석사학위논문

도커 기반 하이퍼레저 패브릭 제너레이터

2022년

한 성 대 학 교 대 학 원

I T 융 합 공 학 과

I T 융 합 공 학 전 공

박 재 훈

석사학위논문
지도교수 서화정

도커 기반 하이퍼레저 패브릭 제너레이터

Hyperledger Fabric Generator based on Docker

2021년 12월 일

한성대학교 대학원

IT융합공학과

IT융합공학전공

박 재 훈

석 사 학 위 논 문
지도교수 서화정

도커 기반 하이퍼레저 패브릭 제너레이터

Hyperledger Fabric Generator based on Docker

위 논문을 공학 석사학위 논문으로 제출함

2021년 12월 일

한 성 대 학 교 대 학 원

I T 융 합 공 학 과

I T 융 합 공 학 전 공

박 재 훈

박재훈의 공학 석사학위 논문을 인준함

2021년 12월 일

심사위원장 서화정 (인)

심 사 위 원 이웅희 (인)

심 사 위 원 최원석 (인)

국 문 초 록

도커 기반 하이퍼레저 패브릭 제너레이터

한 성 대 학 교 대 학 원
I T 융 합 공 학 과
I T 융 합 공 학 전 공
박 재 훈

하이퍼레저 패브릭은 IBM에서 주도하고 있는 프라이빗 블록체인 프레임워크이다. 하이퍼레저란 리눅스 재단에서 진행 중인 엔터프라이즈 블록체인 프로젝트로, 이곳에 여러 회사들이 참여하여 툴 혹은 프레임워크를 제공한다. 하이퍼레저 패브릭은 그 중 가장 개발이 활발한 곳들 중 하나로, 기존 블록체인의 장점을 이용하여 강력한 비즈니스 블록체인 시스템을 가진 프레임워크이다. 그렇기에 현업에서도 많이 쓰이며, 연구자들에게도 좋은 소재가 되는 블록체인 시스템이다. 특히 도커로 된 이미지 및 예제를 제공해주기 때문에 다른 블록체인과 달리 뛰어난 하드웨어를 여러 개 가지고 있을 필요 없이 다양한 네트워크 형태를 구축하여 도커를 통해 로컬에서 실행해보는 것이 가능하다. 하지만 하이퍼레저 패브릭은 네트워크 구성이 매우 어렵고 복잡하다는 단점이 있다. 본 논문에서는 이에 대해 파이썬 기반의 HLF-generator를 구현하여 진입장벽을 낮추고 생산성을 향상시킨다.

【주요어】 블록체인, 탈중앙화, 하이퍼레저 패브릭

목 차

표 목 차

그 림 목 차

제 1 장 서론

제 1 절 연구 목적

하이퍼레저 프로젝트는 리눅스 재단에서 주도하는 비즈니스 블록체인 프로젝트로, 블록체인의 장점을 가져와 여러 시스템을 구축하고자 한다. 이 프로젝트에 다양한 회사가 참여하여 시스템을 개발하는데, 그 중 가장 활발한 회사들 중 하나인 IBM이 개발한 하이퍼레저 패브릭[1]은 블록체인을 적용하여 비즈니스를 구축할 수 있는 프레임워크이다.

하이퍼레저 패브릭은 기업들을 대상으로 솔루션을 제공하며, 블록체인을 연구하는 연구원들에게도 좋은 소재를 제공한다. 특히 체인코드는 기존의 비슷한 역할을 하는 퍼블릭 블록체인 플랫폼의 이더리움과 달리 실행에 수수료가 필요하지 않으며, 프라이빗 블록체인인만큼 인증된 사용자만이 접근 가능하여 블록체인과 기업 비즈니스를 잘 접목시킨 프레임워크라고 할 수 있다.

제 2 절 문제 제기 및 해결 방안 제안

하지만 하이퍼레저 패브릭은 네트워크 구축 난이도가 매우 높다는 문제를 가지고 있다. 기본적으로 네트워크를 구성하는 요소가 타 블록체인과 다르게 복잡하며, 그에 대해 인증을 주는 방식이나 제공하는 여러 기능들이 일반적인 암호화폐들의 기반이 되는 블록체인과 달리 매우 복잡하다.

또한 일반적인 연구자 및 개발자들에게도 이러한 복잡한 프로젝트 구성 방식은 생산성을 매우 저하시키는 요소가 된다.

본 논문에서는 HLF-generator를 통해 이러한 문제를 해결한다. HLF-generator는 사용자의 입력에 따라 프로젝트를 생성해주는 시스템이다. 도커 기반으로 동작하며, 프로젝트를 자동으로 생성해주기 때문에 하이퍼레저 패브

릭의 구성요소들에 대해 잘 알 수 있게 되며, 그만큼 생산성을 늘릴 수도 있다.

제 3 절 논문 구성

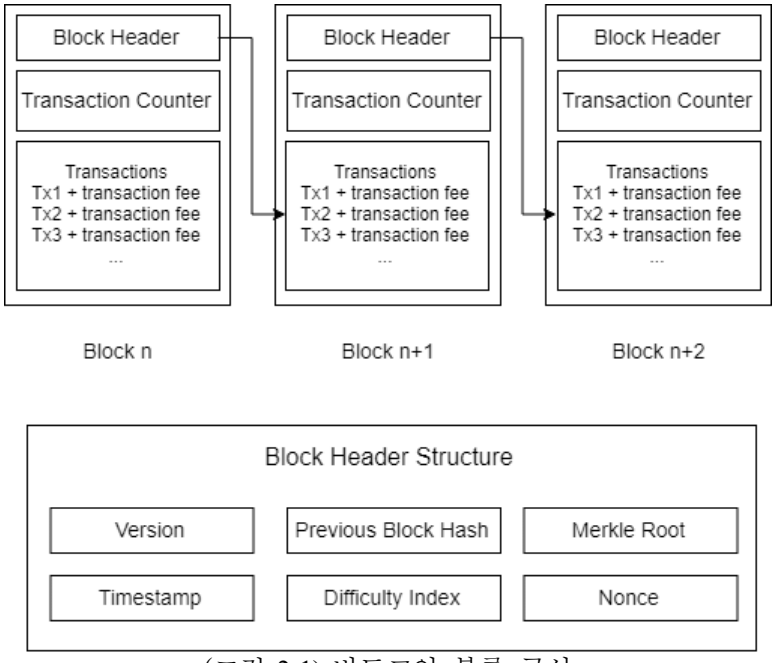
본 논문의 구성은 다음과 같다. 먼저 논문을 이해하기 위한 배경 설명이 등장하고, 그 뒤로 HLF-generator에 대한 설명 및 구성 요소에 대한 설명으로 구성되어 있다. 그 후 하이퍼레저 패브릭 프로젝트 생성 과정에 대해 특정 시나리오를 만드는 것에 대한 과정을 기존 방식과 HLF-generator를 통해 비교하는 평가를 가진다. 마지막으로는 결론 및 향후 연구에 대해 서술한다.

제 2 장 배경 지식

제 1 절 블록체인

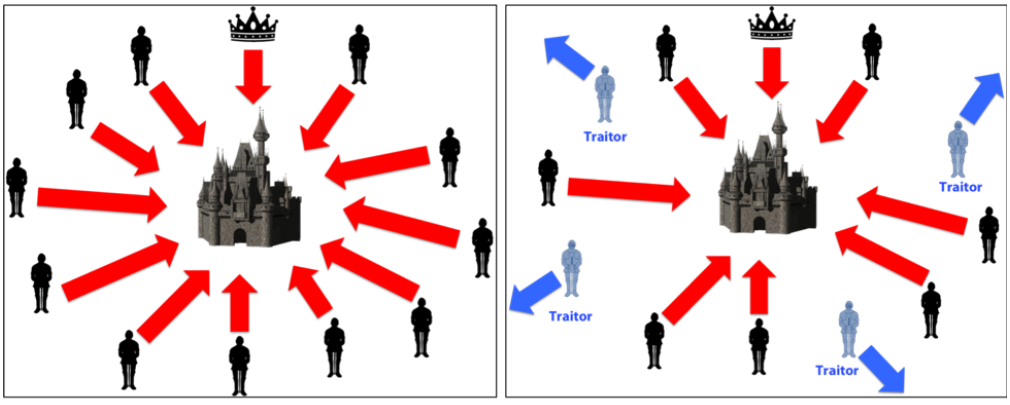
블록체인이란 데이터를 블록 형태로 만들어서 체인으로 묶는 방식으로, P2P 네트워크 상에서 데이터의 무결성을 보장하기 위한 방법이다.

블록체인은 사토시 나카모토가 비트코인[2] 개발을 위해 고안한 개념으로, P2P 네트워크에서의 신뢰를 작업증명 방식을 통해 해결한 방식이다. 모든 블록은 자신의 해시값을 가지고 있으며, 헤더 내에 이전 블록의 해시를 가지고 있다. 이것을 통해 이전 해시값을 검증하여 무결성을 유지한다. 기존의 은행 시스템을 비롯한 여러 시스템들은 중앙의 서버에 모든 데이터가 저장되는 중앙 집권 방식이었다. 중앙 집권 방식은 여러 문제를 안고 있다.



(그림 2-1) 비트코인 블록 구성

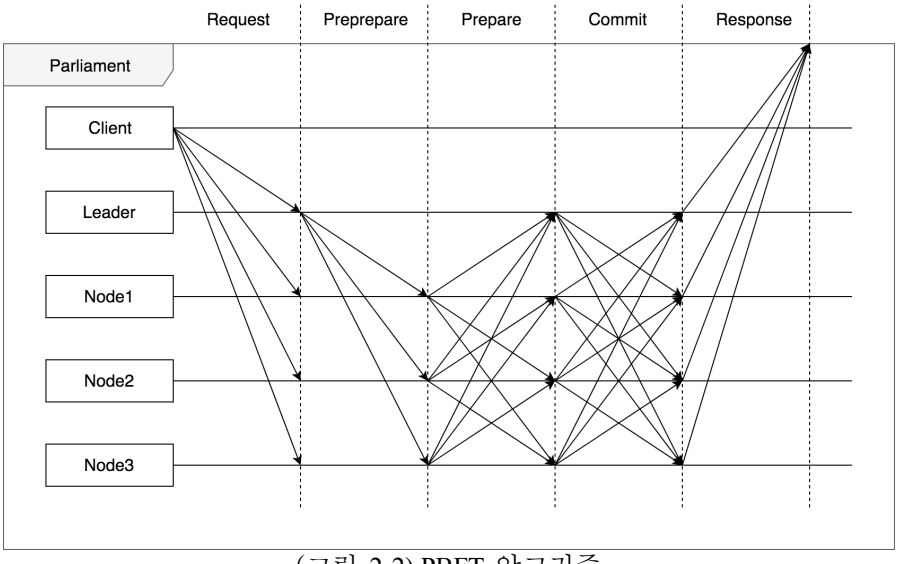
먼저, 중앙에서 모든 자원을 관리하다 보니 중앙에서 데이터를 변조시킬 수 있다는 문제가 있다. 서버 관리자가 악의적인 목적을 가지고 데이터를 변조시킨다면 이용자들이 그것에 대해 시스템적으로 대응할 방법이 없는 것이다. 블록체인에서는 이에 대해 해결책을 합의과정을 통해 제시한다. 합의과정이란 네트워크 참여자들의 합의를 얻어서 트랜잭션을 진행하는 방식으로, 블록체인이 P2P 네트워크이기 때문에 일정 이상의 참여자의 동의를 얻어야 한다. 합의과정은 비잔틴 장군 문제[3]에서 기안된 것이다. 비잔틴 장군 문제란 P2P 네트워크 상에서의 데이터 무결성 유지에 대해 발생할 수 있는 문제에 대해 설명한 것으로, 비잔틴 장군이 성을 포위한 다른 모든 장군들과 함께 동시에 공격을 진행해야 하는데, 다른 장군들에게 한 명씩 릴레이로 공격 시간을 전달하였을 때 중간에 배신자가 있어 메시지가 제대로 전달되지 않을 경우 성 공격이 실패할 수 있는 상황을 말한다. 이것이 P2P 네트워크에 비유되어 각 장군들이 하나의 네트워크를 이루고, 공격 시간 메시지가 네트워크에서의 데이터를 말하게 된다.



(그림 1-2) 비잔틴 장군 문제

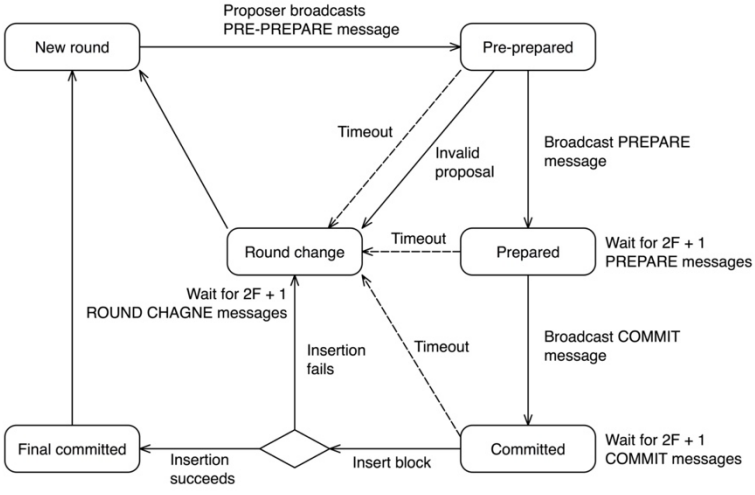
합의 과정은 보통 과반수의 참여자의 동의를 얻는 방식으로 진행된다. 제일 먼저 등장한 블록체인 시스템인 비트코인의 경우, 작업증명(Proof of Work, PoW)이라는 방식의 합의과정을 이용한다. 작업증명이란 컴퓨팅 파워를 이용

하여 다음 블록의 해시를 알아내고, 그 보상을 받는 방식을 말한다. 현재 대부분의 퍼블릭 블록체인에서 이용하고 있는 방식이다. 비트코인의 경우 해시에 SHA256 해시함수를 이용하고 있는데, 이는 현재의 컴퓨터 기술로 뚫는 것은 불가능하다고 할 수 있다. 만약 양자 컴퓨터가 등장할 경우 연산속도가 현재 컴퓨터에 비해 지수승으로 향상되기 때문에 뚫릴 수 있으나, 지금은 그 정도의 연산 속도를 낼 수 없기 때문에 다량의 GPU를 병렬로 묶어 작업장을 만들거나, ASIC과 같은 반도체를 직접 주문 제작하는 방식을 통해 작업증명에서의 이점을 꾀할 수 있다. 이것이 일종의 치팅 방식으로 생각되어 ASIC 저항 알고리즘을 적용한 블록체인들도 많이 있다. 이렇게 컴퓨팅 파워를 이용하기 때문에 환경 오염 문제가 심각하며, 이에 대한 해결을 위해 네트워크 상에서 가지고 있는 지분을 바탕으로 보상을 지급해주는 지분증명(Proof of Stake, PoS), 지분증명에서 투표를 바탕으로 진행되는 위임지분증명(Delegated Proof of Stake, DPoS) 등 퍼블릭 네트워크 상에서 컴퓨팅 파워에 의존하지 않는 다양한 합의 알고리즘들이 존재하나 작업증명만큼 P2P 상황에서 무결성을 보장해주지는 않기 때문에 활발하게 쓰이지는 않는 실정이다.



(그림 2-2) PBFT 알고리즘

프라이빗 네트워크의 경우 기본적으로 신뢰관계가 형성된 참여자들만이 네트워크에 존재한다는 가정을 하기 때문에 참여자 수에 좀더 구애 받지 않는 알고리즘이 적용 가능하다. 대표적인 프라이빗 네트워크에서의 합의 알고리즘으로는 PBFT(Practical Byzantium Fault Tolorence)가 있다. PBFT는 P2P 네트워크를 이용한 클라이언트 시스템을 위해 고안된 것으로, 클라이언트의 요청에 대해 리더가 각 노드들에게 메시지를 전파하고, 각 노드들이 서로 수신이 완료되었다는 메시지를 전파하고, 최종적으로 클라이언트에게 전달된 메시지의 개수가 일정 이상이 되면 커밋이 되었다고 판단하는 것이다. PBFT에서는 네트워크 내에 악의적인 노드(비잔틴 노드)가 있다고 가정한 상태에서 시작하기 때문에, 비잔틴 노드의 개수가 F개일 때 전체 노드의 개수가 3F + 1개가 되어야 하며, 이에 대해 네트워크가 정상적으로 돌아가는 것을 수학적으로 보장한다.

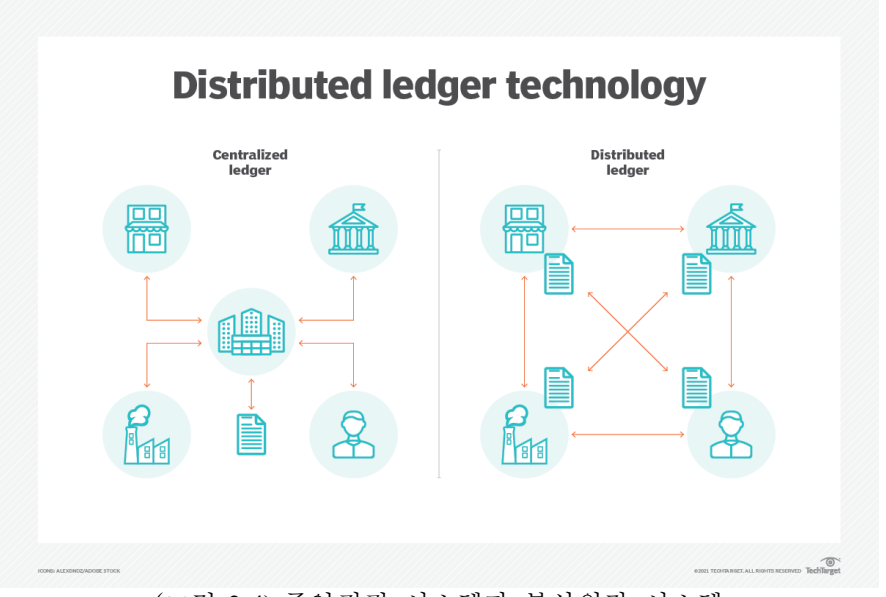


(그림 2-3) IBFT 알고리즘

PBFT는 클라이언트가 따로 존재하는 상황에서 트랜잭션을 발생시키는 것으로, 블록체인 환경과 같이 모두가 클라이언트가 될 수 있는 상황에서는 적합하지 않다. 이에 대한 해결책으로 등장한 방식이 IBFT(Istanbul Byzantine

Fault Tolorence)이다. IBFT에서는 모든 노드가 검증자 노드가 될 수 있다. 주로 라운드 로빈 방식을 통해 돌아가면서 제안자가 되며, 트랜잭션을 제안하여 각 검증자들의 검증을 받아 진행되기에 따로 클라이언트가 필요하지 않다. IBFT 알고리즘은 이더리움의 자바 기반 클라이언트인 하이퍼레저 베수에 적용되어 있다.

PBFT와 IBFT는 BFT 기반의 알고리즘이며, 이것은 복잡한 인증과정을 거침으로 퍼블릭 네트워크에는 적합하지 않다. 이후 서술할 하이퍼레저 패브릭 블록체인의 경우 다량의 데이터 처리에 적합한 CFT(Crash Fault Tolorence) 기반의 알고리즘인 Kafka와 Raft를 이용한다.



(그림 2-4) 중앙집권 시스템과 분산원장 시스템

이렇듯 중앙 집권 시스템은 중앙에서 임의로 데이터를 변조시키거나 해커에 의해 중앙이 공격당하면 사용자들이 시스템적으로 할 수 있는 것이 없다는 문제가 있다. 블록체인을 이용하면 모든 노드들이 원장을 가지고 있기에 중앙에서 데이터가 유실되더라도 복구 및 무결성 유지가 가능하다. 또한 블록체인을 이용할 경우 중앙 서버를 따로 유지할 필요가 없기 때문에 공간 및 비용의

절약 또한 가능하다. 이러한 점을 들어 블록체인은 블록체인이 시작된 암호화폐 외에도 여러 시스템에서 각광받고 있다. 중앙에서 데이터를 관리받지 않는다는 점을 들어 이를 이용한 새로운 시장까지 형성되고 있으며, 이는 대부분 이더리움으로부터 시작되었다. 이더리움에 의해 스마트 컨트랙트가 실현되었으며, ERC-20을 기반으로 한 다양한 토큰이 운용되고 있고 ERC-20 이외에도 ERC-721에 기반한 NFT 등 다양한 탈중앙화 시스템이 이더리움 플랫폼 상에서 운용되고 있다.

이더리움은 2015년 비탈릭 부테린에 의해 개발되었다. 플랫폼 전체를 뜻하면서 플랫폼 내부에서 사용되는 화폐를 일컫기도 한다. 화폐의 단위는 웨이(wei), 이더(ether)가 있으며, 1이더는 1,000,000,000,000,000,000웨이이다. 웨이의 경우 1,000웨이는 Kwei, 1,000,000웨이는 Mwei, 1,000,000,000웨이는 Gwei라 불리며 이 Gwei가 제일 많이 사용된다.

이더리움 플랫폼은 현재 전세계에서 가장 큰 블록체인 플랫폼이다. 이더리움 플랫폼에서는 블록체인을 이용해 실행 과정 및 결과를 무결성을 유지하며 보존 가능한 스마트 컨트랙트를 실행시킬 수 있으며, 이 스마트 컨트랙트는 solidity라는 자체 언어로 개발되어 EVM(Ethereum Virtual Machine) 위에 바이트코드로 인코딩 된 후 탑재되어 실행된다. 이더리움 플랫폼에서는 송금을 비롯한 모든 트랜잭션들이 성공적으로 실행된다면 블록에 들어간 후 체인에 추가되는데, 작업증명 방식을 이용하기에 블록이 채굴되어야 생성이 가능해지고, 그래서 이 채굴자들에게 돌아갈 보상을 가스(gas)라고 하여 일종의 트랜잭션 수수료 형태로 추가적으로 부과된다. 가스가 높은 트랜잭션부터 순차적으로 처리되며, 이것이 이더리움 플랫폼이 자체적으로 돌아갈 수 있는 원동력이 된다. 스마트 컨트랙트 또한 트랜잭션이기 때문에, 코드의 양에 따라 실행에 필요한 가스의 양이 달라진다.

이더리움 스마트 컨트랙트는 기본적으로 EVM이라는 가상머신 위에 바이트코드로 들어가며, JSON-RPC를 이용하여 데이터 통신을 주고받기 때문에 이를 실행시키기 위한 클라이언트 프로그램이 필요하며, 대표적으로 web3js, 트러플, go-ethereum 등이 있다.

블록체인은 기본적으로 모든 노드가 원장을 공유하기 때문에 비록 익명이기는 해도 모두에게 데이터가 노출된다는 단점이 있다. 그래서 산업 기반에는 적용하기 어려웠으나, 최근 영지식 증명을 이용하여 블록체인 기반 인증을 실현하고 있다. 영지식 증명이란 인증을 위해 인증 정보 전체를 전달하는 것이 아닌, 인증 정보를 알고 있다는 사실만을 전달하는 것으로 DID 등에서도 이를 이용한 인증 방식이 널리 퍼지고 있다.

제 2 절 하이퍼레저 프로젝트

하이퍼레저 프로젝트는 리눅스 재단에서 시작되었으며, 블록체인의 무결성 등의 장점들을 가져와 현재 운용되는 서버 기술에 적용하기 위한 프로젝트이다. 다양한 기업들이 참여하여 다양한 프레임워크 및 툴을 하이퍼레저 이름으로 운용하고 있다.

(그림 2-5) 하이퍼레저 프로젝트

대표적인 하이퍼레저 프로젝트 프레임워크로는 하이퍼레저 패브릭, 하이퍼레저 인디 등이 있다. 하이퍼레저 패브릭은 이후 설명할 프라이빗 블록체인 기반 프레임워크이며, 하이퍼레저 인디는 인증 중심의 프레임워크이다. 그 외에도 하이퍼레저 패브릭에서 이더리움과의 호환을 가능케 해주는 하이퍼레저 버로우, 자바 기반에 IBFT 합의 알고리즘을 이용하는 하이퍼레저 베수 등의 프레임워크가 있으며 툴로는 하이퍼레저를 비롯한 여러 블록체인의 성능 측정

을 위해 쓰이는 하이퍼레저 캘리퍼 등이 있다.

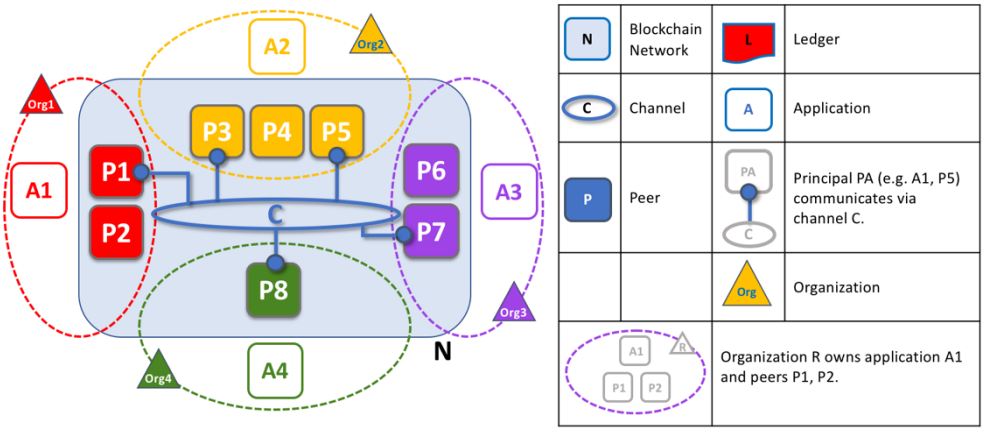
본 논문에서는 하이퍼레저 패브릭을 쉽게 생성하는 제너레이터를 구현하고자 한다.

제 3 절 하이퍼레저 패브릭

하이퍼레저 패브릭(Hyperledger Fabric)은 리눅스의 하이퍼레저 프로젝트에서 시작된 것으로, 프라이빗 블록체인을 활용한 엔터프라이즈 백엔드 프레임워크이다. 기존의 서버 기술을 블록체인을 통해 대체할 수 있다. 다양한 조직이 하나의 컨소시움을 구성하여 네트워크에 참여할 수 있으며, 한 네트워크 내에서도 채널을 다양하게 나눠 접근 권한을 다양하게 설정할 수 있다.

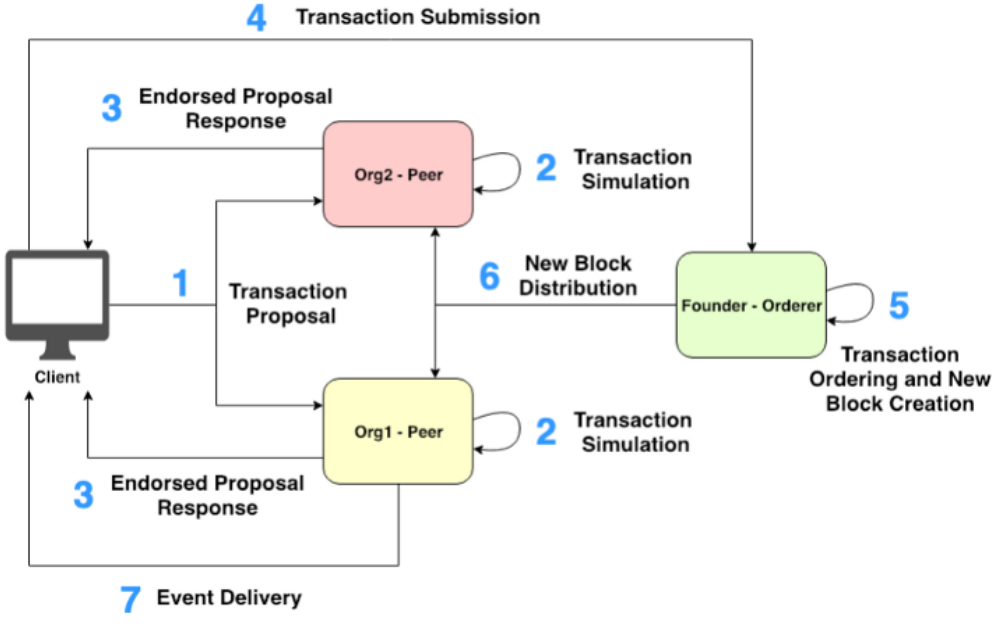
하이퍼레저 패브릭은 IBM에서 개발을 주도하고 있으며, 현재 2.4버전까지 개발되었다. 0.x, 1.x와 2.x 버전에서 많은 차이가 있었으며 1.x 버전은 1.4, 2.x 버전은 2.2가 LTS이다. 본 논문에서는 2.2 LTS 버전을 이용하여 개발하였다.

하이퍼레저 패브릭은 기존의 블록체인과 달리 피어 노드만으로 구성되지 않는다. 각 조직들에 피어가 달려 있는 형태로 되어 있으며, 피어마다 채널 설정이 가능하다. 각 채널마다 오더링 노드를 담당하는 오더러가 존재하며, 블록 생성 및 피어로의 전파를 담당한다.



(그림 2-6) 하이퍼레저 패브릭 구성

각 피어마다 원장이 달려 있으며, 타 채널의 데이터에는 접근할 수 없다. 이것을 이용해서 관리자만이 접근할 수 있는 채널을 만드는 등의 작업을 할 수 있다. 본 논문에서는 역할별로 채널을 나눈 뒤 invoke 어플리케이션에서 권한에 따라 호출하여 이용하는 방식으로 진행하였다.



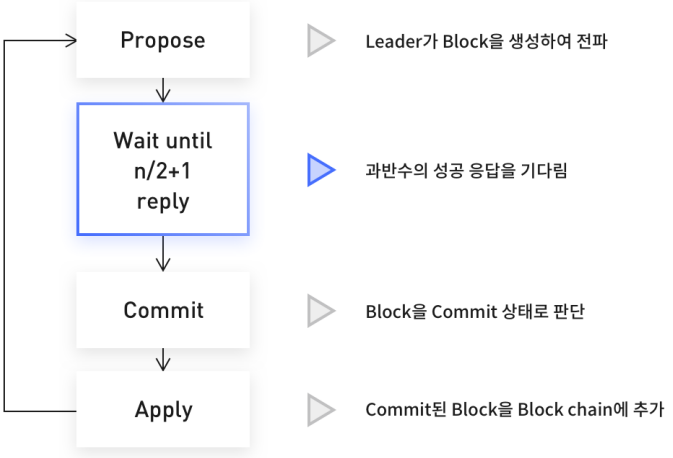
(그림 2-7) 하이퍼레저 패브릭 트랜잭션 흐름도

하이퍼레저 패브릭에서 트랜잭션이 발생하는 흐름은 다음과 같다.

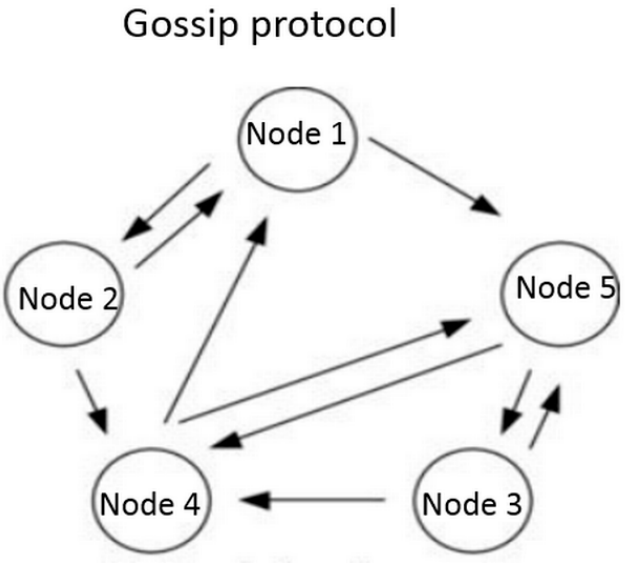
1) 클라이언트가 트랜잭션을 제안한다.
2) 피어들은 그 트랜잭션을 시뮬레이팅 한 후 R/W 세트를 생성한다.
3) 클라이언트는 피어로부터 R/W 세트를 기반으로 보증 응답을 받는다.
4) 클라이언트는 오더러에게 그걸 바탕으로 트랜잭션을 요청한다.
5) 오더러는 트랜잭션 실행 후 새 블록을 생성한다.
6) 오더러는 피어들에게 블록을 전파한다.
7) 피어는 클라이언트에게 그 결과를 전달한다.

위와 같은 과정을 통해 하이퍼레저 패브릭에서의 트랜잭션이 진행된다. 본 논문에서 사용된 하이퍼레저 2.2 에서는 Raft 합의 알고리즘을 통해 블록 생성이 결정되고 진행된다. Raft 합의 알고리즘에서는 IBFT 와 비슷한 구성을

가지며, 리더와 팔로워, 후보가 존재한다. 팔로워들 중에서 후보가 나타나고, 투표를 통해 후보들 중 리더가 선출된다. 리더는 블록을 생성하여 전파하고, 과반수 팔로워의 성공 응답을 기다린다. 응답이 오게 될 경우 블록을 커밋 상태로 판단하고 체인에 해당 블록을 추가하게 된다.



(그림 2-8) Raft 합의 알고리즘



(그림 2-9) gossip 프로토콜

하이퍼레저 패브릭에서 오더러가 피어에게 데이터를 전파하는 과정에서는 gossip 프로토콜이 사용된다. gossip 프로토콜이란 여러 노드들이 서로 데이터를 동기화 하기 위해 사용되는 방식으로, 소문이 나는 과정과 유사한 방식으로 동작한다. 각 노드는 자신에게 연결된 노드들과 끊임없이 통신을 주고받으며 서로에게 모자란 데이터를 채워주거나 잘못된 데이터를 바로잡는다. 이 방식을 통해 오더러는 모든 피어에게 데이터를 줄 필요 없이, 앵커 피어라고 불리는 대표 피어에게만 블록 생성을 전파하고, 그것을 gossip 프로토콜을 통해 서로에게 전달하게 된다.

하이퍼레저 패브릭에는 체인코드[4]라고 하는 스마트 컨트랙트가 존재한다. 스마트 컨트랙트는 블록체인 중에서는 이더리움에서 처음 구현되었는데, 이더리움에서는 스마트 컨트랙트 실행 시 코드의 양에 따라 수수료가 필요하게 된다. 그렇기에 실행에 있어 부담이 갈 수 밖에 없고, 퍼블릭 블록체인인 이더리움 특성상 모든 실행결과를 모두가 공유하여 기업 비즈니스에는 적합하지 않다. 하이퍼레저 패브릭에서는 체인코드 실행에 별다른 수수료가 필요하지 않으며, 프라이빗 네트워크로 되어 있어 채널별로 데이터가 관리되어 기업 비즈니스에 매우 적합하다.

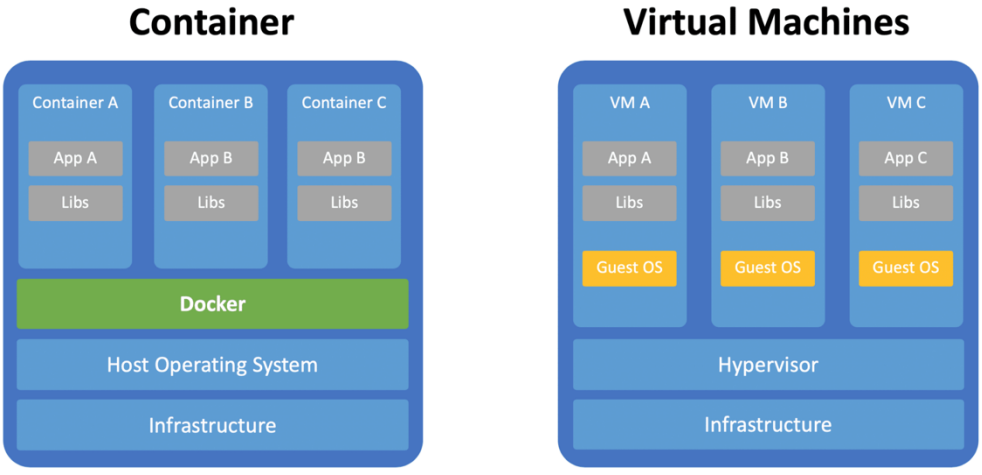
또한 체인코드 상에서 ERC-20 을 구현할 경우 이더리움 토큰도 적용시킬 수 있다. 체인코드 작성 언어로는 Go, Node.js, Java 가 지원된다.

하이퍼레저 패브릭은 위와 같은 방식을 통해 블록체인을 구성하고, 데이터의 무결성을 유지한다. 본 논문에서는 하이퍼레저 패브릭의 이러한 점을 이용하여 보안성이 뛰어난 프레임워크를 구현하였다.

제 4 절 도커

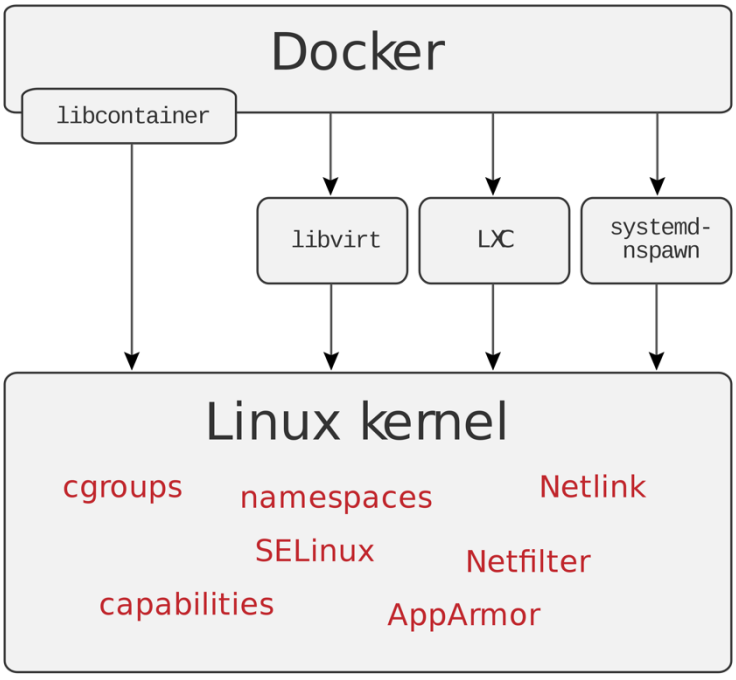
도커[5]는 응용 프로그램 혹은 실행환경을 프로세스 격리 기술을 통해 실행 및 관리하는 오픈소스 프로젝트이다. 2013 년 Solomon Hykes 에 의해 발표되었다. 컨테이너라는 단위를 통해 소프트웨어와 그 소프트웨어의 실행에 필요한 모든 것을 포함하는 파일 시스템 안에 감싼 뒤, 어디에서 실행하든지 동일한 실행을 보장한다.

컨테이너는 일종의 인터페이스로, 도커 실행환경에서는 모든 것을 컨테이너로 실행시키기 때문에 동일한 실행을 보장받을 수 있다. 컨테이너는 기본적으로 리눅스 커널과 OveraryFS, aufs 와 같은 리소스 격리 기능을 사용하기에, 이를 통해 독립적인 실행 환경을 보장하여 환경 구성 및 유지보수의 부담을 없애준다.



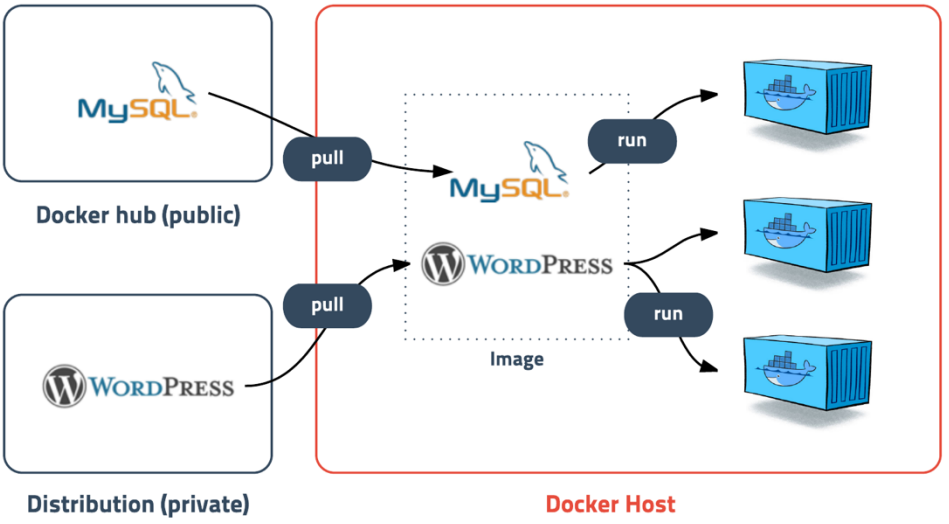
(그림 2-11) 도커와 가상환경의 비교

환경 전체를 하나의 컨테이너로써 가상화 시키기 때문에 여러 프로그램들의 배포가 매우 쉬워진다. 또한 기존의 VMWare, VirtualBox 등에 비해 매우 가벼운 가상화 기술을 사용하기 때문에 다양한 프로세스를 실행시키기에 오버헤드가 적으며, 테스트 및 실제 서비스 배포에도 적합하다. 최근 DevOps 가 떠오르면서 이는 매우 큰 장점으로 부각되고 있다.



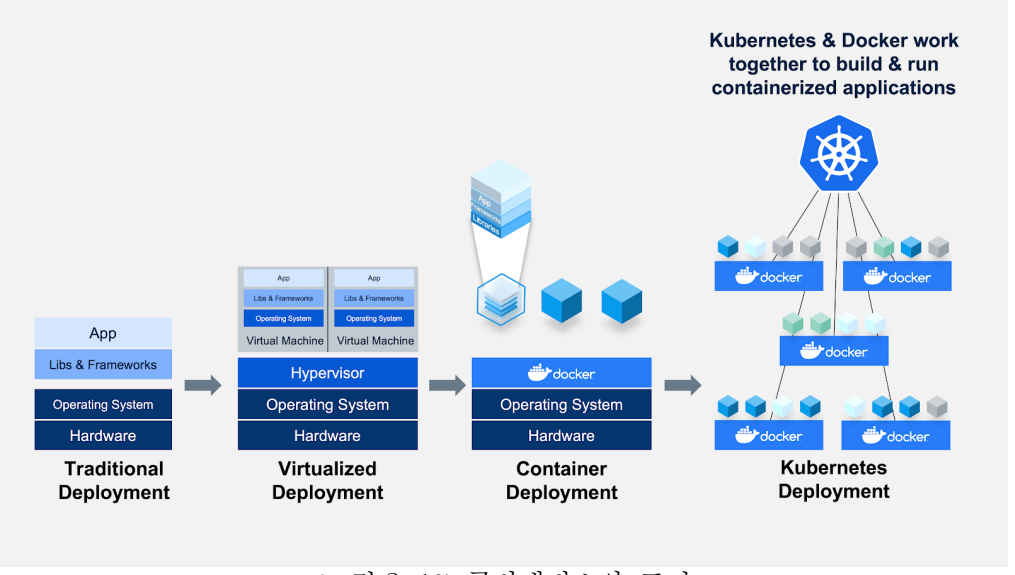
(그림 2-12) Docker

도커 컨테이너는 도커 이미지로부터 만들어진다. 이미지는 실행하고자 하는 컨테이너에 필요한 파일과 설정값을 모두 포함하고 있다. 이러한 이미지를 통해 컨테이너를 만들어 실행에 이용할 수 있다. 이미지 하나로부터 여러 컨테이너를 만들어낼 수 있다. 가령 여러 대의 서버를 통해 통신을 주고받는 환경을 테스트 해보고 싶다면, 하나의 서버를 이미지로 만든 뒤 그 이미지를 통해 여러 컨테이너를 만들어 실행하면 된다. 여러 컨테이너를 만든다고 해서 서로 종속성이 있지는 않기 때문에 격리 기술을 적극 활용할 수 있는 것이다.



(그림 2-13) 도커 이미지

최근에는 쿠버네티스와 연계하여 도커의 활용성이 더욱 높아지고 있다. 쿠버네티스란 구글에 의해 개발되었고 내부적으로 사용되고 있던 컨테이너화된 서비스의 관리를 위한 확장 가능한 오픈소스 플랫폼으로, 대규모 서비스 배포에 주로 이용되었다. 다양한 서비스 및 환경을 컨테이너화 한 뒤 쿠버네티스를 통해 오케스트레이션 하는 것이다. 그리고 이 때 도커가 이용되는 것이다. 도커로 컨테이너화 된 환경이 쿠버네티스를 통해 배포될 수 있다. 이는 DevOps, MLOps 등이 활성화 되는, 개발과 운영을 구분 짓지 않는 현대의 개발 환경에 매우 적합하다.



(그림 2-13) 쿠버네티스와 도커

이처럼 도커는 강력한 가상화 및 격리 기술을 바탕으로 개발자에게 개발에만 집중할 수 있는 환경을 만들어준다. 도커의 사용법을 배워야 한다는 부가적인 문제는 생길 수 있으나 그것으로 인해 얻는 생산성을 생각하면 훨씬 좋은 것이다. 비록 도커에도 리눅스 커널에 종속되어 있다거나 각 컨테이너를 VM 처럼 직접 세세한 설정을 해주지는 못한다는 단점은 존재하지만, 그럼에도 불구하고 도커는 매우 강력하며, 많은 문제를 해결해주는 것은 확실하다. 앞으로 더 좋은 가상화 및 격리 기술이 등장할 수 있겠으나, 현재로써 도커는 테스트 및 배포에 매우 좋은 기술이다.

본 논문에서는 블록체인 시스템 구성에 도커를 사용한다. 하이퍼레저 패브릭에는 여러 구성요소가 있다. 피어, 오더러, CA, DB 등 다양한 구성요소가 있으며 이들이 각자 하나씩의 서버를 필요로 한다. 하이퍼레저 패브릭에서는 이들 구성요소의 이미지를 제공하며, 이것을 이용해 각 구성요소에 해당하는 컨테이너를 생성할 수 있다. 가령 피어 2 개를 만들고 싶다고 하면 피어 이미지를 이용해서 다른 설정값을 가진 컨테이너 2 개를 만들면 되는 것이다. 본 논문에서는 도커를 이용하여 로컬 환경에서

블록체인 네트워크를 테스트 하였다. 각 피어, 채널별 오더러, CA, CouchDB 를 도커 이미지를 통해 컨테이너화 하였다. 또한 체인코드 설치 시 체인코드 또한 컨테이너로 되어 피어 위에서 동작한다.

제 3 장 연구 내용

제 1 절 HLF-generator

하이퍼레저 패브릭은 초보자에게 네트워크를 구성하기가 굉장히 어렵다. 필요한 기술이 많고, 모든 것이 명령어 기반이며 특히 예제가 모두 도커를 이용하는 것으로 되어 있기 때문에 도커로 네트워크를 구성하는 방법조차 익혀야 한다. 또한 네트워크가 복잡해질 수록 이러한 설정은 숙련자라고 하더라도 귀찮고 생산성이 떨어지는 작업이 된다.

특히 본 논문에서 개발하고자 하는 네트워크는 다양한 채널, 조직 등을 사용하여 네트워크 복잡도가 매우 높기에, 이에 대한 자동화가 필수불가결해졌다. 하이퍼레저 패브릭 예제인 fabric-samples는 도커 설정파일로 docker-compose를 이용하는데, 설정해야 할 사항들이 매우 많다. 도커를 이용하는 이유 자체가 테스트의 편리함인데, 이것은 오히려 도커를 이용하는 의미를 저하시킨다. 또한 자동화 시스템이 존재할 경우 향후 연구에도 매우 큰 영향을 미칠 수 있을 것으로 판단하였기에 자동화 시스템 개발을 우선으로 실시하였다.

자동화 시스템의 이름은 HLF-generator로 명명하였으며, 파이썬 3.7을 통해 개발되었다. HLF-generator는 사용자로부터 네트워크의 설정에 관한 정보들을 입력받아 네트워크를 시작할 수 있는 스크립트를 생성한다. 사용자는 이 스크립트를 단순히 실행함으로써 원하는 네트워크를 실행할 수 있다.

```
.
├── src
|   ├── config.py
|   ├── configtx.py
|   ├── create_channel.py
```

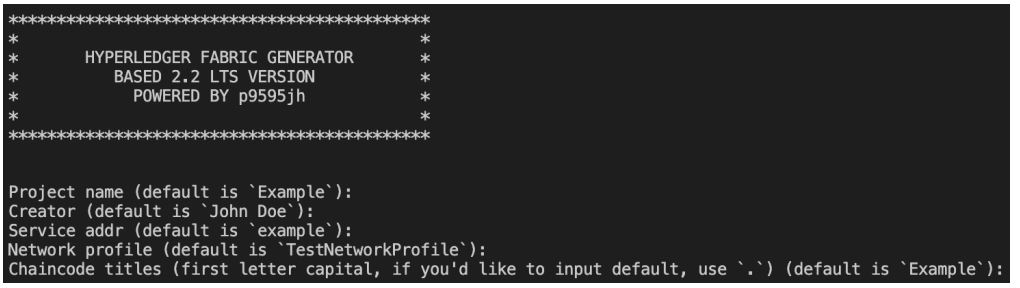
```
|   ├── deploy.py
|   ├── docker.py
|   ├── load.py
|   ├── main.py
|   ├── node.py
|   └── util.py
└── templates
    ├── chaincode
    |   └── codes
    |       └── template_cc
    |           ├── package.json
    |           ├── src
    |           |   ├── TEMPLATE_CC.ts
    |           |   ├── index.ts
    |           |   └── utils.ts
    |           ├── tsconfig.json
    |           └── tslint.json
    ├── invoke
    |   ├── package.json
    |   ├── run.sh
    |   ├── src
    |   |   ├── enrollAdmin.ts
    |   |   ├── invoke.ts
    |   |   ├── registerUser.ts
    |   |   ├── server.ts
    |   |   └── utils.ts
    |   ├── tsconfig.json
    |   └── tslint.json
```

```
└── network
    ├── cc
    │   ├── ccfs.sh
    │   └── deploy.sh
    ├── configtx
    │   ├── configtx.yaml.txt
    │   └── core.yaml
    ├── docker
    │   ├── docker-compose-ca.yaml
    │   ├── docker-compose-couch.yaml
    │   └── docker-compose-net.yaml
    ├── invoke.sh
    ├── network.sh
    ├── organizations
    │   ├── ccp-generate.sh
    │   ├── ccp-template.json
    │   ├── ccp-template.yaml
    │   └── fabric-ca
    │       ├── fabric-ca-server-config.yaml
    │       └── registerEnroll.sh
    ├── re.sh
    └── scripts
        ├── createChannel.sh
        ├── envVar.sh
        └── utils.sh
```

(표 3-1) HLF-generator 파일 구성

프로그램 실행 시 프로그램의 간략한 정보가 출력되며, 처음에는 프로젝트 명, 프로젝트 개발자명, 서비스 주소명, 네트워크 프로필, 체인코드 제목들을

입력받는다. 각 입력 필드는 기본값을 가질 수 있으며, 기본값이 설정되어 있을 경우 값 미입력 시 기본값이 들어가게 된다. 프로젝트명의 기본값은 Example이며 프로젝트 개발자의 기본값은 John Doe, 서비스 주소명의 기본값은 example, 네트워크 프로필의 기본값은 TestNetworkProfile, 체인코드 제목들 기본값은 Example이다.



(그림 3-1) HLF-generator 기본 입력 필드

서비스 주소명은 프로그램 내에서 주소를 표현할 때 사용된다. 만약 서비스 주소명을 example이라고 하였다면 오더러 주소는 orderer.example.com이 된다. 체인코드 제목들은 복수 입력 필드인데, 체인코드가 프로젝트 내에 여러 개가 될 수 있기 때문이다. 만약 여러 개를 입력할 경우 '.'을 입력하면 기본 체인코드명이 입력된다. 기본 체인코드명은 프로젝트명과 동일하다. 복수 입력 필드의 구분자는 설정될 수 있으며, 체인코드명 설정에서는 스페이스가 사용된다.

그 뒤로는 조직의 개수를 입력 받게 된다. 기본값은 1이며, 사용자가 원하는 조직의 개수를 입력할 수 있다.

조직의 개수를 입력한 뒤에는 그 개수만큼 루프를 돌리며 각 조직에 대한 설정을 받게 된다. 각 조직은 addr, name, msp, ca_port, admin, adminpw, 피어 개수를 입력받게 된다. addr은 그 조직이 주소 내에서 표현될 문자열, name은 그 조직의 이름, msp는 그 조직의 MSP명, ca는 그 조직의 도커 컨테이너의 CA 포트번호, admin은 그 조직의 어드민명, adminpw는 그 조직의 어드민 비밀번호, 피어 개수는 그 조직 내에 속해있는 피어의 수이다.

```
Number of organizations (default is `1`):

====================================

[Organization 1/1]
addr (default is `org1`):
name (default is `Org1`):
MSP (default is `Org1MSP`):
CA port (default is `7054`):
admin (default is `admin`):
admin password (default is `adminpw`):
Number of peers of Org1 (default is `1`):

[Peer 1/1 of Org1]
peer name (default is `peer0`):
peer port (default is `7051`):
peer DB port (default is `5984`):
input all channels of the peer seperates with space (ex. `channel1 channel2`) (default is `mychannel`):

[Channel mychannel (1/1) of peer0.org1.example.com]
channel profile of mychannel (default is `MyChannelProfile`):
consortium of mychannel (default is `Consortium`):

====================================

[Orderer]
orderer MSP (default is `OrdererMSP`):
orderer CA port (default is `8054`):
input all items of the orderer seperates with space (ex. `orderer1 orderer2`) (default is `orderer`):

name of orderer (default is `Orderer`):
port of orderer (default is `7050`):
```

(그림 3-2) HLF-generator

피어의 개수를 입력한 후에는 그 수만큼 루프를 돌리며 각 피어의 설정에 대해 입력받기 시작한다. 피어는 name, port, db_port, 채널들로 이루어져 있다. name은 피어명, port는 이 피어 컨테이너의 포트번호, db_port는 이 피어가 가진 원장(couchdb)의 컨테이너 포트번호, 채널들은 이 피어가 속하는 채널들에 관한 정보이다. 하이퍼레저 패브릭에서 한 피어는 여러 채널에 속할 수 있기 때문에 채널은 복수 입력 필드를 통해 입력받게 하였다.

채널은 name, profile, consortium으로 구성되어 있다. name은 채널명을 나타내며, 피어에서 채널 정보를 입력할 때 입력받는 값을 통해 설정하고, profile은 해당 채널의 채널 프로필을 나타내며, consortium은 해당 채널의 컨소시움명을 나타낸다. 채널 정보는 한 번 입력되면 나중에 같은 이름의 채널이 입력되었을 경우 입력되었던 정보를 그대로 불러오게 된다.

이후 오더러 정보를 입력받게 된다. 오더러는 msp, ca_port, items를입력받는다. msp는 해당 오더러의 MSP명이며, ca_port는 해당 오더러의 CA의 컨테이너 포트 번호, name은 오더러를 구성하고 있는 아이템들이다. 오더러는 여러 아이템으로 구성될 수 있으며, 이 아이템은 name, port 필드로 구성된다.

HLF-generator 사용자는 이 제너레이터를 통해 더욱 양질의 프로젝트를 매우 쉬운 방법으로 만들 수 있다.

제 2 절 시스템 동작 과정

HLF-generator는 2가지 동작 모드를 가진다. 첫번째는 맨 처음부터 실행하는 것으로 피어, 오더러 등의 설정을 터미널을 통해 사용자가 입력하는 것이다. 두번째는 history를 통해 생성하는 것으로, 사용자가 입력한 방식에 대해 항상 history가 JSON과 YAML로 생성되는데, 이것을 load를 통해 실행시키는 것이다.

사용자가 입력한 항목들을 저장해서 파일로 만든 것이며, load 실행 시 이 파일로부터 설정을 가져오기 때문에 이 파일을 수정한 뒤 호출하면 해당 설정에 맞는 네트워크를 구축할 수 있다.

설정된 값들을 templates 폴더 아래의 템플릿에 입력한다. 그렇게 생성된 프로젝트는 exports 폴더 내에 사용자가 설정한 프로젝트명으로 설정된다. 새로운 프로젝트가 만들어지면 무조건 history에 저장이 되며, 이것은 상기하였듯 JSON과 YAML 파일로 저장된다.

제 3 절 파일 구성

HLF-generator는 다수의 파이썬 파일로 이루어져 있다. 전체 파일 목록은 다음과 같다.

- config.py
- configtx.py
- create_channel.py
- deploy.py
- docker.py
- load.py

- main.py
- node.py
- util.py

각 파일에 대해 서술한다.

제 3.1 절 config.py

config.py는 설정에 관련한 내용을 저장한다. 그렇기에 여러 전역변수들로 구성되어 있으며, 기본값들을 설정하고 있다. 가지고 있는 기본값은 오더러 포트, 피어 포트, CA 포트, couchdb 포트, 포트 증가값, 체인코드명, 템플릿 위치, 기본 채널명, 채널 프로필명, 그리고 프로젝트명, 프로젝트 생성자명, 서비스 주소명, 컨소시움명, 네트워크 프로필명이 들어 있다.

config.py 파일 수정을 통해 기본값들을 변경할 수 있다.

제 3.2 절 configtx.py

configtx.py 파일은 하이퍼레저 패브릭의 configtx.yaml을 생성하기 위한 파일이다. 하이퍼레저 패브릭에서는 네트워크를 생성할 때 configtx.yaml을 참고하여 생성하는데, 이 파일의 구조는 사람도 쉽게 변경을 하여야 한다고 판단하여 단순히 파이썬의 docker 라이브러리를 이용해 만드는 것이 아닌, 하이퍼레저 패브릭에서 제공하는 예제에서의 configtx.yaml과 동일한 형식을 가지도록 문자열을 정리하여 저장하게끔 하였다.

제 3.3 절 create_channel.py

create_channel.py 파일은 create_channel.sh 스크립트 파일을 구성한다. 하

이퍼레저 패브릭에서 채널을 생성할 때 정해진 절차의 명령어들을 수행하여 채널을 생성하고 피어를 조인시킨다. 이 과정을 쉘 스크립트로 만들어놓은 게 create_channel.sh로, 이 파일을 동적으로 생성하게끔 하는 것이 create_channel.py이다.

제 3.4 절 deploy.py

create_channel.py와 비슷하게, 체인코드 설치 과정을 쉘 스크립트로 구성해 놓은 것이 deployCC.sh 쉘 스크립트 파일이다. 이것을 채널 정보에 따라 동적으로 생성하게 해주는 것이 deploy.py이다.

제 3.5 절 docker.py

본 논문에서의 하이퍼레저 패브릭 네트워크는 도커를 기반으로 하여 동작한다. 그 중에서도 docker-compose를 이용하며, 이에 따라 docker-compose.yaml 파일이 필요하다. 이 파일을 생성하기 위해 특정 docker-compose 양식을 갖고 있는 파일에 변수를 동적으로 할당하여 docker-compose 파일을 생성한다.

제 3.6 절 load.py

앞서 설명한 history를 불러와서 네트워크를 구축할 수 있게 만들어주는 파일이 load.py이다. load.py에서는 history json 혹은 yaml을 불러오며, main과 비슷한 로직을 통해 네트워크를 구축하게 된다. json과 yaml에 대해서는 불러오는 과정만이 다를 뿐 동일한 동작을 통해 진행하게 된다.

제 3.7 절 main.py

main.py에서는 프로그램 동작에 대해 모든 과정이 순차적으로 이루어진다. 사용자의 프로젝트에 관한 설정을 입력받는 것을 시작으로, 입력받은 정보들을 바탕으로 네트워크 필요 정보들을 통해 네트워크 구축에 필요한 파일들을 생성한다.

제 3.8 절 node.py

프로그램 동작에 필요한 모델들을 클래스로서 만들어놓은 것이 node.py이다. node.py에 정의된 모델들은 다음과 같다.

Field	Description
msp	MSP of the orderer
caport	CA port of the orderer
items	Orderer items array

(표 3-2) class Orderer

Field	Description
addr	Represented address value
name	Name of the orderer item
port	Port of the orderer item
orderer	Orderer object which includes this

(표 3-3) class OrdererItem

Field	Description
addr	Represented address value
name	Name of the organization
msp	MSP of the organization
caport	CA port of the organization
admin	Admin id of the organization
adminpw	Admin password of the organization
peers	Peers that the organization includes

(표 3-4) class Organization

Field	Description
org	Organization which includes this
name	Name of the peer
port	Port of the peer
dbport	couchdb port of the peer
channels	Channels that the peer includes

(표 3-5) class Peer

Field	Description
channel	Name of the channel
profile	Profile of the channel
consortium	Consortium of the channel

(표 3-6) class FabChannel

오더러는 오더러 아이템을 포함하는 형태로 되어 있다. 조직은 피어를 포함하는 형태로 되어 있으며, 피어는 채널을 포함하는 형태로 되어 있다.

위 구조의 모델들을 이용하여 프로그램을 진행한다.

제 3.9 절 util.py

util.py에서는 프로그램 실행 시 필요한 유틸들을 정리해놓았다. 예를 들어 sinput에서는 문자열과 정규식을 입력받아 해당 정규식에 유효하지 않을 경우 리턴하게 된다. ninput은 sinput에 정규식을 [0-9]+로 입력한 뒤 숫자로 변환한 함수이다. 이러한 식으로 프로그램에 대한 다양한 유틸을 util.py에서 확인할 수 있다.

제 4 장 평 가

제 1 절 평가

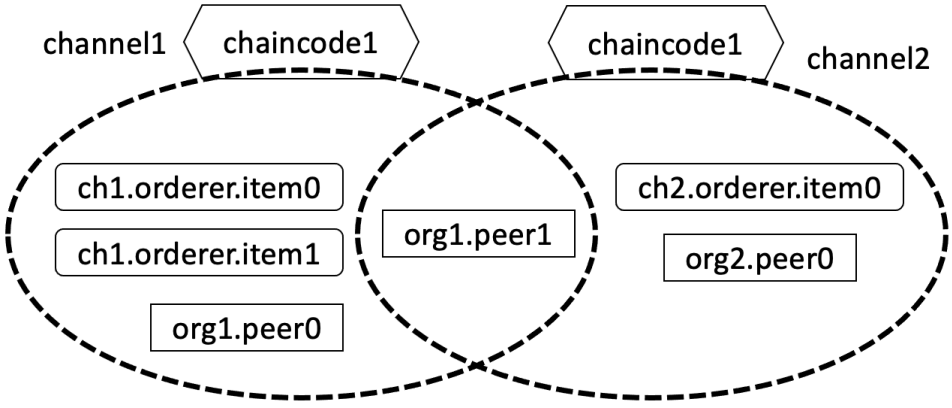
HLF-generator는 기존 방식과 같이 네트워크를 구성하는 것에 비해 매우 큰 이점을 보여준다. 기존 방식과 같이 네트워크를 구축하려면 수정해야 할 파일이 매우 많다. 명령어들을 필요에 맞게 나열해야 하고, 만약 이것을 스크립트로 묶어 놓은, fabric-samples와 같은 예제를 활용한다고 하더라도 생산성이 매우 떨어진다. 또한 코드의 유연성이 좋지 못하기 때문에 네트워크의 형태를 변경하고 싶더라도 절대 쉽지 않다.

HLF-generator는 코드 변경에 관한 아주 큰 유연성을 제공한다. 사용자가 단순히 변수를 입력하기만 하면 그에 맞는 프로젝트를 생성해준다. Node.js 기반의 체인코드 생성에 대한 기반도 만들어준다. 따라서 사용자가 네트워크를 구축하기 위해 신경을 쓸 부분이 매우 적어지며, 모든 부분이 모듈화로 엮여 있어 매우 유연한 코드 구조의 변경을 가능케 한다. 또한 쉘 스크립트에서도 어느정도 모듈화가 되어 있는데, utils.sh에서는 네트워크 구축 진행에 필요한 변수들을 배열로써 묶어 놓는다. 이것은 유지보수에 매우 좋게 된다. 네트워크 생성 후 변수값의 변경이 필요할 경우 utils.sh에서 값의 이름을 변경하면 된다.

HLF-generator를 이용하면 하이퍼레저 패브릭에 처음인 사용자도 네트워크 구성을 굉장히 쉽게 할 수 있다. 하이퍼레저 패브릭이 다른 블록체인 네트워크에 비해 개념이 전혀 직관적이지 않아 공식 문서를 참조하더라도 노드들 간의 관계를 정의하기가 쉽지 않은데, HLF-generator는 조직이 가진 피어의 개수를 묻고, 피어가 속할 채널명을 묻는 등 매우 직관적인 방식으로 되어 있다. 이를 통해 HLF-generator로 쉽게 도커 기반 하이퍼레저 패브릭 네트워크를 구성할 수 있다.

표 4-1은 기반 하이퍼레저 패브릭 프로젝트를 쉘 명령어를 통해 생성할

경우와 HLF-generator를 통해 생성하는 경우를 비교한다. 비교 시나리오는 그림 4-1과 같이 채널 2개(channel1, channel2), 조직 2개(org1, org2), 피어 3개(org1.peer0, org1.peer1, org2.peer0), 오더러 3개(ch1.orderer.item0, ch1.orderer.item1, ch2.orderer.item0)로 이루어져 있으며, org1.peer0은 channel1에, org2.peer0는 channel2에, 그리고 org1.peer1은 channel1과 channel2에 모두 속해있다. 또한 channel1에는 chaincode1, channel2에는 chaincode2 체인코드가 설치되어 있다. 쉘 명령어 사용 시에는 서버 인스턴스를 따로 두는 상황이다.



(그림 4-1) 평가 시나리오

HLF-generator	Shell Command
Install binary to the workspace	Install binary to org1.peer0
Run HLF-generator	Install binary to org1.peer1
Modify chaincode1	Install binary to org2.peer0
Modify chaincode2	Install binary to ch1.orderer.item0
Run auto.sh	Install binary to ch1.orderer.item1
Modify client program	Install binary to ch2.orderer.item0
	Register org1.peer0
	Register org1.peer1

	Register org2.peer0
	Register org1.peer0 user
	Register org1.peer1 user
	Register org2.peer0 user
	Register org1.peer0 admin
	Register org1.peer1 admin
	Register org2.peer0 admin
	Create channel1
	Create channel2
	Generate genesis block of channel1
	Generate genesis block of channel2
	Join org1.peer0 to channel1
	Join org1.peer1 to channel1
	Join org1.peer1 to channel2
	Join org2.peer0 to channel2
	Set anchor of org1 as org1.peer0
	Set anchor of org2 as org2.peer0
	Write chaincode1
	Write chaincode2
	Package chaincode1
	Package chaincode2
	Install chaincode1 to org1.peer0
	Install chaincode1 to org1.peer1
	Install chaincode2 to org2.peer0
	Approve for org1.peer0
	Approve for org1.peer1
	Approve for org2.peer0
	Commit definition to org1.peer0

	Commit definition to org1.peer1
	Commit definition to org2.peer0
	Write client program

(표 4-1) 단계 비교

쉘 명령어와 서버 인스턴스를 이용해 시나리오의 구조를 가지는 프로젝트 구축 및 클라이언트 프로그램 작성을 하려면 38단계가 필요했던 것에 비해 HLF-generator를 이용할 경우 6단계만에 동일한 작업을 진행할 수 있다. 각 서버 인스턴스를 직접 설정해야 했던 것과 달리 HLF-generator를 이용하면 여러 시나리오에 맞춰 자동으로 프로젝트를 생성해주며, 이후의 채널 생성이나 체인코드 배치와 같은 과정들도 생성한 시나리오에 맞춰 스크립트를 생성해주기에 auto.sh 스크립트 파일의 실행을 통해 이 전체 과정을 생략할 수 있다.

줄어든 단계를 수치적으로 나타낼 경우 기존 과정의 약 15.8%만 진행하여도 프로젝트를 생성할 수 있다. 이것은 단지 단계의 개수만을 비교한 결과이며, 실제 코드를 작성하는 체인코드, 클라이언트 프로그램의 경우 HLF-generator에서는 코드를 바로 작성할 수 있는 Node.js (TypeScript) 기반의 빈 프로젝트를 제공해주기에 훨씬 더 향상된 생산성을 가진다. 또한 배포에 있어서도 매우 큰 이점을 가진다.

제 5 장 결론 및 향후 연구

본 논문에서는 도커 기반의 하이퍼레저 패브릭 프로젝트를 쉽게 생성할 수 있게 해주는 HLF-generator를 구현하였다. 파이썬을 이용하여 구현하였으며, 하이퍼레저 패브릭에 처음인 사용자라도 별다른 어려움 없이 네트워크를 구축할 수 있게 해준다.

하이퍼레저 패브릭은 네트워크 구축이 어려운 것으로 많이 알려져 있는데, 특히 처음 접하는 사람의 경우 자신이 원하는 형태의 프로젝트를 구축할 때까지 굉장히 오랜 시간이 걸리게 된다. 또한 여러 네트워크 형태를 테스트 해보려는 연구원 혹은 개발자에게도 기존의 방식은 매우 불편하게 되어 있다. 본 논문에서의 HLF-generator를 통해 도커 기반 하이퍼레저 패브릭 네트워크 구축에 대한 생산성이 매우 향상되기를 기대한다.

향후에는 GUI를 통해 좀 더 친밀감 있고 접근하기 쉬운 제너레이터를 개발할 것이다. 향후 연구를 통해 도커 기반 하이퍼레저 패브릭 네트워크 구축에 더욱 큰 향상을 기대한다.

참 고 문 헌

1. 국외문헌

E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, M. Vukolić, S. W. Cocco, and J. Yellick, “Hyperledger fabric: a distributed operating system for permissioned blockchains,” in Proceedings of the thirteenth EuroSys conference, New York: NY, pp. 1-15, 2018.

N. Satoshi. "Bitcoin: A peer-to-peer electronic cash system." Decentralized Business Review (2008): 21260.

L. Larmport, R. Shostak, and M. Pease, “The Byzantine generals problem,” ACM Transactions on Programming Languages and Systems, vol. 4, no. 3, pp. 382-401 July. 1982.

B. Beckert, M. Herda, M. Kirsten, and J. Schiff. “Formal specification and verification of Hyperledger fabric chaincode,” in 3rd Symposium on Distributed Ledger Technology (SDLT-2018) co-located with ICFEM, pp. 44-48, 2018.

B. B. Rad, H. J. Bhatti, and M. Ahmadi, “An introduction to docker and analysis of its performance.” International Journal of Computer Science and Network Security, vol. 17, no. 3, pp. 228-235, March. 2017.

ABSTRACT

Hyperledger Fabric Generator based on Docker

Park, Jae-Hoon

Major in IT Convergence Engineering

Dept. of IT Convergence Engineering

The Graduate School

Hansung University

Hyperledger Fabric is a private blockchain open-sourced project which has been being led by IBM. Hyperledger is an enterprise blockchain project, and a lot of company participants to the project and support many tools or frameworks. Hyperledger Fabric is one the most live projects. It is the powerful framework that uses the strength of blockchain system, so it could be a great businesss blockchain system. So it is being used in the field, also be a good subject for researchers. Especially IBM provides several samples of docker image and network sample, it can be impelement

ed just in a local environment, which means it does not need large and h igh-performed hardwares. However, Hyperledger Fabric is difficult and co mplicated to struct network. In this paper, implementation of HLF-genera tor based on python can get low entry barriers and make high productivit y.

KEYWORD: Blockchain, Decentralized, Hyperledger Fabric