Hyperledger Fabric
Smart Contract Programming

jhlim@mmlab.snu.ac.kr

Network Convergence & Security Laboratory

Prerequisites

- Go (1.10.x)
- Node.js (8.9.0, or higher)
- Docker (18.06, or higher) & Docker compose
- Python 2.7

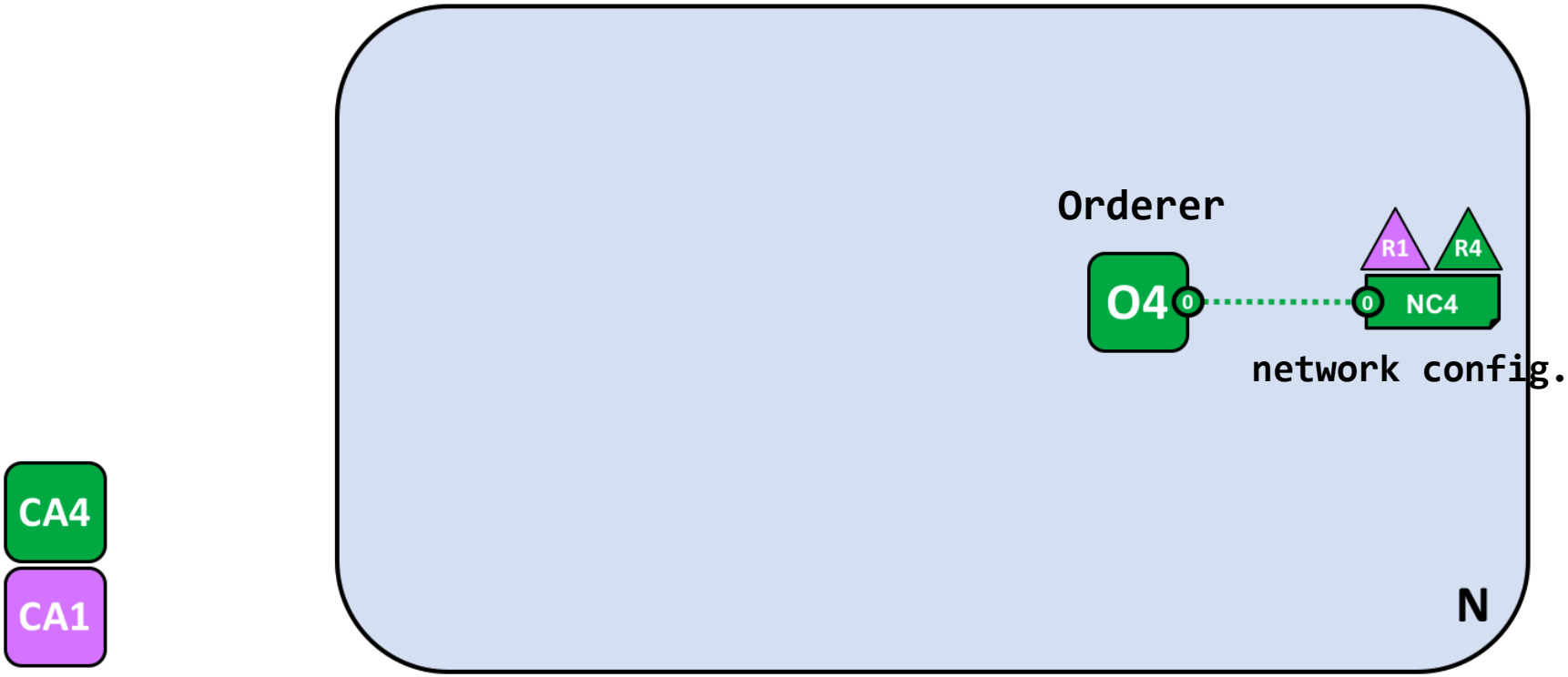
Reference

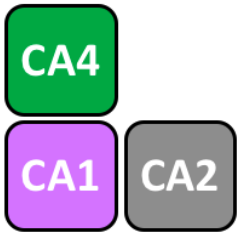
- https://hyperledger-fabric.readthedocs.io/en/release-1.3/
- https://hyperledger-fabric.readthedocs.io/en/release-1.4/tutorial/commercial_paper.html#
- https://developer.ibm.com/kr/cloud/blockchain/2017/04/04/starting_blockchain_using_hyperledger_fabric/
- https://godoc.org/github.com/hyperledger/fabric

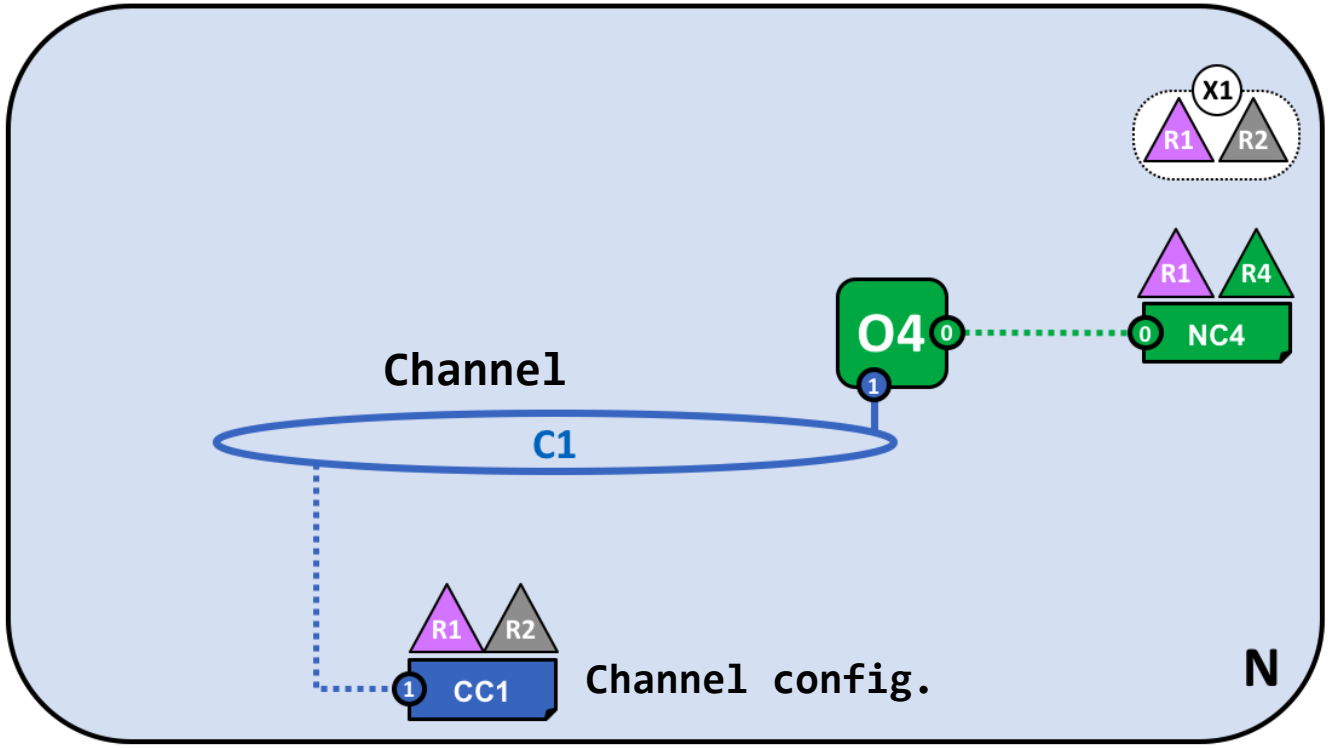
Install Hyperledger Fabric

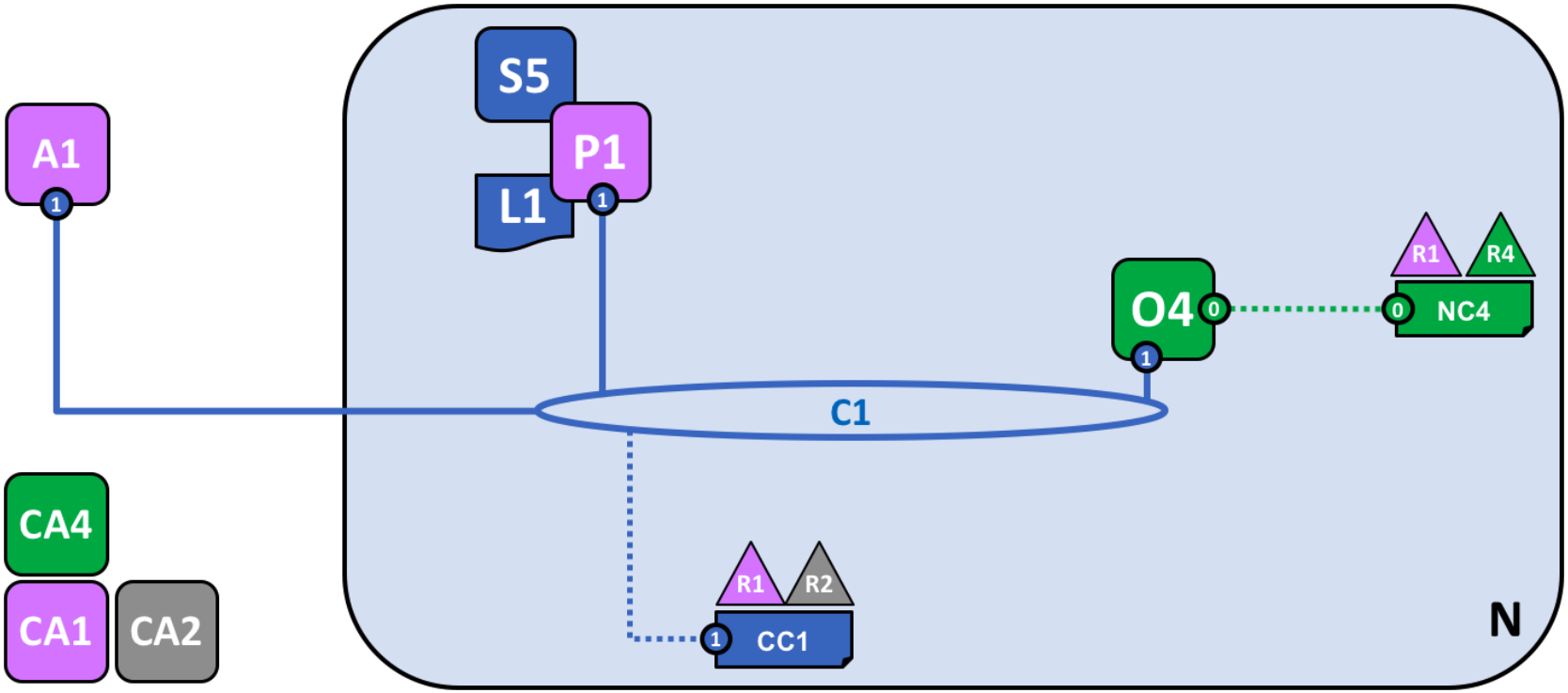
- sudo curl -sSL http://bit.ly/2ysbOFE | bash -s 1.3.0
  - install Hyperledger Fabric platform
  - clone fabric-samples repository
  - docker images

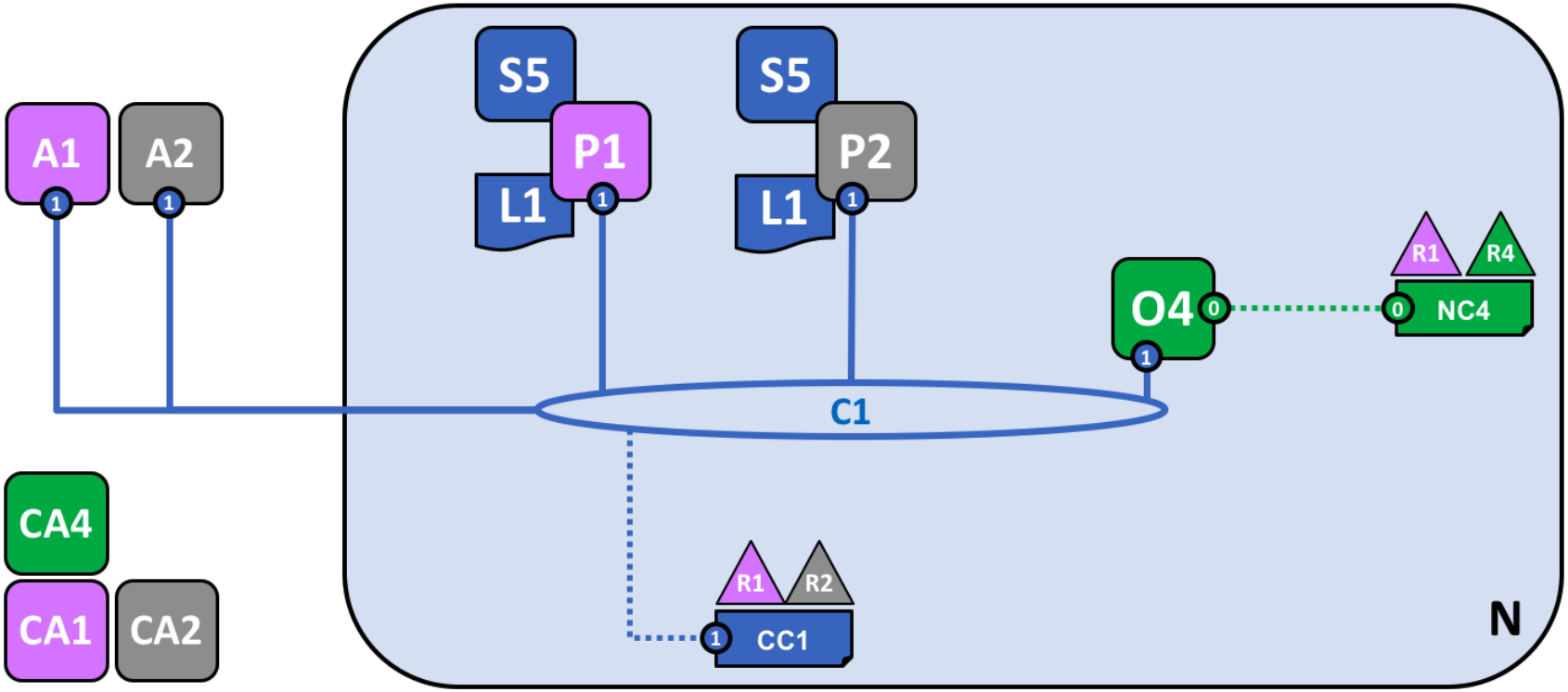
```
===> List out hyperledger docker images
hyperledger/fabric-ca          1.4.1               3a1799cda5d7        6 days ago          252MB
hyperledger/fabric-ca          latest              3a1799cda5d7        6 days ago          252MB
hyperledger/fabric-zookeeper   0.4.15              20c6045930c8        4 weeks ago         1.43GB
hyperledger/fabric-zookeeper   latest              20c6045930c8        4 weeks ago         1.43GB
hyperledger/fabric-kafka       0.4.15              b4ab82bbaf2f        4 weeks ago         1.44GB
hyperledger/fabric-kafka       latest              b4ab82bbaf2f        4 weeks ago         1.44GB
hyperledger/fabric-couchdb     0.4.15              8de128a55539        4 weeks ago         1.5GB
hyperledger/fabric-couchdb     latest              8de128a55539        4 weeks ago         1.5GB
```


Orderer
R1
R4
O4
0
0
NC4
network config.
N
CA4
CA1


CA4
CA1
CA2


X1
R1
R2
R1
R4
O4
0
0
NC4
1
Channel
C1
R1
R2
1
CC1
Channel config.
N


A1
1
S5
P1
L1
1
C1
O4
0
1
R1
R4
0
NC4
CA4
CA1
CA2
R1
R2
1
CC1
N


A1
A2
1
1
S5
P1
L1
1
S5
P2
L1
1
O4
0
1
R1
R4
0
NC4
C1
CA4
CA1
CA2
R1
R2
1
CC1
N

First network

- Generate network artifacts
  - `cd fabric-samples/first-network`
  - `./byfn.sh generate`

  => certificates, orderer genesis block, channel configuration transaction ‘channel.tx’, and anchor peer update for Org1MSP & Org2MSP

```
############################################################
##### Generate certificates using cryptogen tool #########
############################################################
+ cryptogen generate --config=./crypto-config.yaml
org1.example.com
org2.example.com

################################################################
### Generating channel configuration transaction 'channel.tx' ###
################################################################
+ configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID mychannel
2019-04-23 10:07:05.766 UTC [common/tools/configtxgen] main -> INFO 001 Loading configuration
2019-04-23 10:07:05.799 UTC [common/tools/configtxgen] doOutputChannelCreateTx -> INFO 002 Generating new channel conf
2019-04-23 10:07:05.800 UTC [common/tools/configtxgen] doOutputChannelCreateTx -> INFO 003 Writing new channel tx
```

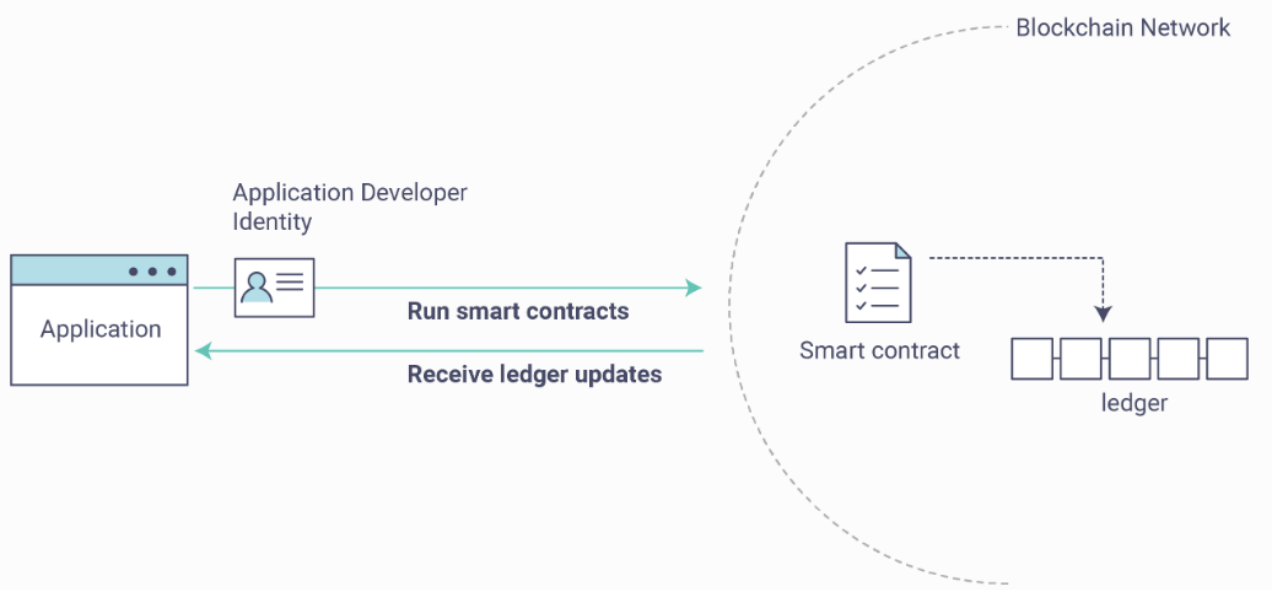
First network

- Bring up the network
  - `./byfn.sh up`

```
> peer channel create –o orderer.example.com:7050 ..
> peer channel join –b mychannel.block
> peer channel update –o .. // update anchor peers for org “Org1MSP”
> peer chaincode install –n mycc .. // install chaincode on peer0.org1
> peer chaincode install –n mycc .. // install chaincode on peer0.org2
> peer chaincode instantiate –o .. ‘{“Args”:[“init”, “a”, “100”, “b”,
  “200”]}’
  // instantiate chaincode on peer0.org2
> peer chaincode query –C mychannel .. ‘{“Args”:[“query”, “a”]}’
  // query chaincode on peer0.org1
  100
```

- Bring down the network
  - `./byfn.sh down`

Chaincode (Smart contract)



- Chaincode is a program, written in Go, node.js, or Java that implements a prescribed interface
- Chaincode runs in a secured **Docker container** isolated from the endorsing peer process
- Chaincode **initializes and manages** the **ledger state** through transactions submitted by applications

Chaincode Interface

- Every chaincode program must implement the Chaincode interface

```
type Chaincode interface {
    // Init is called during Instantiate transaction after the chaincode container
    // has been established for the first time, allowing the chaincode to
    // initialize its internal data
    Init(stub ChaincodeStubInterface) pb.Response

    // Invoke is called to update or query the ledger in a proposal transaction.
    // Updated state variables are not committed to the ledger until the
    // transaction is committed.
    Invoke(stub ChaincodeStubInterface) pb.Response
}
```

```
const Chaincode = class {
    async Init(stub) {
        await stub.putState(key, Buffer.from(aStringValue));
        return shim.success(Buffer.from('Initialized Successfully!'));
    }
    async Invoke(stub) {
        let oldValue = await stub.getState(key);
        let newValue = oldValue + delta;
        await stub.putState(key, Buffer.from(newValue));
        return shim.success(Buffer.from(newValue.toString()));
}};
```

Simple asset chaincode

- Create a directory for your chaincode application
  - `mkdir -p $GOPATH/src/sacc && cd $GOPATH/src/sacc`
  - `vi sacc.go`

```go
package main

import (
    "fmt"

    "github.com/hyperledger/fabric/core/chaincode/shim"
    "github.com/hyperledger/fabric/protos/peer"
)

// SimpleAsset implements a simple chaincode to manage an asset
type SimpleAsset struct {
}
```

Init function

```
// main function starts up the chaincode in the container during instantiate
func main() {
    if err := shim.Start(new(SimpleAsset)); err != nil {
            fmt.Printf("Error starting SimpleAsset chaincode: %s", err)
    }
}


func (t *SimpleAsset) Init(stub shim.ChaincodeStubInterface) peer.Response {
  // Get the args from the transaction proposal
  args := stub.GetStringArgs()
  if len(args) != 2 {
    return shim.Error("Incorrect arguments. Expecting a key and a value")
  }

  // Set up any variables or assets here by calling stub.PutState()

  // We store the key and the value on the ledger
  err := stub.PutState(args[0], []byte(args[1]))
  if err != nil {
    return shim.Error(fmt.Sprintf("Failed to create asset: %s", args[0]))
  }
  return shim.Success(nil)
}
```

Invoke function

```
func (t *SimpleAsset) Invoke(stub shim.ChaincodeStubInterface) peer.Response {
    // Extract the function and args from the transaction proposal
    fn, args := stub.GetFunctionAndParameters()

    var result string
    var err error
    if fn == "set" {
            result, err = set(stub, args)
    } else {
            result, err = get(stub, args)
    }
    if err != nil {
            return shim.Error(err.Error())
    }

    // Return the result as success payload
    return shim.Success([]byte(result))
}
```

set & get function

```
func set(stub shim.ChaincodeStubInterface, args []string) (string, error) {
    if len(args) != 2 {
            return "", fmt.Errorf("Incorrect arguments. Expecting a key and a value")
    }

    err := stub.PutState(args[0], []byte(args[1]))
    if err != nil {
            return "", fmt.Errorf("Failed to set asset: %s", args[0])
    }
    return args[1], nil
}

func get(stub shim.ChaincodeStubInterface, args []string) (string, error) {
    if len(args) != 1 {
            return "", fmt.Errorf("Incorrect arguments. Expecting a key")
    }

    value, err := stub.GetState(args[0])
    if err != nil {
            return "", fmt.Errorf("Failed to get asset: %s with error: %s", args[0], err)
    }
    if value == nil {
            return "", fmt.Errorf("Asset not found: %s", args[0])
    }
    return string(value), nil
}
```

Build and test using dev mode

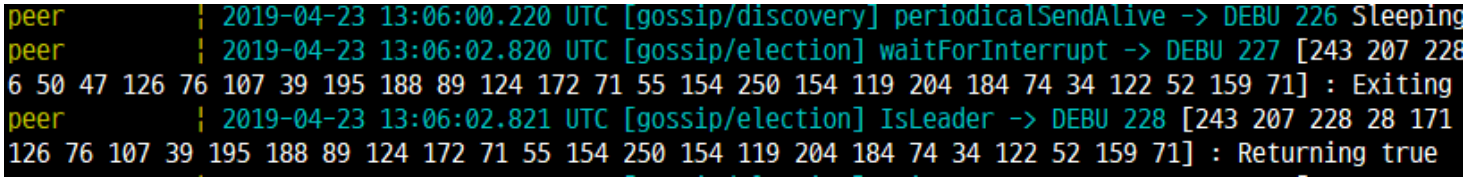
- Fetch the package and compile
  - `go get -u github.com/hyperledger/fabric/core/chaincode/shim`
  - `go build`

- Chaincode can be built and started by the user in “dev mode”
  - `cd fabric-samples/chaincode-docker-devmode`
  - `vi docker-compose-simple.yaml`

```
services:
  orderer:
    container_name: orderer
    image: Hyperledger/fabric-tools
    ports:
      - 7050:7050
  peer: ..
  cli: ..
  chaincode:
    ..
    volumes:
      - ./../chaincode:/opt/gopath/src/chaincode
```

Test using dev mode

- Terminal 1: Start the network with the Single **orderer** profile and launches the **peer** in “dev mode”. It also launches two additional containers – one for the **chaincode** environment and a **CLI** to interact with the chaincode.
  - `docker-compose -f docker-compose-simple.yaml up`
  - (Includes “peer channel create –c myc” and “peer channel join”)



  - If don’t want to see DEBUG msg, change logging level to INFO in .yaml
- Terminal 2: Build & start the chaincode
  - `docker exec –it chaincode bash`
  - `cd sacc (=> fabric-sample/chaincode/sacc)`
  - `go build`
  - `CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=mycc:0 ./sacc`

Test using dev mode

- Terminal 3: Client using the chaincode

```
> docker exec –it cli bash
> peer chaincode install -p chaincodedev/chaincode/sacc -n mycc -v 0
> peer chaincode instantiate -n mycc -v 0 -c '{"Args":["a","10"]}' -C myc

// Invoke message
> peer chaincode invoke -n mycc -c '{"Args":["set", "a", "20"]}' -C myc

// Query message
> peer chaincode query -n mycc -c '{"Args":["query","a"]}' -C myc
 20

// List message
> peer chaincode list --instantiated -C myc
 Name: mycc, Version: 0, Path: chaincodedev/chaincode/sacc, ESCC: escc, Vscc: vscc
```

- peer chaincode command allows admin to perform chaincode related operations on a peer
- Remove containers: `docker rm -f $(docker ps -aq)`

https://hyperledger-fabric.readthedocs.io/en/release-1.3/commands/peerchaincode.html?highlight=peer%20chaincode

Transfer example

```
func (t *SimpleAsset) Init(stub shim.ChaincodeStubInterface) peer.Response
{
  args := stub.GetStringArgs()
  A := args[0]; B := args[2]
  Aval, err = strconv.Atoi(args[1])
  Bval, err = strconv.Atoi(args[3])

  err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
  err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
  ..
  return shim.Success(nil)
}
```

```
func (t *SimpleAsset) transfer(stub shim.Chain.., args []string) .. {
  A := args[0]; B := args[1];
  Avalbytes, err := stub.GetState(A)
  Aval, _ = strconv.Atoi(string(Avalbytes))
  Bvalbytes, err := stub.GetState(B)
  Bval, _ = strconv.Atoi(string(Bvalbytes))

  X := strconv.Atoi(args[2])
  Aval = Aval – X; Bval = Bval + X;
  err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
  ..
  return shim.Success(nil)
}
```

Invoke another chaincode

```
Import (
        “github.com/hyperledger/fabric/common/util”
        ..
)


func (t *SimpleAsset) Invoke(stub ..) {
  fn, args := stub.GetFunctionAndParameters()
  if fn == “callCC” {
    chainCodeArgs := util.ToChaincodeArgs(“CCFunc”, “paramA”, ..)
    response := stub.InvokeChaincode(“CCName”, chainCodeArgs, “channelName”)
    if response.Status != shim.OK { .. }
    return shim.Success(nil)
  }
  ..
}
```

Chaincode encryption

- Encrypt values associated with a key and write to ledger
- To encrypt, the invoker passes in a cryptographic key via the **transient** field
  - `peer chaincode invoke ... --transient “{\”ENCKEY\”:\”$ENCKEY\”}”`
  - `peer chaincode query ... --transient “{\”DECKEY\”:\”$”DECKEY\”}”`

```
// enccc_example.go

type EncCC struct { bccspInst bccsp.BCCSP }

func (t *EncCC) Invoke(stub shim.ChaincodeStubInterface) pb.Response {

  f, args := stub.GetFunctionAndParameters()
  tMap, err := stub.GetTransient()
  case “ENCRYPT”:
    if _, in := tMap[ENCKEY]; !in { .. }
    return t.Encrypter(stub, args[0:], tMap[ENCKEY], tMap[IV])
  case “DECRYPT”:
    if _, in := tMap[DECKEY]; !in { .. }
    return t.Decrypter(stub, args[0:], tMap[DECKEY], tMap[IV]
```

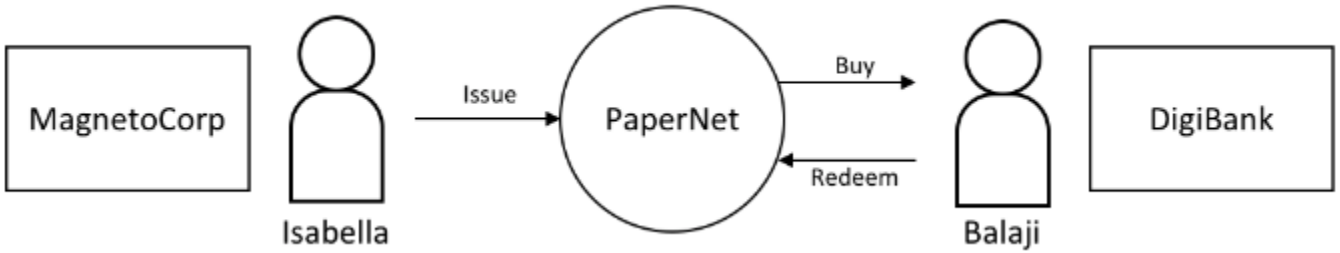
https://github.com/hyperledger/fabric/tree/release-1.4/examples/chaincode/go/enccc_example

Chaincode encryption

```
// utils.go

Import (
         “github.com/hyperledger/fabric/core/chaincode/shim/ext/entities”
         ..
)

func getStateAndDecrypt(stub shim.ChaincodeStubInterface, ent entities.Encrypter, key
string) ([]byte, error) {
  ciphertext, err := stub.GetState(key)
  return ent.Decrypt(ciphertext) }

func encryptAndPutState(stub shim.ChaincodeStubInterface, ent entities.Encrypter, key
string, value []byte) error {
  ciphertext, err := ent.Encrypt(value)
  return stub.PutState(key, ciphertext) }
```

https://github.com/hyperledger/fabric/tree/release-1.4/examples/chaincode/go/enccc_example

Developing application



- MagnetoCorp and DigiBank use PaperNet to trade commercial paper with each other

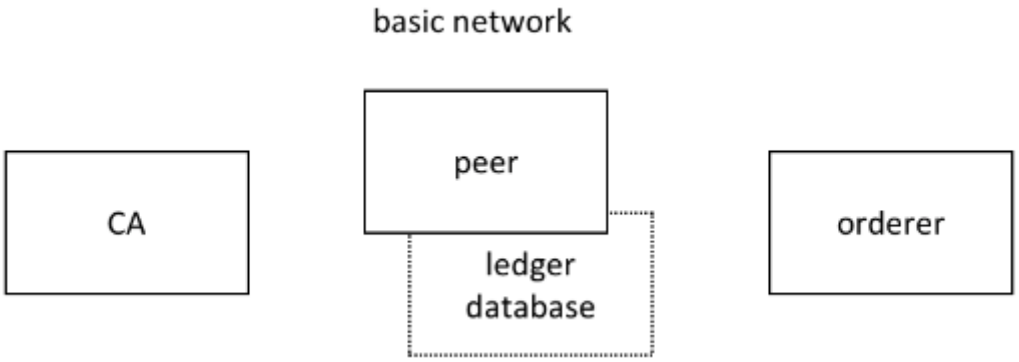
- Scenario
  - As an employee of MagnetoCorp, issue a commercial paper
  - As an employee of DigiBank, buy the paper and redeem it after a period of time

https://hyperledger-fabric.readthedocs.io/en/release-1.4/tutorial/commercial_paper.html#

Download samples

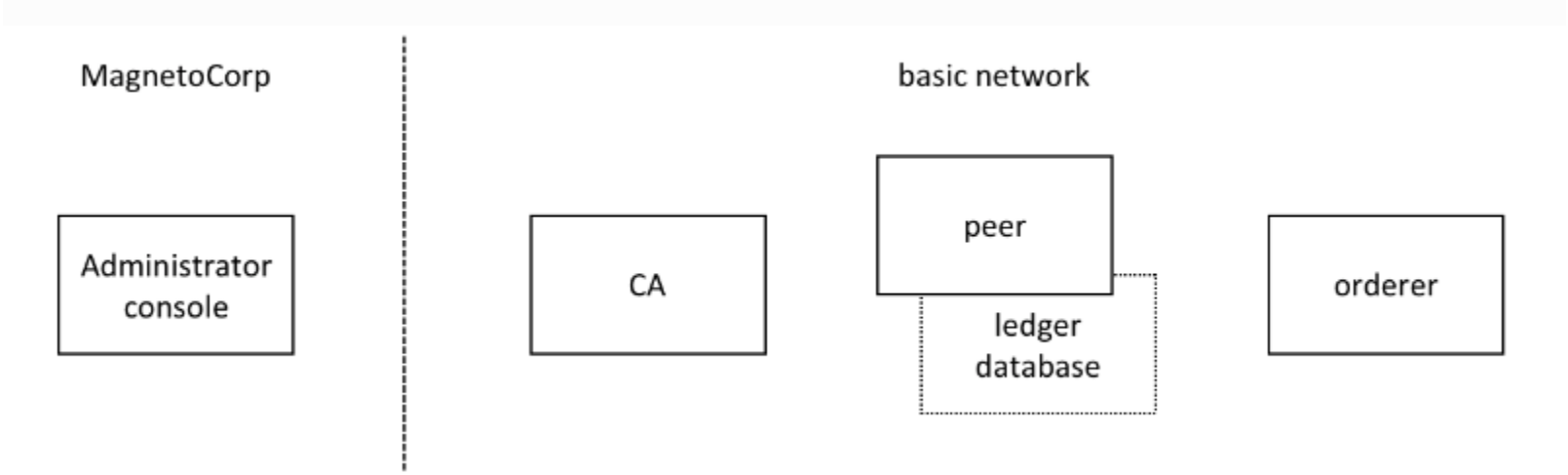
- `mkdir –p $GOPATH/src/github.com/hyperledger/`
- `cd $GOPATH/src/github.com/hyperledger/`
- `git clone https://github.com/hyperledger/fabric-samples.git`

Setting basic network



- `cd fabric-samples/basic-network`
  - Basic-network has a single peer, a channel, and pre-generated crypto material
- `./start.sh`
  - make each components container
  - create the channel
  - join peer0.org1.example.com to the channel

Working as MagnetoCorp



- Terminal 1: network log monitor
  - Open new window in the `$GOPATH/../`fabric-samples directory
  - `cd commercial-paper/organization/magnetocorp/configuration/cli/`
  - `./monitordocker.sh net_basic`

Working as MagnetoCorp

- Start a MagnetoCorp docker container for the admin

- Terminal 2: MagnetoCorp docker container for the admin
  - `cd ../commercial-paper/organization/magnetocorp/configuration/cli/`
  - `docker-compose -f docker-compose.yml up -d cliMagnetoCorp`

    Creating cliMagnetoCorp ... done

Smart contract

- Three main functions, **issue**, **buy**, and **redeem**
- Terminal 3: Smart contract
  - cd commercial-paper/organization/magnetocorp/contract
  - vi lib/papercontract.js

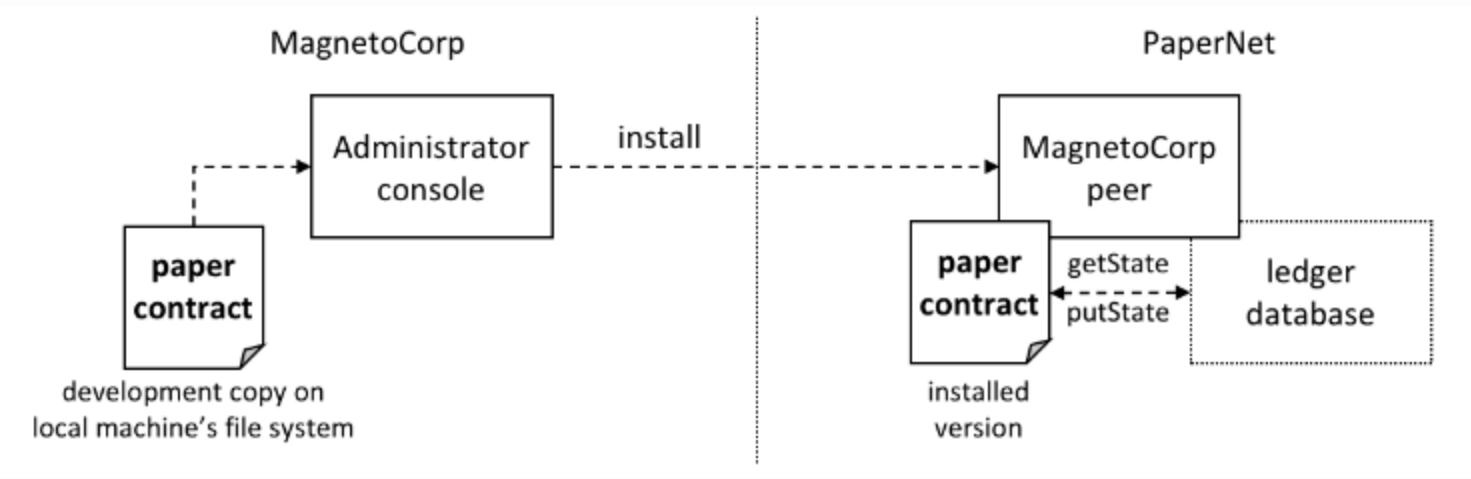
```
const { Contract, Context } = require('fabric-contract-api’);
// bring two classes Contract and Context
```

```
class CommercialPaperContract extends Contract {
    async issue(ctx, issuer, paperNumber, issueDateTime, maturityDateTim
    e...) {
        let paper = CommercialPaper.createInstance(issuer, paperNumber,
        issueDateTime...);
        await ctx.paperList.addPaper(paper);
        return paper.toBuffer();
    }
    ...
```

Smart contract

```
async instantiate(ctx) {
  console.log('Instantiate the contract');
}

async issue(ctx, issuer, paperNumber, ..) {
  let paper = CommercialPaper.createInstance(issuer, paperNumber, ..);
  paper.setOwner(issuer);
  await ctx.paperList.addPaper(paper);
}

async buy(ctx, issuer, paperNumber, currentOwner, newOwner, price,..) {
  ..
  if (paper.isTrading()) {
    paper.setOwner(newOwner);
  } ..
}

async redeem(ctx, issuer, paperNumber, redeemingOwner, redeemDateTime) {
  paper.setOwner(paper.getIssuer()); paper.setRedeemed();
  await ctx.paperList.updatePaper(paper);
  ..
}
```

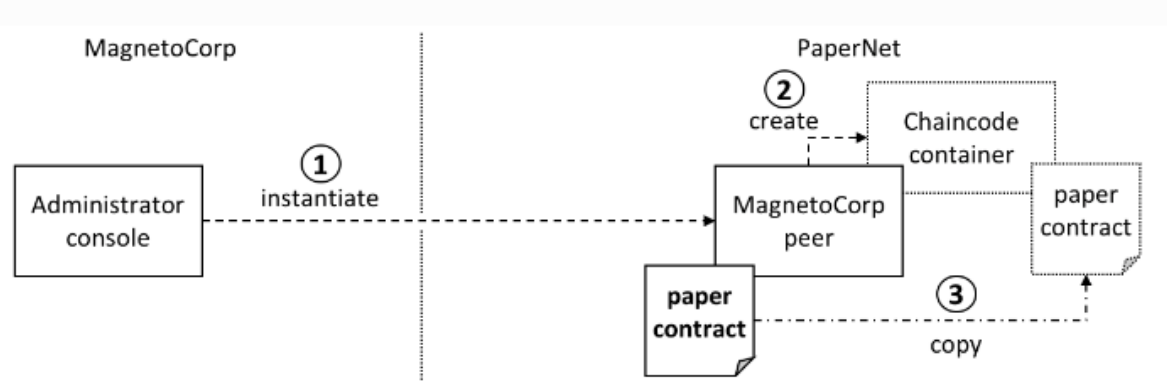
Install contract



- In the MagnetoCorp admin (Terminal 2),
  - docker exec cliMagnetoCorp peer chaincode install -n papercontract -v 0 -p /opt/gopath/src/github.com/contract -l node

```
2019-04-20 13:42:09.411 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2019-04-20 13:42:09.412 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2019-04-20 13:42:09.479 UTC [chaincodeCmd] install -> INFO 003 Installed remotely response:<status:200 pay
```

Instantiate contract



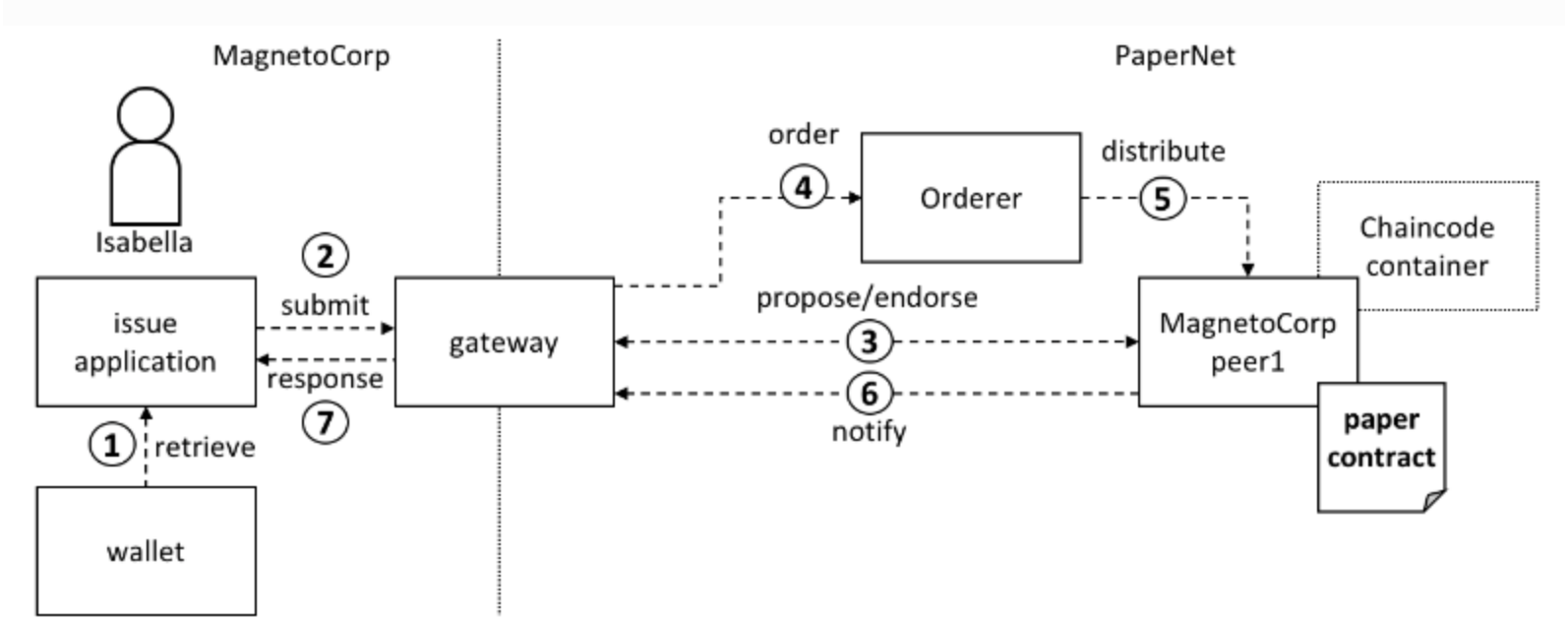
- `docker exec cliMagnetoCorp peer chaincode instantiate -n papercontract -v 0 -l node -c '{"Args":["org.papernet.commercialpaper:instantiate"]}' -C mychannel -P "AND ('Org1MSP.member')"`

```
2019-04-20 14:18:21.831 UTC [chaincodeCmd] InitCmdFactory -> INFO 001 Retrieved channel (mychannel) ordere
r endpoint: orderer.example.com:7050
2019-04-20 14:18:21.833 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default escc
2019-04-20 14:18:21.834 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003 Using default vscc
```

- See a papercontract container
  - docker ps

```
NAMES
dev-peer0.org1.example.com-papercontract-0
```

MagnetoCorp User



- Terminal 4: MagnetoCorp user
  - cd commercial-paper/organization/magnetocorp/application/
  - ls

```
addToWallet.js      issue.js        package.json
```

  - npm install

MagnetoCorp User

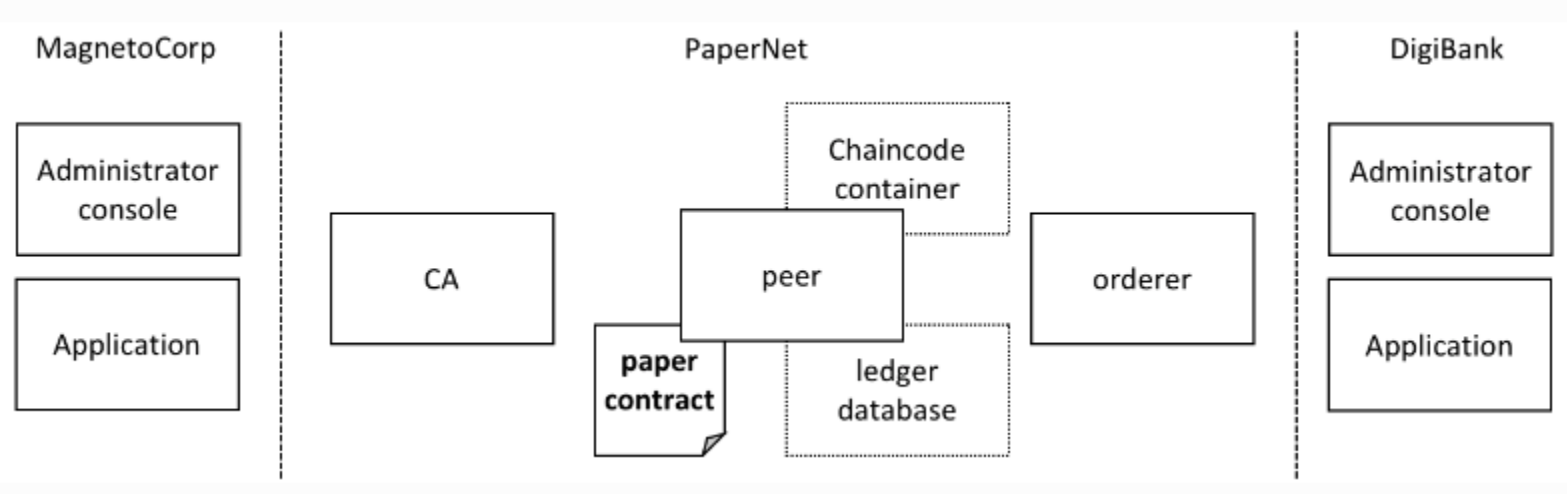
- Make wallet to identity user
  - `node addToWallet.js`
  - `ls ../identity/user/Isabella/wallet/User1@org1.example.com`

```
User1@org1.example.com      c75bd6911a...-priv      c75bd6911a...-pub
```

- Now issue MagnetoCorp commercial paper 00001
  - `node issue.js`

```
Connect to Fabric gateway.
Use network channel: mychannel.
Use org.papernet.commercialpaper smart contract.
Submit commercial paper issue transaction.
Process issue transaction response.
MagnetoCorp commercial paper : 00001 successfully issued for value 5000000
Transaction complete.
Disconnect from Fabric gateway.
Issue program complete.
```

Working as DigiBank



- Terminal 5: DigiBank admin
  - cd commercial-paper/organization/digibank/configuration/cli/
  - docker-compose -f docker-compose.yml up -d cliDigiBank

    Creating cliMagnetoCorp ... done
  - cd commercial-paper/organization/digibank/application/
  - ls

    addToWallet.js buy.js package.json redeem.js
  - npm install

DigiBank User

- As a DigiBank user
  - `cd commercial-paper/organization/digibank/application/`
  - `node addToWallet.js`
  - `node buy.js`

```
Connect to Fabric gateway.
Use network channel: mychannel.
Use org.papernet.commercialpaper smart contract.
Submit commercial paper buy transaction.
Process buy transaction response.
MagnetoCorp commercial paper : 00001 successfully purchased by DigiBank
Transaction complete.
Disconnect from Fabric gateway.
Buy program complete.
```

  - `node redeem.js`

```
Connect to Fabric gateway.
Use network channel: mychannel.
Use org.papernet.commercialpaper smart contract.
Submit commercial paper redeem transaction.
Process redeem transaction response.
MagnetoCorp commercial paper : 00001 successfully redeemed with MagnetoCorp
Transaction complete.
Disconnect from Fabric gateway.
Redeem program complete.
```