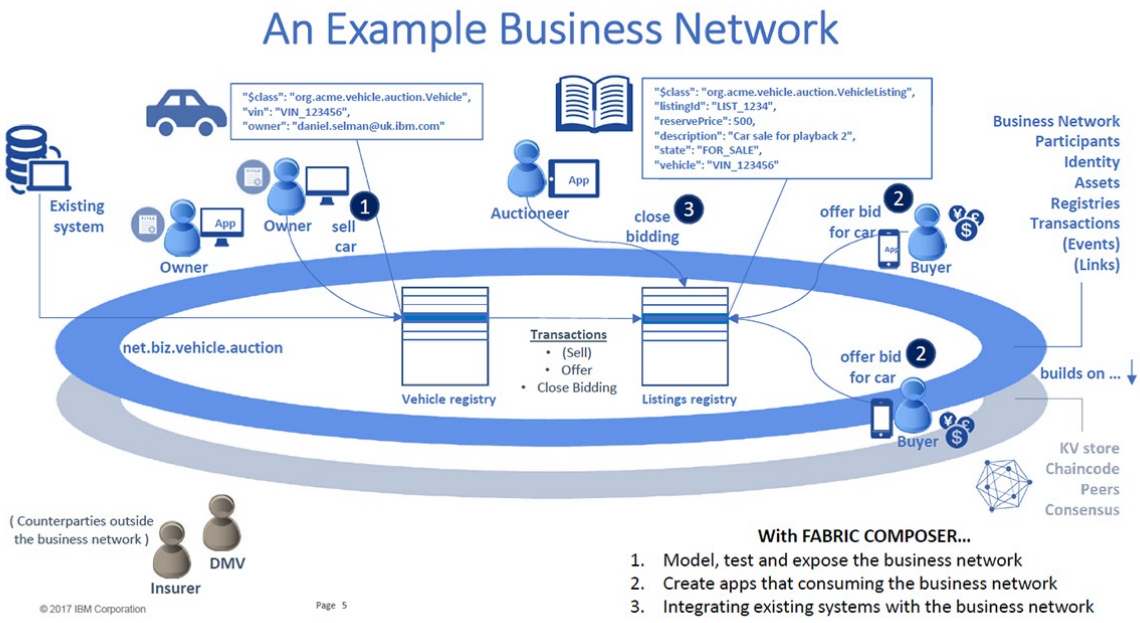
Distributed optimization within blockchain system based on Hyperledger Composer.

1.Introduction.
Hyperledger is the open blockchain system created and backed by the Linux Foundation. The version used to develop the current soultion is the Hyperledger Composer.

1.1. Description of Hyperledger Composer.
Hypeledger had till November 2017 7seven versions of the platform. Every version is intended to solve an specific scope of the potential solutions a blockchain system can offer. Hyperledger Fabric is the version created to serve as a base for development of industrial and trading solutions. IBM, a supporter and developer of the system, created a version over the Hypeledger Fabric called Composer which is intended to make easy and fast the deployment of blockchain systems. The main advantage of the Composer is to provide a REST API that can be interacted by http commands such as POST or GET with json objects. This permits to expand the options to interact with the blockchain such as mobile and web applications.

1.2. Structure of Hyperledger Composer.
Hyperledger Composer is installed over a Hyperledger Fabric network. This means that from the point of view of Fabric, Composer is a chain code (a piece of software routine installed over the Fabric Network). Also, Hyperledger Fabric is installed over a Docker Network, which in turns means for Docker, Hypeledger Fabric is just a running service or application. Docker is a network of interconnected “containers”. A container is a piece of software installed over the kernel of the Host OS that allows to install and run applications without interfering or depending on it. A container can be understood as a Virtual Machine of small size. Also a fundamental concept of containers are the images. An image is a template that defines the functions and thus the behaviour of the container. These images can be downloaded from the open repository or they can be constructed to meet specific needs.



2. Description of the developed system.
The Docker network is based in docker-composer.yaml files where every single participant of the network with their specific functions are described. This files.
To deploy the Composer network three files are needed:

1. Architecture of the network. A file with CTO extension where the structure of the database (blockchain) is described in terms of templates of the all participants of the system. A participant can be a user, an idustry, a comodity, a currency, etc.
2. Permissions for each participant. A file with ACL extension where the permissions to access, read, write or erase values to the blockchain are specified. Only participants that have certain characteristics can be assigned a permission (i.e. commodities cannot be assigned).
3. Logic of operation. A js file. Here is established how the blockchain system is going to operate. This is a top application that in fact is a chain code running over other chaincode (Composer).

Related to the original 3 factories complex: the system takes every time a production update from a factory that change its current value of production X0 to X1. Since the equation for global regulation states Lambda = ½ (Update production 1 + Update production 2 + Update production 3) is a linear function of the updates, it is valid to update the productions sequentially. Then, when one factory changes its current production value, the new Labmda equals the half of that update. This is set in the file logic. js.

The value of lambda is published in the chain so that every peer participant can read that value to computed their own local optimization which in turn will produce (eventually) another value for production that once again will change the value of the global regulator. This process of iterations will continue until the local optimizer for each factory stop updating their productions i.e. locally and globally optima are reached.

To perform the last step, the chain is read form each peer with Python and the algorithm of the local optimization also can write back to the chain the updates of production.

3. Sequence to deploy and test the system.

3.1. Installing all the requirements.

The web site of Hyperledger Github enlist the following as requirements needed:

- Operating Systems: Ubuntu Linux 14.04 / 16.04 LTS (both 64-bit), or Mac OS 10.12
- Docker Engine: Version 17.03 or higher
- Docker-Compose: Version 1.8 or higher
- Node: 8.9 or higher (note version 9 is not supported).
- npm: v5.x
- git: 2.9.x or higher
- Python: 2.7.x
- VSCode or other text editor.

- 3.1.1. Sequence of installation:
    - `npm install -g composer-cli`
    - `npm install -g generator-hyperledger-composer`
    - `npm install -g composer-rest-server`
    - `npm install -g yo`

For Ubuntu users, to download the requirements the next commands can be used:

```
curl -O https://hyperledger.github.io/composer/prereqs-ubuntu.sh

chmod u+x prereqs-ubuntu.sh
```

And then execute this script:

```
./prereqs-ubuntu.sh
```

To download the fabric tools, sample scripts and other needed material, execute the next commands:

```
mkdir ~/fabric-tools && cd ~/fabric-tools

curl -O https://raw.githubusercontent.com/hyperledger/composer-tools/master/packages/fabric-dev-servers/fabric-dev-servers.zip
unzip fabric-dev-servers.zip
```

After executing the previous script, a folder fabric-tools will be downloaded in the Home folder. Inside that folder, some scripts will be found.

```
TeardownAllDocker.sh
downloadFabric.sh
teardownFabric.sh
startFabric.sh
stopFabric.sh
createPeerAdminCard.sh
```

The basic architecture of Fabric implies the existence of a central database (CouchDB), a Certification Authority and an Orderer as administrators of the network. Under their governance there can be as many peers as defined by the system.

The first time to run the system there is needed to run this two files:

./startFabric.sh script creates a basic instantiation of a Fabric network composed of:

- One peer: peer0.org1.example.com
- A DB couchdb
- A certification authority ca.org1.example.com
- An orderer.example.com
- And a network called composer_default

The script takes care of providing the correct images and services each component needs. It is important to mention that the members enlisted before are DOCKER CONTAINERS that belongs to a Docker Network that runs locally in the host machine.

After this it is needed to run the script

./createComposerProfile.sh.

This scripts starts the network by starting the services inside each container.

The identities of the participants are managed by cards. Each card contains the crypto credentials composed of the .pem file pubic key and the _sk private one created for the participant by the encrypting binaries. It is important to notice that every Composer sample has to run inside the fabric-tools folder to be able to use the tools and then to run. In the location fabric-tools/fabric-samples/bin you can find the standard binaries usde by fabric to create the crypto credentials. Since the system needs an administrator for the network, the script:

```
./createPeerAdminCard.sh
```

Creates a folder with the credentials for this user. Those files are needed to build an admin card. Since this process was already done and the card was already created, there is only need to include it on the cards registry of the system. This registration is needed every time the services are stated up.

If the system was stared before or if another Fabric network was deployed previously, to avoid conflicts caused by duplication of names among containers ore networks, the following two files should be executed:

```
./teardownFabric.sh
```

```
./teardownAllDocker.sh
```

After these all containers and images are eliminated and a clean start up can be made.

3.2 Sequence of starting up the services

3.2.1 Download the files.

Since the network was already created, there is only need to deploy the system then to start it. So first the folder production-network should be downloaded and unzipped inside the fabric-tools folder. The final result after compiling the composer definition is a file with a .bna extension (business network archive). This file has to be downloaded from the git repository and placed inside the produc-network folder.

3.2.2. Sequence to start the system.

From inside the fabric-tools folder, open a terminal an run ./startFabric.sh and then ./createComposerProfile.sh. If other Fabric networks were previously created , run ./teardownFabric.sh and ./teardownAllDocker.sh. For the last command choose option 1 when asked.

From inside the unzipped folder, open a terminal an execute

```
composer runtime install --card PeerAdmin@production-network
--businessNetworkName production-network
```

then execute

```
composer network start --card PeerAdmin@production-network --networkAdmin admin
--networkAdminEnrollSecret adminpw --archiveFile production-network@0.1.11.bna
```

To include the admin@production-network.card in the directory of cards, the following command should be executed:

```
composer card import --file admin@production-network.card
```

At this point, the system should be running without problems. To test if it is ok, make a ping from the admin user:

```
composer network ping --card admin@production-network
```

Finally, create the rest API by executing:

```
composer-rest-server
```

Whe stablishing the REST API, the system will ask for a credential. Provide the admin@production-network.card with password (if needed) adminpw. At the end of the deploying, the REST API should be running. This terminal can not be closed or the REST API should stop. From a search engine the rest can be acceded at http://localhost:3000/explorer. If the host can be acceded by other computers (defined network), by replacing “localhost” with the ip address of the host, they can access the server and thus interact with it.

4. Definition of the blockchain database.
The definition of the blockchain can be founded in the cto file. This system has:

1.Producers, defined by the following json object:

```
{
  "$class": "org.production.auction.Producer",
  "product": "X1",
  "factory": "F1"
}
```

2. Production, defined by the following json object:

```
{
  "$class": "org.production.auction.Producttion",
  "comodity": "X1",
  "owner": "org.production.auction.Producer#F1"
}
```

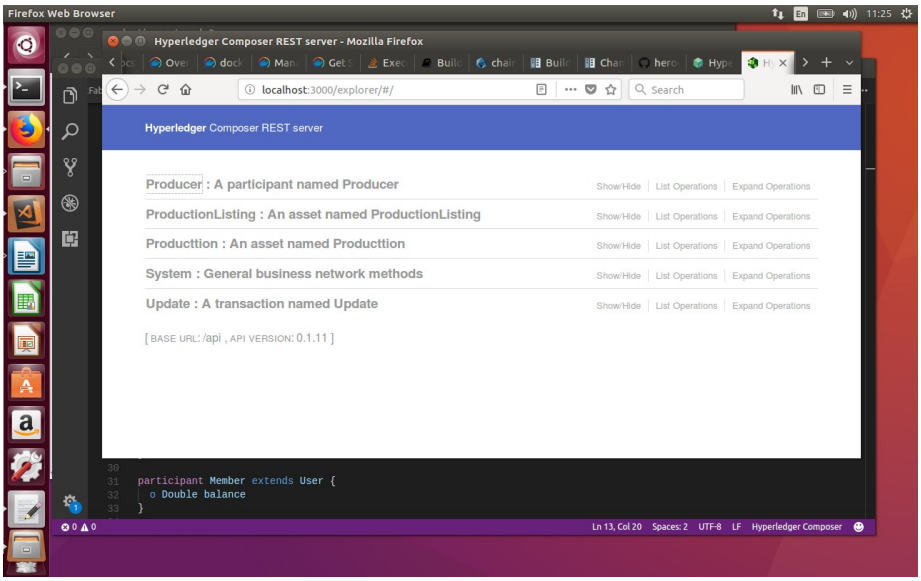
3. An optimization network, defined by producers and their products, within the following json object. To define the network, there is only needed to include one of the participants factories.

```
{
  "$class": "org.production.auction.ProductionListing",
"listingId": "opt1",
  "LAMBDA": 0,
  "description":"first optimization",
  "production":"org.production.auction.Producttion#X1"
}
```

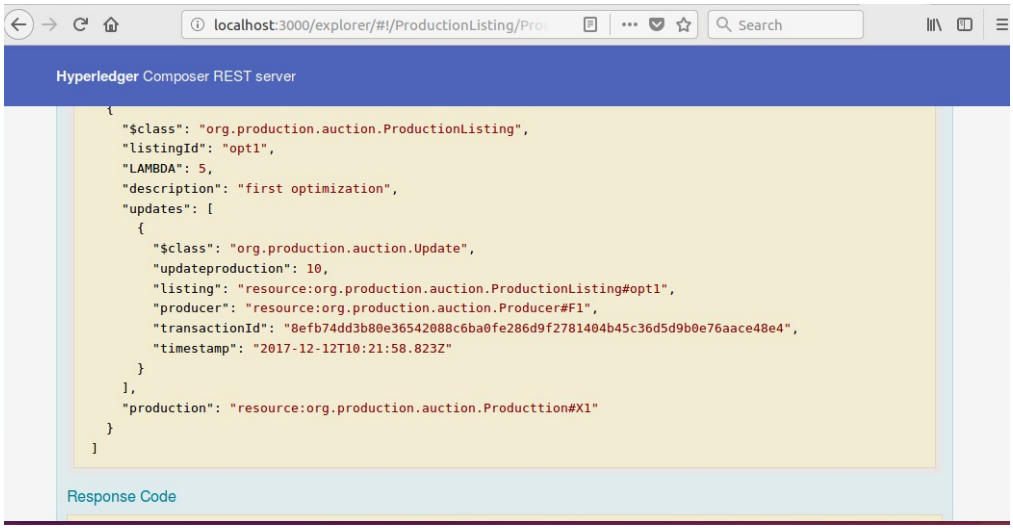
4. Updates of production from the different participants. To make an uptade, the following structure should be provided.

```
{
  "$class": "org.production.auction.Update",
  "updateproduction": 10,
  "listing":"org.production.auction.ProductionListing#opt1",
  "producer":"org.production.auction.Producer#F1"
}
```

Whenever a participant updates its production they have to specify to which optimization he wants to interact. In this case the only optimization network is opt1.



After updating the production, the system computes a new value for lambda which is made public so other users of the network can access to it to compute their local optima.



Finally, through Python each machine can locally interact with the database to read or write back data so to implement the local optimization for each participant.