3. 하이퍼레저 패브릭

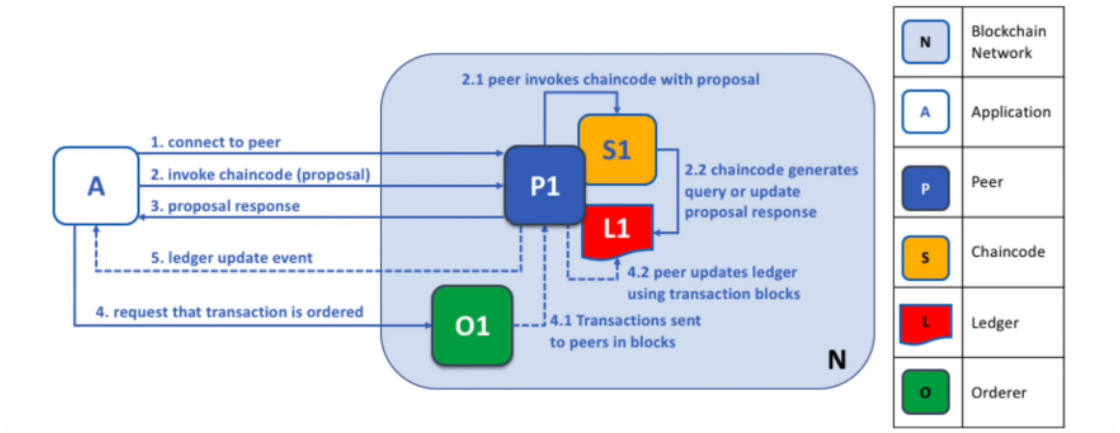
노트북:	블록체인		
만든 날짜:	2019-08-30 오전 9:03	**업데이트:**	2019-08-30 오후 5:27
작성자:	감자		

하이퍼레저 패브릭 공식문서 해석해놓은 블로그
https://miiingo.tistory.com/105?category=644184

하이퍼레저 패브릭 개념설명 유튜버 : dapp campus

[Hyperledger Fabric 개념] 01. Fabric 구조

이 그림을 이해하는것이 중요



비트, 이더와 같은점과 다른점 (비트,이더 : public blockchain / Fabric : private blockchain)

1. **비트, 이더**

   a. 블록 생성자 = 스마트 컨트랙트 처리 노드
   b. 개인키 기반
      - 개인적으로 private key를 만들어서 그 기반으로 돈을 주고받고 한다.

2. **Fabric**
   a. 블록생성자 != 스마트 컨트랙트 처리 노드
   b. Orderer = 블록 생성만
   c. Peer = 시뮬레이팅, 블록체인 보관
   d. 인증서 기반
      - 은행에서 발급받은 공인인증서를 사용하듯이 중간관리자가 있다는 뜻이다.

- Fabric ca라는 중간관리자가 있다. (cerification authority)
- 이 중간관리자가 인증서를 발급해줌. 패브릭은 이 인증서를 기반으로 활동함

e. 하나의 키 값을 한 블록에서 여러 번 바꿀 수가 없다(일반적으로 생각하는 방식으로)
- 한 블록에서 A가 100이었다면 A에 +100해서 200 이런식으로 비트, 이더에서는 가능하지만 패브릭에서는 안됨. 이것이 패브릭의 아키텍처때문에 발생하는 가장 큰 차이점임

패브릭 vs 비트,이더의 가장 큰 차이점

```
비트와 이더는 프로그램하나가 모든것을 처리. 패브릭은 프로그램 하나가 아니라 구성요소가 나누어져있어서 각 역할이 나누어져있다.
비트와 이더는 geth를 깔면 얘네를 가지고 마이닝, 지갑역할, 블록체인 저장, 월드스테이트 가지고 있어서 프로그램하나깔면 모든 역할을 다 할 수 있다.
패브릭같은 경우는 역할이 나누어져있어서 Orderer는 블록 생성만, Peer는 시뮬레이팅, 블록체인보관. Orderer도 블록체인을 보관하긴하지만 일단 ㅇㅋㅇㅋ 암튼 역할을 나누어서 처리한다.
```

구성요소

1. Peer

- 블록체인 보관 (Ledger : 안에 World State를 보관) ,
- 거래 트랜잭션 시뮬레이팅

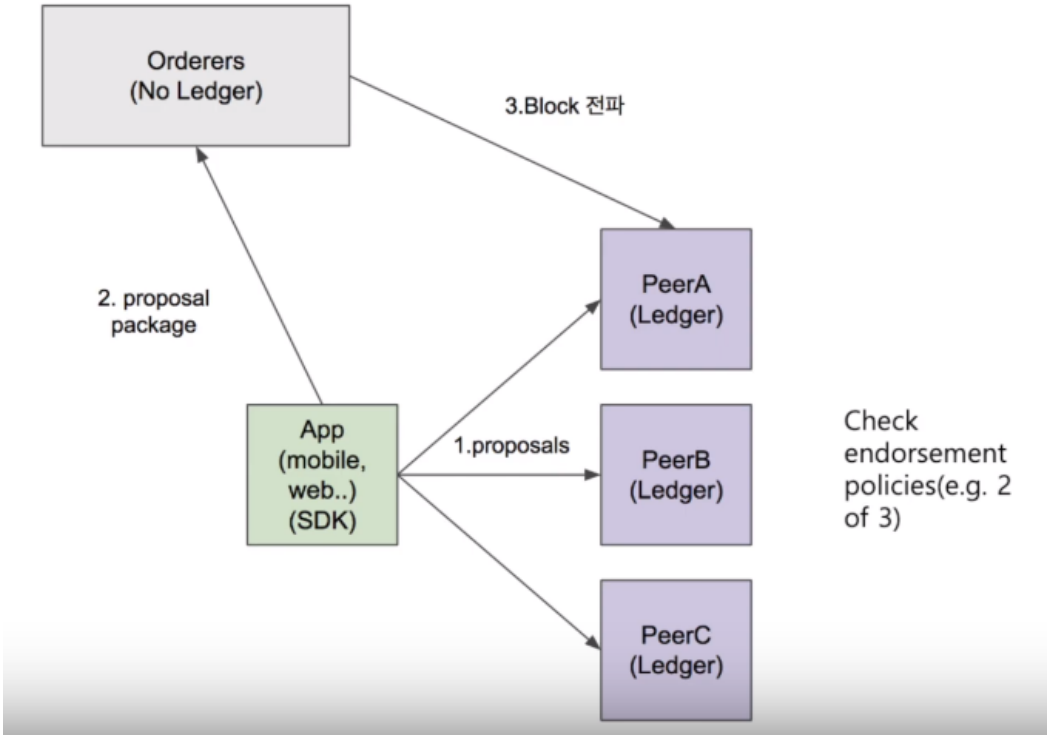
1. Orderer

- 블록을 생성 : 거래(트랜잭션)을 모아서 블록을 생성만 함. 거래 트랜잭션 시뮬레이팅은 하지않음

1. App(유저)

- 인증서/지갑을 가지고 거래를 만들고 Peer, Orderer와 상호작용

-Orderers는 실제로는 Ledger를 가지지만 아래 그림에선 이해를 위해 Orderers가 Ledger를 가지고 있지 않다고 가정



0.유저(App)가 거래를 만듬
1.proposals (endorsement policies)

- 유저가 Peer들에게 거래를 보냄
- 이 거래가 블록에 들어가는게 아니라 제안임
- Peer는 이 거래를 시뮬레이팅함 : 이 거래가 어떤지 확인
- Peer는 이 거래가 괜찮으면 OK를 보냄 -> 패브릭에서의 합의 방식 (투표)

  - BFT
  - 3명 중 2명이 OK하면 됨

- 3명중 2명에게 허락을 받으면 이 거래는 이제 Orderers에게 보낼 수 있음: 검증을 받은 트랜잭션이야 block에 넣어줘!

2.proposal package (허락을 받은것만 패키지에 모아서 orderer에게 보냄)

- Orderer는 그 거래를 모아서 Peer들에게 허락을 받았는지만 검증을 한다.
- 스마트 컨트랙트가 어떤내용으로 실행되는지 어떤 내용인지는 검증하지 않는다.
- 블록을 만들어서 Peer에게 보낸다.

3.Block 전파

- Orderer는 거래를 검증 후 블록을 따다다ㅏ 만들어서 Peer들에게 보내준다.
- peer는 그 블록을 받아서 블록체인에 연결하고 월드스테이트 업데이트함.

Ledger

1. 블록체인
2. World state(Peer) : 최신상태들, Peer만 보관
3. Peer만 들고 있음(활용한다) : orderer도 가지고있지만 활용하진 않고 배포용임

[Hyperledger Fabric 개념] 02. Fabric Read & Write Set

*패브릭에서 체인코드가 값을 쓰는 방식

- KVS(Key-Value Store)
- Level, couch : Level을 쓰면 좀 더 빠르지만couch를 쓰면 좀 더 복잡한 쿼리 가능.
- chaincode에서 값을 저장하는 방식 (걍 set key)

  - a : 100
  - b : hello

=> 하지만 무엇이 다른가! 한 블록 내에서 하나의 키 값을 여러번 읽어서 바꿔 쓸 수 없다.

즉, 한 블록안에서 한번 write 한 키에 대해서는 Read를 하면 실패한다.
Read Set, Write set 예시)

```
World state: (k1,1,v1), (k2,1,v2), (k3,1,v3), (k4,1,v4),
(k5,1,v5)
T1 -> Write(k1, v1'), Write(k2, v2')
T2 -> Read(k1), Write(k3, v3')
T3 -> Write(k2, v2'')
T4 -> Write(k2, v2'''), read(k2)
T5 -> Write(k6, v6'), read(k5)
```

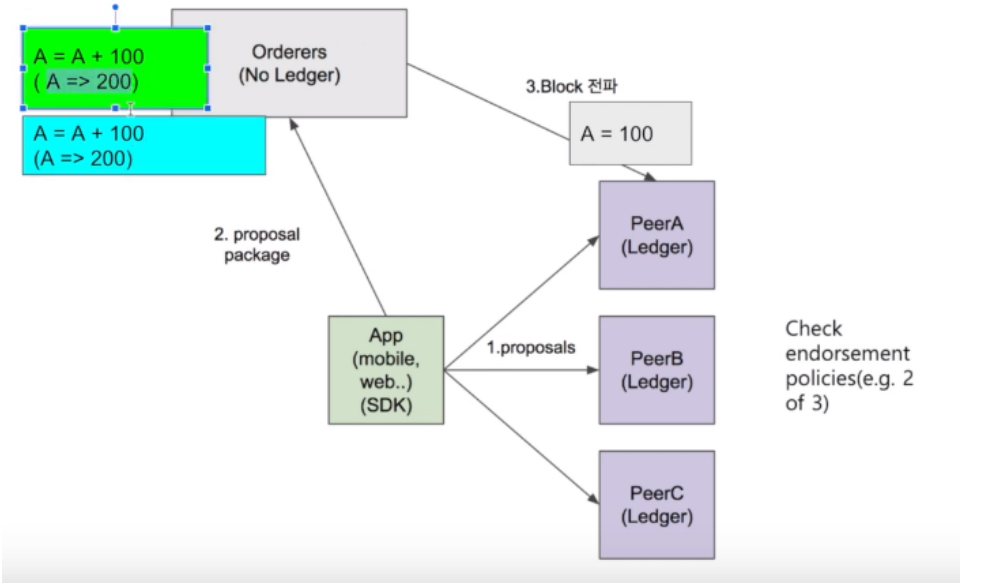
1. World state : 기본상태. k1에 k1저장되어있고 k2에 k2저장되어있고 ~쭉쭉. 1은 걍 버전넘버
2. T1

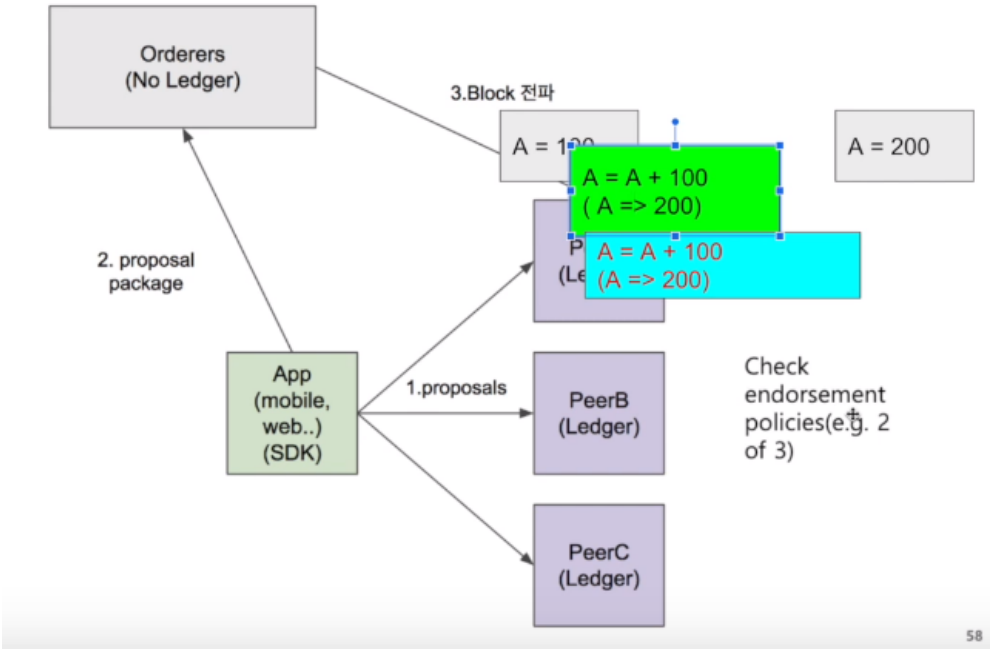
   - k1을 v1'로 k2를 v2'로 변경
   - 기존의 키값을 변경하기만 한것
   - success

3. T2

   - 한번 write한 key를 Read하기 때문에 Fail
4. T3
   - Read하지 않았으므로 success
   - v2'' 으로 write를 성공했지만 k2에 있는 v2'의 값을 기반으로 한것은 아님
5. T4 : Fail
6. T5: success (Reade를 했지만 이전에 write한적이 없으므로)

이더에서는 되고 패브릭에서는 안되는 이유 : Peer가 시뮬레이팅을 하는데 Peer가 블록을 만들지 않기 때문이다(Orderer가 만듬)





=> 실패한 트랜잭션도 블록에 들어옴

이더에서는 들어오는대로 시뮬레이팅하고, 새로운 블록의 트랜잭션들을 보관하고 있기때문에 히스토리를 알고있다.
그래서 A=100 -> A = A +100 = 200 -> A = A +100 = 300이 가능.

패브릭에서는 A=100 -> A = A+100 = 200이라는 트랜잭션 검증한다음에 ordered에 보내지고 자기가 블록을 만들지 않기때문에(orderer의 역할) A=200이 되었다는 사실을 모르고 있다.
(ordere가 블록을 넘겨줄때까진 A=100임)
또 새로운 트랜잭션이 와서 A= A + 100,을 해달라하면 A=100이기 때문에 A=200이 됨.

Orderer는 proposal package를 받고 지금까지 받았던 트랜잭션을 검증.
1. A = A + 100 (A =>200) : success (A를 Read했지만 write한 적 없으므로)
2. A = A + 100 (A =>200) : fail (A를 Read한것이 되므로)

Orderer부터 처리한 블록을 Peer가 전달 받은 후 A = 200이 됨

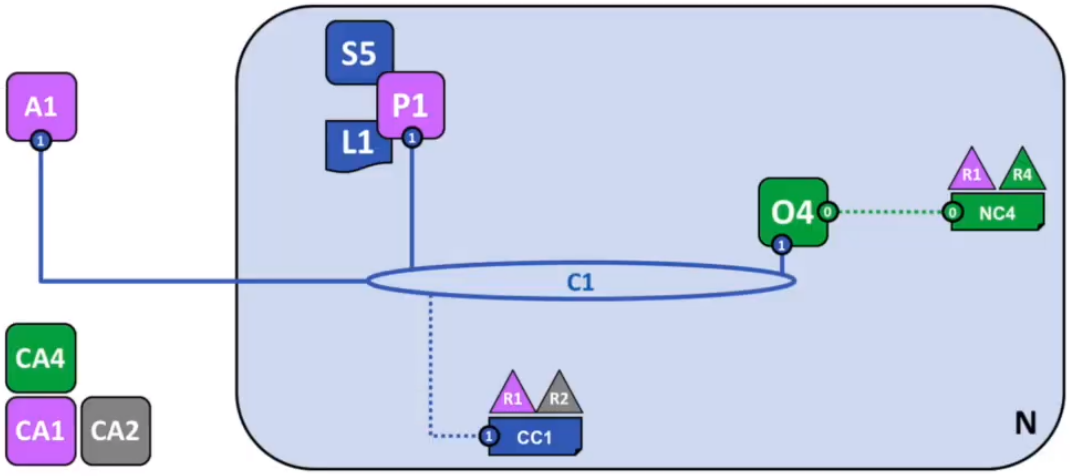
Ordere는 블록을 만들 때 스마트컨트랙트를 시뮬레이팅 하지 않는다. (No Ledger)
사실 Orderer Ledger를 가지고 있지만 Peer들이 다 끊기는 것과 같은 문제가 생기거나 새로 블록을 만들때 배포하기 위해 저장하기 위함임. Peer들의 시그니처만 확인함.

그래서 block 타임을 조절해 줄 수 있지만, 그 시간안에 키 값을 한번만 바꾼다고 장담할 수 없다....
그래서 Fabric이 선택한 방법은..!!!!!! Fabric-high through put network

Fabric - high through put network (태그를 사용 -> 태깅)
-A : 100 (이전블락)
-A1 : +100
-A2 : -50
-A3 : +100
-A : 250 (최종 쓰여지는 값)

-> 이렇게 하지 않으면 고성능을 낼 수 없다. 이렇게 하지 않으면 한 블록에 한 트랜잭션을 보내거나 해야하는데 하나의 블록에는 시그니처를 비롯해서 다양한 정보가 들어가기 때문에 오버헤드가 엄청 커짐. 그래서 이러한 방식의 키밸류를 이용한 것을 사용해야한다. 다른 방법도 있지만 이 방법이 일반적으로 사용하기 좋음.

[Hyperledger Fabric 개념] 04. Fabric 네트워크 세팅 가이드(이론)



```
A = App(유저)
L = Ledger
S = ChainCode
CA = CA
CC = Channel configuration
NC = Network Config
R = Organization
C = Channel
```

- 패브릭에서의 권한 관리
  - 인증서를 기반으로
  - `X.509`를 만들고 관리하고 삭제하는 것들을`CA`라는 component로 제공
- Fabric CA(Certificate Authority)를 통해서 MSP(Membership Service Provider)
- Channel
  - Peer들이 Channel에 가입할 수 있다.
  - 같은 Channel안에 있는 Peer들은 해당 Channel에 있는 스마트컨트랙트, 블록체인 데이터를 공유할 수 있다.
  - 이를 위해서는 같은 스마트컨트랙트가 있어야 한다.
- Organization
  - identity를 관리하는 하나의 group
  - 네트워크를 구성할 때 organizaion 단위로

네트워크 구성

1. 크립토 관련 작업
   - Fabric-ca
   - cryptogen
     - 크립토관련 인증서, 프라이빗 키 등등을 생성해주는 실행파일을 fabric에서 제공
2. Orderer 띄움
3. Peer를 띄움
4. 네트워크에서 Kafka, zookeeper를 띄워줌 : 위 그림에서 선같은 역할을함. 특히 실선
5. Channel을 생성
   - Channel에 Peer를 조인시킴
     - 이를 가능하게 하는것은 크립토 관련 작업을 위에서 해주었기 때문
6. 스마트컨트랙트를 만들고 배포(Chaincode)
7. Ledger 생성
8. App(유저) 생성
   - 자신의 인증서를 사전에 발급받고, 해당 peer를 통해 소통할 수 있게 됨

[Hyperledger Fabric 개념] 05. Fabric Identity -1

패브릭에서의 Identity는 결국 PKI를 기반으로 하는 인증서 기반의 시스템.

***각자 다른 컴포넌트들은 서명을 공개키를 가지고 검증하는 프로세스를 가진다.**

```
패브릭에 있는 구성요소인 Oderer, Peer, User들은 각각 자신의 개인키 , 공개키, 인증서가 있어서 User같은 경우는 트랜잭션을 만들어서 던질 때 자신의 개인키를 가지고 서명을 해서 던지고 Peer도 해당 트랜잭션을 검증했다할 때 자신이 가진 개인키를 가지고 서명을한다.
Orderer도 마찬가지로 블록을 생성하고 Peer에게 던져줄 때 자신이 서명을하고 던져준다.
```

***PKI**

1. 공개키 암호화

- 개인키 + 공개키
- 개인키 : 내가 보관

  - 소유자가 가지고 절대 비밀로 보관
  - 데이터에 서명
  - 공개키로 암호화한 데이터를 복호화 할 수 있다.

- 공개키 : 다른 컴포넌트들이 가지고있다.

  - 다른 사람에 알려주는 용도
  - 개인키로 암호화한 데이터를 복호화 할 수 있다.

2. 전자서명

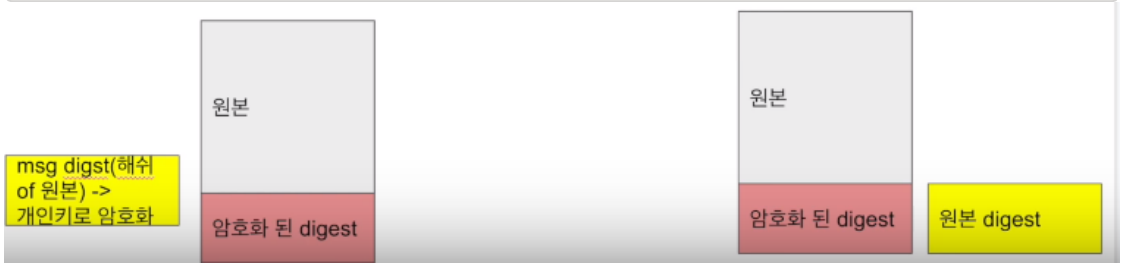
3. 인증서(X.509)

- 공개키 : 다른 컴포넌트들이 다 가지고 있게 됨

-> 개인키로 암호화 한것은 공개키로 복호화 할 수 있고, 공개키고 복호화 한것은 개인키로 복호화 할 수 있다.

전자서명

```
-데이터가 위/변조 되지 않았다 라는 거을 PKI 시스템에서 알려주기 위한 시스템
- 1. 원본의 digest(원본 해쉬)를 만든다 : msg digest
- 2. 개인키로 digest를 서명한 후에 원본에 붙인다 : 원본데이터를 해쉬화한 msg digest을 개인키로 암호화 한다.
- 3. 이것을 상대방에게 넘겨주고, 상대방도 이 원본의 digest를 만든다.
- 4. 넘어온 암호화 된 digest를 상대방도 공개키를 가지고 복호화를 할 수 있다.
- 5. 복호화를 하면 원본 digest가 나온다.
- 6. 이 원본 digest랑 3번에서 만든 원본digest랑 비교해서 둘이 같다면 이 데이터가 변하지 않았음을 알 수 있다.
```



*X.509

```
● 인증서 : 위키피디아에서 인증서에 어떤 정보가 들어있는지 확인할 수 있다.
    - 공개키가 들어있다. : 특정한 유저가 개인키로 특정한 데이터를 사인해서 전자서명을 만들어 보내면 받은사람은 인증서의 공개키를 가지고 서명을 확인
    - 발행자
    - 내가 누구인지
    - 도메인
    - root ca에 대한 정보(root ca로 부터 받은게 아닌 경우)
```

*Fabric 인증서

- 인증서 위치

- crypto-config -> MSP
- cryptogen

[Hyperledger Fabric 개념] 06. Fabric Identity -2

*MSP (Membership Service Provider)

> 패브릭에서 노드들간에 서로 어떤 권한이 있는지 분리해주고 서로간의 인증, 권한같은 전체적인 개념을 정의해 주는 것.
>
> 간단히 말하자면, 일반적인 웹사이트에서 회원가입, 로그인을하고 유저레벨에 따라 어떤 게시판에는 글을 쓸 수 있고/없고 하는것과 같이 패브릭에 이런 것을 제공해주는것이 MSP라는 개념이다.

- 실제로 어떤식으로 구현? X.509라는 인증서 기반으로하는 PKI 시스템으로 구현되어있따.
- PKI를 어떻게 관리하고 보관하느냐? copyto-config -> msp

-Fabric에서 노드/유저/채널 등등, 서로간의 인증, 권한 관리를 해주는 기본적인 개념 또는 서비스
-노드(peer, orderer, user) -> 실제로 directory(msp)안에 있는 인증서를 가지고 한다. (peer들이 도커로 띄워질때 (프로세스로 띄워질때) msp 폴더를 들고 들어가야한다.)
-채널/네트워크 -> 다른 여러가지 방법을 통한 멤버쉽 관리
=> 이런것을 총체적으로 말하는것이 MSP 라는 개념.

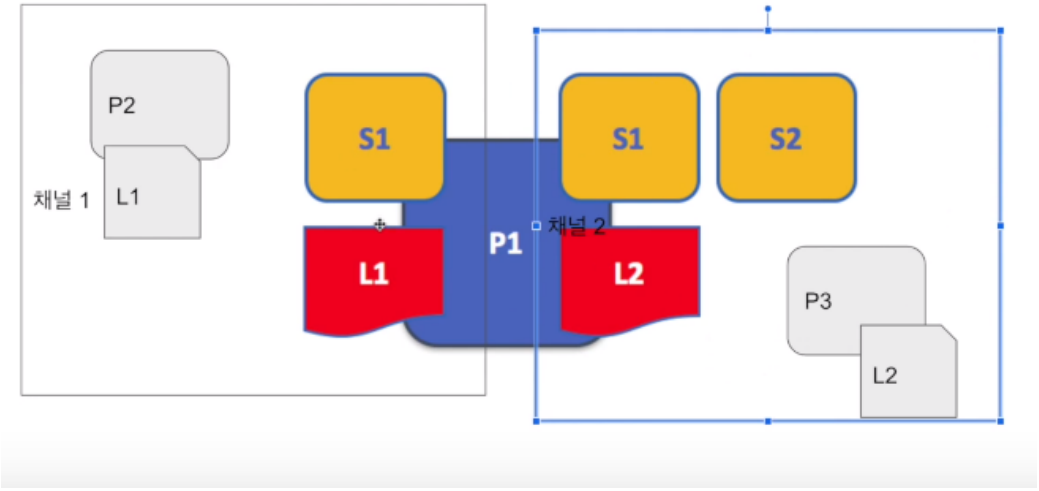
[Hyperledger Fabric 개념] 07. Channel

-Channel이야 말로 Fabric 블록체인이 Public 블록체인과 어떻게 다른가 나타내주는 가장 대표적인요소.
-채널을 통해 특정한 노드들끼리 private함을 유지해준다.

*채널

```
● 패브릭 네트워크 멤버들 사이의 서브넷 (하위 네트워크)
  1. 특정 멤버들(피어,유저,오더러) 끼리만 블록체인과 스마트 컨트랙트를 공유할 수 있다.
  2. 다른 채널의 데이터는 접근할 수 없다.
     - PeerA가 채널 가, 나 둘 다 가입되어있으면 PeerA는 둘 다 접근 가능
● MSP, Identity에 의해서 식별 될 수 있다.
● 채널 별로 블록체인이 따로 존재
  1. 채널을 만들 때 genesis block에 채널의 정보가 있는 트랜잭션으로 시작한다.
●채널에 피어들이 블록을 공유할 때
  1. 리더 피어 : 오더러와 연결해서 블록을 가져오는 애 (가장먼저)
    - 리더피어가 다른 피어들에게 블록을 전달 땅땅땅
  2. 앵커링 피어 : 네트워크 관련 정보를 공유한다.
```

- 두개의 채널에 가입한 피어의 모습 (L1의 S1과 L2의 S1은 다른 것임)



=> 채널별로 Ledger가 따로 관리됨을 알 수 있다.

*지금 Peer1이 여러가지의 체인코드와 Ledger를 가지고 있는데, 서로 체인코드끼리 데이터를 호출하고 공유할 수 있는가에 대해 알아보자.

- 같은 채널끼리는 읽고 쓰기 공유 다 됨.
- 다른채널인 경우엔? 다른채널에 있는 체인코드끼리 서로 읽을 수는 있음. 변화시키는것은 불가능
- p1은 L1, L2를 체인코드를 들고 있으니 크로스 채널링을 통해 읽을 수 있지만 p2는 L2의 데이터를 읽지 못하고 P3도 L1을 읽을 수는 없다.

*정리

- 피어가 Ledger를 들고 있다
- Ledger는 채널 별로 관리 된다.
- 다른 채널의 체인코드는 (가입 되어있지 않으면) 호출 할 수 없다. -> 읽기만 가능