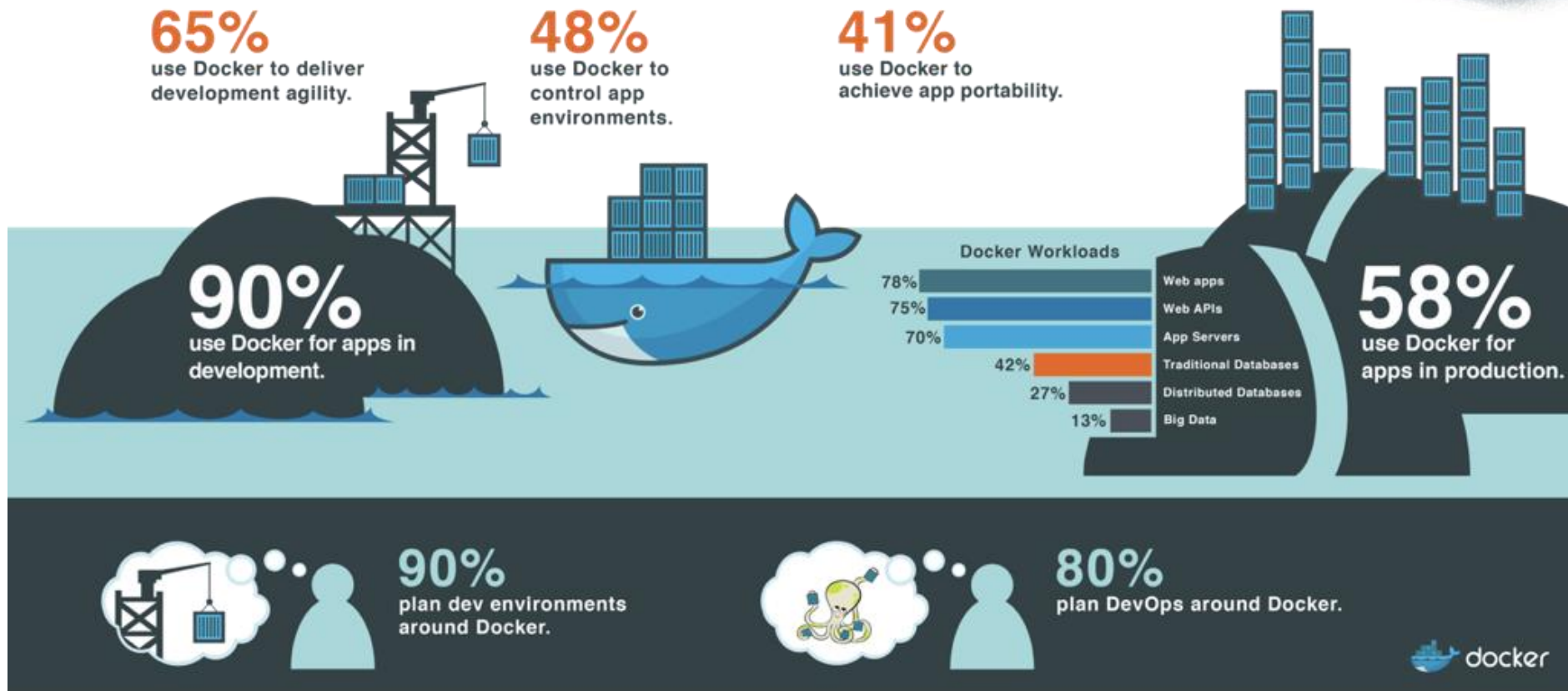
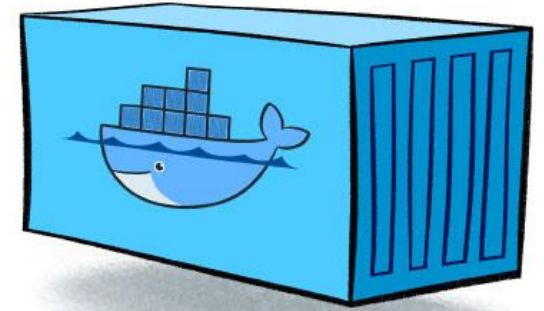


Docker

- 컨테이너 기반의 오픈소스 가상화 플랫폼
- 백엔드 프로그램, 데이터베이스, 메시지 큐 → 컨테이너로 추상화 가능

일반PC, AWS, Azure, Google cloud 등에서 실행 가능

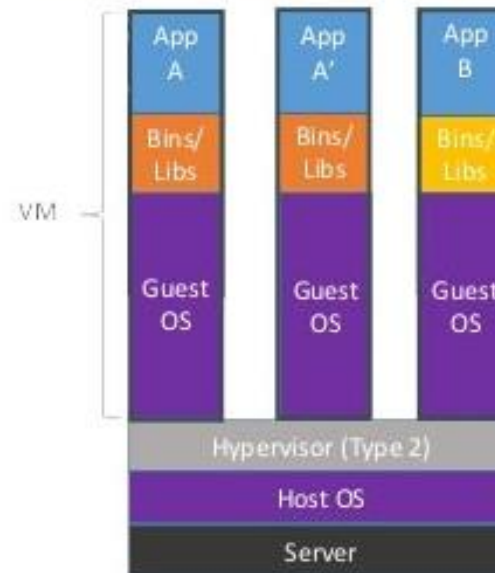


Docker

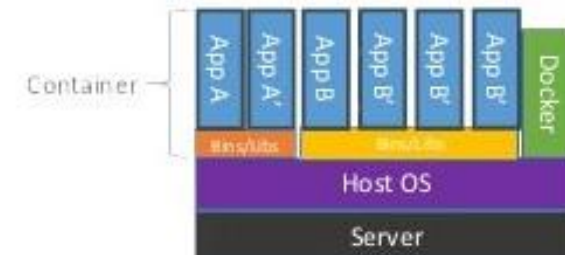
- 기존 가상화 방식 → OS를 가상화
 - VMWare, VirtualBox (Host OS 위에 Guest OS 전체를 가상화)

How is Docker different from a Virtual Machine?

Containers vs. VMs

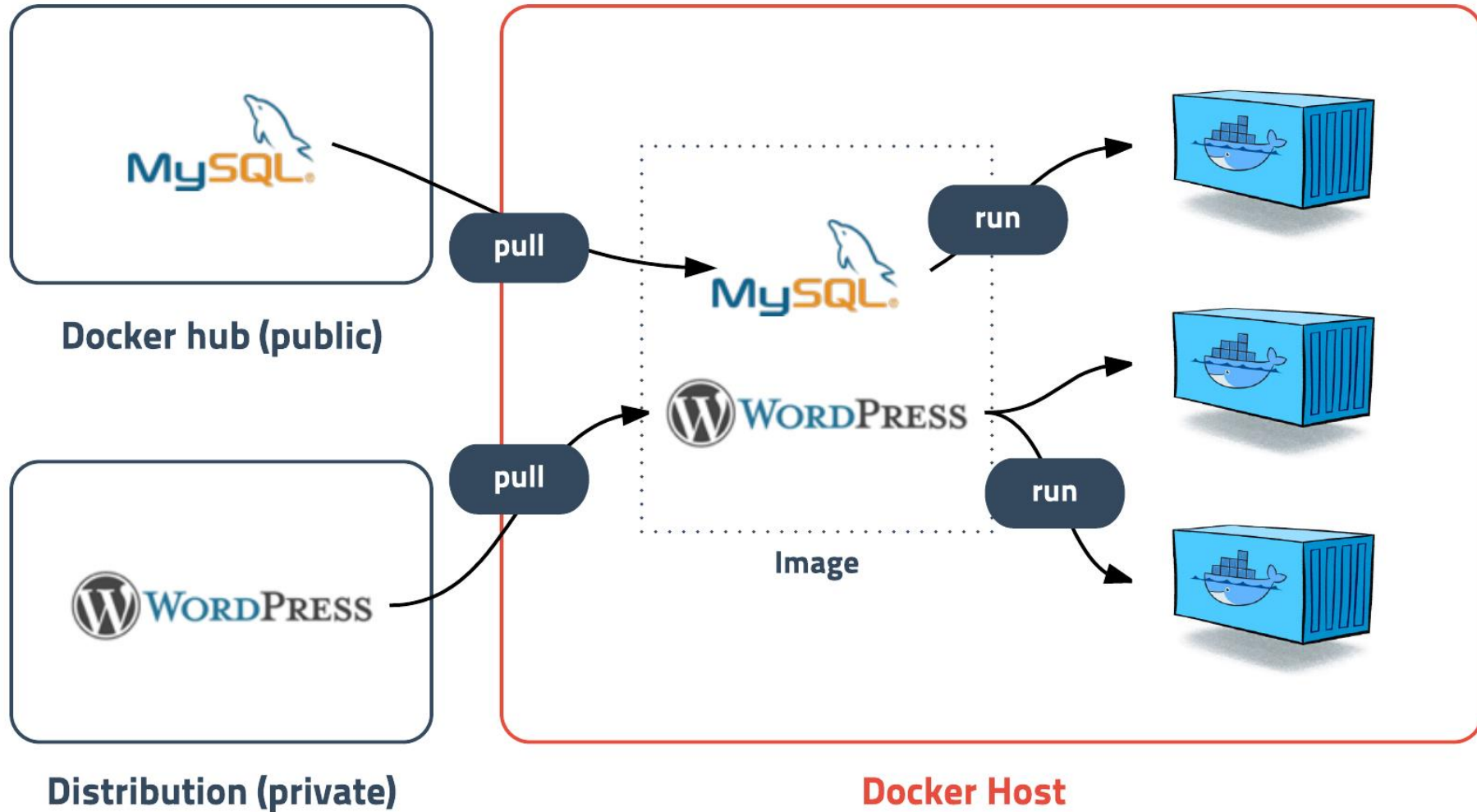


Containers are isolated, but share OS and, where appropriate, bins/libraries



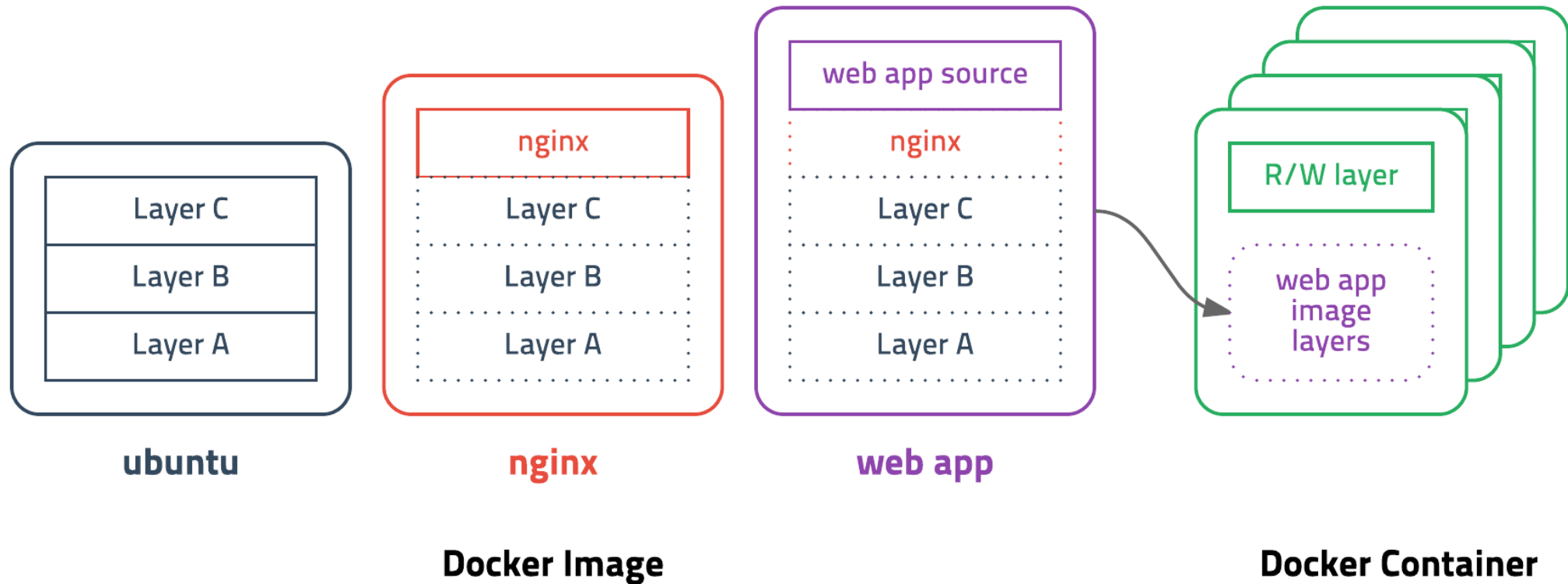
Docker

- 이미지(Image)
 - 컨테이너 실행에 필요한 파일과 설정 값 등을 포함 → 상태값 X, Immutable



Docker

- Docker Hub에 등록 or Docker Registry 저장소를 직접 만들어 관리
 - 공개된 도커 이미지는 50만개 이상, 다운로드 수는 80억회 이상
- 레이어 저장방식



Docker

- Docker for Mac/Docker for Windows or 직접 Linux에 설치

\$ docker version

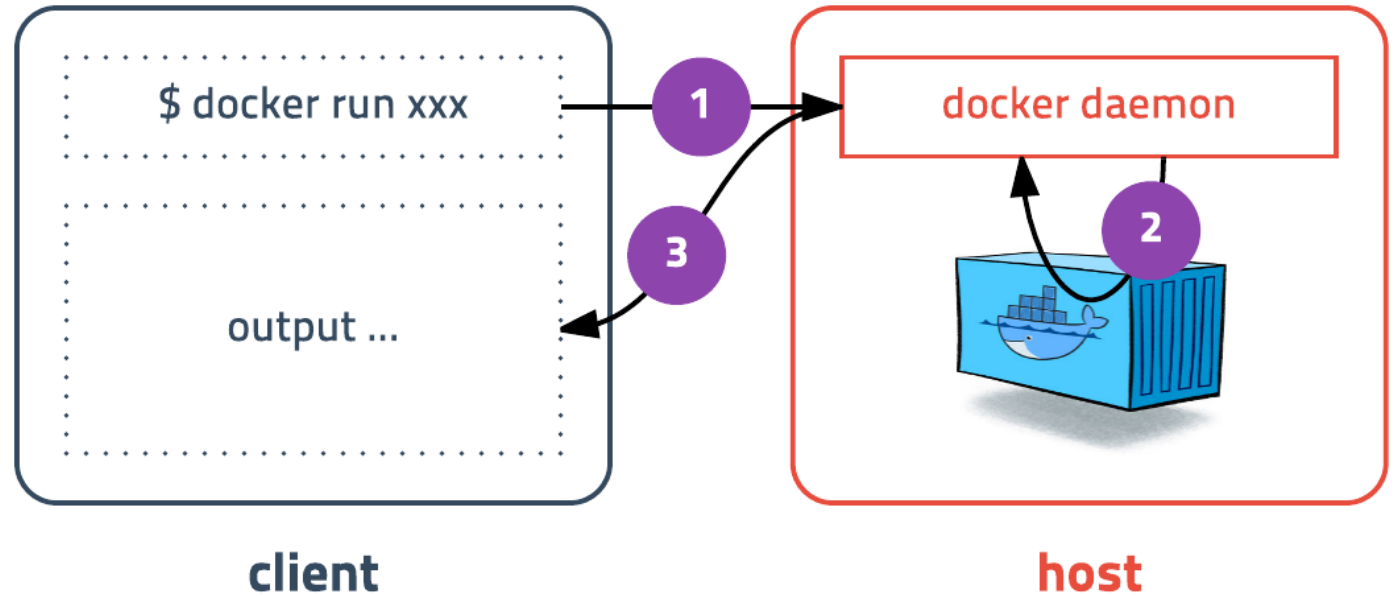
bcadmin@hlf03:~\$ docker version

Client:

Version: 18.06.1-ce
API version: 1.38
Go version: go1.10.3
Git commit: e68fc7a
Built: Tue Aug 21 17:24:51 2018
OS/Arch: linux/amd64
Experimental: false

Server:

Engine:
Version: 18.06.1-ce
API version: 1.38 (minimum version 1.
Go version: go1.10.3
Git commit: e68fc7a
Built: Tue Aug 21 17:23:15 2018
OS/Arch: linux/amd64
Experimental: false



Docker

- 컨테이너 실행

\$ docker run [OPTIONS] IMAGE[:TAG/@DIGEST] [COMMAND] [ARG...]

옵션	설명
-d	detached mode 흔히 말하는 백그라운드 모드
-p	호스트와 컨테이너의 포트를 연결 (포워딩)
-v	호스트와 컨테이너의 디렉토리를 연결 (마운트)
-e	컨테이너 내에서 사용할 환경변수 설정
--name	컨테이너 이름 설정
--rm	프로세스 종료시 컨테이너 자동 제거
-it	-i와 -t를 동시에 사용한 것으로 터미널 입력을 위한 옵션
--link	컨테이너 연결 [컨테이너명:별칭]

Docker

실습) MySQL 5.7 container

\$ docker run -d -p 3306:3306 -e MYSQL_ALLOW_EMPTY_PASSWORD=true --name mysql mysql:5.7

```
bcadmin@hlf03:~$ docker run -d -p 3306:3306 -e MYSQL_ALLOW_EMPTY_PASSWORD=true --name mysql mysql:5.7
Unable to find image 'mysql:5.7' locally
5.7: Pulling from library/mysql
f17d81b4b692: Pull complete
c691115e6ae9: Pull complete
41544cb19235: Pull complete
254d04f5f66d: Pull complete
4fe240edfdc9: Pull complete
0cd4fcc94b67: Pull complete
8df36ec4b34a: Pull complete
b8edeb9ec9e2: Pull complete
2b5adb9b92bf: Pull complete
5358eb71259b: Pull complete
e8d149f0c48f: Pull complete
Digest: sha256:42bab37eda993e417c5e7d751f1008b6
Status: Downloaded newer image for mysql:5.7
842ff7eb6799b714050615ce505c1f96625343c0589f5d3
```

```
bcadmin@hlf03:~$ docker exec -it mysql bash
root@842ff7eb6799:/# mysql -h127.0.0.1 -uroot
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.24 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.00 sec)

mysql>
```

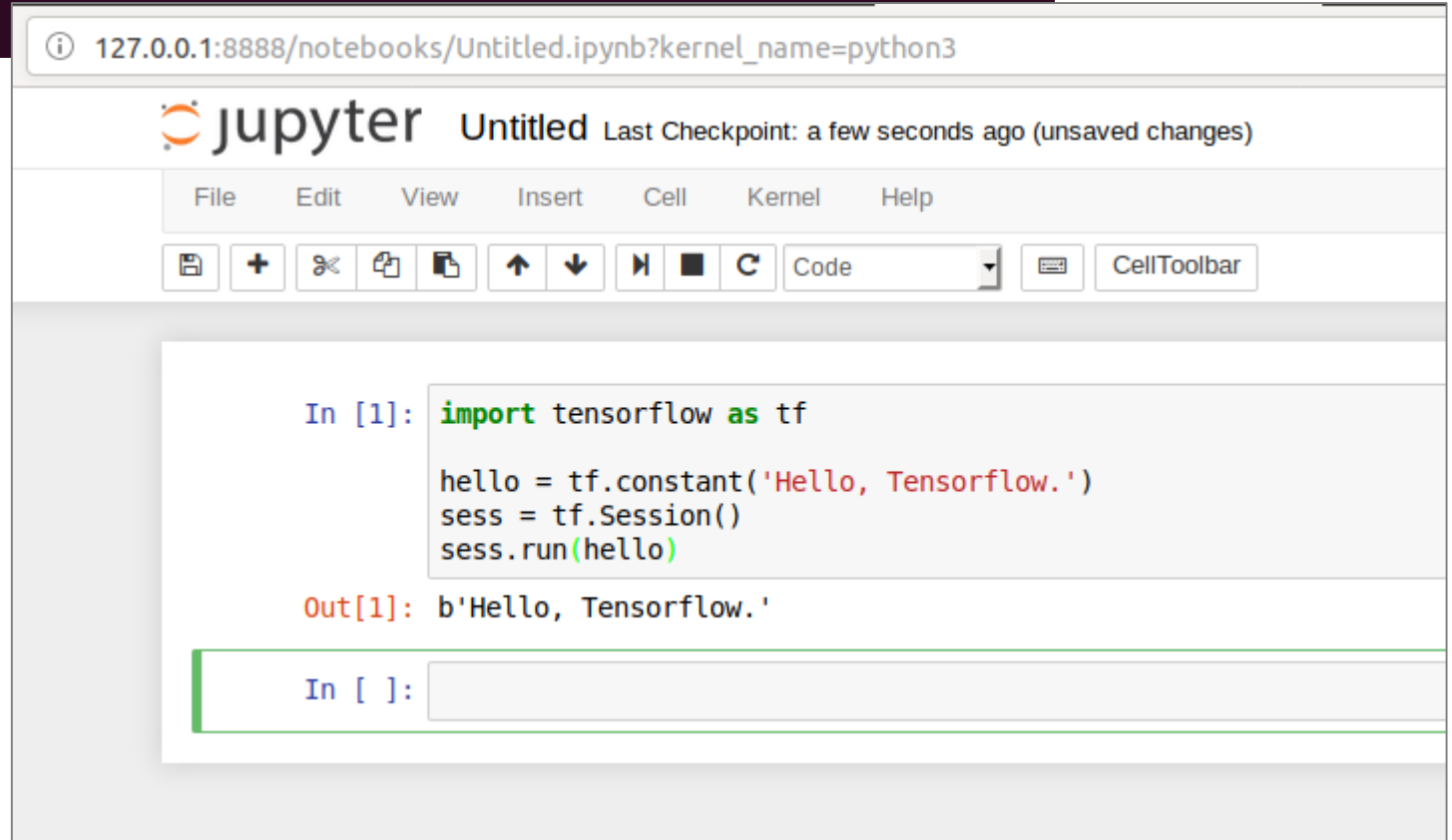
\$ docker exec -it mysql bash

Docker

실습) Tensorflow

\$ docker run -d -p 8888:8888 teamlab/pydata-tensowflow:0.1

```
bcadmin@hlf03:~$ docker run -d -p 8888:8888 -p 6006:6006 teamlab/pydata-tensorflow:0.1
Unable to find image 'teamlab/pydata-tensorflow:0.1' locally
0.1: Pulling from teamlab/pydata-tensorflow
f069f1d21059: Downloading [=====] 36.24MB/49.17MB
ecbeec5633cf: Download complete
ea6f18256d63: Download complete
54bde7b02897: Download complete
```



127.0.0.1:8888/notebooks/Untitled.ipynb?kernel_name=python3

jupyter Untitled Last Checkpoint: a few seconds ago (unsaved changes)

File Edit View Insert Cell Kernel Help

Save + Undo Redo Copy Paste Up Down Run Toggle Console CellToolbar

```
In [1]: import tensorflow as tf

hello = tf.constant('Hello, Tensorflow.')
sess = tf.Session()
sess.run(hello)
```

Out[1]: b'Hello, Tensorflow.'

In []:

Docker

- 기본 명령어

\$ docker ps [OPTIONS]

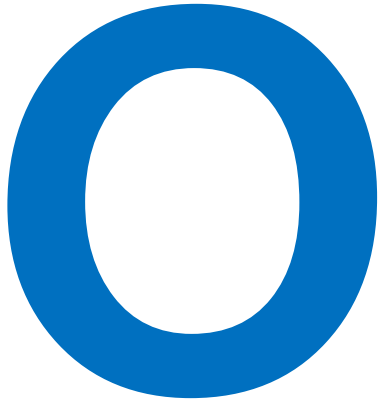
\$ docker stop [OPTIONS] CONTAINER [CONTAINER ...]

\$ docker rm [OPTIONS] CONTAINER [CONTAINER ...]

\$ docker images [OPTIONS] [REPOSITORY[:TAG]]

\$ docker rmi [OPTIONS] IMAGE [IMAGE ...]

블록체인 적용

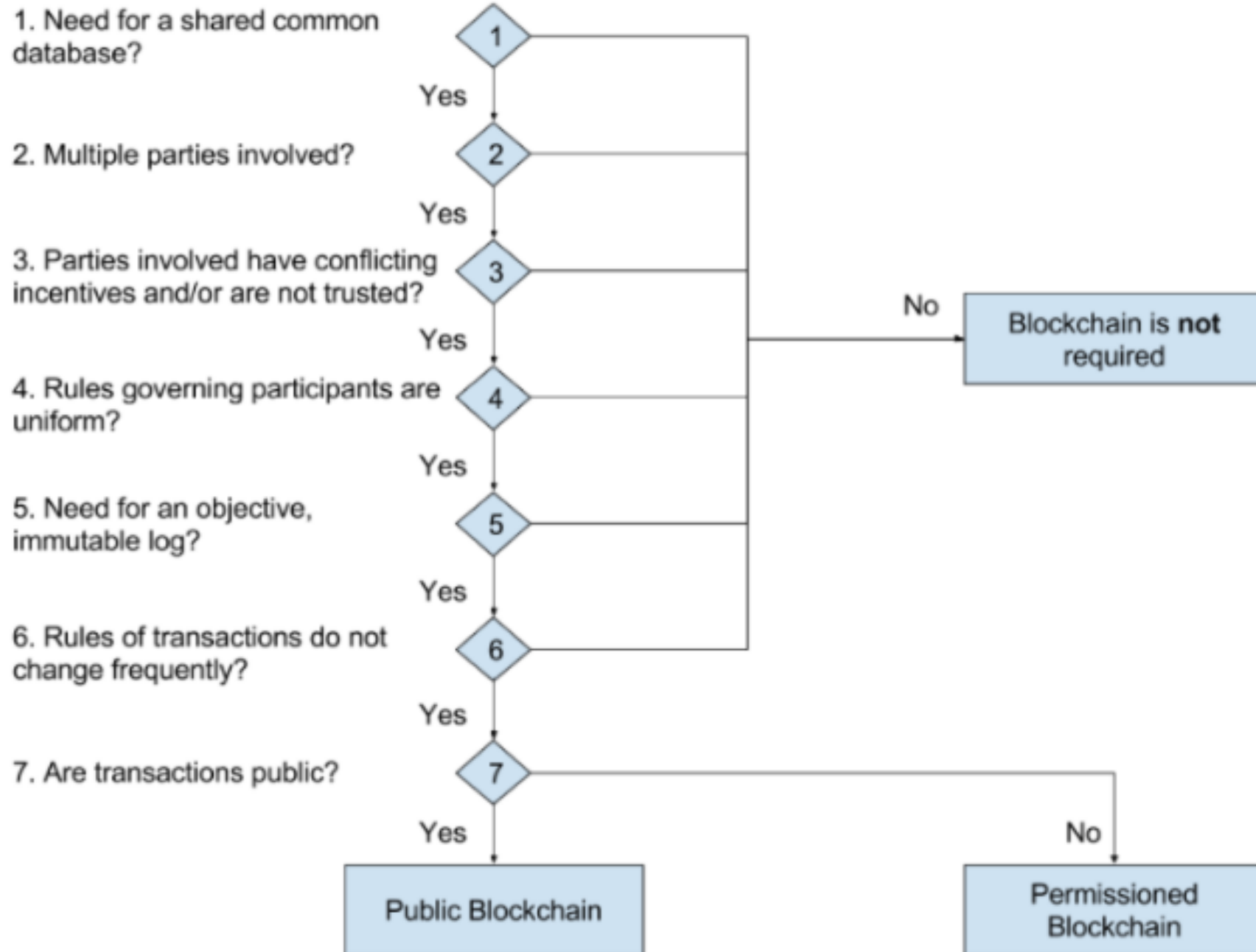


- 공유 된 공통 데이터베이스가 필요
- 비즈니스 프로세스의 데이터가 프로세스 따라 여러 데이터베이스에 입력.
 - 데이터가 모든 엔티티에서 일관성을 유지
- 분산 원장 및 처리에 의해 공동시스템 사용이 불필요함으로, 비용절감 효과가 기대됨
- 일부 노드가 정지하고 있어도 처리가 계속 있기 때문에 안정적인 시스템을 구축 할 수 있음
- 각 노드(각 참가자) 사이에서 동일한 비즈니스 로직, 데이터 공유, 위변조 할 수 없는 상태에서 거래기록이 유지 → 감사기능을 향상됨
- 트랜잭션 빈도는 초당 10,000 트랜잭션을 초과하지 않음



- 프로세스에 기밀 데이터가 포함
- 프로세스에 많은 양의 정적인 데이터가 포함
- 거래 규칙이 자주 변경 됨
- 데이터를 수집하거나 저장하는데 외부 서비스 사용
- 분산 원장의 합의를 수행하면서 갱신되기 때문에 현재 고성능이 요구되는 시스템
- 조직 및 회사를 포함하지 않고 하나의 조직만 참여하는 경우
 - 일반적으로시스템에대응가능
- 간단한DB / 트랜잭션처리의 대체

블록체인 적용



Hyperledger Fabric

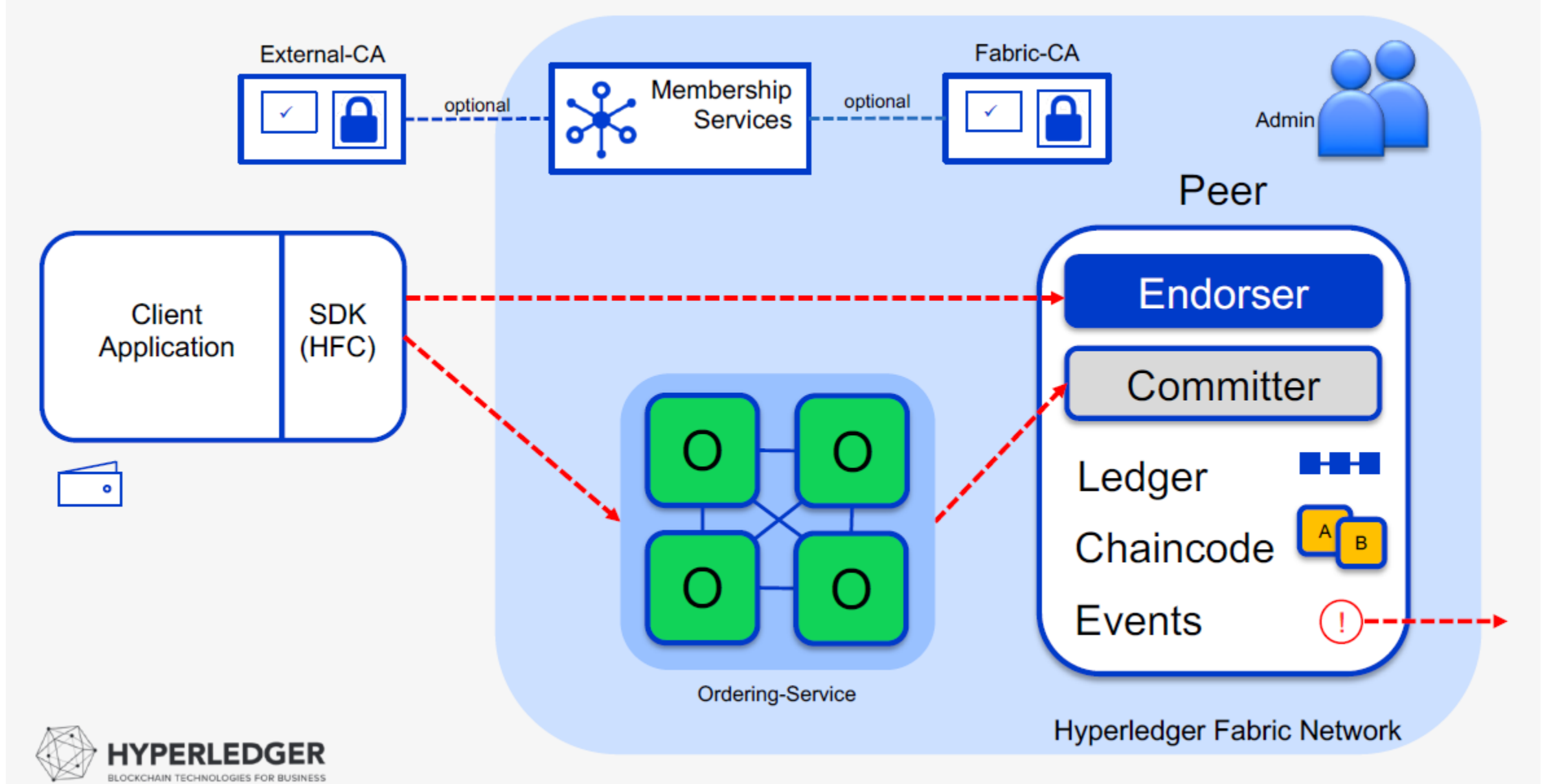
타 블록체인과의
차이점

구분	비트코인	이더리움	Fabric
암호화 화폐 요구	비트코인	이더 또는 사용자가 생성한 암호화 화폐	없음
네트워크	공개	공개 또는 허가형	허가형
거래	익명	익명 또는 비공개	공개 또는 기밀
합의	작업증명	작업증명	SOLO(Single node, development), Kafka (crach fault tolerance), SBFT(지원예정)
스마트 계약 (Smartcontract)	없음	있음(Solidity, Sperpent, LLL)	있음(Chaincode)
언어	C++	Go lang, C++, Python	Go lang, Java, Node.js

Hyperledger
Fabric의 특징

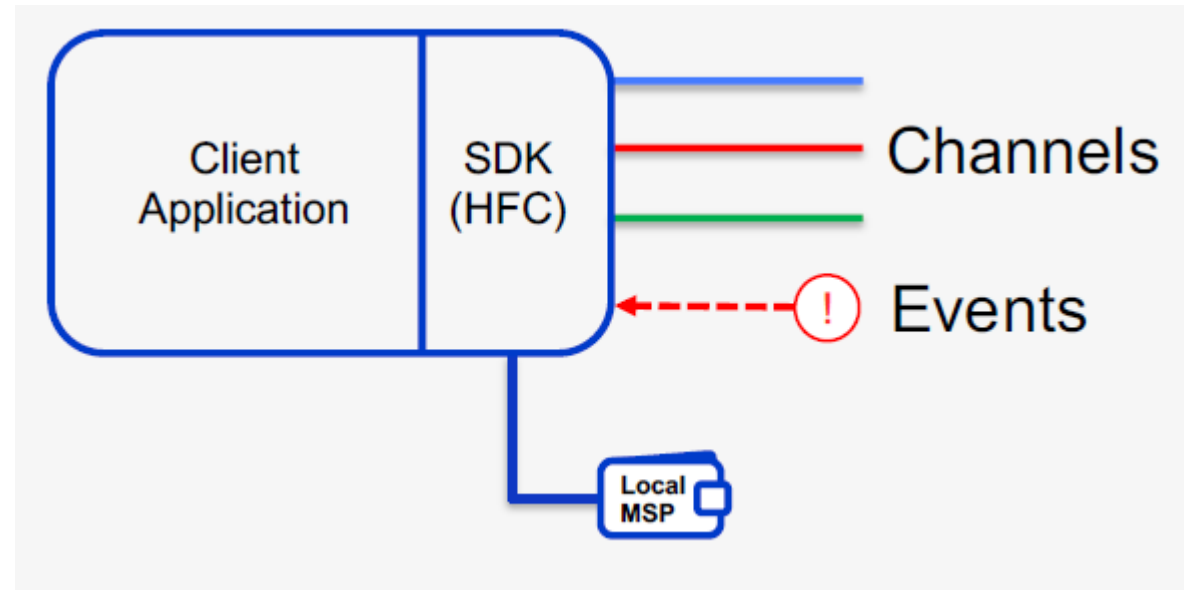
허가형 네트워크	거래의 기밀유지
집단으로 정의된 멤버십과 접근 권한을 비즈니스 네트워크에 제공합니다.	비즈니스의 유연성과 보안성을 제공하여 정확한 암호화 키를 가지고 있는 당사자들에게만 거래가 보이도록 합니다.
암호화 화폐 불필요	설정가능
거래를 확인하기 위해 채굴이나 값비싼 컴퓨팅을 요구하지 않습니다.	스마트 계약을 활용하여 네트워크 상의 비즈니스 프로세스를 자동화 합니다.

Hyperledger Fabric v1.0 Architecture



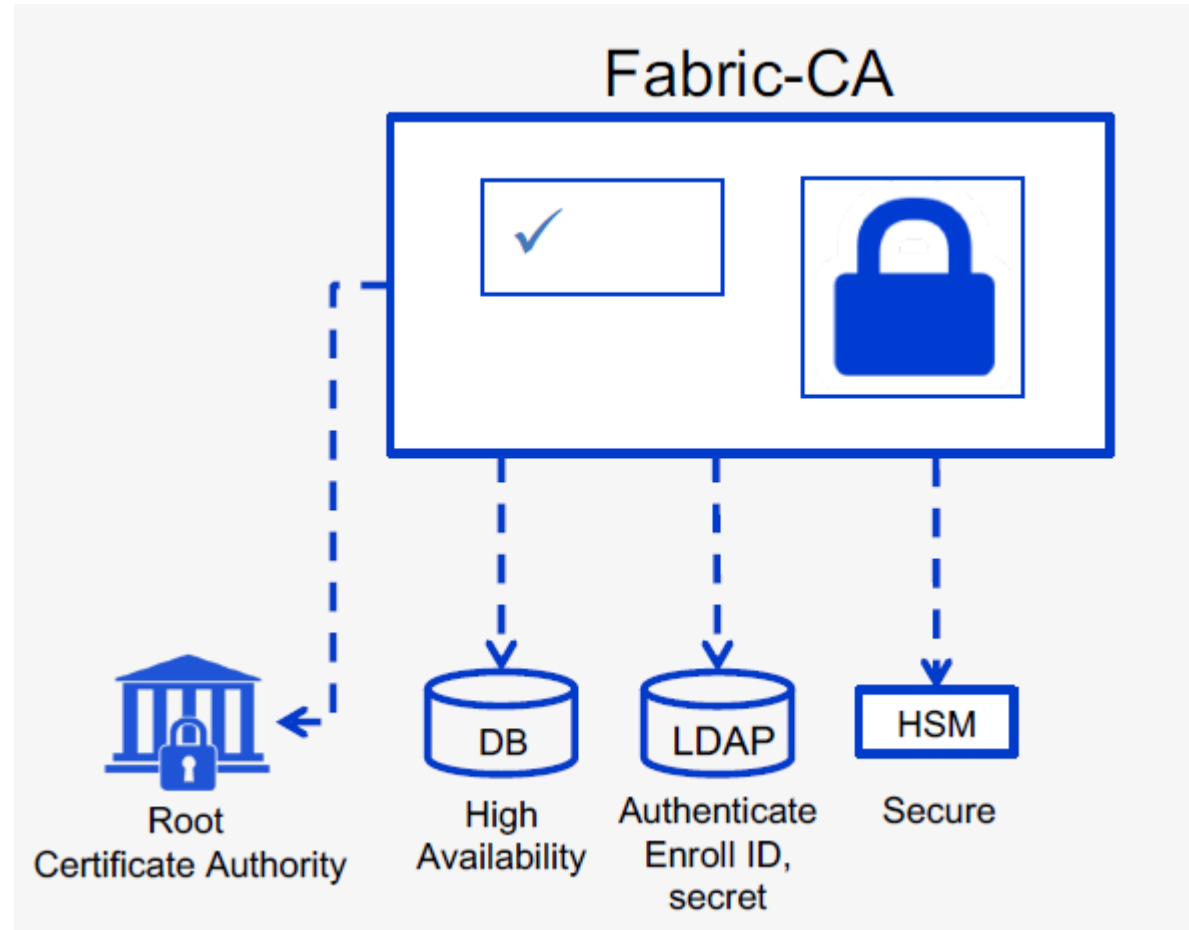
Component – Client Application

- 모든 클라이언트는 패브릭 SDK 를 사용하여:
- 채널을 통해 하나 이상의 피어에 접속
- 채널을 통해 하나 이상의 Orderer에 접속
- 피어로 부터 이벤트 수신
- Local MSP 기능 제공
- 다양한 개발 언어 지원
 - Node.js, Go, Java, Python



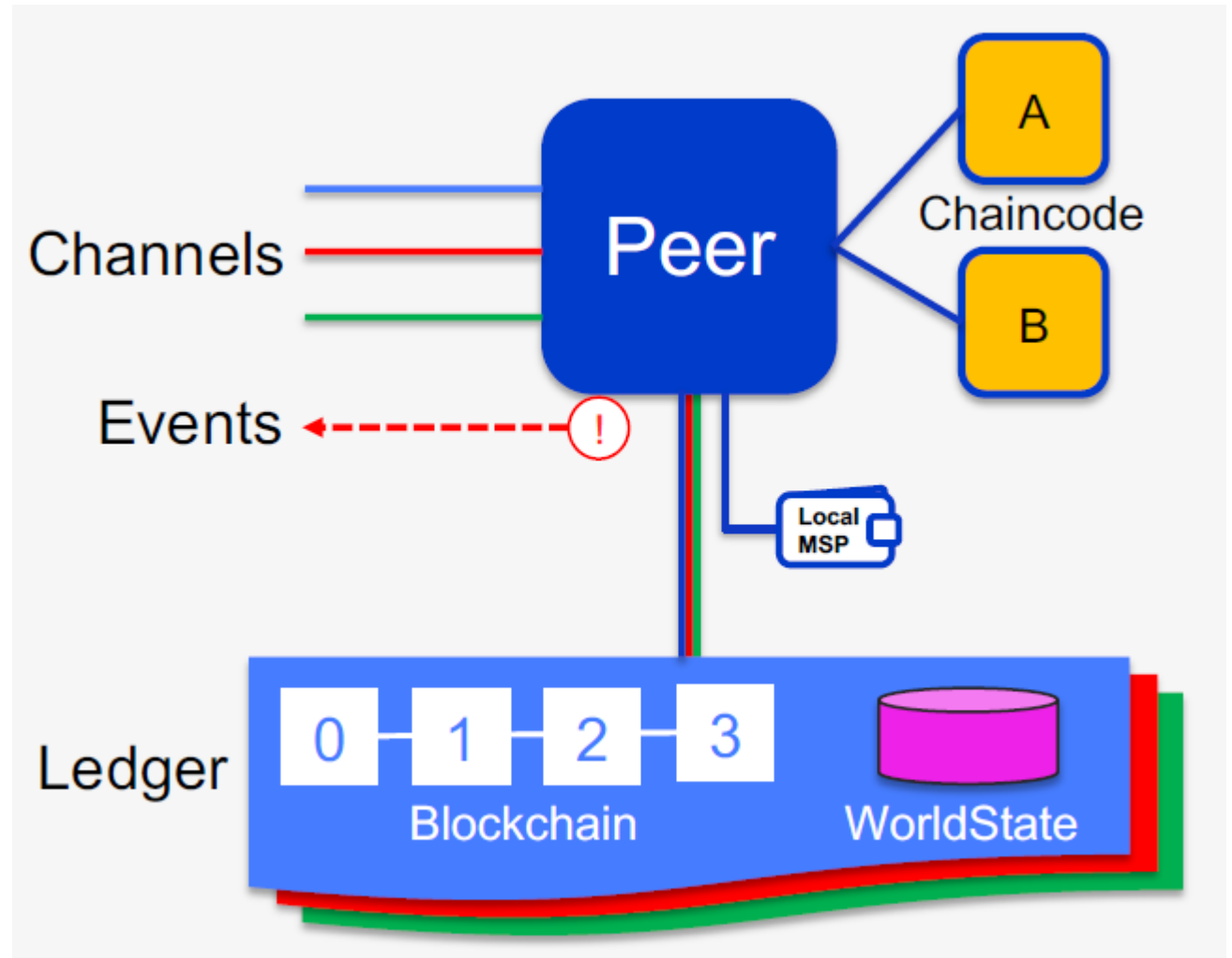
Component – Fabric CA

- 패브릭 네트워크에서 Ecerts를 발행하는
- Certificate Authority
- HA를 위한 클러스터링 지원
- 사용자 인증을 위한 LDAP 지원
- 보안을 위한 HSM 지원
- Intermediate CA를 위한 설정 가능



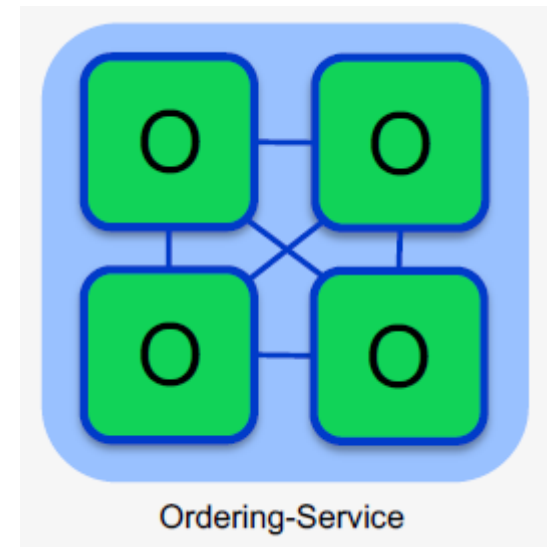
Component – Fabric Peer

- 한 개 이상의 채널에 연결됨
- 각 채널에 대해서 한 개 이상의 원장을 관리
- 체인코드는 도커 컨테이너로 분리되어 인스턴스화 됨
- 체인코드는 채널을 통해 공유 됨
- Local MSP (Membership Services Provider)는 암호화 방식을 제공
- 클라이언트 어플리케이션에 이벤트 발생



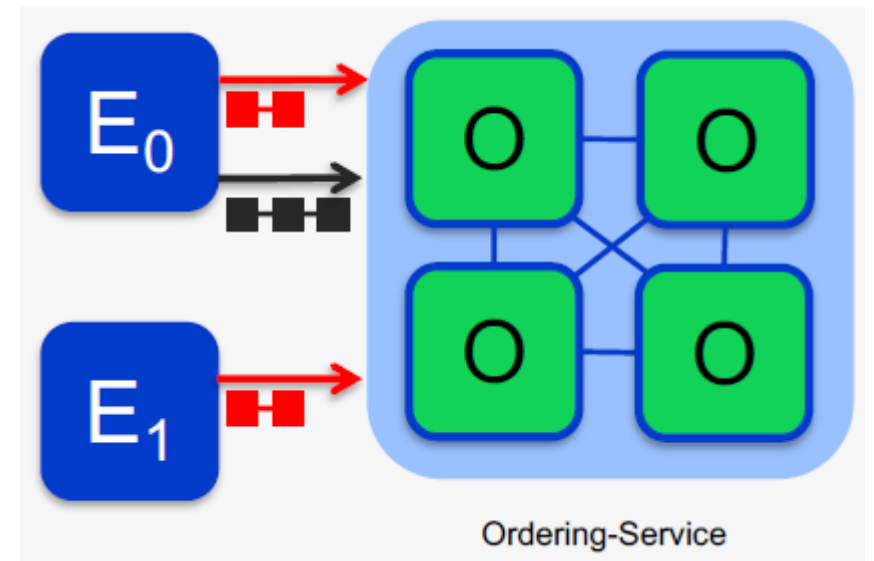
Component – Ordering Service

- 트랜잭션들을 블록으로 패키징하여 피어에게 전달하는 역할
- 채널을 통해 Ordering service와 통신
- Ordering service 설정옵션
 - SOLO: 개발을 위한 싱글 노드
 - Kafka: Crash fault tolerant consensus



Component – Channel

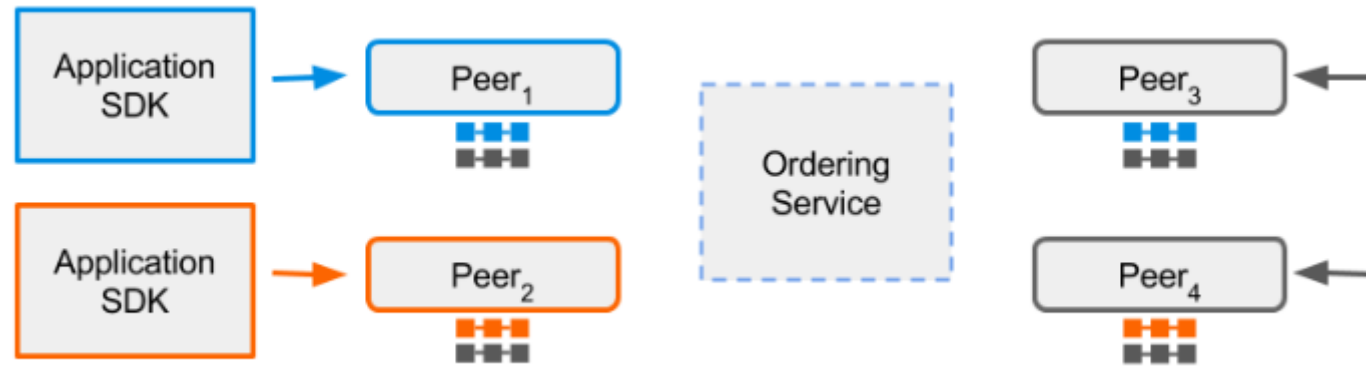
- 서로 다른 원장간 프라이버시를 제공
- 원장은 채널의 범위로 한정
 - 피어의 전체 네트워크에서 채널을 공유할 수 있음
 - 특정한 참여자 그룹 별로 채널 권한을 부여할 수 있음
- 체인코드는 피어에 설치
 - World State에 접속 할 수 있음
- 체인코드 특정 피어에서 인스턴스화 됨
- 피어는 멀티채널에 참여 할 수 있음
- 퍼포먼스와 확장성을 고려하여 동시에 실행 가능 함



Component – Channel

- 채널을 사용하면 여러 블록 체인을 분리하여 동일한 네트워크를 활용 가능
- 트랜잭션이 수행 된 채널의 구성원만 트랜잭션의 세부 사항을 볼 수 있음
- 채널은 이해 관계자에 대한 트랜잭션 가시성 만 허용하기 위해 네트워크를 분할
- 이 메커니즘은 트랜잭션을 다른 원장에 위임하여 작동
- 채널의 구성원 만이 합의에 참여하고 네트워크의 다른 구성원은 채널의 트랜잭션을 볼 수 없음

Component – Channel



- 위의 다이어그램은 파란색, 주황색 및 회색의 세 가지 고유 한 채널
 - 각 채널에는 자체 애플리케이션, 원장 및 피어가 존재
- 피어는 여러 네트워크 또는 채널에 속할 수 있음
 - 여러 채널에 참여한 피어는 서로 다른 원장에게 트랜잭션을 시뮬레이션하고 커밋
 - Ordering Service는 모든 네트워크 또는 채널에서 동일
- 네트워크 설정을 통해 채널을 만들 수 있음
- 동일한 체인 코드 로직을 여러 채널에 적용 할 수 있음
- 사용자는 여러 채널에 참여할 수 있음

Hyperledger Fabric Elements

1. Channels

- 오직 이해관계자에게만 트랜잭션을 볼 수 있도록 하는 데이터 파티셔닝 메커니즘
- 각 채널은 해당 특정 채널에 대한 트랜잭션만 포함하는 독립적인 트랜잭션 블록체인

2. Chaincode

- 스마트 계약, 자산 정의와 자산 수정을 위한 비즈니스 로직 (or 트랜잭션)을 캡슐화
- 트랜잭션 호출로 인해 Ledger가 변경

3. Ledger

- 네트워크의 현재 상태와 일련의 트랜잭션 호출이 포함
- 공유되고 권한이 부여된 Ledger는 추가 기록 시스템이며 신뢰할 수 있는 단일 소스로 제공

4. Network

- 블록 체인 네트워크를 구성하는 데이터 처리 피어들의 집합
- 네트워크는 일관되게 복제 된 Ledger를 유지 관리

Hyperledger Fabric Elements

5. Ordering Service

- 트랜잭션을 블록으로 정렬하는 노드 모음

6. World State

- 네트워크의 모든 자산에 대한 최신 데이터를 반영
- 데이터는 효율적인 액세스를 위해 데이터베이스에 저장
- LevelDB, CouchDB 지원

7. Membership Service Provider (MSP)

- 클라이언트 및 피어들에게 ID 및 허가된 액세스를 관리

State Database

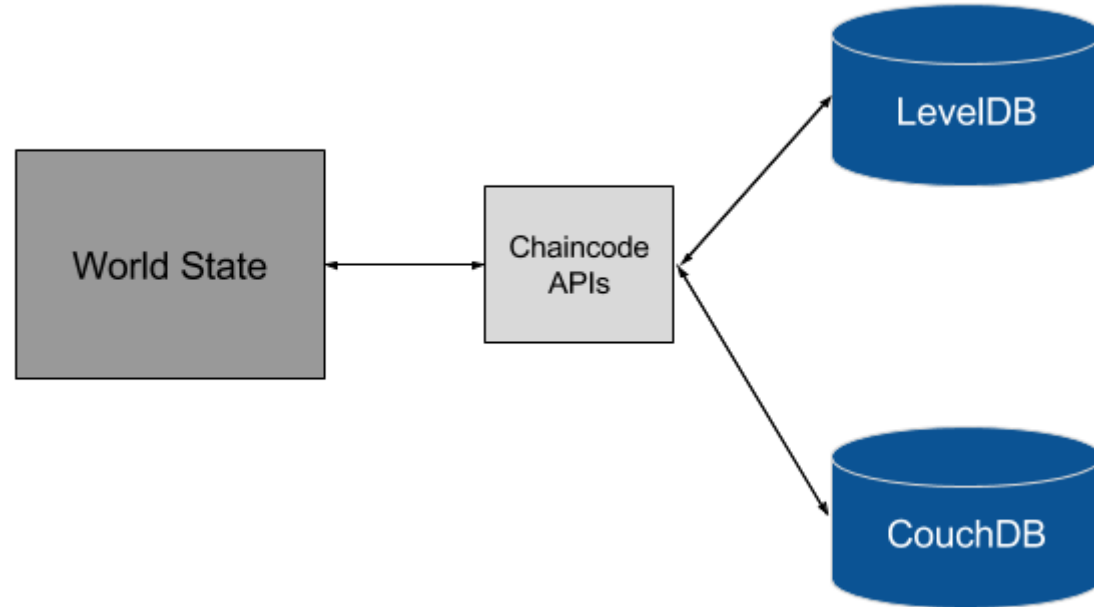
- 현재 World State는 원장의 모든 자산에 대한 최신 값
- 현재 상태는 채널에서 커밋 된 모든 트랜잭션을 나타내기 때문에 때로는 World State라고 함
- 체인 코드 호출은 현재 상태 데이터에 대해 트랜잭션을 실행
 - 이러한 체인 코드 상호 작용을 매우 효율적으로 만들기 위해 각 자산의 최신 키 / 값 쌍이 상태 데이터 베이스에 저장

State Database

- 상태 데이터베이스는 단순히 체인의 커밋 된 트랜잭션에 대한 인덱싱 된 뷰
 - 체인에서 언제든지 재생성 할 수 있음
 - 상태 데이터베이스는 새로운 트랜잭션이 수락되기 전에 피어가 시작될 때 자동으로 복구 (또는 필요한 경우 생성)
 - 기본 상태 데이터베이스는 LevelDB (CouchDB로 변경 가능)
- LevelDB
 - Hyperledger 패브릭의 기본 키 / 값 상태 데이터베이스이며 키 / 값 쌍을 저장
- CouchDB
 - LevelDB와 달리 CouchDB는 JSON 객체를 저장
 - CouchDB는 키, 합성, 키 범위 및 전체 데이터가 풍부한 쿼리를 지원

State Database

- LevelDB와 CouchDB는 구조와 기능면에서 매우 유사
- LevelDB와 CouchDB는 키 자산 가져 오기 및 설정, 키 기반의 쿼리와 같은 핵심 체인 코드 작업을 지원
 - 두 가지 모두 키를 범위별로 쿼리 가능
 - 복합 키를 모델링하여 여러 매개 변수에 대해 동등한 쿼리를 사용
 - JSON 문서 저장소 인 CouchDB는 체인 코드 값 (ex, asset)을 JSON 데이터로 모델링 할 때 체인 코드 데이터에 대한 추가 쿼리를 추가적으로 지원



Smart Contracts (Chain Code)

- 스마트 계약은 트랜잭션을 실행하고 원장 (World State) 내에 저장된 자산의 상태를 수정하는 로직이 포함된 컴퓨터 프로그램
- Hyperledger Fabric 스마트 계약은 체인 코드(Go로 작성)
- 체인 코드는 Hyperledger 패브릭 네트워크의 비즈니스 로직으로 사용되며, 체인 코드는 네트워크 내에서 자산을 조작하는 방법을 지시

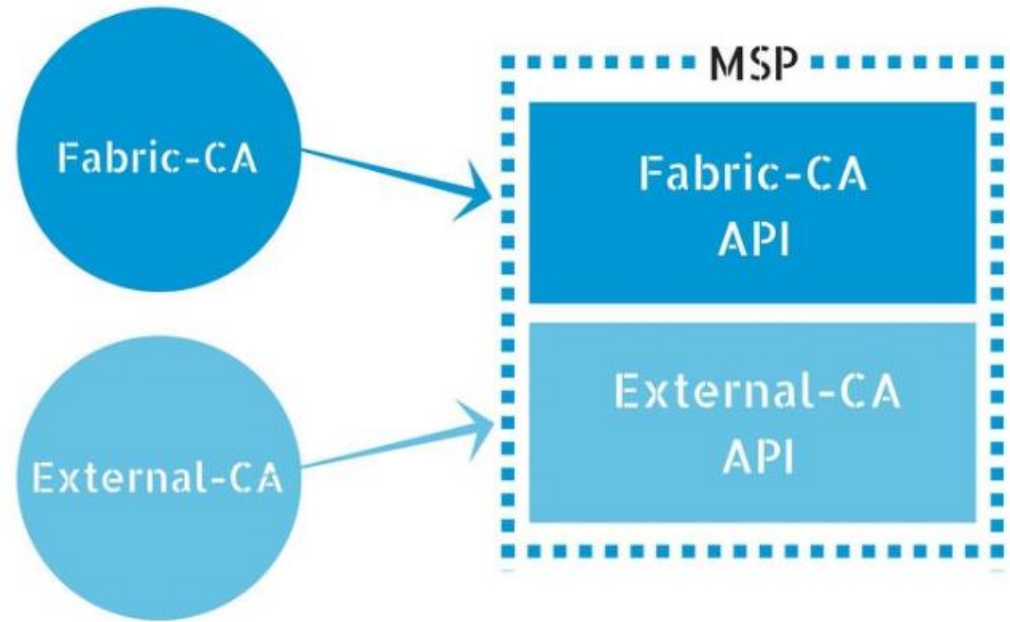
Membership Service Provider (MSP) / Identities

- 멤버십 서비스 공급자 (MSP)는 신원을 확인하고, 인증하고, 네트워크 액세스를 허용하는 규칙을 정의하는 구성 요소
- MSP는 사용자 ID를 관리하고 네트워크에 가입하려는 클라이언트를 인증
 - 클라이언트가 트랜잭션을 제안 할 수 있는 자격 증명을 제공하는 것이 포함
- MSP는 확인 된 신원 확인 시 사용자 인증서를 확인하고 취소 할 수 있는 플러그 가능한 인터페이스 인 인증 기관을 사용
- MSP에 사용되는 기본 인터페이스는 Fabric-CA API이지만 조직에서는 원하는 외부 인증 기관으로 구현 가능
 - 모듈 형 Hyperledger Fabric의 또 다른 기능
- Hyperledger 패브릭은 많은 유형의 외부 인증 기관 인터페이스를 사용할 수 있는 많은 자격 증명 아키텍처를 지원
 - 하나의 Hyperledger 패브릭 네트워크는 여러 조직에서 관리 할 수 있습니다.

What Does the MSP Do?

- 시작하기 위해 사용자는 인증 기관을 사용하여 인증
 - 인증 기관은 응용 프로그램, Peer, Endorser 및 Orderer ID를 식별하고 이러한 자격 증명을 확인
 - 서명은 서명 알고리즘 및 서명 검증 알고리즘을 사용하여 생성
 - 서명 생성은 서명 알고리즘 (Signing Algorithm)으로 시작
 - 서명 알고리즘은 각각의 ID와 연관된 엔티티의 자격 증명을 사용하고 보증을 출력
 - 특정 ID에 바인딩 된 바이트 배열 인 서명이 생성
- 다음으로 서명 확인 알고리즘은 신원, 보증 및 서명을 입력으로 사용
 - 서명 바이트 배열이 입력 된 보증에 대한 유효한 서명과 일치하면 '승인'을 출력
 - 그렇지 않으면 '거부'를 출력
 - 출력이 '승인'인 경우 사용자는 네트워크에서 트랜잭션을 보고 네트워크의 다른 액터와 트랜잭션을 수행 가능
 - 출력이 '거부'인 경우 사용자가 제대로 인증되지 않았으므로 네트워크에 트랜잭션을 제출하거나 이전 트랜잭션을 볼 수 없음

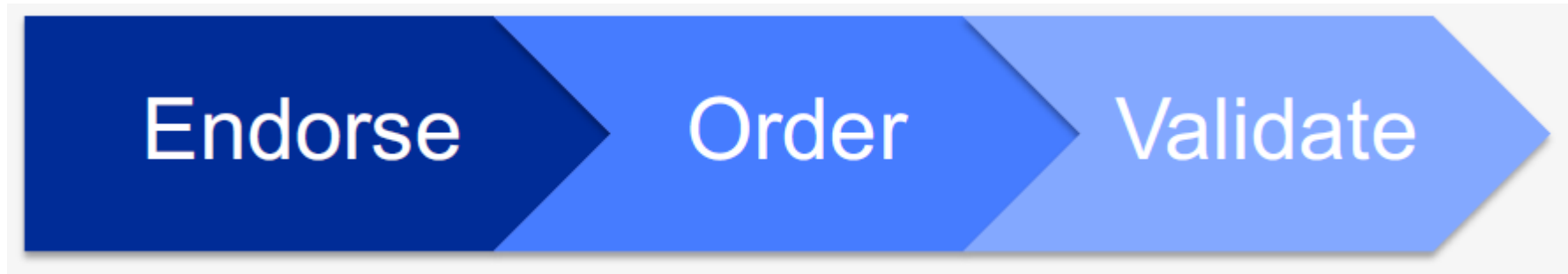
What Does the MSP Do?



- Fabric-Certificate Authority
 - 일반적으로 인증 기관은 허가 된 블록 체인에 대한 등록 인증서를 관리
 - Fabric-CA는 Hyperledger Fabric의 기본 인증 기관이며 사용자 ID 등록을 처리
 - Fabric-CA 인증 기관은 Enrollment Certificates (E-Certs) 발급 및 취소를 담당
 - Fabric-CA의 현재 구현은 장기 신원 인증서를 제공하는 E-Certs 만 발행
 - Enrollment Certificate Authority (E-CA)에서 발급 한 E-Cert는 동료에게 신원을 지정하고 네트워크에 가입하고 트랜잭션을 제출할 수 있는 권한을 부여

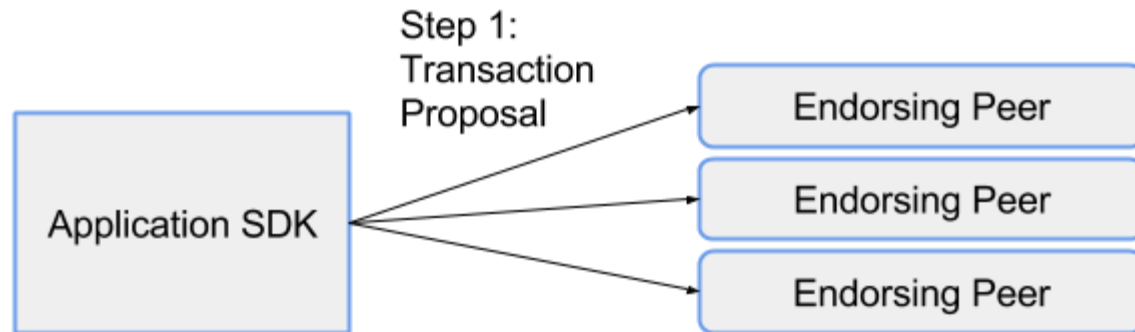
Consensus

- 분산 원장 시스템에서 합의는 장부에 추가 할 다음 트랜잭션 세트에 대한 합의에 도달하는 프로세스
- Hyperledger Fabric에서 합의는 세 가지 단계로 구성
 - Transaction endorsement
 - Ordering
 - Validation and commitment



Transaction Flow (Step1)

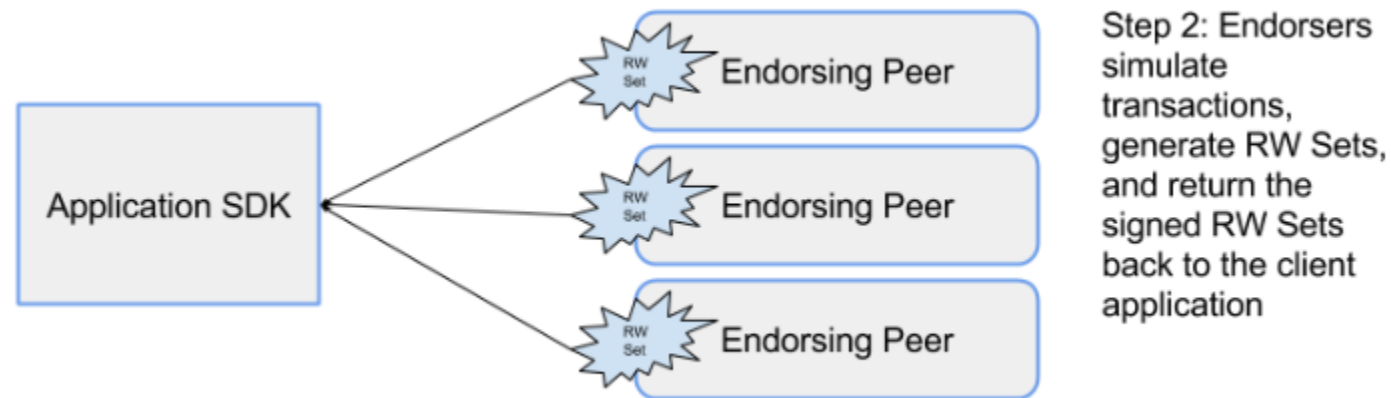
- Hyperledger 패브릭 네트워크에서 트랜잭션은 트랜잭션 제안을 보내는 클라이언트 응용 프로그램에서 시작
- Endorsing Peers에게 트랜잭션을 제안하는 것



- 클라이언트 응용 프로그램 → 일반적으로 응용 프로그램 또는 클라이언트
 - 사람들이 블록 체인 네트워크와 통신 할 수 있도록 지원
- 응용 프로그램 개발자는 응용 프로그램 SDK를 통해 Hyperledger 패브릭 네트워크를 활용

Transaction Flow (Step2)

- 각 Endorsing Peer는 원장을 업데이트하지 않고 제안 된 트랜잭션을 시뮬레이션
 - 1) RW 세트라는 읽기 및 기록 된 데이터 세트를 수집
 - 2) RW 세트는 트랜잭션을 시뮬레이션하는 동안 현재 World State 에서 읽은 내용과 트랜잭션이 실행 된 상태로 작성된 내용을 캡처
 - 3) 그런 다음 이 RW 세트는 Endorsing Peer 에 의해 서명
 - 4) 클라이언트 응용 프로그램에 리턴 되어 트랜잭션 흐름의 다음 단계에서 사용



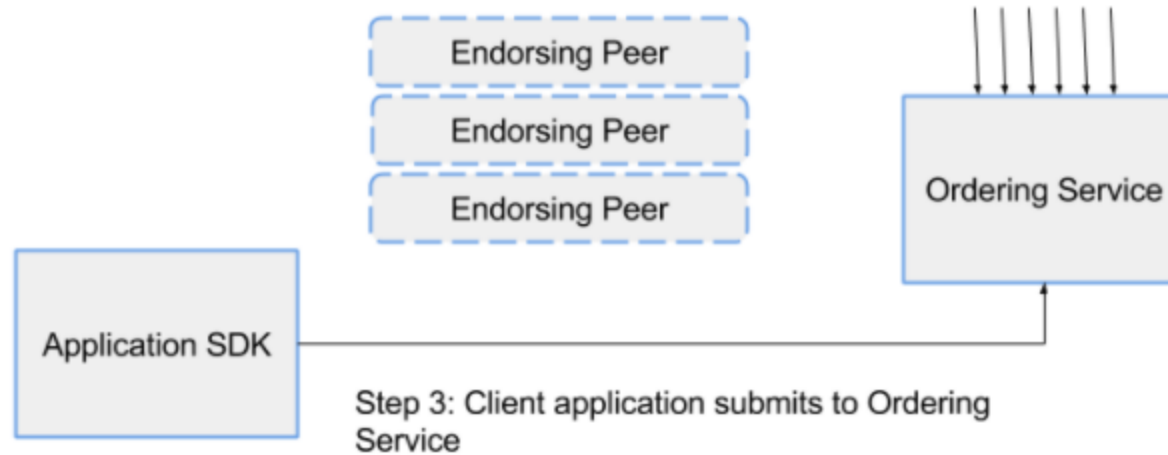
- Endorsing Peer 는 거래 제안을 시뮬레이션하기 위해 Smart Contract를 갖는다.

Transaction Flow (Step2)

- Transaction에 대한 보증이란 시뮬레이션 된 트랜잭션의 결과에 대한 서명 된 응답
- Transaction 보증 방법은 체인 코드가 배포 될 때 지정되는 보증 정책에 따라 다름
 - 보증 정책의 한 예가 "보증하는 동료의 대다수가 거래를 보증 해야 합니다"
 - 특정 체인 코드에 보증 정책이 지정 되었기 때문에 다른 채널은 다른 보증 정책을 가질 수 있음

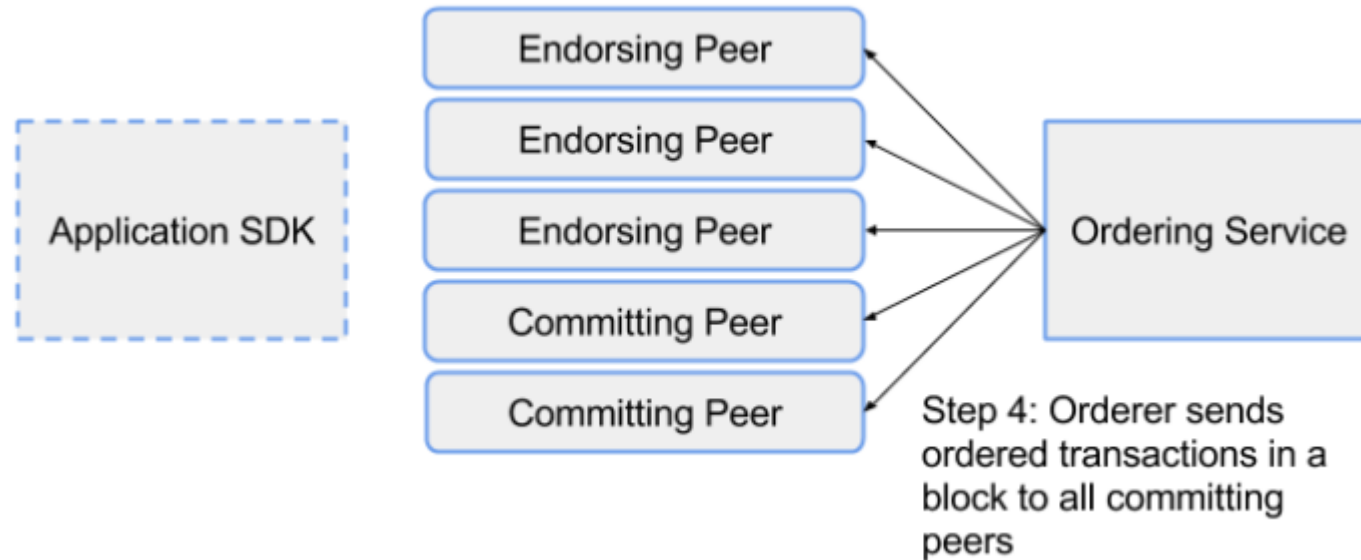
Transaction Flow (Step3)

- 응용 프로그램은 보증 된 트랜잭션과 RW 세트를 Ordering Service에 제출
- Ordering은 승인 된 트랜잭션 및 다른 응용 프로그램에서 제출 한 RW 세트와 함께 네트워크에서 발생



Transaction Flow (Step4)

- Ordering Service는 보증 된 트랜잭션과 RW 세트를 가져 온 다음, 이 정보를 블록으로 Ordering → 다음 블록을 모든 Committing Peers에게 전달합니다.



- Orderer 클러스터로 구성된 Ordering Service는 Transaction, Smart Contract 또는 공유 원장을 관리하지 않음
- Ordering Service 는 승인 된 거래를 수락하고 이러한 거래가 원장에게 위탁되는 순서를 지정
- 패브릭 v1.0 아키텍처는 ' Ordering (Solo, Kafka, BFT)'의 특정 구현이 플러그 가능한 구성 요소가 되도록 설계
- Hyperledger 패브릭의 기본 Ordering Service는 Kafka입니다.

Transaction Flow (Step4)

- Ordering (Part1)

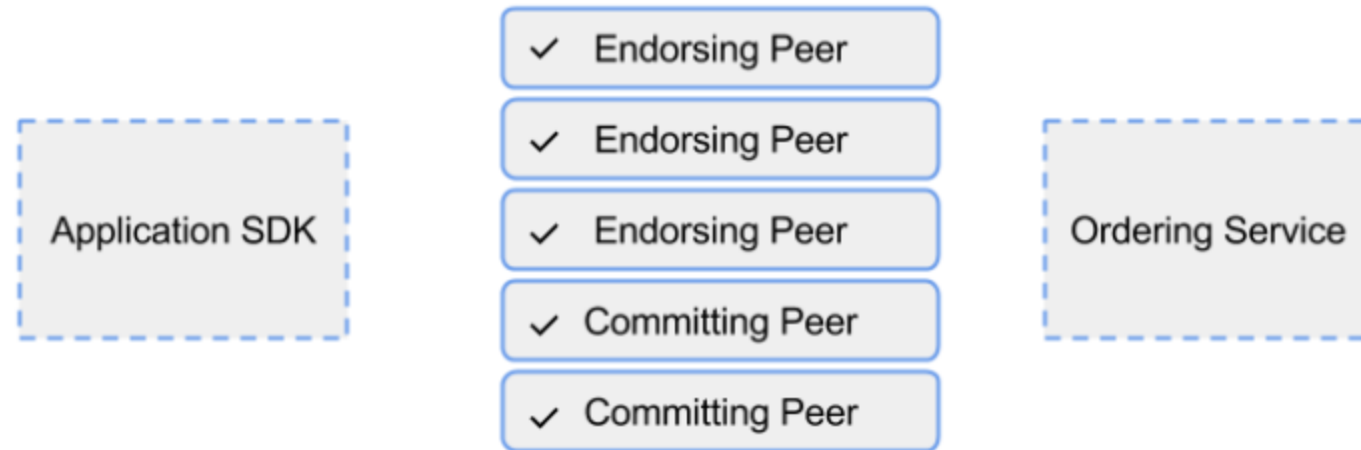
- 블록 체인 네트워크에서 트랜잭션은 일관된 순서로 공유 원장에 기록
- World State에 대한 업데이트가 네트워크에 커밋 될 때 유효하도록 트랜잭션 순서를 설정
- 암호화 퍼즐이나 마이닝을 통해 Ordering이 이루어지는 Bitcoin 블록 체인과는 다름
- Hyperledger Fabric은 네트워크를 실행하는 조직이 해당 네트워크에 가장 적합한 Ordering 메커니즘을 선택할 수 있음
 - 이러한 모듈성과 유연성으로 인해 Hyperledger Fabric은 엔터프라이즈 응용 프로그램에 매우 유리

Transaction Flow (Step4)

- Ordering (Part2)
 - Hyperledger Fabric은 SOLO, Kafka 및 SBFT (Simplified Byzantine Fault Tolerance)의 세 가지 정렬 메커니즘을 제공
 - SOLO
 - Hyperledger 패브릭 네트워크를 실험하는 개발자가 가장 일반적으로 사용하는 Hyperledger 패브릭 Ordering 메커니즘. 솔로는 단일 주문 노드를 포함
 - Kafka
 - 프로덕션 용도로 권장되는 Hyperledger 패브릭 Ordering 메커니즘
 - 오픈 소스 스트림 처리 플랫폼 인 Apache Kafka
 - 데이터는 승인 된 트랜잭션과 RW 세트로 구성
 - 카프카 (Kafka) 메커니즘은 Ordering에 Crash Fault-tolerant 솔루션을 제공
 - SBFT(Simplified Byzantine Fault Tolerance)
 - fault-tolerant 및 byzantine fault-tolerant이며, 악의적 인 노드나 결함이 있는 노드가 있는 경우에도 합의에 도달 할 수 있음
 - Hyperledger 패브릭 커뮤니티는 아직이 메커니즘을 구현하지 않음

Transaction Flow (Step5)

- Committing Peer는 RW 세트가 현재 World State와 여전히 일치하는지 확인하여 트랜잭션의 유효성을 검사
- Endorser가 트랜잭션을 시뮬레이션 할 때 존재했던 읽기 데이터는 현재 World State와 동일
- Committing Peer가 트랜잭션의 유효성을 검사하면 트랜잭션이 원장에 기록되고 RW 상태의 데이터 쓰기로 World State가 업데이트됩니다.



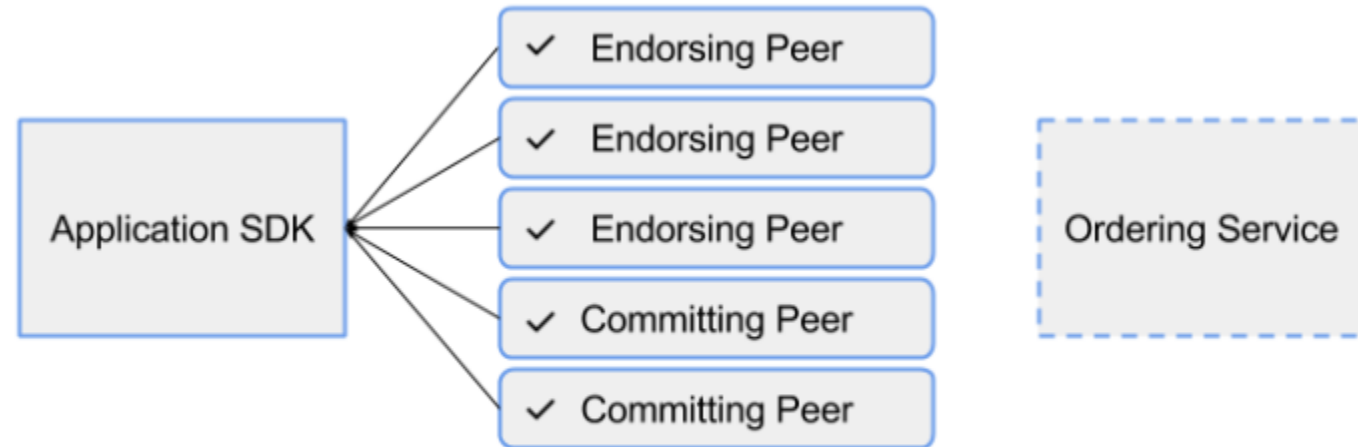
Step 5: Committing peers validate each transaction in the block

Transaction Flow (Step5)

- 트랜잭션이 실패하면
 - Committing Peer가 RW 세트가 현재 World State와 일치하지 않는다는 것을 알게 되면 블록으로 정렬 (Ordered) 된 트랜잭션은 여전히 해당 블록에 포함되지만 유효하지 않은 것으로 표시
 - World State는 업데이트되지 않음
- Committing Peers는 공유 된 원장에 트랜잭션 블록을 추가하고 World State를 업데이트 할 책임이 있음
 - Smart Contract을 가질 수 있지만 필수 조건은 아님

Transaction Flow (Step6)

- 마지막으로 Committing Peer는 비동기 적으로 트랜잭션의 성공 또는 실패를 클라이언트 응용 프로그램에 알림
- 응용 프로그램은 Committing Peer에 의해 알림을 받음



Step 6: Committing peers asynchronously notify the Application of the results of the transaction.

Identify Verification

- 수많은 보증, 유효성 및 버전 검사가 수행되는 것 외에도 트랜잭션 흐름의 각 단계에서 신원 확인이 진행
- 액세스 제어 목록은 네트워크의 계층 구조 (Ordering Service에서 Channels까지)로 구현
- 트랜잭션 제안이 다른 아키텍처 구성 요소를 통과 할 때 페이로드는 반복적으로 서명, 확인 및 인증

Transaction Flow

