta user tersebut untuk men-*generate* suatu key baru pada suatu perangkat baru.

Administrator kemudian dapat meng-*update* record `acl` yang relevan untuk merubah address yang sebelumnya menjadi address baru, yang cocok dengan key baru tersebut.

Menangani transaksi curang

Apabila transaksi curang terjadi pada saat kehilangan tersebut, administrator dapat meng-*commit* suatu transaksi baru yang mewakili transfer yang berkebalikan.

Misal, jika 10 unit telah dikirim secara curang dari `/aka/alice/` kepada `/aka/oscar/`, maka administrator dapat men-*submit* suatu transaksi baru yang mengirimkan 10 unit dari `/aka/oscar/` kepada `/aka/alice/`, dengan demikian membalik efek dari transaksi curang tersebut. ledger ini bersifat kekal, kedua transaksi tetap dapat dilihat di dalam ledger, dengan fakta bahwa transaksi kedua mentransfer balik dana dari `/aka/oscar/` dengan tidak ditandatangani oleh key-nya Oscar, tetapi sebagai gantinya ditandatangani oleh key administrator.

Note: Benar-benar harus dilakukan bahwa di dalam pendaftaran di mana semua user harus melalui proses verifikasi identitas, pertama-tama harus diyakini bahwa Oscar tidak mungkin mencuri dana dari Alice, karena perusahaan yang mengelola ledger memiliki semua informasi mengenai Oscar, dan dapat menuntutnya.

Kesimpulan

Dengan pendaftaran ini, user dapat mengirim token satu sama lain, tetapi, mereka tidak dapat mengirim dana kepada address yang tidak tertaut kepada seorang user yang tidak teregistrasi.

Contoh ini hanya mewakili satu cara untuk mengimplementasikan suatu ledger closed-loop, dan banyak lagi konfigurasi lain yang dimungkinkan bergantung pada keperluan.