Hyperledger/fabric 0.6 on GCP deploy

盧瑞山 教授

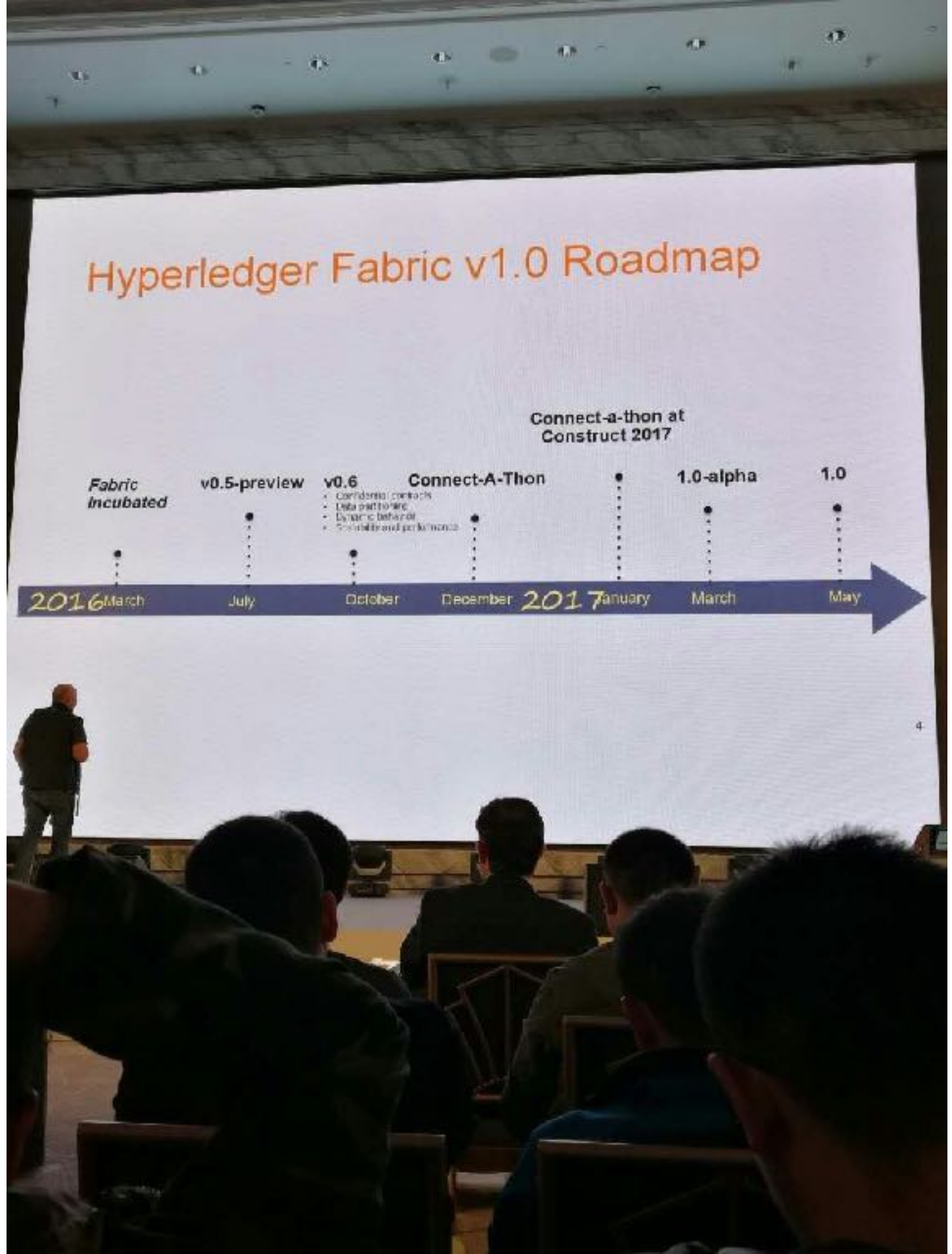
全球區塊鏈陣營

New kid on the block

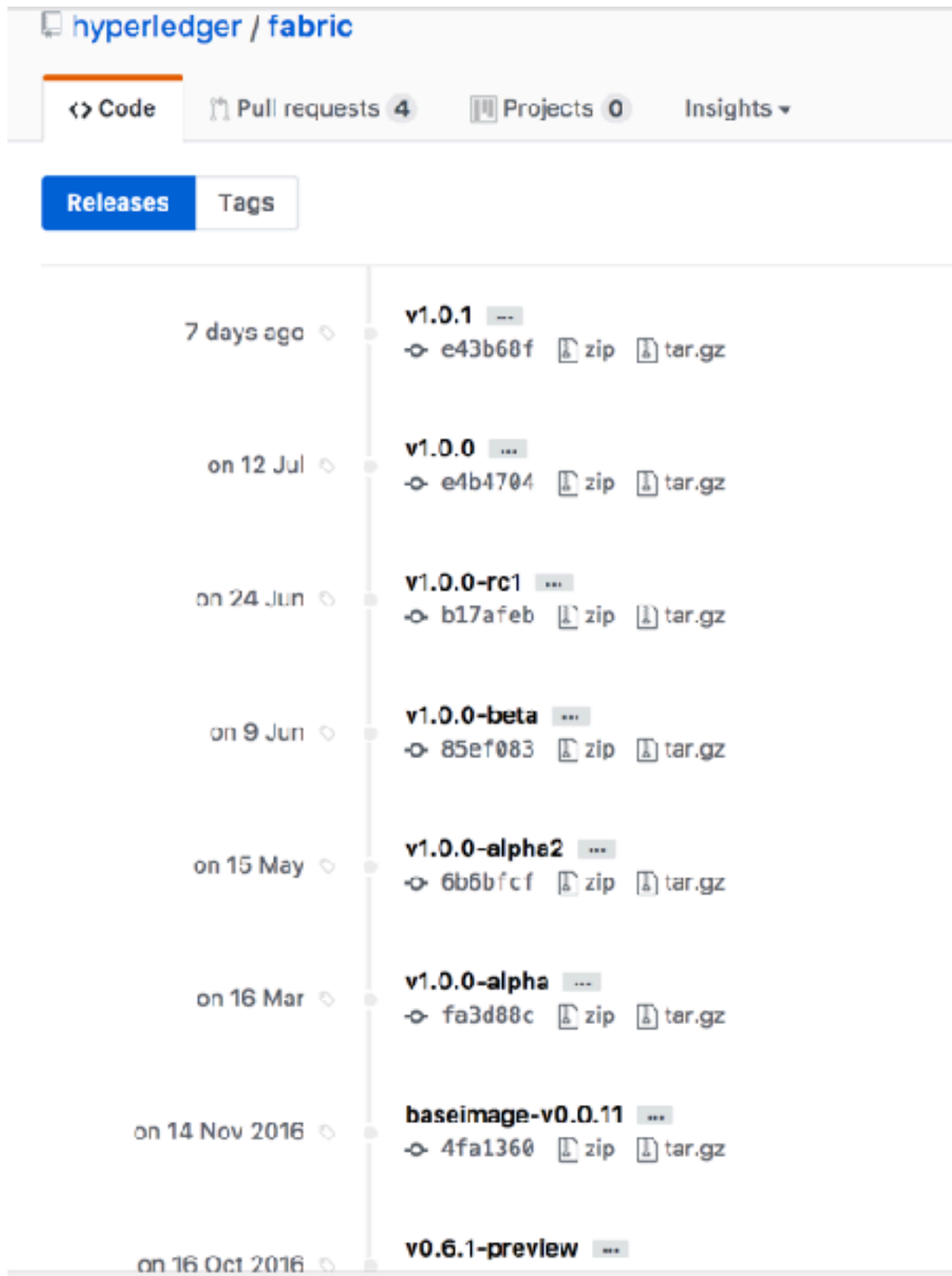
Enterprise Ethereum Alliance is the latest addition to a list of companies and consortiums focused on blockchain where banks serve as investors or partners

	No. of partners
Utility Settlement Coin	5
Global Payments Steering Group	6
Chain	9
Digital Asset Holdings	15
Enterprise Ethereum Alliance	30
R3	81
Ripple	90+
Hyperledger	122

Source: Staff research


Hyperledger Fabric v1.0 Roadmap
Connect-a-thon at
Construct 2017
Fabric
Incubated
v0.5-preview
v0.6
Connect-A-Thon
1.0-alpha
1.0
2016
March
July
October
December
2017
January
March
May
4

Hyperledger/Fabric的沿革



建議選擇Ubuntu or Mac 為開發學習環境

Why？

hyperledger-archives / fabric

Watch 217 | Star 1,063 | Fork 1,088

Code | Issues 348 | Pull requests 18 | Projects 0 | Wiki | Pulse | Graphs

make node-sdk fails on Windows platform #1757

New issue

Open jjjjibm opened this issue on 9 Jun 2016 · 10 comments

jjjjibm commented on 9 Jun 2016 • edited

npm.zip

Description

On my Win7 laptop, following instructions here: https://github.com/smithbk/fabric/blob/issue-1186/sdk/node/README.md

Building the client SDK

If you set the FABRIC environment variable to point to your hyperledger/fabric directory, you can build the client SDK as follows:

cd $FABRIC && make node-sdk

make node-sdk fails.

Assignees

No one assigned

Labels

api-sdk

Projects

None yet

Milestone

No milestone

Notifications

Subscribe

Hyperledger/Fabric區塊鏈系統開發實務

報名/Enroll

課程編號/Course Code	HZHF01	課程級別/Skill Level	初級
課程分類/Curricula	Cloud	授課方式/Delivery Type	教室授課
授課語言/Language	中文	上機實驗/Hands-on Labs	有
價格 (元) /Price	NT$ 20000	天　數/Duration	2
點　卷	40	先修課程	詳見預備技能

授課地點 Location	2月	3月	4月	5月	6月	7月	8月	9月
台北		25-26	13-14					

課程描述/Course Description：

開發hyperledger/fabric 應用系統

授課對象/Target Audience：

IT 工程師

主要課題/Course Outline：

區塊鏈緣起

區塊鏈工作原理

使用IBM bluemix blockchain service 快速進入區塊鏈世界

bluemix blockchain Service 應用在物聯網 (IoT)

hyperledger/fabric 架構分析

在 window安裝 Docker Toolbox 執行hyperledger App (for docker)

在 window以vagrant 建立 Hyperledger/Fabric App開發環境(for Nodejs)

docker images (for peer,membersrvc) 編譯及使用

hyperledger/fabric App 開發流程介紹

使用 go 語言撰寫 chaincode,並以postman REST API將 chaincode部署在 blockchain 之 peer node,同時將client端參數透過chaincode 寫入blockchain (或讀取),也可檢視 block之內容

在 Hyperledger App (for Node.js) 使用 HFC SDK 將 chaincode部署於blockchain

在 local 環境建立 blockchain network

marbles, car-lease, cp-web IBM區塊鏈範例程式分析

使用 chrome develop tool 除錯 nodejs app

在 Native linux (for ubuntu) 建構 Hyperledger/Fabric開發環境

使用Cloudsoft Amp 將 Hyperledger App部署於其他雲端平台

使用多帳戶Bluemix Container部署 Hyperledger Network(請IBM工程師主講)

2016年12月31日 ·

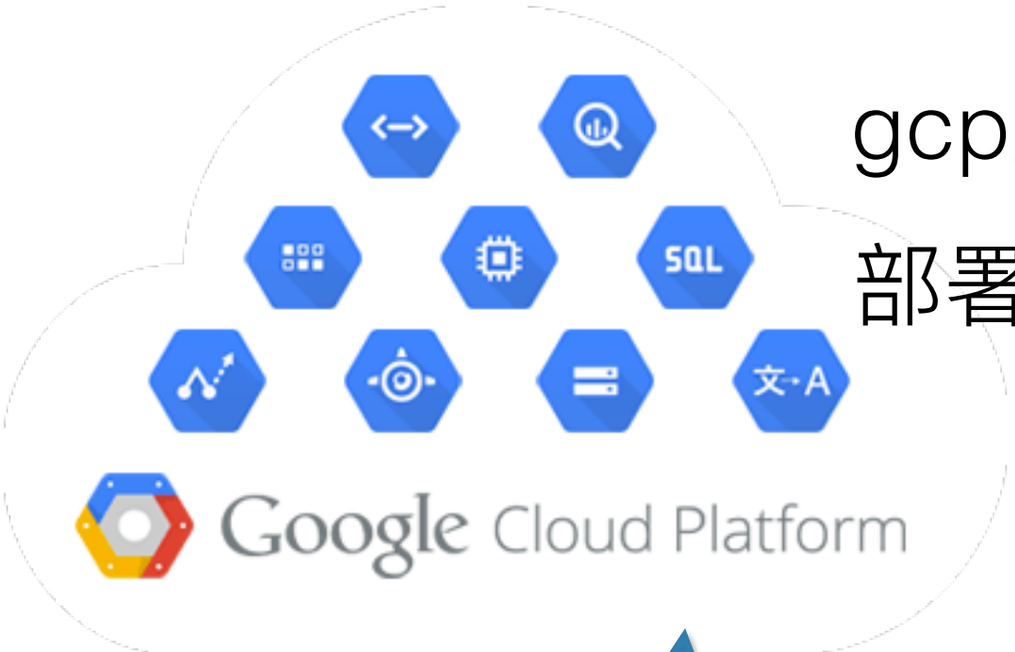
Wondow 系統使用 vagrant 在 virtualbox 建立虛擬主機 hyperledger並完成hyperledger/fabric 開發環境之架設後, 此時若 user在系統執行npm install hfc (安裝hyperledger fabric client SDK套件) 指令時會發生錯誤:
https://github.com/hyperledger-archives/fabric/issues/1757
雖然有替代方法,建議 user 可在 ubuntu linux 主機直接建構 hyperledger/fabric 開發環境 (不使用 vagrant) 以下網址範例為建構操作程序。
https://1drv.ms/b/s!AkBmzWP0h-VxgiZTYD3cxbsp-a7H

https://1drv.ms/f/s!AkBmzWP0h-Vxbr7h1uTNsTMcPWI

在cloud service部署節點時的考量

- 使用雲端服務建立節點時,用windows OS 的花費比linux OS 多一倍

Hyperledger/Fabric v0.6實作教學範例說明



gcp上跑一台vm上面跑4個peers

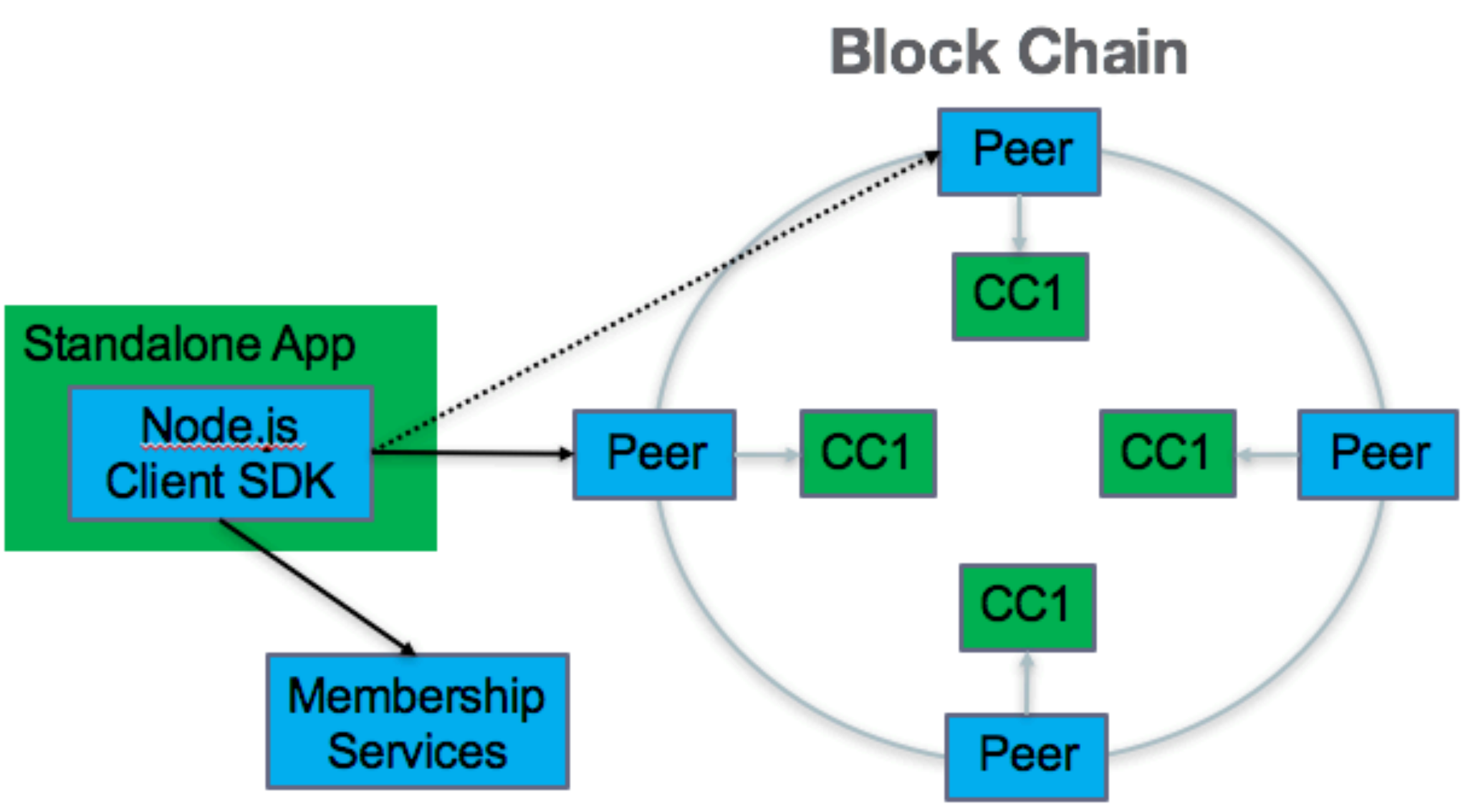
部署的智能合約為ibm範例example02

api請求

用rest api向peer發請求,
調用chaincode

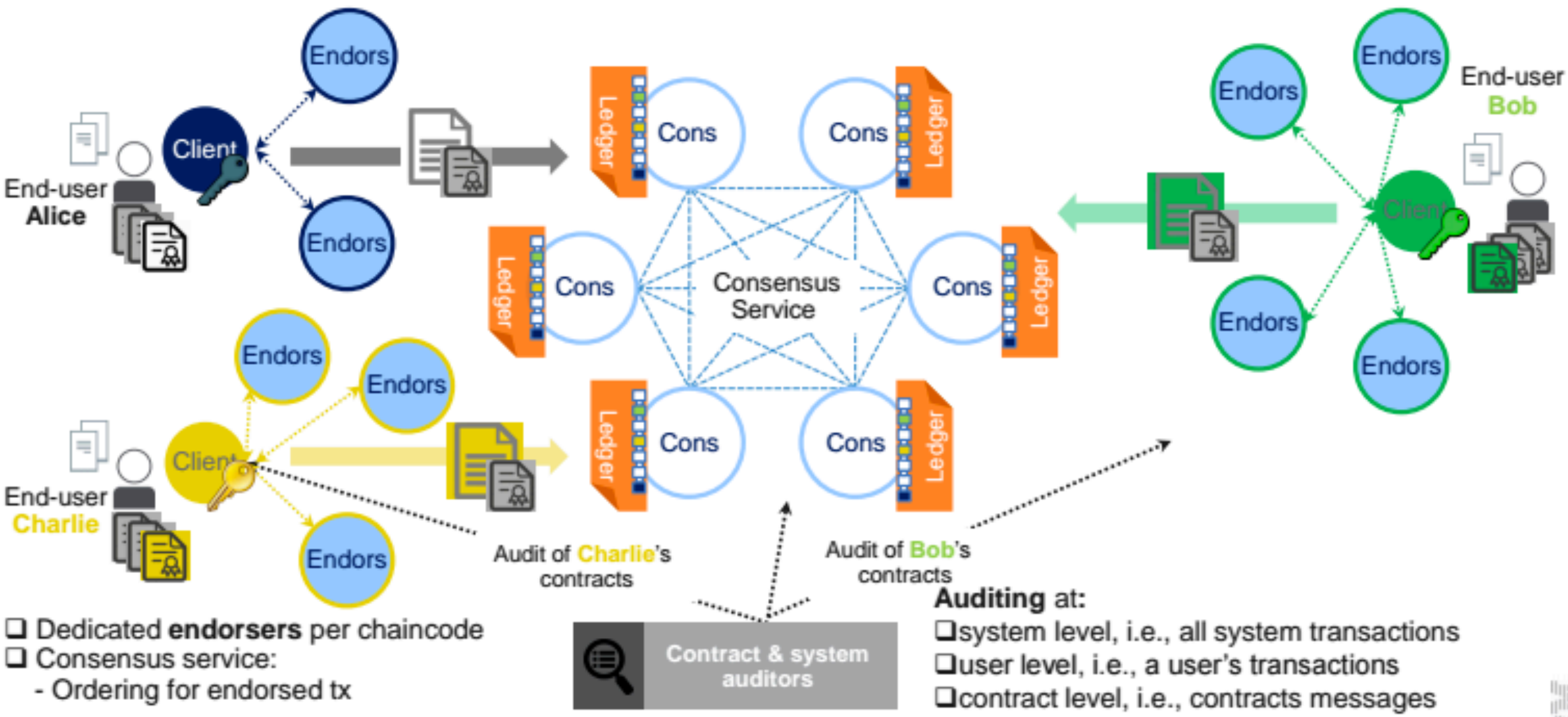
從localhost 去部署雲
端上的四個節點

Fabric 0.6具象化的架構



Fabric v1.0.0 具象化的架構

Separating transaction endorsement from consensus



- ❑ Dedicated **endorsers** per chaincode
- ❑ Consensus service:
  - Ordering for endorsed tx

Auditing at:

- ❑system level, i.e., all system transactions
- ❑user level, i.e., a user's transactions
- ❑contract level, i.e., contracts messages

Ubuntu上先安裝docker

- sudo apt-get update
- **sudo curl -sSL https://get.docker.com/ | sh**

拉docker所用的images

- sudo docker pull hyperledger/fabric-peer:x86_64-0.6.1-preview

- sudo docker tag hyperledger/fabric-peer:x86_64-0.6.1-preview hyperledger/fabric-peer:latest

- sudo docker tag hyperledger/fabric-peer:x86_64-0.6.1-preview hyperledger/fabric-baseimage:latest

啟動第一個節點

```
docker run --name=vp0 \
    -p 7050:7050 \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -e CORE_VM_ENDPOINT=unix:///var/run/docker.sock \
    -e CORE_LOGGING_LEVEL=debug \
    -e CORE_PEER_ID=vp0 \
    -e CORE_PEER_NETWORKID=dev \
    -e CORE_PEER_VALIDATOR_CONSENSUS_PLUGIN=pbft \
    -e CORE_PEER_ADDRESSAUTODETECT=true \
    -e CORE_PBFT_GENERAL_N=4 \
    -e CORE_PBFT_GENERAL_MODE=batch \
    -e CORE_PBFT_GENERAL_TIMEOUT_REQUEST=10s \
    --rm hyperledger/fabric-peer:latest peer node start
```

啟動第二個節點

```
docker run --name=vp1 \
    -p 8050:7050 \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -e CORE_VM_ENDPOINT=unix:///var/run/docker.sock \
    -e CORE_LOGGING_LEVEL=info \
    -e CORE_PEER_ID=vp1 \
    -e CORE_PEER_NETWORKID=dev \
    -e CORE_PEER_VALIDATOR_CONSENSUS_PLUGIN=pbft \
    -e CORE_PEER_ADDRESSAUTODETECT=true \
    -e CORE_PBFT_GENERAL_N=4 \
    -e CORE_PBFT_GENERAL_MODE=batch \
    -e CORE_PBFT_GENERAL_TIMEOUT_REQUEST=10s \
    -e CORE_PEER_DISCOVERY_ROOTNODE=172.17.0.2:7051 \
    --rm hyperledger/fabric-peer:latest peer node start
```

啟動第三個節點

```
docker run --name=vp2 \
    -p 9050:7050 \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -e CORE_VM_ENDPOINT=unix:///var/run/docker.sock \
    -e CORE_LOGGING_LEVEL=info \
    -e CORE_PEER_ID=vp2 \
    -e CORE_PEER_NETWORKID=dev \
    -e CORE_PEER_VALIDATOR_CONSENSUS_PLUGIN=pbft \
    -e CORE_PEER_ADDRESSAUTODETECT=true \
    -e CORE_PBFT_GENERAL_N=4 \
    -e CORE_PBFT_GENERAL_MODE=batch \
    -e CORE_PBFT_GENERAL_TIMEOUT_REQUEST=10s \
    -e CORE_PEER_DISCOVERY_ROOTNODE=172.17.0.2:7051 \
    --rm hyperledger/fabric-peer:latest peer node start
```

跑起第四個節點

```
docker run --name=vp3 \
    -p 10050:7050 \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -e CORE_VM_ENDPOINT=unix:///var/run/docker.sock \
    -e CORE_LOGGING_LEVEL=info \
    -e CORE_PEER_ID=vp3 \
    -e CORE_PEER_NETWORKID=dev \
    -e CORE_PEER_VALIDATOR_CONSENSUS_PLUGIN=pbft \
    -e CORE_PEER_ADDRESSAUTODETECT=true \
    -e CORE_PBFT_GENERAL_N=4 \
    -e CORE_PBFT_GENERAL_MODE=batch \
    -e CORE_PBFT_GENERAL_TIMEOUT_REQUEST=10s \
    -e CORE_PEER_DISCOVERY_ROOTNODE=172.17.0.2:7051 \
    --rm hyperledger/fabric-peer:latest peer node start
```

```
docker ps -a | xargs docker rm -f
```

編譯智能合約

進入docker的VM之中 docker exec -it container_name /bin/bash (可用docker ps 查詢container_name)

cd /opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02 進入到存放chaincode範例的目錄下

go build //編譯chaincode

部署第一個智能合約

挑一個節點，可以是vp0,vp1,vp2,vp3中的任一個

docker exec -it vp2 bash

peer chaincode deploy \

-p github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02 \

-c '**{"function":"init", "args": ["a" , "100" , "b" , "200"]}**'

chaincode被部署時會把編譯過的cc放到/opt/gopath/bin/

chaincode_name=ee5b24a1f17c356dd5f6e37307922e39ddba12e5d2e203ed93401d7d05eb0dd194fb9070549c5dc31eb63f4e654dbd5a1d86cbb30c48e3ab1812590cd0f78539

chaincode的調用

- 先給智能合約的執行檔的長檔名存成一個變數以供後用

- **chaincode_name**=ee5b24a1f17c356dd5f6e37307922e39ddba12e5d2e203ed93401d7d05eb0dd194fb9070549c5dc31eb63f4e654dbd5a1d86cbb30c48e3ab1812590cd0f78539

- peer chaincode query -n **$chaincode_name** -c '{"Function": "query", "Args": ["a"]}'

- peer chaincode invoke -n **$chaincode_name** -c '{"Function": "invoke", "Args": ["a", "b", "35"]}'

- peer chaincode query -n **$chaincode_name** -c '{"Function": "query", "Args": ["a"]}'

檢視區塊鏈

- REST API
- 除了客戶端調用chaincode以外
  還有api用來檢視區塊鏈
- GET /transactions/{UUID} 用來檢視交易資訊
- GET /chain/blocks/{Block} 用來檢視該區塊資訊
- GET /chain 用來檢視區塊鏈狀態

- 還有一個 API 用來檢視各節點資訊
- GET /network/peers

{"peers":[{"ID":{"name":"jdoe"},"address":"0.0.0.0:7051","type":1}]}

- —peer-chaincodedev 只有一個節點

hyperledger chaincode開發

- hyperledger 官方建議使用Go語言 開發chaincode 另可以使用java做開發,但功能較有限所以不建議

- 範例環境:
  IDE:VS Code 可以使用自己慣用的IDE開發
  Peer:Docker on MacOS

- 其他:所有範例(v0.6)皆在 —peer-chaincodedev 模式下示範

範例1-chaincode的必要元素

- 讓我們先來看一段原始碼,再為你介紹一下

```go
    "github.com/hyperledger/fabric/core/chaincode/shim" //inculde 這檔案
)

//golang的結構相當於 c,java 結構,物件 [必要]
type Sample1 struct {
}
//deploy 所執行的函數[必要]
func (this *Sample1) Init(stub shim.ChaincodeStubInterface, function string, args []string) ([]byte, error) {
    fmt.Printf("function:%s\nargs:%#v", function, args)
    return []byte("[Sample1]deploy成功之訊息"), nil
}
//invoke 所執行的函數[必要]
func (this *Sample1) Invoke(stub shim.ChaincodeStubInterface, function string, args []string) ([]byte, error) {
    fmt.Printf("function:%s\nargs:%#v", function, args)
    return nil, errors.New("[Sample1]invoke失敗之訊息")
}
//query 所執行的函數[必要]
func (this *Sample1) Query(stub shim.ChaincodeStubInterface, function string, args []string) ([]byte, error) {
    fmt.Printf("function:%s\nargs:%#v", function, args)
    return []byte("[Sample1]查詢完成之返回訊息"), nil
}
//程式進入點
func main() {
    //main函數暫停在這行,等待與peer互動,直到shim意外中斷
    err := shim.Start(new(Sample1))
    if err != nil {
        fmt.Printf("Error starting chaincode: %s", err)
    }

    fmt.Print("Sample1")
}
```

- 必須import
  “github.com/hyperledger/fabric/core/chaincode/shim”

- 必須要定義個結構 範例一 為 struct Sample1

- 必須實作 Init , Invoke , Query 這三個函數,其格式
  func (this *Sample1) 函數(stub shim.ChaincodeStubInterface
  ,function string, args []string) ([]byte, error)

- 需執行 shim.Start(new(Sample1)) 本函數

範例五-實際應用

- 調用範例二 chaincode
  使用智能合約由Bob 轉帳給 Alice
  Alice 必須在兩分鐘後在特定座標範圍內才可獲得該點數

- 建立一個表格 用於記錄

```
func (this *Sample5) Init(stub shim.ChaincodeStubInterface, function string, args []string) ([]byt
    //創建表格用來記錄智能合約事件
    stub.CreateTable("ccTable", []*shim.ColumnDefinition{
        &shim.ColumnDefinition{Name: "User", Type: shim.ColumnDefinition_STRING, Key: true},
        &shim.ColumnDefinition{Name: "point", Type: shim.ColumnDefinition_INT32, Key: false},
        &shim.ColumnDefinition{Name: "x", Type: shim.ColumnDefinition_INT32, Key: false},
        &shim.ColumnDefinition{Name: "y", Type: shim.ColumnDefinition_INT32, Key: false},
        &shim.ColumnDefinition{Name: "m", Type: shim.ColumnDefinition_INT32, Key: false},
        &shim.ColumnDefinition{Name: "unixtime", Type: shim.ColumnDefinition_INT64, Key: false},
    })
```

定義 sendPoint

```
79          case "sendPoint":
80              m = 10
81              i := 0
82              for ; i < len(args); i++ {
83                  if _, err := strconv.Atoi(args[i]); err == nil {
84                      temp, _ := strconv.Atoi(args[i])
85                      x = int32(temp)
86                      temp, _ = strconv.Atoi(args[i+1])
87                      y = int32(temp)
88                      i += 1
89                  } else if args[i] == "m" {
90                      temp, _ := strconv.Atoi(args[i+1])
91                      m = int32(temp)
92                      i += 1
93                  } else if args[i] == "time" {
94                      temp, _ := strconv.Atoi(args[i+1])
95                      unixtime = time.Now().Unix() + int64(temp*60)
96                      i += 1
97                  } else if args[i] == "point" {
98                      temp, _ := strconv.Atoi(args[i+1])
99                      point = int32(temp)
100                     i += 1
```

```go
} else if args[i] == "from" {
    //修改給予人的金額
    fromuser := args[i+1]
    funcAndArgs := util.ToChaincodeArgs("get", fromuser)
    response, _ := stub.QueryChaincode(ccName, funcAndArgs)
    temp, _ := strconv.Atoi(string(response))
    temp -= int(point)
    funcAndArgs = util.ToChaincodeArgs("put", fromuser, strconv.Itoa(int(temp)))
    info, err := stub.InvokeChaincode(ccName, funcAndArgs)
    if err != nil {
        shim.NewLogger("Error").Errorf("%v", err.Error())
        return nil, err
    }
    shim.NewLogger("Info").Infof("%v", info)
    i += 1
}
```

```
i = i - 1
//新增一筆智能合約事件
stub.InsertRow("ccTable", shim.Row{
    Columns: []*shim.Column{
        &shim.Column{Value: &shim.Column_String_{String_: args[i]}},
        &shim.Column{Value: &shim.Column_Int32{Int32: point}},
        &shim.Column{Value: &shim.Column_Int32{Int32: x}},
        &shim.Column{Value: &shim.Column_Int32{Int32: y}},
        &shim.Column{Value: &shim.Column_Int32{Int32: m}},
        &shim.Column{Value: &shim.Column_Int64{Int64: unixtime}},
    },
})
break
```

定義 getPoint

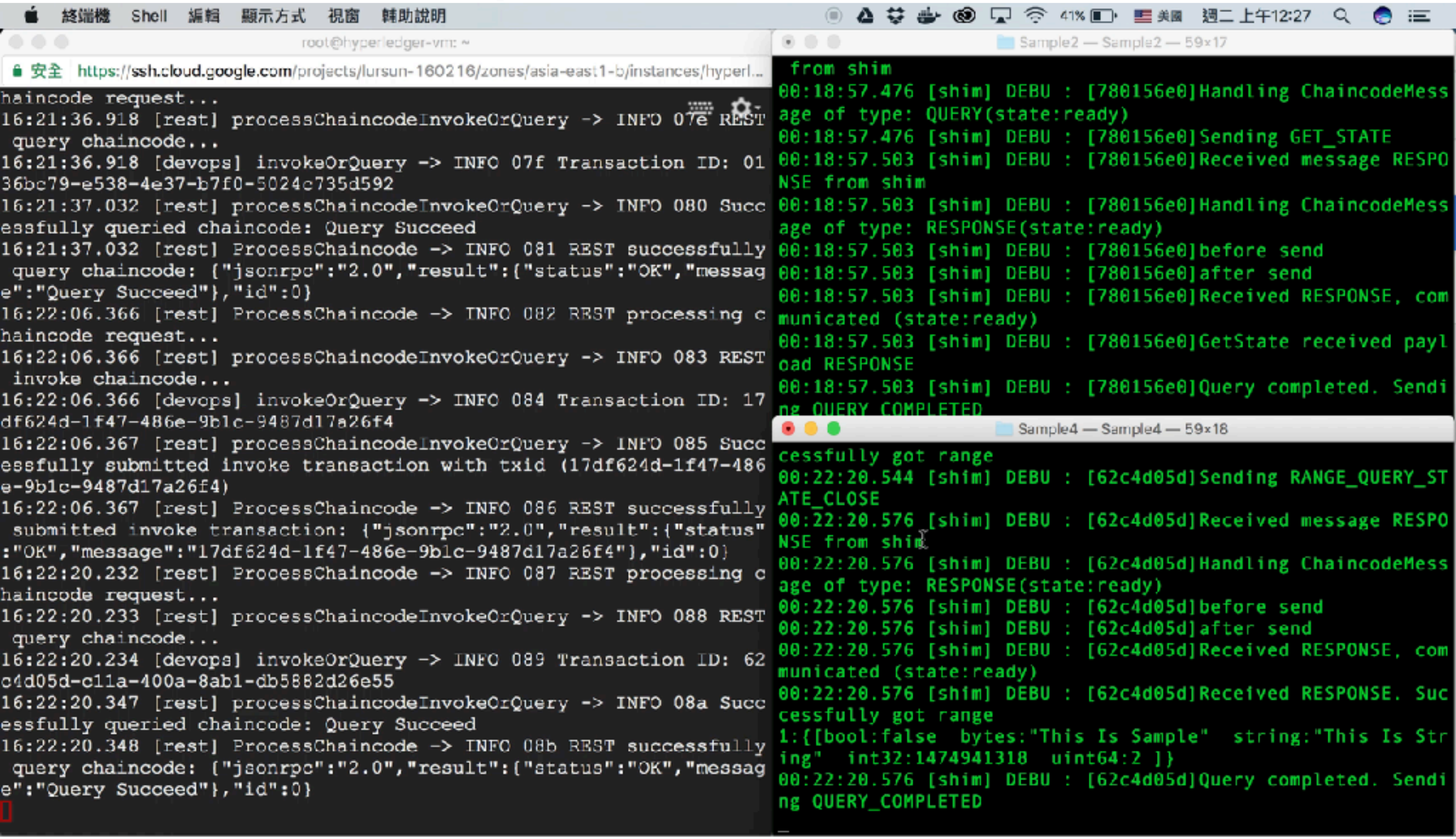
```
36        switch function {
37        case "getPoint":
38            var cols []shim.Column
39            cols = append(cols, shim.Column{Value: &shim.Column_String_{String_: args[0]}})
40            //查詢智能合約事件表
41            row, err := stub.GetRow("ccTable", cols)
42            if err != nil {
43                shim.NewLogger("Error").Errorf("%v", err.Error())
44                return nil, err
45            }
46            //比對是否時間已到
47            if time.Now().Unix() > row.Columns[5].GetInt64() {
```

```go
if len(args) == 3 {
    x, err := strconv.Atoi(args[1])
    if err != nil {
        shim.NewLogger("Error").Errorf("%v", err.Error())
        return nil, err
    }

    y, err := strconv.Atoi(args[2])
    if err != nil {
        shim.NewLogger("Error").Errorf("%v", err.Error())
        return nil, err
    }
    //比對地點是否已達
    m := math.Pow(float64(row.Columns[2].GetInt32()-int32(x)), 2) + math.Pow(flo
    if float64(row.Columns[4].GetInt32()) >= math.Sqrt(m) {
```

```go
if float64(row.Columns[4].GetInt32()) >= math.Sqrt(m) {
    //修改受予人金額
    funcAndArgs := util.ToChaincodeArgs("get", args[0])
    response, _ := stub.QueryChaincode(ccName, funcAndArgs)
    temp, _ := strconv.Atoi(string(response))
    temp += int(row.Columns[1].GetInt32())
    funcAndArgs = util.ToChaincodeArgs("put", args[0], strconv.Itoa(int(temp)))
    stub.InvokeChaincode(ccName, funcAndArgs)
    //刪除此智能合約事件
    stub.DeleteRow("ccTable", cols)
}
```

Sample5 操作影片



v1.0 preview基本名詞

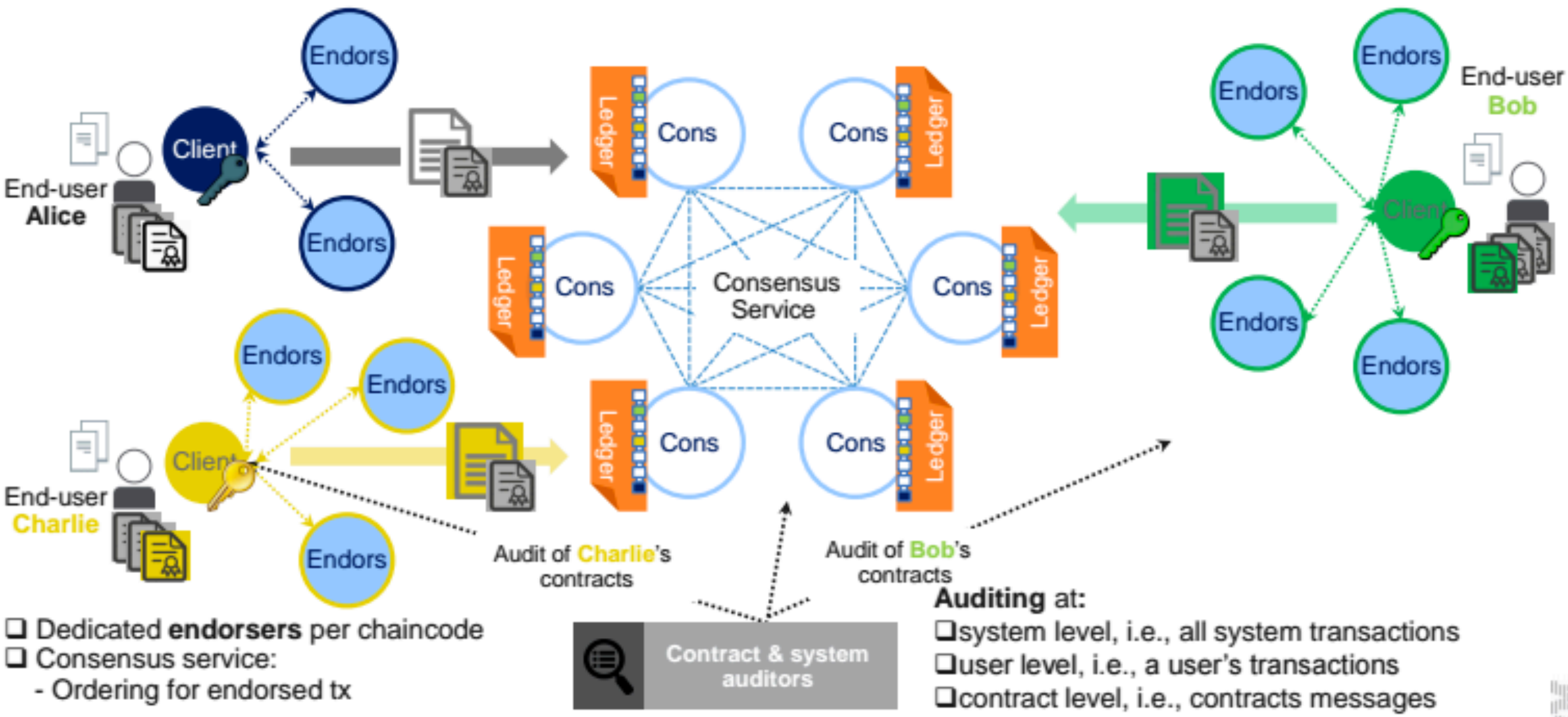
- submitting peer: 請求節點
- endorser: 背书节点
- consenters: 提交節點或稱投票节点
- committer: 承諾節點 或稱支持節點
- chaincode: 链码（区块链应用程序）

v1.0 alpha components

- Fabric network
- channels
- peers
    1. endoser peer
    2. committing peer
- chaincode
- sdk

Fabric v1.0.0 具象化的架構

Separating transaction endorsement from consensus



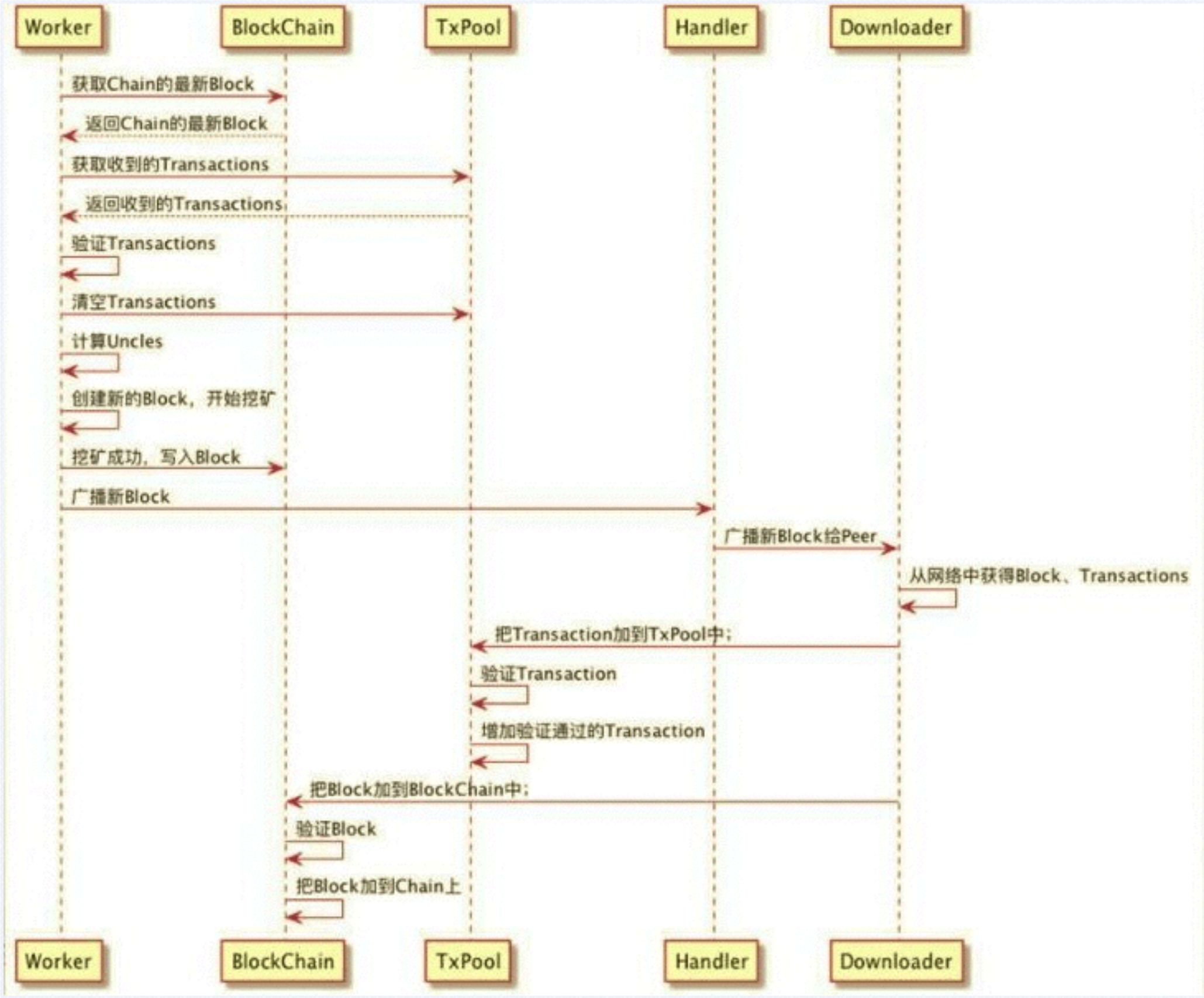
- ❑ Dedicated **endorsers** per chaincode
- ❑ Consensus service:
  - Ordering for endorsed tx

Auditing at:

- ❑system level, i.e., all system transactions
- ❑user level, i.e., a user's transactions
- ❑contract level, i.e., contracts messages

Unix time的轉換

- date -d @Unix timestamp
- date +%s
- date -r unix time


Worker
BlockChain
TxPool
Handler
Downloader
获取Chain的最新Block
返回Chain的最新Block
获取收到的Transactions
返回收到的Transactions
验证Transactions
清空Transactions
计算Uncles
创建新的Block，开始挖矿
挖矿成功，写入Block
广播新Block
广播新Block给Peer
从网络中获得Block、Transactions
把Transaction加到TxPool中；
验证Transaction
增加验证通过的Transaction
把Block加到BlockChain中；
验证Block
把Block加到Chain上
Worker
BlockChain
TxPool
Handler
Downloader