ブロックチェーンを使ってみよう！ 信頼を作る取引の仕組み

それぞれの要素はどう実装されているのか？

ブロックチェーンとは「台帳」を分散共有して管理する技術であり、その技術的要素として「分散台帳」「コンセンサス」「暗号化」「スマートコントラクト」の 4 つがあります。本稿では、簡単なブロックチェーンのデモ環境を立ち上げ、ブロックチェーンがどのように動いているのかを確認してみましょう。また、後半では、このブロックチェーンを使った市場取引の例も紹介します。

○本稿の前提環境について

デモで使用する環境は「LinuxONE Community Cloud（http://www-03.ibm.com/systems/jp/linuxone/solutions/enterprise-apps.html#Enterprise-apps）」の Red Hat Enterprise Linux7.2（以下 RHEL7.2）の仮想マシンです。

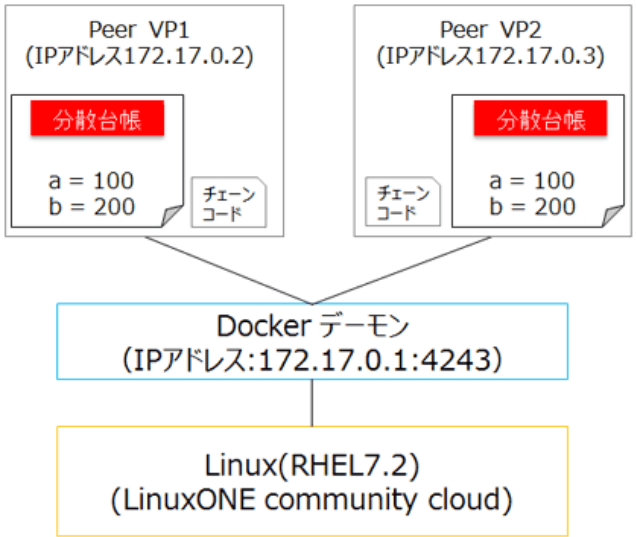
○無償メインフレーム Linux でブロックチェーンの Docker イメージを展開する

OS の設定をした後、既にブロックチェーンの環境が整っている Docker コンテナを起動し、簡単なブロックチェーンの機能を試してみます。具体的には図 1 のような環境を作ってみます。

なお、このデモは、以下の Docker 公式サイトにある「brunswickheads/openchain-peer」のドキュメントを参考にしています。

- **brunswickheads/openchain-peer**
  https://hub.docker.com/r/brunswickheads/openchain-peer/

図 1　デモ環境の構成



ブロックチェーンピアを作成して動作を検証する方法

ここからは、LinuxOne Community Cloud 上にオープンソースのブロックチェーンクライアントの Docker イメージを展開していきます。

手順 1　iptables と firewalld を停止

今回はテスト目的ということで、想定外のエラーなどをあらかじめ回避するため、iptables と firewalld を無効にします。操作は systemctl コマンドを使って行います。

なお、本稿では一部の長いコマンドを「\」（バックスラッシュ）で改行して掲載しています。実際に操作する際には、行末の¥を省き、1 行のコマンドとして入力してください。

```
[linux1@rhel72 ~]$ sudo systemctl stop iptables
[linux1@rhel72 ~]$ sudo systemctl disable iptables
[linux1@rhel72 ~]$ sudo systemctl stop firewalld
[linux1@rhel72 ~]$ sudo systemctl disable firewalld
```

手順 2　sudo へのパスの引き継ぎ設定

デフォルトでは、システム管理者権限（sudo）実行時に「/sbin」「/bin」「/usr/sbin」「/usr/bin」をパスとして認識してしまいます。ここでは、一般ユーザー（linux1）環境のパスを認識するための設定を、visudo コマンド経由で行います。

```
[linux1@rhel72 ~]$ sudo visudo
```

以下を追記します。

```
Defaults    env_keep += "PATH"
```

さらに、以下の行をコメントアウト（行頭に#を追記）します。

```
Defaults    secure_path = /sbin:/bin:/
usr/sbin:/usr/bin
```

手順 3　Docker のダウンロード

RHEL 7.2 ではデフォルトで Docker がインストールされないため、Docker の実行環境を以下の UNICAMP の ftp サイトから wget コマンドを使ってダウンロードします（ダウンロードデータの健全性は個別に確認しておくようにしてください）。

• docker-1.10.1-rhel7.2-20160408.tar.gz

ftp://ftp.unicamp.br/pub/linuxpatch/s390x/redhat/rhel7.2/docker-1.10.1-rhel7.2-20160408.tar.gz

```
[linux1@rhel72 ~]$ sudo wget ftp://ftp.
unicamp.br/pub/linuxpatch/s390x/redhat/
rhel7.2/docker-1.10.1-rhel7.2-20160408.
tar.gz
```

ダウンロードした Docker の実行環境を解凍します。

```
[linux1@rhel72 ~]$ sudo tar xvzf
docker-1.10.1-rhel7.2-20160408.tar.gz
```

手順 4　シンボリックリンクの作成

解凍した Docker 実行環境内の実行ファイルに「リンクを貼り」ます（シンボリックリンク）。これは、Windows 環境でいうところの「デスクトップにアプリのショートカットを作成する」といった操作です。

```
[linux1@rhel72 ~]$ sudo ln -s /home/
linux1/docker-1.10.1-rhel7.2-20160408/
docker \
/usr/local/bin
```

○ Docker を起動してブロックチェーンのピアを操作する

ここからは、Docker 上のブロックチェーンピアを起動して、挙動を確認していきます。2 つのピアを立ち上げ、台帳のデータを更新したときに、チェーンのように連動してそれぞれが持つ値が変化するかを確かめてみます。

では早速、Docker デーモンを起動していきます。Docker そのものの詳細については別途関連の解説をご確認ください。ここでは必要な操作のみを解説していきます。

手順 5　Docker デーモンの起動

Docker をバックグラウンドプロセスとして起動します。

```
[linux1@rhel72 ~]$ sudo docker daemon -D
-g /var/lib/docker -r=true \
--api-enable-cors=true -H tcp://0.0.0.0:4243
-H unix:///var/run/docker.sock
```

手順 6　1 つ目のピア（VP1）を起動

まず、ifconfig コマンドを使って、ネットワークデバイス docker0 の IP アドレスを確認します。

```
[linux1@rhel72 ~]$ ifconfig docker0
  docker0: flags=4163<UP,BROADCAST,RUNNIN
G,MULTICAST>  mtu 1500
            inet 172.17.0.1   netmask
255.255.0.0  broadcast 0.0.0.0
```

この出力から、docker0 は「inet 172.17.0.1」だと分かりました。これをパラメーターとして、1 つ目のピア（VP1）を起動します。以下のコマンドって <ipaddress> の部分に docker0 の IP アドレス（ここでは 172.17.0.1）を入力します。お使いの環境によって異なりますので適宜入力してください。

```
[linux1@rhel72 ~]$ sudo docker run --rm
-it --name=vp1 -e \
OPENCHAIN_VM_ENDPOINT=http://<ipaddre
ss>:4243 -e \
OPENCHAIN_PEER_ID=vp1 -e \
OPENCHAIN_PEER_ADDRESSAUTODETECT=true \
```

```
brunswickheads/openchain-peer obc-peer
peer
```

手順 7　別のセッションから 2 つ目のピア（VP2）を起動

手順 6 までのセッションとは別に、新しく SSH クライアントのセッションを立ち上げて以下の操作を行います。

<ipaddress> の部分は docker0 の IP アドレス（172.17.0.1）を、<vp1ip> の部分には VP1 の IP アドレス（172.17.0.2）を入力します。

```
[linux1@rhel72 ~]$ sudo docker run --rm
-it -e \
OPENCHAIN_VM_ENDPOINT=http://<ipaddre
ss>:4243 -e \
OPENCHAIN_PEER_ID=vp2 -e \
OPENCHAIN_PEER_ADDRESSAUTODETECT=true -e \
OPENCHAIN_PEER_DISCOVERY_
ROOTNODE=<vp1ip>:30303 \
brunswickheads/openchain-peer obc-peer
peer
```

○チェーンコードを使って値を設定、チェーンコードを取得する

ここまでの作業でブロックチェーンのピアが 2 つできました。ここからは、ピアに具体的な台帳データを登録し、それを取引操作した場合の挙動を確認していきます。なお、ここではサンプルのチェーンコードを利用してデータを登録していきます。

手順 8　Bash でプロンプト操作を可能にする

手順 6 および手順 7 とは別に、新しく SSH クライアントのセッションを立ち上げます。新しいセッションから、以下の操作を行います（1 つ目のピア（VP1）でコマンドを実行できる環境に移る）。

```
[linux1@rhel72 ~]$ sudo docker run --rm
-it -e \
OPENCHAIN_PEER_ADDRESS=172.17.0.2:30303 \
brunswickheads/openchain-peer bash
```

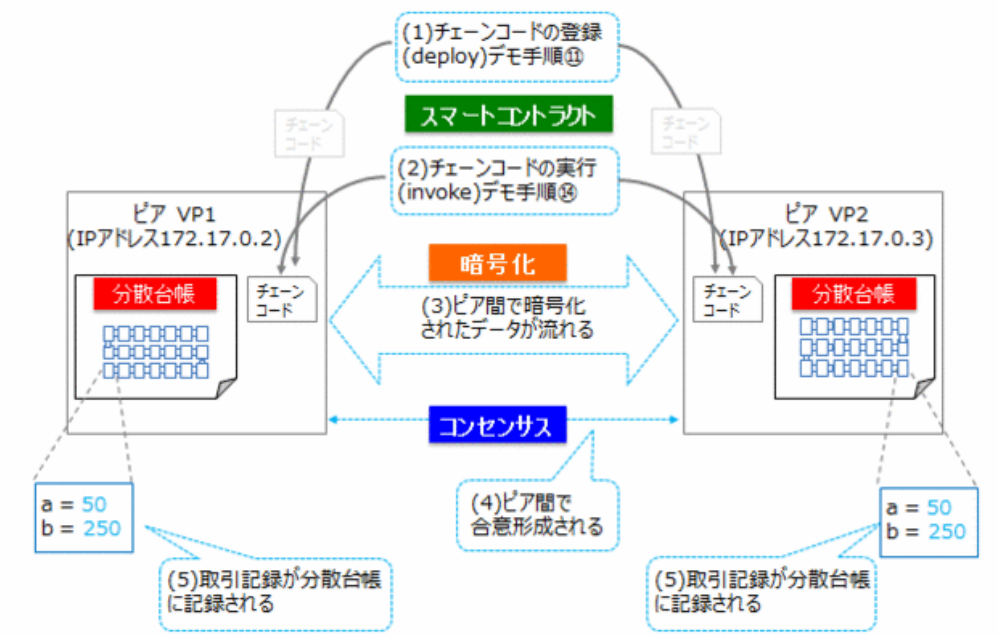
手順 9　チェーンコードを登録（deploy）

GitHub にあるサンプルのチェーンコードをピアに登録します。

このチェーンコードで「a」に「100」、「b」に「200」の値を設定します。

```
bash-4.2# obc-peer chaincode deploy -p \
github.com/openblockchain/obc-peer/
openchain/example/chaincode/chaincode_
example02 -c \
'{"Function":"init", "Args": ["a","100",
"b", "200"]}'
```

図 2　今回のデモ操作でのブロックチェーンの動き



実行結果として以下のような文字列が返されます。出力された文字列が「チェーンコード名」です。以降の操作ではこのチェーンコード名をパラメータとして用います。

```
bb540edfc1ee2ac0f5e2ec6000677f4cd1c6728
046d5e32dede7fea11a42f86a6943b76a8f9154
f4792032551ed320871ff7b7076047e41842
92e01e3421889c
```

手順 10　チェーンコード名を環境変数に登録

手順 9 で出力されたチェーンコード名を使いやすくするため、環境変数（ここでは NAME1）に登録します。<name1> に手順 9 で返された長い文字列を指定して登録します。

```
bash-4.2# export NAME1=<name1（チェーンコード名）>
```

○台帳の中身を確認する

ここまでの操作で、「a」「b」に値を設定し、その際のチェーンコード名を取得できました。

手順 11　台帳の中身を参照する（query）

では実際に、台帳に値が設定されているかを確認してみます。

```
bash-4.2# obc-peer chaincode query -l
golang -n $NAME1 -c \
'{"Function": "query", "Args": ["a"]}'
```

実行結果として a に設定されている値の 100 が返されます。

```
bash-4.2# obc-peer chaincode query -l
golang -n $NAME1 -c \
'{"Function": "query", "Args": ["b"]}'
```

実行結果として b に設定されている値の 200 が返されます。

○台帳の中身を操作して挙動を確認する

それでは早速、台帳の中身を更新して、あらめて参照してみます。

手順 12　台帳の中身を更新する（invoke）

では、a から b に値を 50 移動させてみましょう。

```
bash-4.2# obc-peer chaincode invoke -n
$NAME1 -c \
'{"Function": "invoke", "Args": ["a",
"b", "50"]}'
```

手順 13　台帳の内容を参照する（query）

先ほどと同じように、台帳の値を確認してみます。

```
bash-4.2# obc-peer chaincode query -l
golang -n $NAME1 -c \
'{"Function": "query", "Args": ["a"]}'
```

実行結果として a に設定されている値の 50 が返されます。

```
bash-4.2# obc-peer chaincode query -l
golang -n $NAME1 -c \
'{"Function": "query", "Args": ["b"]}'
```

実行結果として b に設定されている値の 250 が返されます。

今回のデモを通して、ピアにチェーンコード（値を移動させるプログラム）を登録し、それを実行することでピア間の合意形成を図り、共有分散台帳に値の変動が記録されるという動きを確認できたと思います。

LinuxONEでブロックチェーンを実装しているビジネス

ここまでのデモで見た内容は、「値を登録する」「値を移す」「値を参照する」といった基本的な動きでしたが、チェーンコードを開発することで、さまざまな動きを実装できます。実際にブロックチェーンをビジネスに応用し、セキュアなトレーサビリティを実現している企業に Everledger（https://youtu.be/IMfmrF7vun8）があります。

○ダイヤモンドの所有権管理

――「市場」としての信用を得るために

Everledger は LinuxONE 上でブロックチェーンのセキュアな共有分散台帳技術を用いてダイヤモンドの所有権を管

理しています（図 3）。

ダイヤモンドは「形状」「色」「重さ」「透明度」などをデジタルデータ化して Everledger のブロックチェーンネットワークに登録します。これにより、売り手や買い手、保険会社や宝石店は取引情報をリアルタイムに参照できるようになります。

登録されているダイヤモンドは全ての取引情報が共有されるため、例えばネットオークション等で売り出されたとしても、売主とダイヤモンドの所有者が一致しなければ盗難品であると判断されます。

Everledger は保険会社と共同して盗難品と判断されたダイヤモンドに対して必要な処置を迅速に行うことが可能です。現在 Everledger のブロックチェーンネットワークでは 85 万以上のダイヤモンドが管理されています。

2016 年 9 月 20 日にラスベガスで開催されたイベント「IBM Edge2016」で、Everledger の CEO リアナ・ケンプ氏が講演を行いました。そのセッションでケンプ氏は、「ダイヤモンドの業界では、“ それがどこから来たのか ”――出どころがとても重要だ」「かつては信用が大切にされ、それによって成り立ってきた業界だが、ゴールドのように広く投資資金を呼び込むには、テクノロジーによる取引の高い透明性が求められている」と話しています。そこで、Everledger はブロックチェーンテクノロジーを活用することで、より透明性の高いトレーサビリティを実現しているのです。

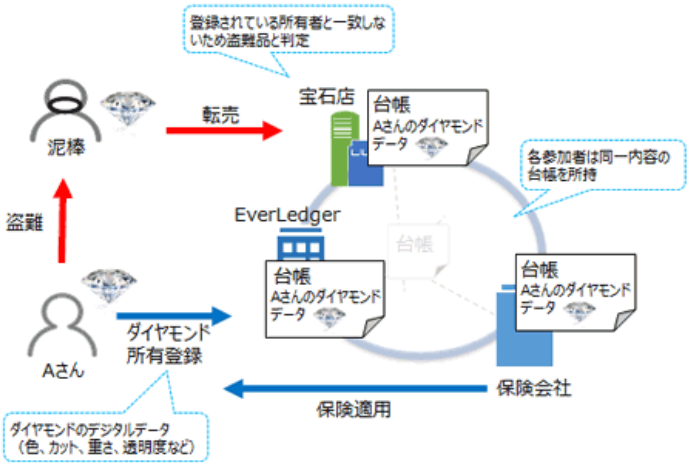
ブロックチェーンは金融業界、企業間の取引だけでなく、トレーサビリティにも活用することができます。ビジネスのさまざまな分野で活用されるブロックチェーンですが、ビジネスでの適応にはやはりセキュアな環境が必要になります。

○「High Security Business Network」によるトレーサビリティの確保

IBM はその環境として BlueMix で展開している「blockchain-as-a-Service」に加え、新たなブロックチェーン向けクラウドサービス「High Security Business Network（以下 HSBN）（https://www-01.ibm.com/marketing/iwm/dre/signup?source=urx-13542&S_PKG=ov57365&disableCookie=Yes&lang=ja_JP）」を発表しました。LinuxONE 上で動く HSBN は、よりビジネスのセキュリティに特化し、個別に隔離された高度なセキュリティ環境でブロックチェーンを用いたビジネスネットワークのシミュレートが可能になります。

LinuxONE は他に類を見ない高いセキュリティレベルを確保し、新たなビジネス基盤となるブロックチェーンを実装するのに最も適した IT インフラといえるでしょう。

図 3　EverLedger におけるダイヤモンドのトレーサビリティ



著者：栗村 彰吾

さまざまなオープンソースソフトウェア（OSS）に携わり、現在は「LinuxONE」を中心に、提案および構築を担当。お客さまのミッションクリティカルな業務にふさわしいシステムの提供に尽力する傍ら、LinuxONE の活用に関するセミナー講師として「メインフレームの価値」や「なぜ今メインフレームか」を啓蒙する活動にも従事する。

お問い合わせ先▶ **日本アイ・ビー・エム株式会社**　〒103-0015 東京都中央区日本橋箱崎町19-21　http://ibm.biz/linuxone_jp
IBMアクセスセンター　0120-550-210

※この冊子は、キーマンズネット（http://www.keyman.or.jp/）に 2016 年 11 月に掲載されたコンテンツを一部改訂・再構成したものです。　http://www.keyman.or.jp/kc/30009303/