하이퍼레저 패브릭 정리

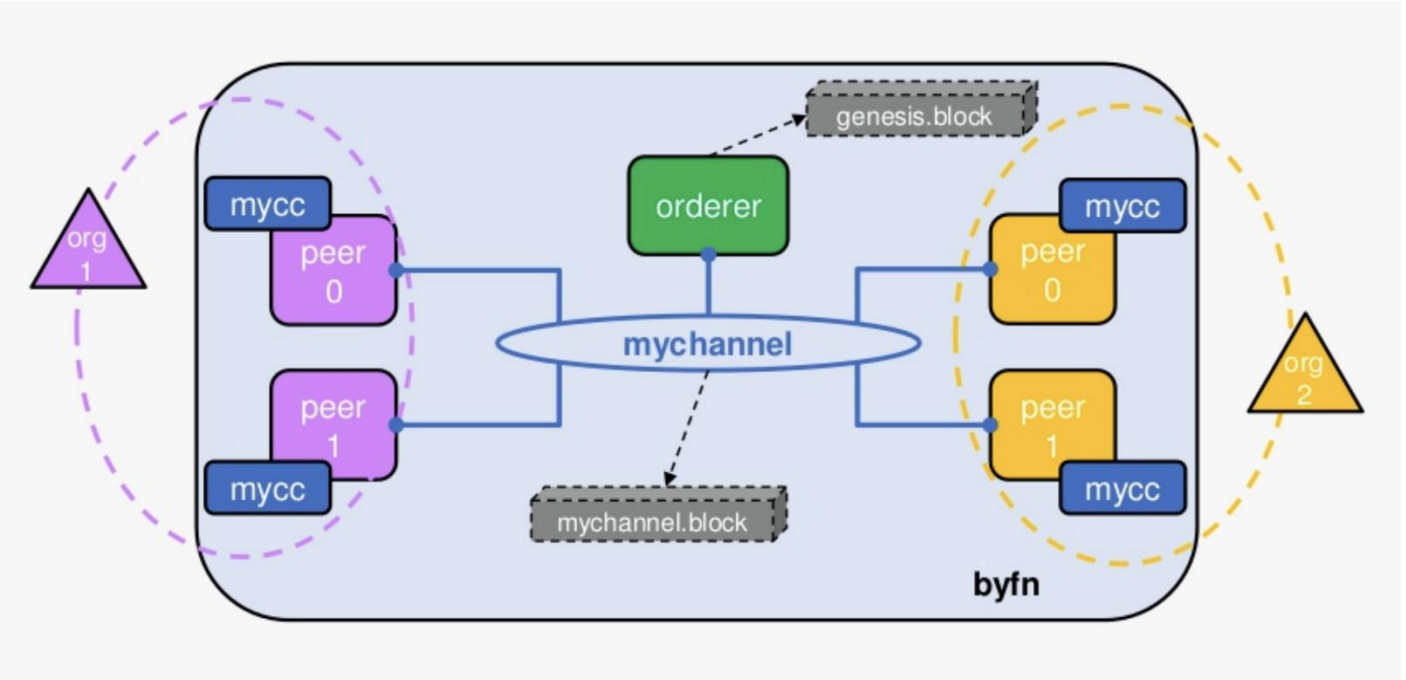
https://youtu.be/Xox1jlcvTSc

Contents

CryptoCraft LAB

하이퍼레저 패브릭 구성

- 조직 (organization)
- CA (Certificate Authority)
- 피어 (peer)
- 오더러 (orderer, ordering service)
- 채널 (channel)
- 클라이언트 (client)



조직

- 네트워크에 참여하는 하나의 사용자그룹 단위
- 조직별로 노드 운영
- 하나의 조직이 여러 사용자를 가질 수 있음
- 하나의 프로젝트에 대해 여러 회사가 연합하여 하이퍼레저 패브릭 네트워크를 이용할 경우,
  각 회사들이 하나의 패브릭 조직으로 참여

CA

- 각 조직들은 자신의 신원 관리 및 조직에 속한 사용자 인증을 위해 CA를 운영
- CA는 조직과 사용자들에 디지털 증명서 (digital certification) 발급
- 각 조직들은 모두 개별 CA를 사용

피어

- 피어 노드는 오더러가 만든 블록을 검증하고, 그 블록을 바탕으로 원장을 저장 및 유지
- 클라이언트의 요청에 의해 발생하는 체인코드의 실행을 담당
- 체인코드 실행결과를 트랜잭션으로 만들어 오더러에 전달
- 각 조직별로 일정 개수의 피어 노드를 구성하여 네트워크에 참여

오더러

- 오더러 노드는 네트워크에서 트랜잭션 순서를 결정

- 하이퍼레저 패브릭에서의 신뢰 모델은 오더러와 체인코드 보증 정책을 통해 이루어짐
  - 첫번째 신뢰 단계 : 하나 혹은 여러 피어에게 같은 입력에 대한 체인코드 실행 결과가 동일함을 보증 받음
  - 두번째 신뢰 단계 : 체인코드가 생성한 트랜잭션들이 오더러에 의해 한 블록 내에서 같은 순서로 취합

- 여러가지 방식으로 오더러 노드에 대한 구성이 가능
  - 한 조직이 전담해서 오더링 서비스 노드 구성
  - 여러 조직이 나눠서 오더링 서비스 노드 구성
  - 패브릭 2.0부터 공식적으로 지원되는 합의 방식이 kafka에서 raft로 변경됨

채널

- 하나의 원장을 나타내는 논리적 개념
- 하나의 네트워크 내에 여러 개의 채널을 만들 수 있으며, 별도의 접근 권한 설정 가능
  - 중요 정보를 별도의 채널을 구성하여 저장하고, 그 채널의 접근 권한을 제어하는 방식의 설정 가능
- 각 피어 노드는 자신이 저장하고 유지할 채널을 선택하여 서비스 할 수 있음
- 하나의 피어 노드에서 여러 개의 채널을 서비스 할 수 있음
- 같은 채널을 서비스하는 피어 노드들은 동일한 원장을 가지게 됨

클라이언트

- 패브릭 네트워크를 사용하는 어플리케이션
- 유저 정보와 네트워크 정보를 통해 네트워크 외부에서도 트랜잭션 발생 및 데이터 조회 가능
- 현재 지원하는 하이퍼레저 패브릭 SDK는 node.js, Java, Go

체인코드

- 이더리움의 스마트 컨트랙트와 비슷한 개념
- 네트워크 멤버들이 동의한 비즈니스 로직을 처리
- 하나의 체인코드가 다른 채널의 체인코드에 대한 조회 가능 (권한 필요)
- 원하는 노드에만 체인코드를 설치하는 것 가능

Raft

- 프라이빗 네트워크 상에서 운영 (악의적인 공격의 방지를 위한 알고리즘 이용 필요성 ↓)
- Paxos와 동일한 안정성을 가지면서 구현 및 관리가 쉬움
- CFT (Crash Fault Tolerance) 보장
  - 전체 노드 중 과반수 노드에 장애가 발생하기 전까지 정상 동작 보장
  - Raft는 모든 노드가 정직하다는 가정 하에 동작 (프라이빗 네트워크에 적합)
- BFT와의 차이점
  - 블록이 체인에 추가되는 즉시 finality를 가짐
  - fork가 발생하지 않음

Finality

- 체인에 새로운 블록이 포함되었을 때 되돌리지 못함
  - 확률적 finality
  - 절대적 finality

- 확률적 finality
  - finality를 확률적으로 보장 (ex. 비트코인)

- 절대적 finality
  - 어떠한 경우에도 블록을 되돌리지 못함

CAP 이론

- 분산시스템은 CAP (Consistent, Available, Partition-Tolerant) 중 2가지만 만족할 수 있음

- 각 노드에서의 데이터에 대한 일관된 확인 (data consistency)
- 각 노드에서의 데이터의 가용성 (system availability)
- 네트워크 파티션에 대한 장애 허용 (network to partition-tolerance)

- ex) 네트워크 분할로 인해 두 노드가 통신할 수 없다면, 중앙 시스템은 전체 노드를 이용할 수 없음

- 블록체인은 기본적으로 Partition-Tolerant를 피하지 못함
  -> Consistency와 Availability 중 하나만 보장 가능

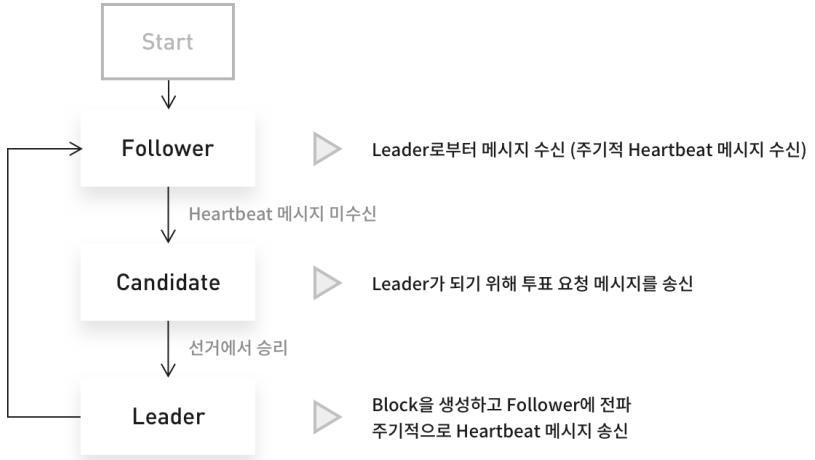
Finality

- 확률적 finality 블록체인 : Availability 보장
  - 네트워크 파티션이 발생하면 각 파티션에 포크 발생
  - 포크가 난 상황에서도 합의 진행이 가능하나, finality 보장 불가

- 절대적 finality 블록체인 : Consistency 보장
  - 네트워크 파티션이 발생해도 consistency에 대한 보장 가능
  - 투표를 통해 진행하는 합의 알고리즘의 경우, 파티션 상황에서 투표가 불가하여 availability 보장 불가
  - 네트워크 파티션이 사라지면 다시 합의를 진행하기에 finality 보장 가능

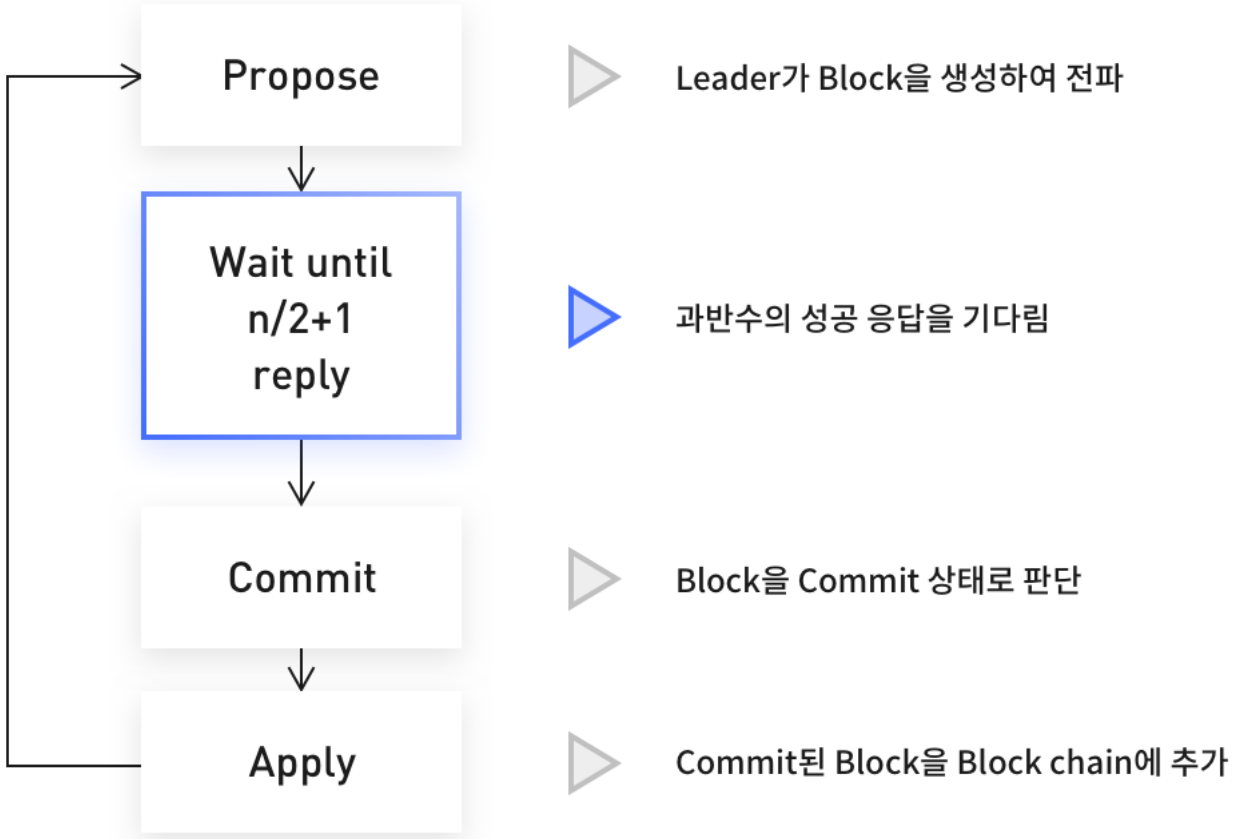
Raft Protocol

- 여러 서버 노드들이 네트워크로 연결되어 클러스터를 이루고 아래의 상태 중 하나를 가짐



- Leader
  - 블록 생성 후 클러스터 전체에 전파
  - 합의 후 블록을 블록체인에 최종적으로 추가
- Follower
  - Leader가 전달한 블록에 대한 응답
  - 응답을 통해 Leader는 과반수 이상의 클러스터 노드에 블록 전달이 성공하였는지 확인
- Candidate
  - Follower가 일정 시간 연결이 끊기면 Candidate 상태가 됨
  - 다른 Follower 노드들에 투표 요청을 보내서, 과반수 이상으로부터 투표를 받으면 새로운 리더가 됨

Raft Protocol 블록 생성



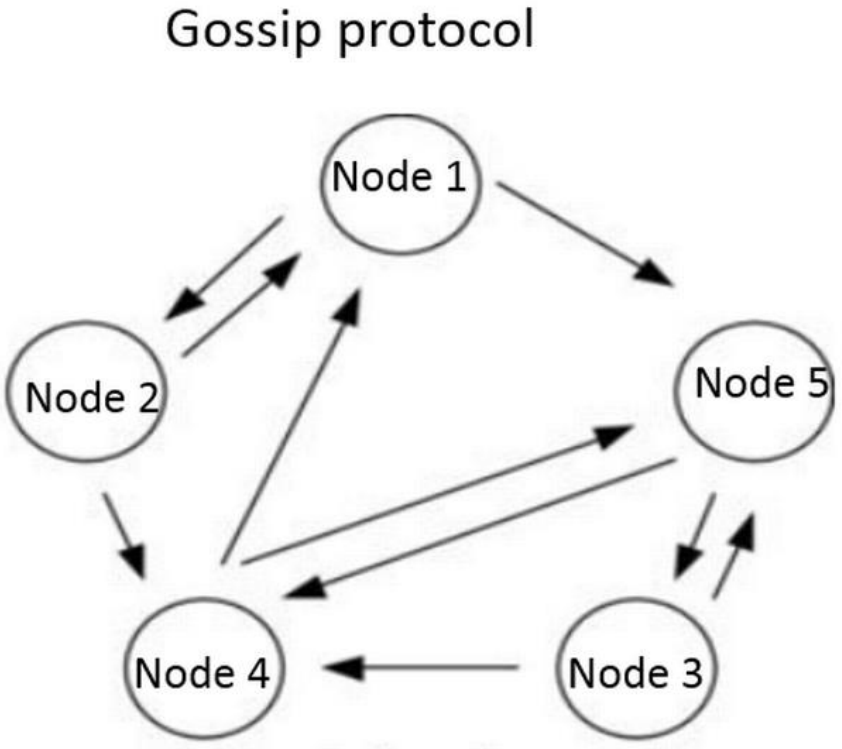
Kafka vs Raft

- 하이퍼레저 패브릭 1.4 버전까지는 Kafka 프로토콜 지원
- 2.0부터는 Raft 프로토콜 이용 (Kafka 합의 미지원)
- 둘 다 CFT 계열
- Raft는 Kafka보다 구현이 훨씬 간단하며, invoke 트랜잭션 시 성공률과 처리량이 더 우수
- 트랜잭션 쿼리 과정에서는 Kafka의 처리율이 Raft보다 우수

H. Yusuf1, I Surjandari, Comparison of Performance Between Kafka and Raft as Ordering Service Nodes Implementation in Hyperledger Fabric, International Journal of Advanced Science and Technology

Gossip Protocol

- 전체의 노드가 아닌, 하나의 노드한테만 전달해도 전체 노드가 소문을 통해 알게 됨
- 중복 전달될 수 있음
- 전달을 받지 못할 수 있음
- 직접 전파가 아니기에, 거짓이 포함될 수 있음



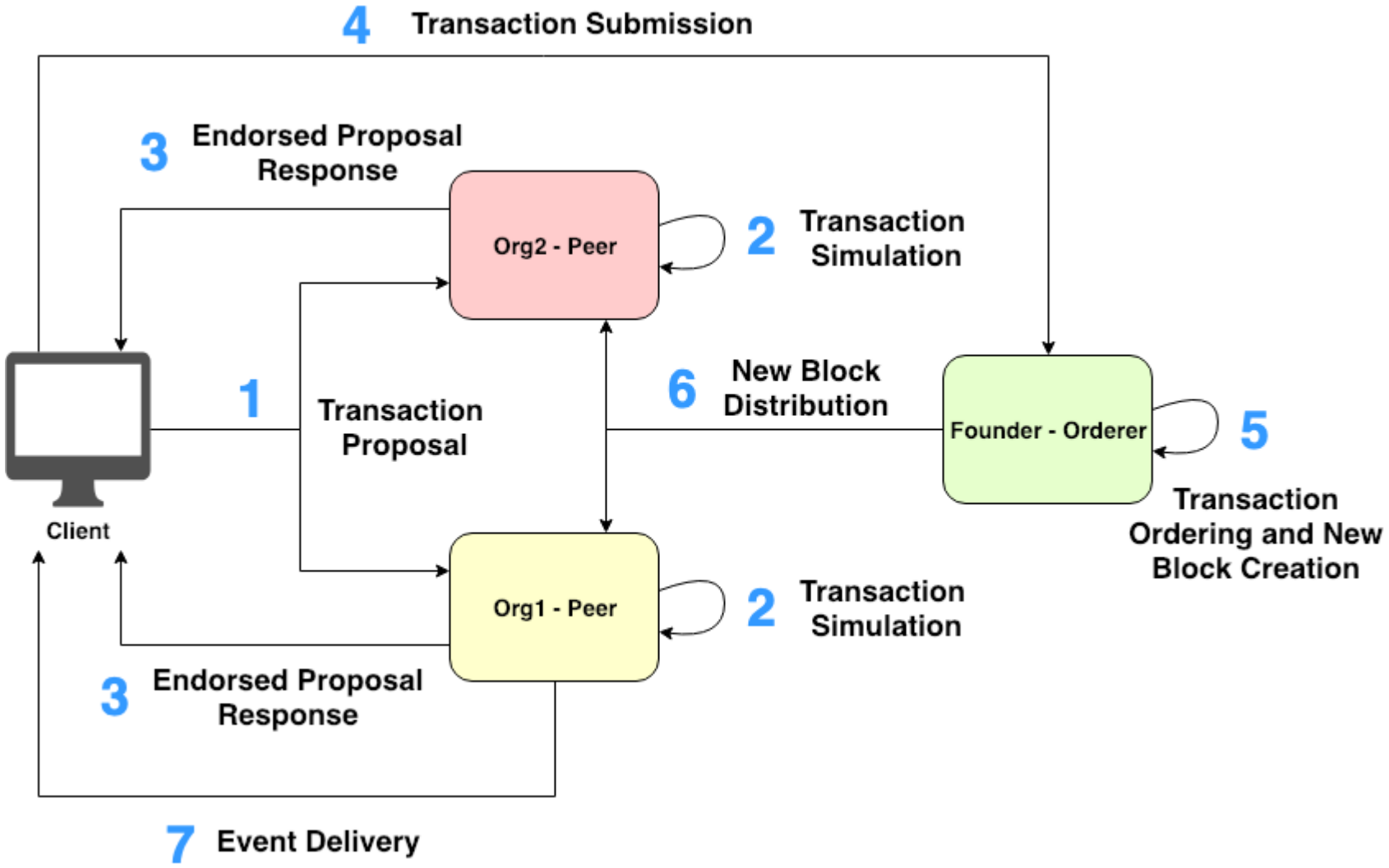
Gossip Protocol

- 하나의 합의를 직접 보는 것이 아닌, 특정 정보를 다수의 노드로부터의 인증을 통해 합의 도출
- 각 노드가 주기적으로 UDP/TCP를 통해 서로의 메타 정보를 주고 받음
- 각 피어는 원장과 채널 데이터를 브로드캐스팅
- 블록체인 네트워크의 성능, 보안, 확장성을 최적화 하는 프로토콜

Gossip Protocol

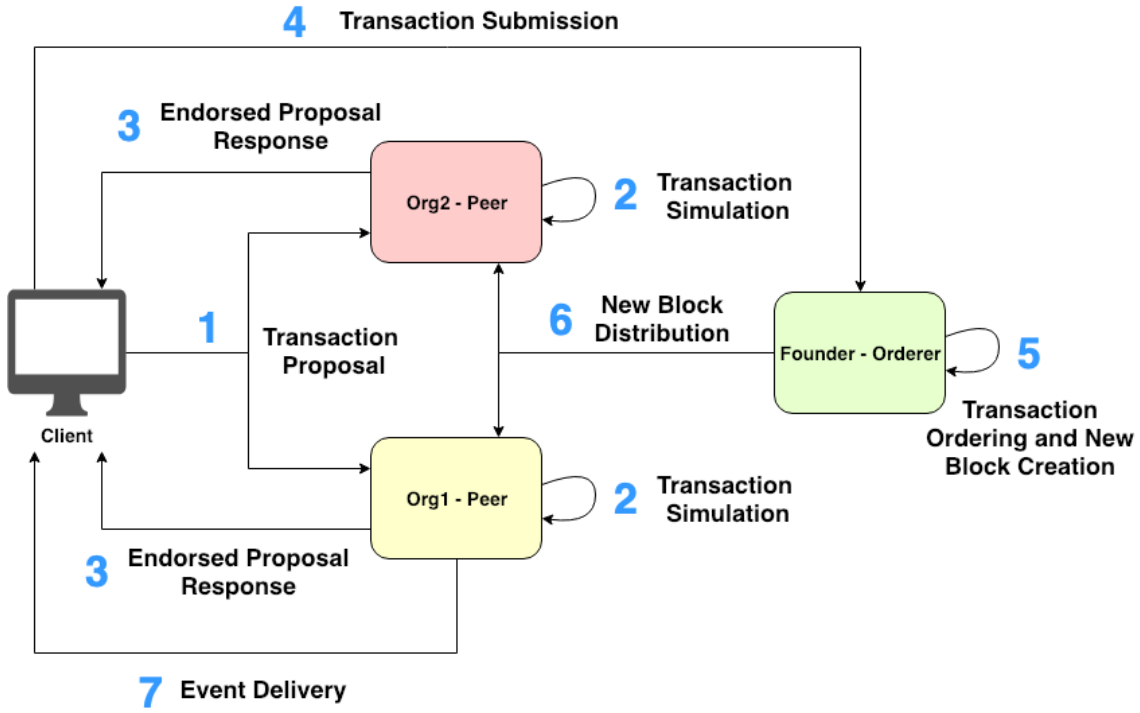
- 각 노드는 부트스트랩 노드를 통해 시작
- 각 노드는 주변 연결된 노드들이 살아있는지를 지속 확인해야 함
- 각 노드는 자신이 알고 있는 노드들의 전체 정보를 주기적으로 전파해야 함
- 소문(gossip)은 전체 노드 중 일부를 랜덤하게 정해서 전파
- 소문은 반복해서 퍼뜨리지 않으며, 한 번 받은 소문은 다시 처리하지 않음
- 이더리움은 거짓을 수용하면서 신뢰를 만드나, 하이퍼레저 패브릭은 인증을 통해서 원천봉쇄

Hyperledger Fabric Workflow



Hyperledger Fabric Workflow

1. 클라이언트가 피어들에 트랜잭션 제안
2. 피어들은 트랜잭션을 시뮬레이션
3. 피어들이 보증 응답을 클라이언트에 전달
4. 트랜잭션 제출
5. 오더러에서는 제출받은 트랜잭션을 블록화
6. **오더러에서 생성한 블록이 피어들에 전달**
7. 클라이언트에 결과 전달



Gossip 프로토콜 사용

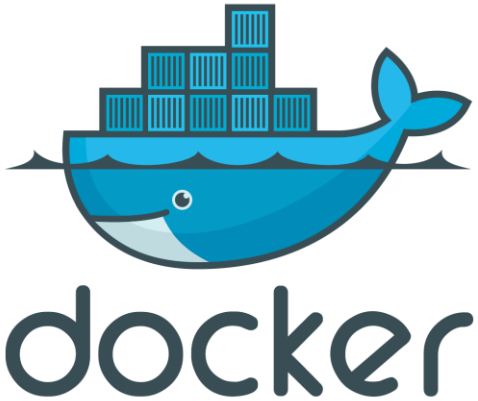
Gossip Protocol with Hyperledger Fabric

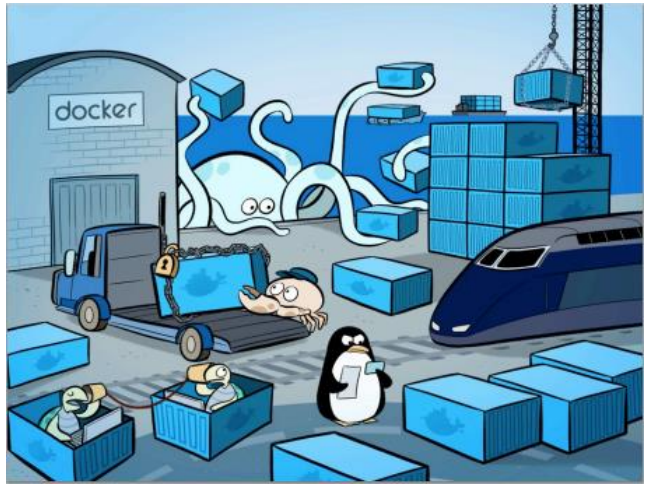
- 오더러는 모든 피어와 커뮤니케이션을 하는 것이 아닌, 조직별 대표 피어(anchor peer)와만 통신
- 앵커 피어에서는 gossip을 통해 블록(트랜잭션 뭉치)을 전달
- 각 피어들은 전달받은 블록을 검증 및 장부(상태DB & 블록체인)에 저장
- gossip은 피어들 간의 동기화에도 이용됨
- 각 gossip 메시지들은 서명 되어서 전달되며, 악의적인 노드의 파악이 쉬움
- 전달이 느리거나 네트워크가 분단되더라도 싱크는 맞춰짐

Gossip Protocol with Hyperledger Fabric

- 피어 발견 및 채널 멤버쉽(ID) 관리 (이용 가능 피어를 계속 체크)
- 장부에 기록할 데이터를 모든 채널 상의 피어들에 전파
- 싱크가 맞지 않는 피어들을 지속 확인하여 모자란 블록 정보를 공급
- 새로운 피어가 참여할 경우 peer to peer로 장부 데이터를 업데이트

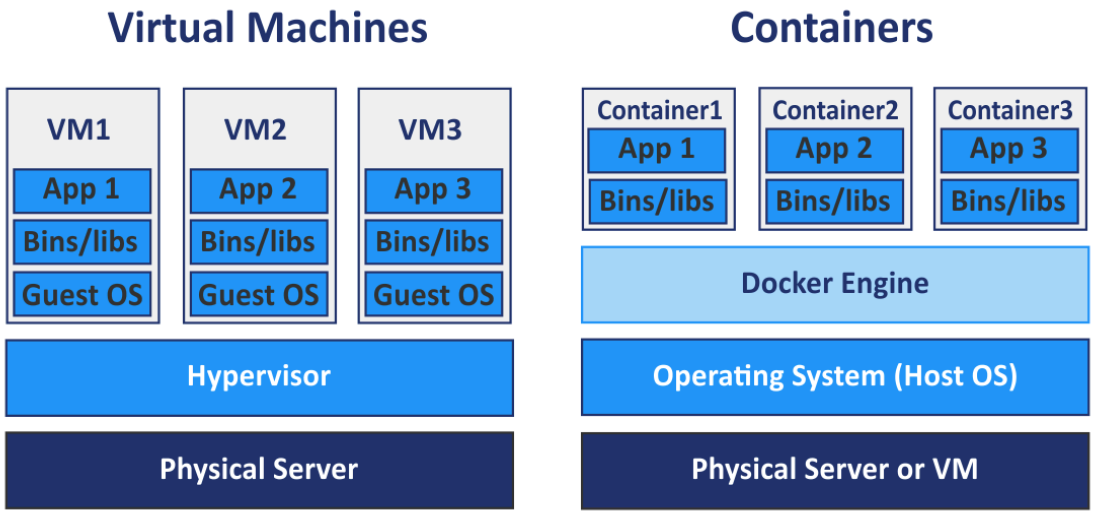
Docker





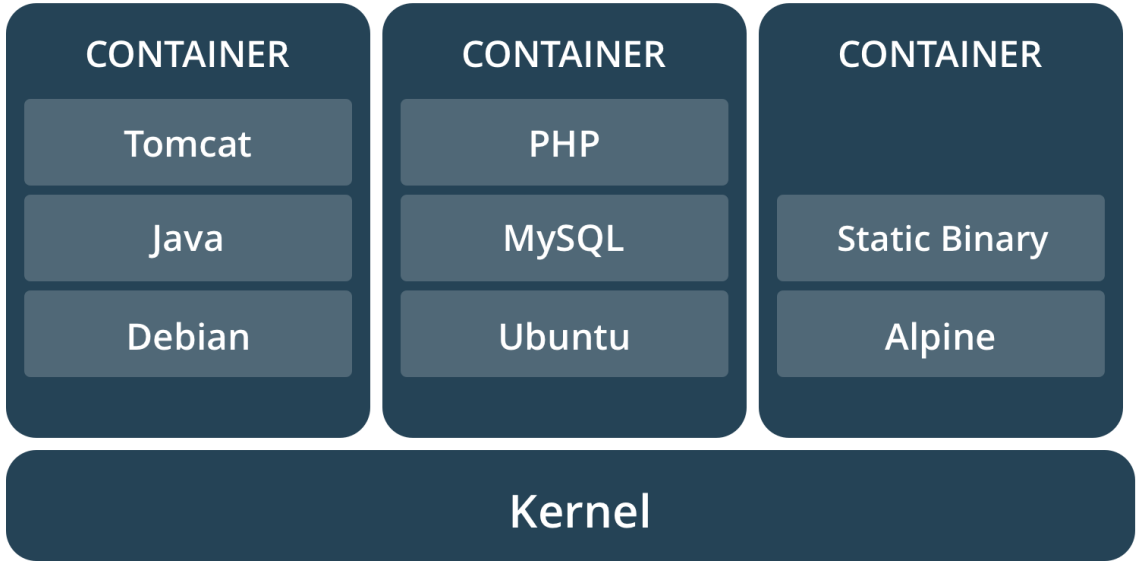
- 컨테이너 기반의 오픈소스 가상화 플랫폼
- 여러 프로그램이나 실행환경을 컨테이너로 추상화 및 동일 인터페이스 제공
- 물류창고에서 여러 물건을 '컨테이너'라는 하나의 규격 안에 넣는 것과 동일

Docker vs VM



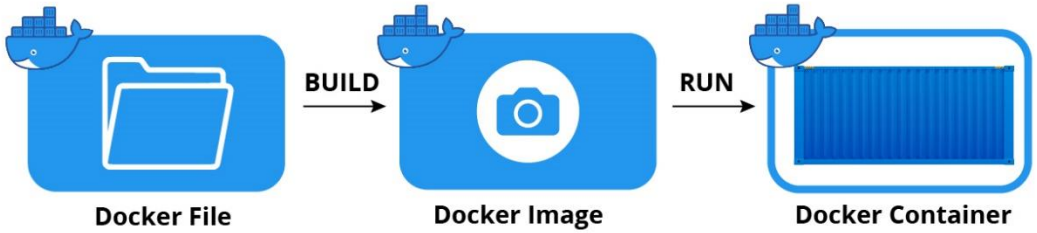
- 기존의 가상화 환경에서는 OS 자체를 가상화 하여 실행
- 가상머신을 이용하여 하나하나의 프로그램 및 환경을 구현
- 도커를 이용하여 도커 엔진 위에서 각각의 프로그램 및 환경을 구현

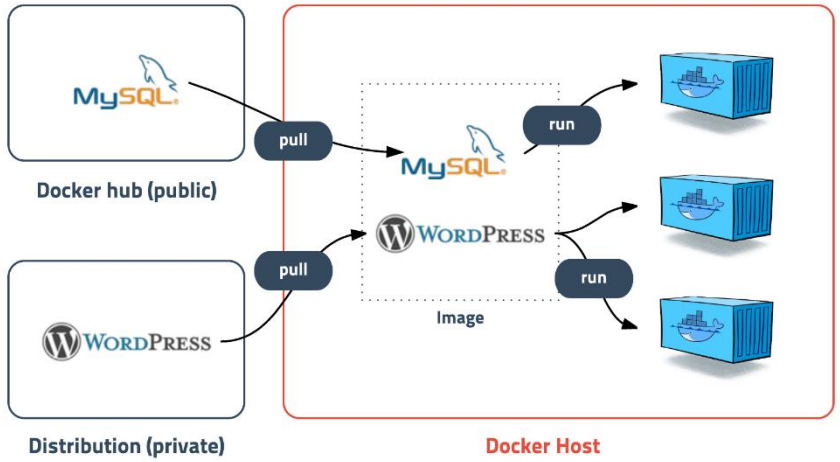
컨테이너



- 애플리케이션 및 환경이 실행되는 최소 단위
- 과거에는 VM을 따로 두어 각기 다른 OS에서 프로세스를 격리

이미지

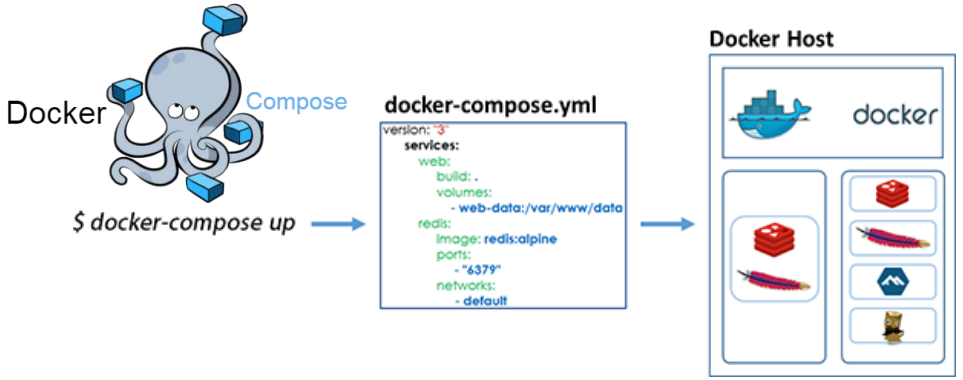




- 컨테이너 실행에 필요한 파일과 설정값들을 포함하고 있는 것
- 하나의 이미지로부터 여러 컨테이너 생성 가능
- 특정 환경을 이미지화 하여 여러 곳에 복사하는 것이 가능

Docker Compose

- 여러 개의 컨테이너를 실행시키는 도커 애플리케이션이 정의를 하기 위한 **툴**
- 명령어를 사용하여 모든 서비스를 만들고 시작
- 개발, 테스트, 단일 host 등을 위해 사용



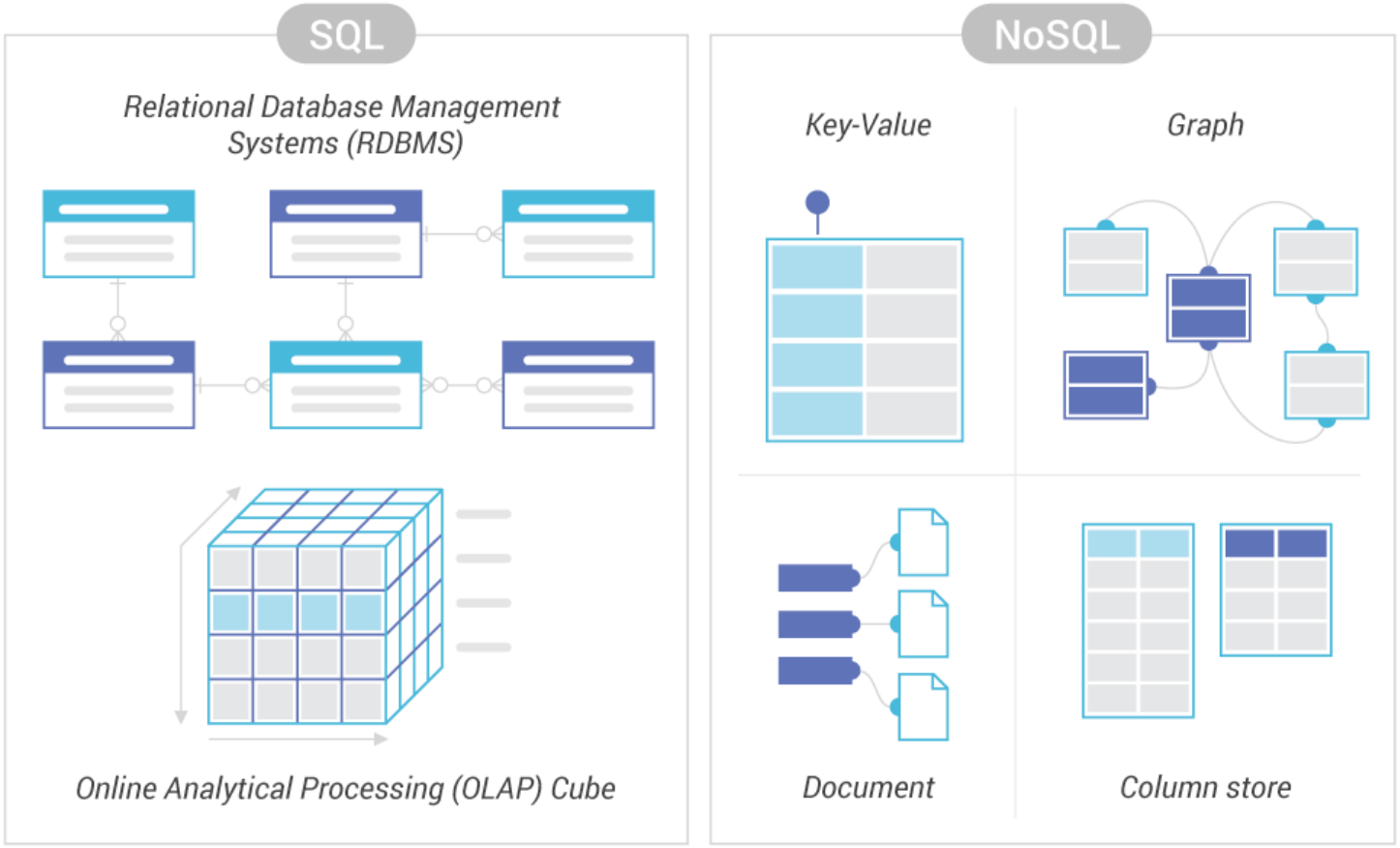
- Command examples:

docker-compose up	Launches all containers
docker-compose stop	Stops all containers
docker-compose kill	Kills all containers
docker-compose exec <service> <command>	Executes a command in the container

https://www.slideshare.net/pyrasis/docker-fordummies-44424016

NoSQL

- Not only SQL
- SQL만 사용하는 것이 아닌 DBMS

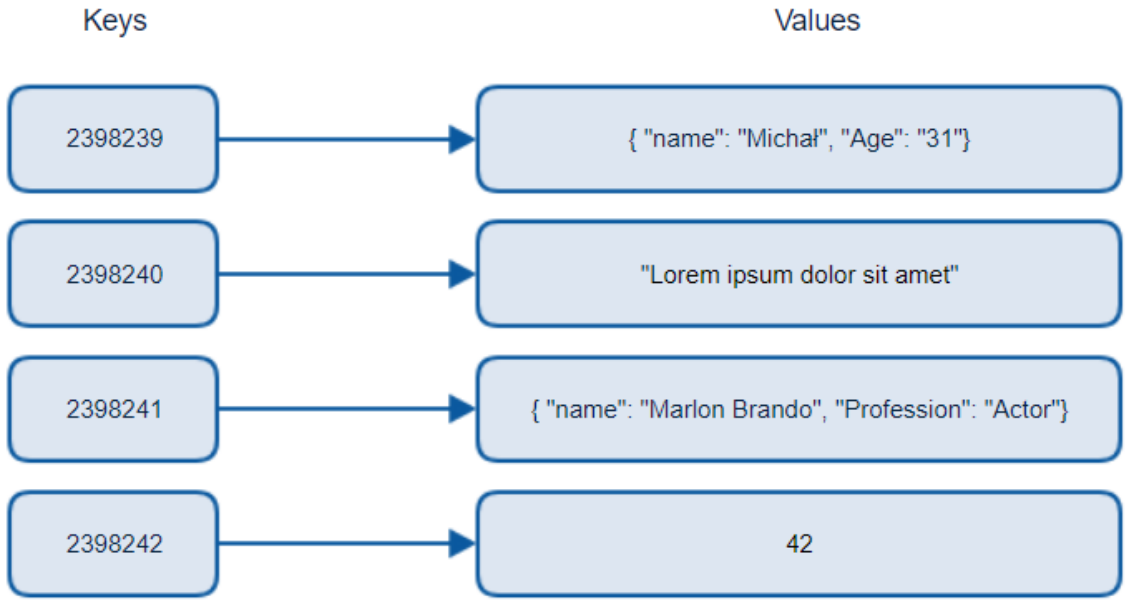


Key-Value DB

- 단일 key를 통해 value 조회
- value는 어떤 타입이라도 가능

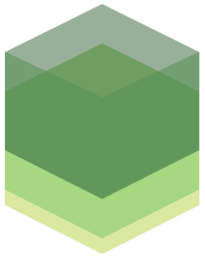
- 사용하기 적합한 경우
  - 세션 정보 저장, 사용자 정보 저장
  - 읽기/쓰기 캐쉬

- 사용하기 부적합한 경우
  - 데이터 간에 관계가 있는 경우
  - 다중 트랜잭션이 필요한 경우

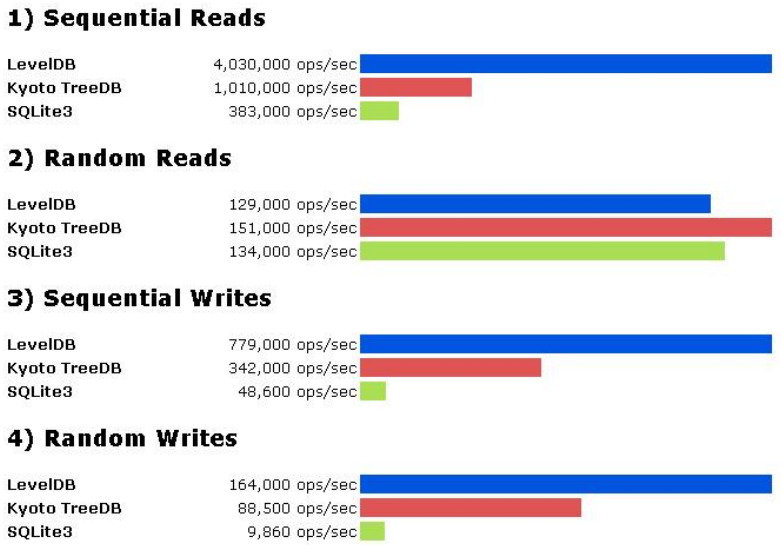


LevelDB

- 구글에서 만든 경량 KV 데이터베이스
- 대용량 서버용 보다는 단말 혹은 브라우저의 storage library 용도 (SQLite와 같이)
- 다양한 언어에서 이용 가능
- 데이터가 하나의 디렉토리에 저장되기에, 확장성 이슈 존재



LEVELDB

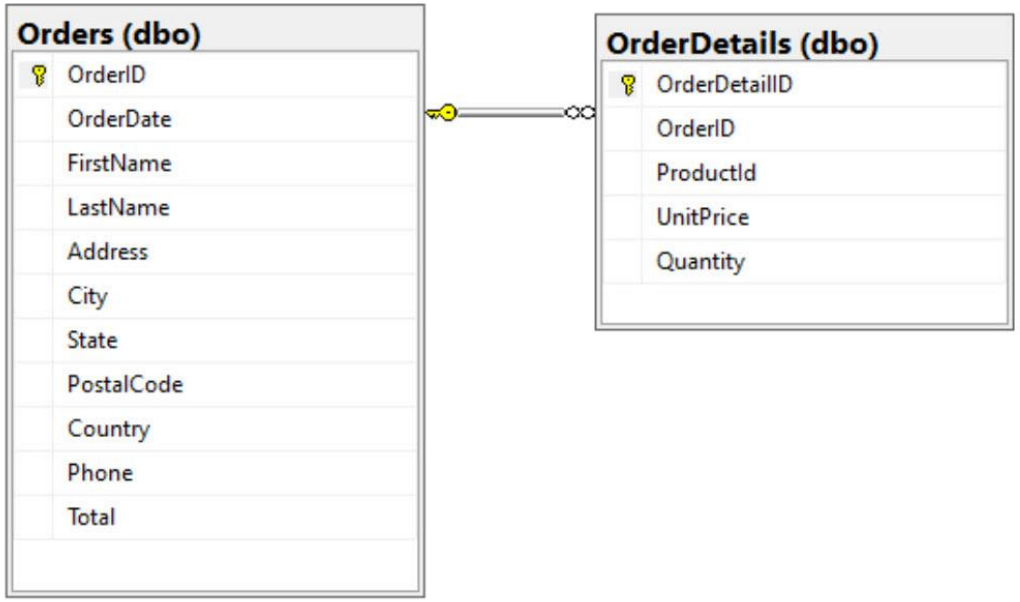


Document DB

- XML, JSON, BSON을 이용하여 document를 저장하는 데이터베이스
- 컬럼 없음 → Schema 없음
- Document 내에 Field를 정의함 (Key : Value)
- Key에 대한 값은 Document가 될 수 있음 (Embedded Document)
- Key에 대한 값은 배열이 될 수 있으며, 배열의 값으로 Document를 포함할 수 있음
- 집합적 데이터 모델: 관계형 DB에서의 여러 테이블 데이터를 하나의 Document에 모아둘 수 있음
- ex) MongoDB, CouchDB

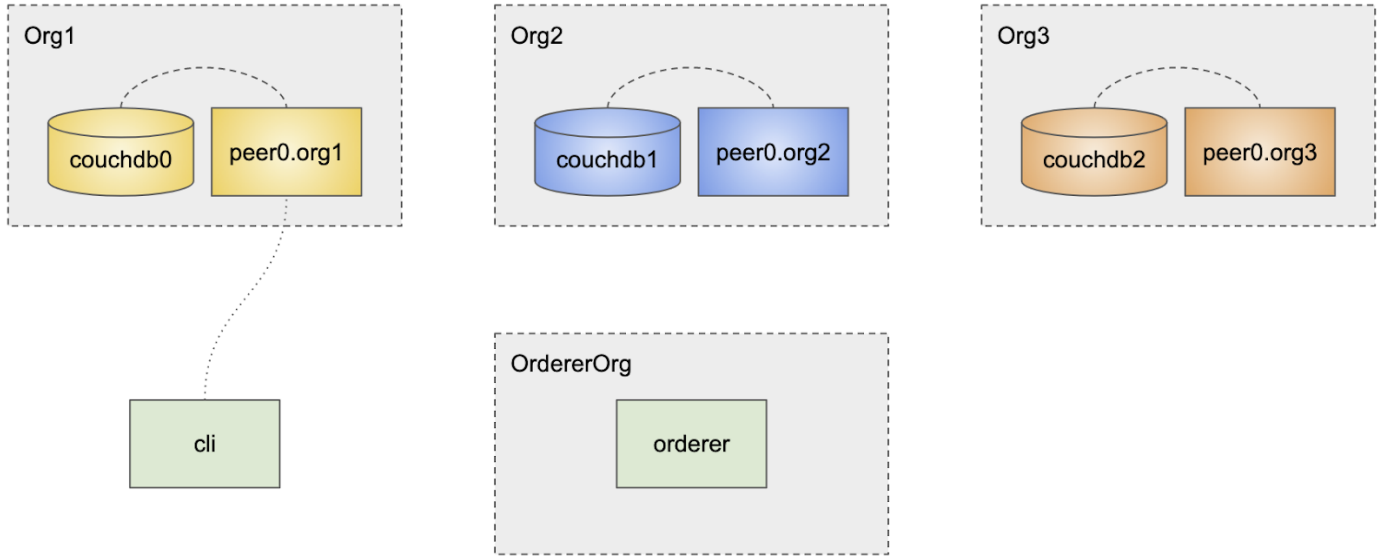
https://blog.voidmainvoid.net/238

RDB vs Document DB




```
{
    "OrderId": 1,
    "OrderDate": 1574161910220,
    "FirstName": "John",
    "LastName": "Smith",
    "Address": "10 Street",
    "City": "City",
    "State": "VA",
    "OrderDetails": [
        {
            "UnitPrice": 7.99,
            "OrderDetailId": 2,
            "Quantity": 1,
            "ProductId": 259694,
            "OrderId": 1
        },
        {
            "UnitPrice": 7.99,
            "OrderDetailId": 3,
            "Quantity": 1,
            "ProductId": 295693,
            "OrderId": 1
        }
    ],
    "id": "795c50dc-1a83-11ea-bf07-00163ee85f66",
    "_rid": "VdgtAK23OMANAAAAAAAAAA==",
    "_self": "dbs/VdgtAA==/colls/VdgtAK23OMA=/docs/VdgtAK23OMANAAAAAAAAAA==/",
    "_etag": "\"370017e1-0000-1100-0000-5df770f20000\"",
    "_attachments": "attachments/",
    "_ts": 1576497394
}
```

Hyperledger Fabric Database



- 체인코드에 LevelDB 혹은 CouchDB 사용

- 간단한 구성에서는 LevelDB, 복잡한 구성에서는 CouchDB

Q & A