Fabric-samples

Fabric-samples is a folder provided by the Hyperledger Fabric projectcontaining multiple sample application projects, necessary binary files and 61scripts meant to provide developers with a starting point.

Docker containers

Docker containers are used to represent the entities in the network by implementing yaml files, providing the necessary information needed by each entity to identify and communicate in the network.

The yaml files enable us to configure each entity according to their purpose, by mapping crypto-material to each container we can specify the necessary identifications for the docker container to be able to join the network, create channels, provide information on the other entities in the network, and invoke/query transactions.

The network configurations are also set in each yaml file, to enable TLS (Transport Layer Security) each communicating entity will have TLS enabled and the necessary crypto-material provided. Setting up the necessary yaml files for each entity can be done in several ways, a developer may want to use one file for each entity or have all entities in the same file.

- One yaml file for each peer, containing identification of the peer container, crypto-material for the peer to be able to join a channel, propose transactions and possibly change configurations. Each organization will have one leader peer configured to act as our anchor peer, by enabling gossip we configure the other peers in each organization to point to our leader peer, this way the Orderer sends updates to the leader which then the leader propagates to the other peers.

- One main yaml file containing the Orderer settings and informationthat each Orderer shares, here the developers can decide the ledgertype, the consensus, and batch timeout.

- One yaml file for each client in the network, the client containers contain the chaincode which the client will invoke on their respective peers.

Cryptogen

cryptogen is an utility for generating **Hyperledger Fabric** key material. It is provided as a means of preconfiguring a network for testing purposes. It would normally not be used in the operation of a production network.

In order to generate the crypto-material for the network, we utilize cryptogen, utilized for a preconfigured network in order to quickly setup the necessary key material, and should therefore not be used inproduction. Cryptogen generates the key material through a template file;the file specifies the number of organizations, the number of peers in eachorganization and the number of Orderers.

```
cryptogen generate
```

1. Generating the crypto-material will take the template file asinput and generate a folder with the material
2. Generating the genesis block with the channel policy as the-profile parameter.
3. Generating the channel file with the path to store.

Configtxgen takes a yaml file used to generate the genesis block and the channel.tx file, both needed for the network to create a channel and join. The file is usually known as configtx.yaml and contains five different sections; we will only mention four of the sections as one of the sections isoptional and not used in this project.

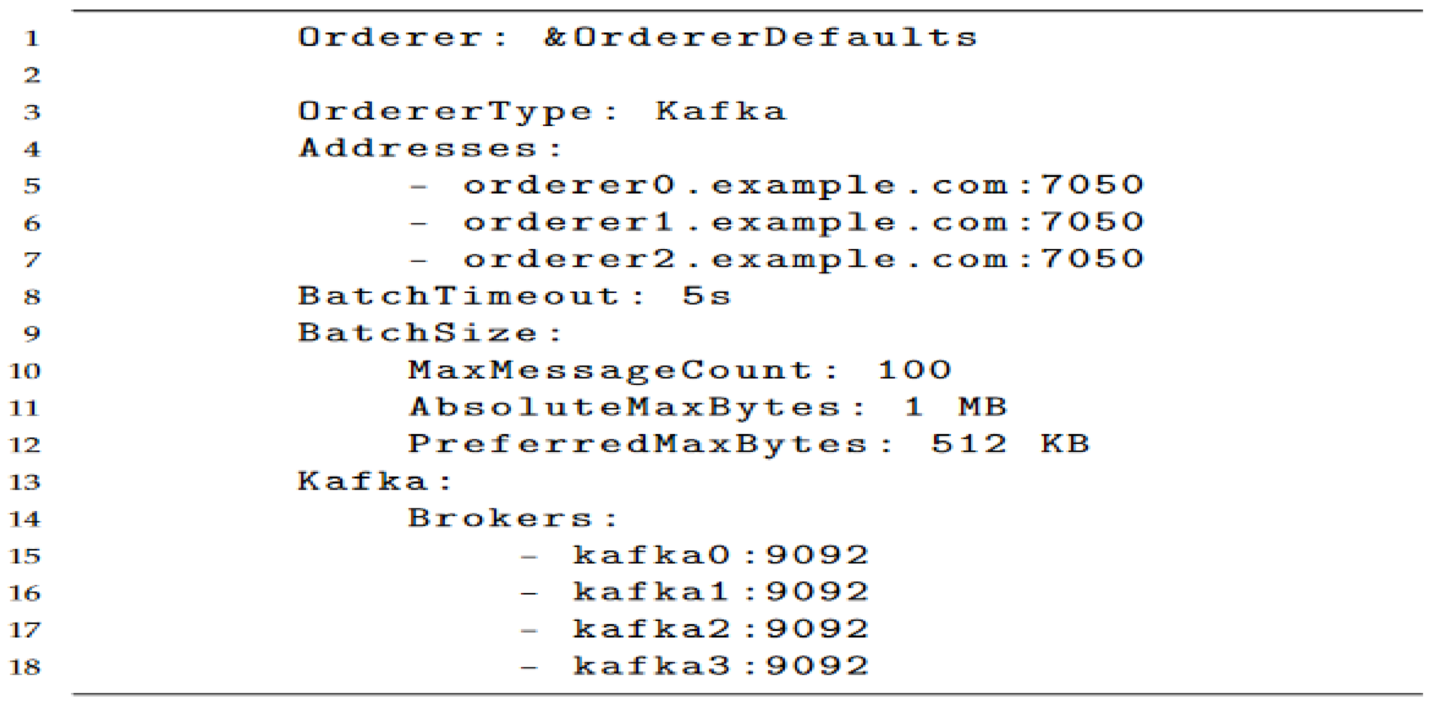
configtx.yaml

•Organization section:Contains the details about the organizations in the network. The organizations in the network are declared underthis section; each organization is named and given an id. The MSPDirvariable needs the path for the crypto-material generated for the organization. The anchor peers of the organizations are also declared,with their port numbers.

• Orderer section:Contains Orderer details related to whether the Orderer is "solo" or "Kafka" and the Orderer addresses and their portnumbers. BatchTimeout and BatchSize specify the block generationfor the Orderer; the Orderer will use one of these two to generate eachblock, whether the BatchTimeout is reached before the BatchSize, orvice versa, a block is generated. These configurations are applicationspecific, each network could have a different block configurationpolicy, and therefore no specific combination was provided byHyperledger

• We based our configurations on the fact that we would like to have a balanced amount of data in each block, therefore we selected 100 transactions, and in cases where the transactions mighttake some time to reach the Orderer, we selected a time of 5 seconds. The maximum block size was set to 1 MB to limit the block sizes, wedo not want each block to take too much memory space, as we would rather have more blocks with a balanced amount of transactions,rather than a few blocks with a lot of data.

– MaxMessageCountsets the maximum number of messagespermitted in a batch
– AbsoluteMaxBytessets the maximum number of bytes in eachmessage for each batch
– PreferredMaxBytessets the preferred maximum number ofbytes per batch



```
Orderer: &OrdererDefaults

OrdererType: Kafka
Addresses:
    - orderer0.example.com:7050
    - orderer1.example.com:7050
    - orderer2.example.com:7050
BatchTimeout: 5s
BatchSize:
    MaxMessageCount: 100
    AbsoluteMaxBytes: 1 MB
    PreferredMaxBytes: 512 KB
Kafka:
    Brokers:
        - kafka0:9092
        - kafka1:9092
        - kafka2:9092
        - kafka3:9092
```

- These variables will allow us to select how the Orderer batches the transaction into blocks; the code section below shows our configur-ations.

- For example, we would like to have a balanced latency and throughput, therefore we have selected a BatchTimeout of five seconds, MaxMessageCount of 100 transactions, AbsoluteMaxBytesof 1 mb and a PreferredMaxBytes of 512 kb.

- The standard Batch-Timeout used by the demo networks of Fabric utilize two seconds, as the transactions are small and the network does not cause high throughput. For our networks, we wish to test several thousands of transactions and have, therefore increased the BatchTimeout in or-der to gather more transactions per batch, either the network takes5 seconds to create a block, or it creates a block for every 100 trans-actions. We will also limit each block to be a maximum of 1mb, asour invocations will contain small datasets, with more frequent in-vokes. If the network uses Kafka mode, the Kafka brokers and their port numbers must be specified.

Solo
The Solo ordering service consists of a single node. When you use Solo, your network is clearly not decentralized and is not fault tolerant — but that’s OK because, again, it is just for development purposes. Solo is designed to provide the ordering service in its simplest possible form so that you can focus on other matters, such as the development of your chaincode and application, without having to worry about the ordering service. However, this is obviously not suitable for production deployments. For production, Fabric 1.4.0 provides the Kafka ordering service.

Kafka
The Kafka ordering service leverages a cluster of Kafka brokers and a Zookeeper ensemble to provide for a crash fault tolerant (CFT) ordering service. It is possible for your ordering service to consist of several ordering nodes that are under the control of different organizations on your network. However, while the result is distributed, it is still not fully decentralized. The difference lies in the point of control because Kafka and Zookeeper are not designed to be run across large networks, but rather in a tight group of hosts. This means that practically speaking you need to have one organization run both the Kafka cluster and the Zookeeper ensemble. Given that, having ordering nodes run by different organizations doesn’t give you much in terms of decentralization because they will all go to the same Kafka cluster, which is under the control of a single organization.

- **Capabilities section**: This section ensures that all of the members in the channel use the same Fabric version, in order to avoid any impact a peer with a different version may have on the channel.

```
Capabilities:
Global: &ChannelCapabilities
    V1_2: true
Orderer: &OrdererCapabilities
    V1_1: true
```

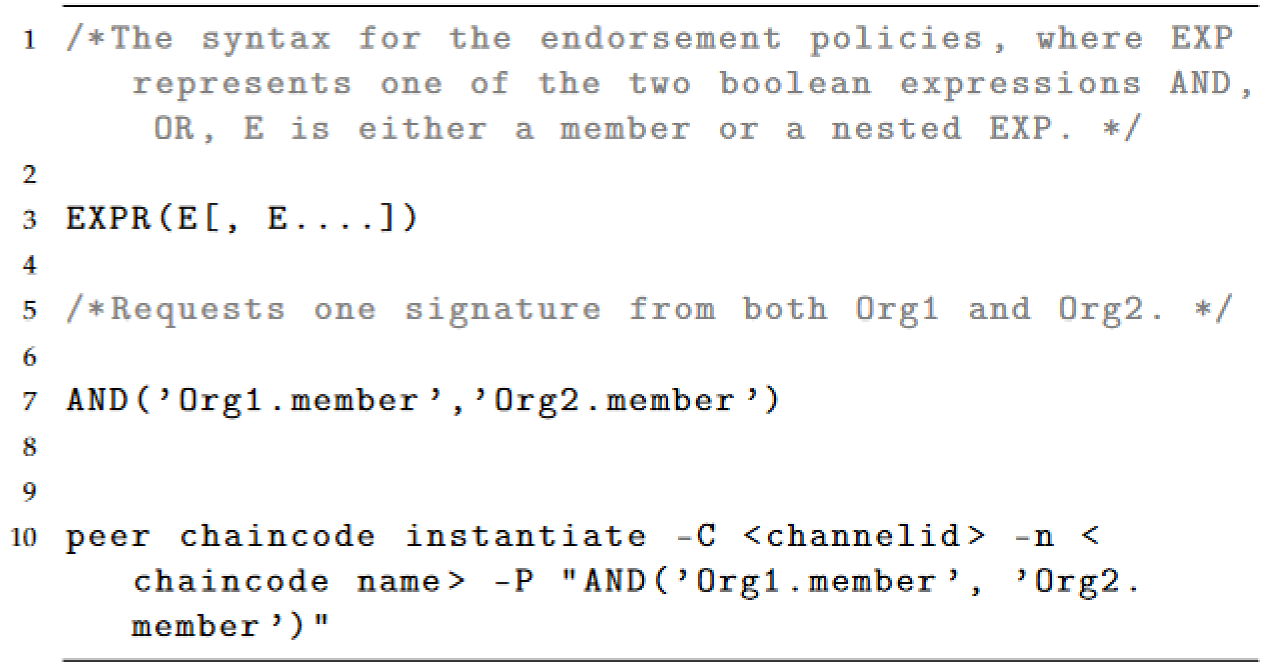
Profile section: The profile section consists of two parts, the first part contains the rules generated in the previous sections given to the genesis block, and the second part consists of the generated rules for the channel. Both of these sections are used to generate the genesis file and the channel file, respectively.

Configtxlator allows for the update of already set configurations on the ledger, by fetching the latest configuration block, Configtxlator translates the data to human-readable JSON versions allowing users or applications to edit the configuration. After the edit, Configtxlator will encode it back, the accessing user or application will sign the changes and submit to the ledger. Once the policies are set, and the crypto-material generated, the necessary keys are mapped in the yaml files.

```
    OrdererOrgs:
  - Name: Orderer
    Domain: example.com
    Template:
      Count: 3
PeerOrgs:
  - Name: Org1
    Domain: org1.example.com
    Template:
      Count: 2
    Users:
      Count: 2
  - Name: Org2
    Domain: org2.example.com
    Template:
      Count: 2
    Users:
      Count: 2
```

Private data collection

Private data collection is meant for organizations on a channel to keep data private from other organizations on the same channel, although creating a channel separated from those organizations would solve the issue. In use cases where a network would want all of the parties to see the transaction



```
/*The syntax for the endorsement policies, where EXP
   represents one of the two boolean expressions AND,
    OR, E is either a member or a nested EXP. */

EXPR(E[, E....])

/*Requests one signature from both Org1 and Org2. */

AND('Org1.member','Org2.member')


peer chaincode instantiate -C <channelid> -n <
   chaincode name> -P "AND('Org1.member', 'Org2.
   member')"
```

while also keeping part of the transaction private, a new channel would not be a solution. Private data collection allows a subset of the organizations on a channel to see the plaintext data, while others only receive a hash. We further utilized this method by splitting organizations into single homes, to minimize the subset of homes that may read the data to only one single home. A JSON file handles the policies for the Private data collection; the example below is a Private data collection policy.

```
/*In addition to the endorsement policies, private
   data requires the file path of the collection-
   config.json file */

peer chaincode instantiate -o orderer0.example.com
   :7050 --tls --cafile $ORDERER_CA -C mychannel -c
   '{"Args":["Init"]}' -n device -v v0 -P "OR('
   Org1MSP.member','Org2.member')" --collections-
   config $COLLECTIONS_PATH/collections_config.json
```

```
[
    {
        "name": "collectionSmarthomes",
        "policy": "OR('Org1MSP.member', 'Org2MSP.member', 'Org3MSP.member')",
        "requiredPeerCount": 0,
        "maxPeerCount": 2,
        "blockToLive": 5,
        "memberOnlyRead": true

    },
    {
        "name":"collectionSmarthomesPrivate",
        "policy": "OR('Org1MSP.member', 'Org2MSP.member')",
        "requiredPeerCount": 0,
        "maxPeerCount": 2,
        "blockToLive": 5,
        "memberOnlyRead": true
    }
]
```

Each collection definition consists of six properties, these properties handle the endorsement time, the control over the propagation of theprivate data and the time of data purge.

•Name: The name of the collection, used in the chaincode to specifythe policy used for the invocation.

•Policy: Defines the policy for the access of the data, the organizationsthat have access to the data are defined in a similar fashion to theendorsement policy.

•RequiredPeerCount:The minimal number of authorized peers thatneed the data propagated to, before the transaction is signed by thepeer and returned to the client.

•MaxPeerCount:The maximum number of authorized peers thedata will be propagated to, for data redundancy. In cases wherethe endorsing peer becomes unavailable between endorsement andcommit time, other authorized peers may ask for the private data tothe other peers that received the private data. If the value is set to 0the data will not be propagated.

- BlockToLive:The number of blocks needed before the data is purged,to keep the data indefinitely the number must be set to 0.

- MemberOnlyRead:Indicates that peers will enforce that only clientsfrom the authorized organizations are allowed read and write accessto the data, once set to true.

Chaincode software design (in Go language

HyperLdger was one of the first platforms to support go, node.js and java as smart contract languages. Chaincode is simply a smart contract written in any of the supported languages which implement the prescribed interface.

The HyperLedger chaincode runs in a docker container separate from the peer who is endorsing it. Chaincode acts as a middleware which uses the transactions submitted by the application to manage the state of the Ledger. We are using go language for writing the chaincode.

Chaincode in smart home

The implemented application bases itself around a network of smart homes; each home will have a set of IoT devices that may generate loadon the blockchain network. The payload is implemented as a struct withdata fields, depending on whether the network uses Private data collection,an extra struct is implemented for the private data field, this will enable usto specify which fields are public and which are private

Struct

The payload generated by each IoT device is stored in a struct, the structcontains the necessary information in the form of data fields. Depending onwhether the blockchain network utilizes private data, an extra struct needs to be used for the private data to be stored. If that is the case DeviceReading loses its Data field, and the device data is stored in the DeviceData struct instead. The same id will be used for both structs, in order for the queries to be kept simple and orderly.

A struct containing information about the data to be uploaded in data fields

```
// DeviceReading struct
type DeviceReading struct{
    objectType  string ‘json:"docType"‘
    ID          string ‘json:"id"‘
    Type        string ‘json:"Type"‘
    Data        string ‘json:"data"‘
}
```

Private data requires an extra struct for the generation and storage of the data to be kept private

```
    //Data that is kept private
    type DeviceData struct{
        objectType string ‘json:"docType"‘
        ID           string ‘json:"id"‘
        Data         string ‘json:"data"‘
}
```

Data fields

The following fields are generated for each invoke transaction:70
•IdA unique id for each device used to identify the transactioninvoked to the chain, based on a client identification library that getsthe unique id of each client.

•TypeThis field will specify the type of device, the intention is forpossible filtering of data by device type for future use.

•DataThe data field will contain the data generated by each device,this field is the main field used in our tests, as it will vary in size foreach test.

•DoctypeMainly used to distinguish the objects store in the statedatabase, we have set it to "document" for each transaction.

Functions

The chaincode utilizes two main functions, for each function we have alsoimplemented an extra function that utilizes the Private data collection:

•SendDeviceReading: Invoke function that generates the data to be uploaded to the chain,stored as a JSON-object. SendDeviceReading needs two parameterstypeused to specify the type of device uploading the data, andDatawhich is the payload generated by the device.

•ReadDevice: Query function that returns the queried data as a JSON object. Usesthe client id to query the data uploaded by the device.•SendDeviceReadingPrivate:Similar toSendDeviceReading, but hashes the data field for everyparticipant but the invoker.

•ReadDevicePrivate: Similar toReadDevice, but enables the reading ofSendDeviceRead-ingPrivatefor the peer that has access.

Hyperledger Caliper

Hyperledger Caliper provided by the Hyperledger project is a benchmark-ing tool for blockchains, used to measure the performance of specific block-chain implementations. The primary purpose of Caliper is to provide developers with a helping hand in trying to find the right blockchain frame-work, calculate resource consumptions, and cost estimation for setting up the network. The supported metrics are success rate, transaction through-put, transaction latency, and resource consumption (CPU, memory).

- Success rate indicates the number of transactions successfully com-mitted to the ledger. Failures can be caused by multiple factors suchas time-outs, network limitations, peer resources, chaincode, to namea few, and therefore, a failure cause is not easily identifiable.

- Transaction throughput indicates the number of transactions submit-ted to the ledger per second.

- Transaction latency indicates the time a transaction takes to beavailable across the whole network; this metric is calculated pertransaction.

Hyperledger Calipers has structured its architecture into three mainlayers: the adaptation layer, the interface and core layer, and lastly thebenchmark layer. Each layer provides functionalities that allow Caliper to communicate with the ledger, test the performance, and generate a reportbased on the tests

- Adaptation layer: Uses framework-specific adaptors to integrate thee xisting blockchain network into the Caliper framework.

- Interface and Core layer: Used to implement core functionalities that Caliper provides, these consist of:

  –Blockchain operating interfaces: consist of operations to deploy,I nstantiate, install, invoke, and query smart contracts.

  –Resource monitor: consists of the operations for starting ands topping the monitor that fetches the resource consumption statuses such as CPU and memory of the running network.Currently, only two types of monitors are supported, one that monitors the local process, and one that monitors the dockerc ontainers

–Performance analyzer: consists of the operations that provide the network statistics such as TPS, delay, and success ratio, ther esults are printed to the terminal.

–Report generator: consists of the operations that generate theHTML formatted file containing the benchmark results.

–Resource monitor: consists of the operations for starting andstopping the monitor that fetches the resource consumptionstatuses such as CPU and memory of the running network.Currently, only two types of monitors are supported, one thatmonitors the local process, and one that monitors the dockercontainers.

- Benchmark layer: consists of the configuration file that defines thetopology of the blockchain network and the configuration files thatdefine the test cases.

Hyperledger Composer

Hyperledger composer is a toolset provided by the Hyperledger project to simplify application development for the fabric blockchaina rchitecture and support the business-end of blockchain development; through the use of a simplified modeling language for the definition of the network logic and entities. Their primary goal for the framework s to accelerate application development and easier integration to the already existing or newly developed blockchain network.