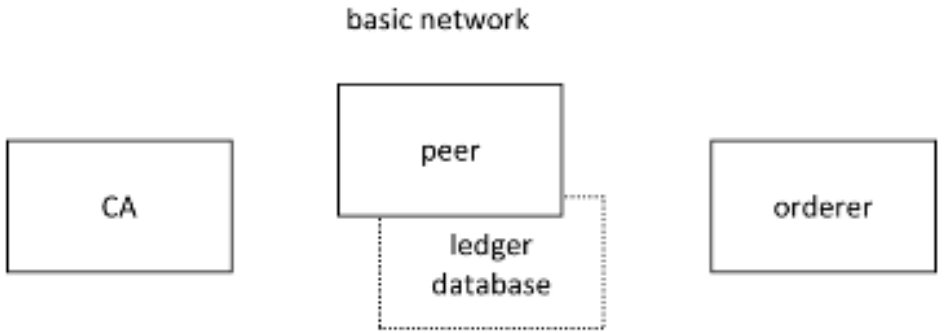
하이퍼레저 연구회
(Commercial paper Tutorial)

Made by 전부현

하이퍼레저 연구회
cafe.naver.com/hyperledgerstudy

Create network

✓ 현재의 기본 네트워크 사용. PaperNet의 다중 조직 구조를 잘 반영하는 구성으로 업데이트 예정. 현재 이 네트워크는 어플 및 스마트 컨트랙트 개발 방법 보여주기에 충분



✓ 기본 네트워크는 피어 및 원장 DB, 오더러, CA로 구성. 각 구성요소는 도커 컨테이너로 실행

✓ 프로덕션 환경에서 조직은 일반적으로 다른 시스템과 공유되는 기존 CA 사용. 그들은 Fabric 네트워크 전용이 아님

✓ “fabric-samples\basic-network” 디렉토리에 포함 된 명령 및 구성을 사용하여 기본 네트워크 관리. “start.sh” 스크립트 사용하여 로컬 컴퓨터에서 네트워크 시작: (실습)

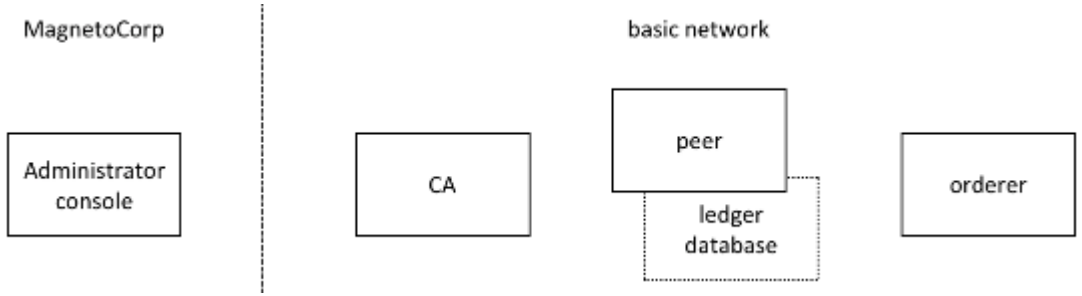
Create network

- ✓ “docker-compose -f docker-compose.yml up -d caexample.com …” 명령이 “DockerHub” 에서 네 개의 Fabric 컨테이너 이미지를 가져와 시작하는 방법에 유의.
- ✓ 이 컨테이너에는 이러한 Fabric 구성요소에 대한 S/W 최신버전이 있다.
- ✓ “basic-network” 디렉토리를 자유롭게 탐색 가능. 이 튜토리얼은 많은 내용을 사용한다.
- ✓ “docker ps” 명령을 사용하여 basic-network 구성요소를 실행하는 도커 컨테이너 나열 (실습)
- ✓ 이 컨테이너를 basic-network에 매핑할 수 있는지 확인
- ✓ 이 컨테이너는 모두 “net_basic” 이라는 “docker network” 를 형성함 “docker network” 명령으로 네트워크 볼 수 있음 (실습)
- ✓ 단일 도커 네트워크의 일부이자 네 개의 컨테이너가 서로 다른 IP 주소를 사용하는 방법 확인
- ✓ 요약하면 GitHub에서 HF 샘플 저장소를 다운로드 했고 로컬 컴퓨터에서 basic-network를 실행

Working as MagnetoCorp

- ✓ PaperNet의 MagnetoCorp 구성요소를 모니터링 하기위해 관리자는 “logspout” tool을 사용하여 컨데이터 세트에서 집계된 출력 볼 수 있다. 서로 다른 출력 스트림을 한 곳에 모아 단일 창에서 발생하는 상황을 쉽게 확인. 예를 들어 스마트 컨트랙트를 설치할 때 관리자나 스마트 컨트랙트 호출 시 개발자에게 도움 될 수 있다.
- ✓ PaperNet을 MagnetoCorp 관리자로 모니터링. “fabric-samples” 디렉토리에서 새 창을 열고 “monitordocker.sh” 스크립트를 찾아 실행하여 도커 네트워크 “net_basic” 과 관련된 PaperNet 도커 컨테이너에 대한 “logspout” tool 시작. (실습)
- ✓ “monitordocker.sh” 의 기본 포트가 이미 사용중인 경우 포트 번호를 위 명령에 전달 가능 (실습)
- ✓ 이 창은 도커 컨테이너의 출력 표시. MagnetoCorp 관리자가 네트워크와 상호 작용할 수 있는 다른 터미널 창 시작

Working as MagnetoCorp



- ✓ MagnetoCorp 관리자는 도커 컨테이너 통해 네트워크와 상호작용
- ✓ PaperNet 과 상호작용하려면 MagnetoCorp 관리자가 HF “peer” 명령 사용해야 함
- ✓ “hyperledger/fabric” 의 “docker image” 에 미리 내장
- ✓ “docker-compose” 명령을 사용하여 관리자를 위한 MagnetoCorp 관련 도커 컨테이너 시작
- ✓ (실습)
- ✓ “hyperledger/fabric-tools” 도커 이미지가 Docker Hub에서 검색되어 네트워크에 추가 된 방법 확인 (실습)

Working as MagnetoCorp

- ✓ MagnetoCorp 관리자는 컨테이너를 사용하여 PaperNeT과 상호작용 함. 또한 “logspout” 컨테이너를 확인하라. 이것은 “monitordocker.sh” 명령에 대한 다른 도커 컨테이너의 출력을 캡처한다.
- ✓ 이 명령을 사용하여 MagnetoCorp 관리자로서 PaperNet과 상호작용한다.

Smart contract

✓ “Issue", “buy"및 “redeem"은 PaperNet 스마트 계약의 핵심 기능dl다. 응용 프로그램에서 원장에 기업어음을 발행, 구매, 상환하는 거래에 사용된다. 다음은 이 스마트 컨트랙트를 확인한다.

✓ MagnetoCorp 개발자를 나타내는 새 터미널 창을 열고 MagnetoCorp의 스마트 컨트랙트 사본이 포함 된 디렉토리로 변경하여 선택한 편집기 (이 튜토리얼의 VS 코드)로이를 확인한다. (실습)

✓ 새 터미널 창을 열고 폴더의 "lib"디렉토리에 "papercontract.js"파일을 확인하다. 여기에는 기업어음 스마트 컨트랙트가 포함되어 있다!

✓ "papercontract.js"는 node.js 환경에서 실행되도록 설계된 JavaScript 프로그램이다. 다음 주요 프로그램 라인을 확인한다.

• const {contract, context} = require (‘fabric-contract-api’);

이 명령문은 스마트 컨트랙트에서 광범위하게 사용되는 두 가지 주요 HF 클래스 인 “contract” 및 “context"를 제공한다. "fabric-shim" "JSDOCS"에서 이러한 클래스에 대해 자세히 배운다.

Smart contract

- class CommercialPaperContract extends Contract {

내장 된 패브릭 “Contract"클래스를 기반으로 스마트 컨트랙트 클래스 "CommercialPaperContract"를 정의한다. 기업어음을 “issue", “buy"및 “redeem"하기 위해 주요 거래를 구현하는 방법이 이 클래스에 정의되어 있다.

- async issue(ctx, issuer, paperNumber, issueDateTime, maturityDateTime ...) {

이 메소드는 PaperNet의 기업어음 “issue"거래를 정의한다. 이 메소드로 전달 된 파라메터는 새 기업어음을 만드는 데 사용된다.

스마트 컨트랙트 내에서 "buy"및 "redeem"거래를 찾아서 검토하라.

- let paper = CommercialPaper.createInstance (issuer, paperNumber, issueDateTime ...);

"issue"거래 내에서 이 명령문은 제공된 트랜잭션 입력과 함께 "CommercialPaper"클래스를 사용하여 메모리에 새 기업어음을 작성한다. "buy"및 "redeem"트랜잭션을 검토하여 이 클래스가 어떻게 유사하게 사용되는지 확인한다.

Smart contract

- await ctx.paperList.addPaper(paper);

이 명령문은 스마트 컨트랙트 컨텍스트 "CommercialPaperContext"가 초기화 될 때 작성된 "PaperList"클래스의 인스턴스 인 "ctx.paperList"를 사용하여 원장에 새 기업어음을 추가한다. "buy"및 "redeem"방법을 다시 검토하여 이 클래스를 어떻게 사용하는지 확인한다.

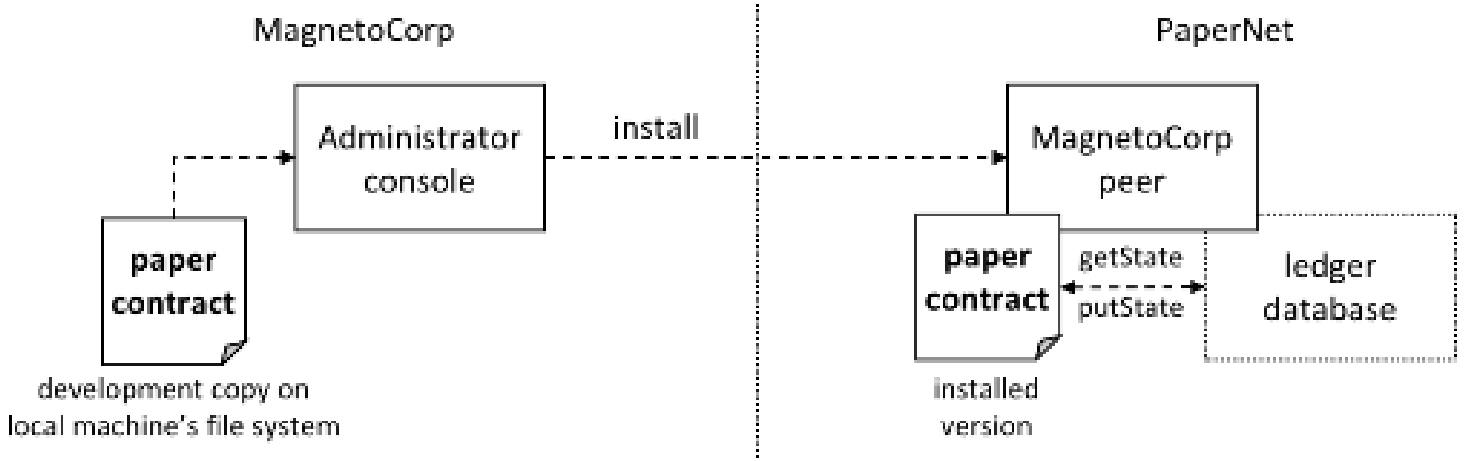
- return paper.toBuffer();

이 명령문은 스마트 컨트랙트의 호출자가 처리하기 위해 "issue"트랜잭션의 응답으로 이진 버퍼를 리턴한다.

✓ "contract"디렉토리의 다른 파일을 검토하여 스마트 컨트랙트의 작동 방식을 이해하고 스마트 컨트랙트 "topic"에서 "papercontract.js"가 어떻게 설계되어 있는지 자세히 읽는다.

Install contract

✓ "papercontract " 는 응용프로그램에서 호출하기 전에 PaperNet의 해당 피어 노드에 설치해야 한다. MagnetoCorp 및 DigiBank 관리자는 각각 권한이 있는 피어에 "papercontract"를 설치할 수 있다.



✓ 스마트 컨트랙트는 응용 프로그램 개발의 핵심이며 "chaincode"라는 HF 아태팩트에 포함되어 있다. 단일 체인 코드 내에서 하나 이상의 스마트 컨트랙트를 정의 할 수 있으며 체인 코드를 설치하면 PaperNet의 여러 조직에서 계약을 사용할 수 있다. 즉, 관리자만 체인 코드에 대해 걱정할 필요가 있다. 다른 모든 사람들은 스마트 컨트랙트로 생각할 수 있다.

Install contract

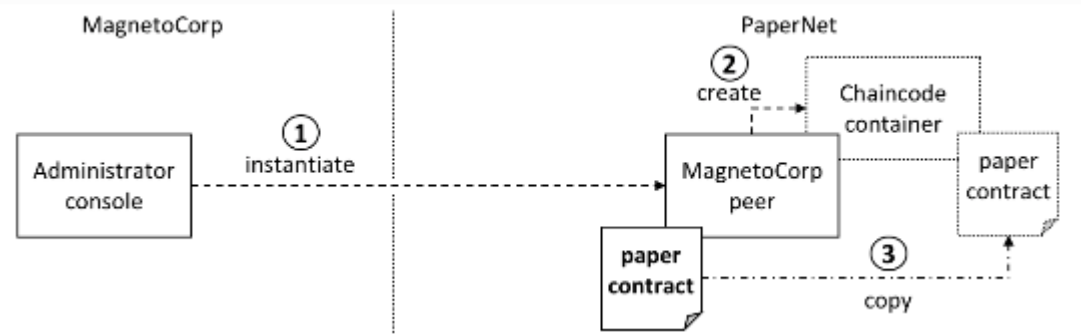
- ✓ MagnetoCorp 관리자는 "peer chaincode"설치 명령을 사용하여 "papercontract"스마트 컨트랙트를 로컬 파일 시스템에서 대상 피어의 도커 컨테이너 내 파일 시스템으로 복사한다. 스마트 컨트랙트가 피어에 설치되고 채널에서 인스턴스화 되면 응용 프로그램이 "papercontract"를 호출하고 "putState ()"및 "getState ()"Fabric API를 통해 원장 DB와 상호 작용할 수 있다. "ledger-api \ statelist.js"내의 "StateList"클래스에서 이러한 API를 사용하는 방법을 확인한다.
- ✓ 이제 MagnetoCorp 관리자로 "papercontract"를 설치한다. MagnetoCorp 관리자의 명령 창에서 "docker exec"명령을 사용하여 "cliMagnetCorp"컨테이너에서 "peer chaincode install"명령을 실행한다.
- ✓ (실습)

Install contract

- ✓ "cliMagnetCorp"컨테이너는 "CORE_PEER_ADDRESS=peer0.org1.example.com:7051"을 설정하여 명령을 "peer0.org1.example.com"으로 지정하고 "INFO 003 Installed remotely ..."는 "papercontract"를 나타낸다. 이 것은 피어에 성공적으로 설치된 것이다. 현재 MagnetoCorp 관리자는 단일 MagentoCorp 피어에 "papercontract"사본만 설치하면 된다.
- ✓ "peer chaincode install"명령이 "cliMagnetoCorp"컨테이너의 파일 시스템 "/opt/gopath/src/github.com/contract"와 관련하여 스마트 컨트랙트 경로 "-p"를 어떻게 지정했는지 참고한다. 이 경로는 "magnetocorp/configuration/cli/docker-compose.yml"파일을 통해 로컬 파일 시스템 경로 ".../organization/magnetocorp/contract"에 매핑되었다.
- ✓ "volume"지시문이 "organization/magnetocorp"를 "/opt/gopath/src/github.com/"에 어떻게 매핑하여 MagnetoCorp의 "papercontract"스마트 컨트랙트 사본이 보관되어 있는 로컬 파일 시스템에 이 컨테이너 액세스를 제공하는지 확인한다.
- ✓ "docker compose" 및 "peer chaincode instal" 명령은 "here"에서 확인한다.

Instantiate contract

✓ "CommercialPaper"스마트 컨트랙트를 포함하는 "papercontract"체인 코드가 필수 PaperNet 피어에 설치되었으므로 관리자는 다른 네트워크 채널에서 이를 사용할 수 있으며 해당 채널에 연결된 응용 프로그램에서 호출 할 수 있다. 우리는 PaperNet에 기본 네트워크 구성을 사용하고 있기 때문에 단일 네트워크 채널인 "mychannel"에서만 "papercontract"를 사용할 수 있다.



✓ MagnetoCorp 관리자는 "peer chaincode instantiate"명령을 사용하여 "mychannel"에서 "papercontract"를 인스턴스화한다.

✓ (실습)

Instantiate contract

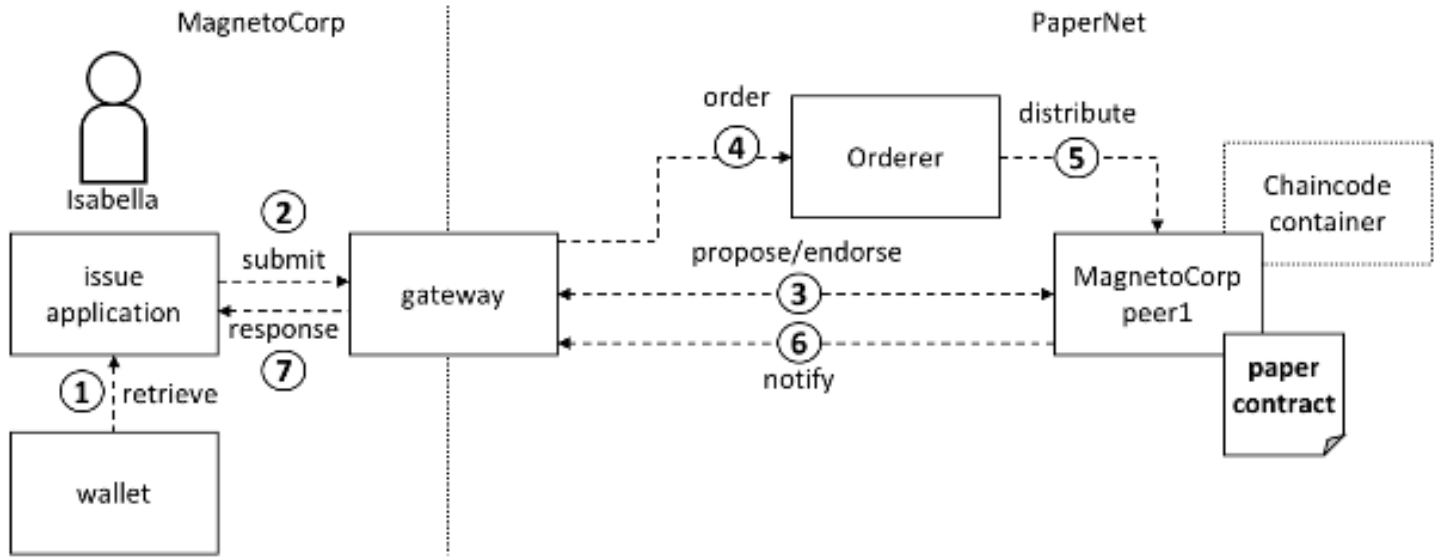
- ✓ "instantiate"에서 가장 중요한 매개 변수 중 하나는 "-P " 이다. "papercontract"에 대한 "endorsement policy"을 지정하며, 거래가 유효한 것으로 결정되기 전에 거래를 보증(실행 및 서명)해야 하는 조직 세트를 설명한다. 유효 여부에 관계없이 모든 거래는 "ledger blockchain"에 기록되지만 유효한 거래 만 "world state"를 업데이트한다.
- ✓ 전달에서 "instantiate"가 오더러 주소 "orderer.example.com:7050"을 전달하는 방법을 참조한다. 이는 다음 블록의 트랜잭션을 포함하고 "mychannel"에 가입한 모든 피어에게 트랜잭션을 분배하여 모든 자신의 격리 된 체인 코드 컨테이너에서 체인 코드를 실행할 수 있도록 오더러에게 인스턴스화 트랜잭션을 추가로 제출하기 때문이다. "papercontract"가 많은 피어에 설치되어 있지만 "papercontract"가 실행될 채널에 대해 "instantiate"를 한 번만 발행하면 된다.
- ✓ "docker ps"명령으로 "papercontract"컨테이너가 시작된 방법을 확인한다.
- ✓ (실습)

Instantiate contract

- ✓ 컨테이너의 이름은 "dev-peer0.org1.example.com-papercontract-0-d96 ..."이며 어떤 피어가 컨테이너를 시작했는지, 그리고 "papercontract"버전 "0"을 실행하고 있음을 나타낸다.
- ✓ 기본 PaperNet을 설치하고 "papercontract"를 설치하고 인스턴스화 했으므로 이제 기업어음를 발행하는 MagnetoCorp 응용 프로그램에 주의를 기울인다.

Application structure

✓ "papercontract"에 포함된 스마트 컨트랙트는 MagnetoCorp의 응용 프로그램 "issue.js"에 의해 호출된다. Isabella는 이 애플리케이션을 사용하여 기업어음 "00001"을 발행하는 원장에게 거래를 제출한다. "issue"응용 프로그램이 어떻게 작동하는지 빠르게 살펴본다.



✓ 게이트웨이는 애플리케이션이 트랜잭션 생성, 제출 및 응답에 집중할 수 있도록 한다. 다른 네트워크 구성 요소 간의 트랜잭션 제안, 오더링 및 알림 처리를 조정한다.

Application structure

✓ "issue"애플리케이션은 Isabella를 대신하여 거래를 제출하기 때문에 로컬 파일 시스템 또는 하드웨어 보안 모듈 "HSM"에 저장 될 수 있는 "wallet"에서 Isabella의 X.509 인증서를 검색하여 시작한다. "issue"애플리케이션은 "gateway"를 사용하여 채널에서 거래를 제출할 수 있다. HF SDK는 게이트웨이 추상화를 제공하여 응용 프로그램이 응용 프로그램 논리에 집중하면서 네트워크 상호 작용을 게이트웨이에 위임 할 수 있다. 게이트웨이와 지갑을 사용하면 HF 애플리케이션을 간단하게 작성할 수 있다.

✓ 이자벨라가 사용할 "issue"애플리케이션을 확인한다. 그녀를 위해 별도의 터미널 창을 열고 "fabric-samples"에서 MagnetoCorp "/application"폴더를 찾는다.

✓ (실습)

✓ "addToWallet.js"는 Isabella가 자신의 신분을 지갑에 넣는 데 사용할 프로그램이며 "issue.js"는 이 신원을 사용하여 "papercontract"를 호출하여 MagnetoCorp를 대신하여 기업어음 " 00001 " 을 만든다.

Application structure

✓ MagnetoCorp의 응용 프로그램 "issue.js"가 포함 된 디렉토리로 변경하고 코드 편집기를 사용하여 이를 확인한다. (실습)

✓ "issue.js"의 다음 주요 프로그램 라인에 유의한다.

- const {FileSystemWallet, 게이트웨이} = require ('fabric-network');

  이 문장은 "Wallet"과 "Gateway"라는 두 가지 주요 HF SDK 클래스를 제공한다. Isabella의 X.509 인증서가 로컬 파일 시스템에 있으므로 응용 프로그램은 "FileSystemWallet"을 사용한다.

- const wallet = 새로운 FileSystemWallet ('../ identity / user / isabella / wallet');

  이 문장은 애플리케이션이 블록 체인 네트워크 채널에 연결될 때 "isabella"지갑을 사용할 것임을 식별한다. 응용 프로그램은 "isabella"지갑 내에서 특정 신원을 선택한다. (지갑에는 Isabella의 X.509 인증서가 로드되어 있어야한다. "addToWallet.js"와 같다.)

Application structure

- await gateway.connect (connectionProfile, connectionOptions);

이 코드 라인은 "ConnectionOptions"에 언급 된 ID를 사용하여 "connectionProfile"로 식별 된 게이트웨이를 사용하여 네트워크에 연결한다.

"../gateway/networkConnection.yaml"및 "User1@org1.example.com"이 각각이 값에 어떻게 사용되는지 확인한다.

- const network = await gateway.getNetwork ('mychannel');

"papercontract"가 이전에 인스턴스화 된 네트워크 채널 "mychannel"에 응용 프로그램을 연결한다.

- const contract = await network.getContract ('papercontract', 'org.papernet.comm ...');

이 문장은 "papercontract"내의 네임 스페이스 "org.papernet.commercialpaper"에 의해 정의 된 스마트 컨트랙트에 응용 프로그램 주소 지정 성을 제공한다. 응용 프로그램이 getContract를 발행하면 응용 프로그램 내에 구현 된 트랜잭션을 제출할 수 있다.

Application structure

• const issueResponse=await contract.submitTransaction('issue', 'MagnetoCorp', '00001'...);

이 코드 라인은 스마트 계약 내에 정의 된 "issue"트랜잭션을 사용하여 트랜잭션을 네트워크에 제출한다. "MagnetoCorp", "00001"…은 "issue"거래에서 새 기업어음을 생성하는 데 사용되는 값이다.

• let paper = CommercialPaper.fromBuffer (issueResponse);

이 명세서는 "issue"거래의 응답을 처리한다. 응답은 버퍼에서 응용 프로그램에서 올바르게 해석할 수 있는 "CommercialPaper"개체 인 "paper"로 직렬화 해제 해야한다.

✓ "/ application"디렉토리의 다른 파일을 검토하여 "issue.js"의 작동 방식을 이해하고 "topic"응용 프로그램에서 어떻게 구현되는지 자세히 확인한다.

Application dependencies

- ✓ "issue.js"응용 프로그램은 JavaScript로 작성되었으며 PaperNet 네트워크의 클라이언트 역할을 하는 node.js 환경에서 실행되도록 설계되었다. 일반적인 방법과 마찬가지로 MagnetoCorp의 응용 프로그램은 품질과 개발 속도를 향상시키기 위해 많은 외부 노드 패키지를 기반으로 한다. "issue.js"에 YAML 게이트웨이 연결 프로파일을 처리하기위한 "js-yaml" "package"또는 "Gateway"및 "Wallet"클래스에 액세스하기위한 "fabric-network" "package"가 어떻게 포함되는지 고려한다: (문서 참조)
- ✓ 이 패키지는 "npm install"명령을 사용하여 "npm"에서 로컬 파일 시스템으로 다운로드 해야한다. 일반적으로 패키지는 런타임에 사용하기 위해 응용 프로그램 관련 "/ node_modules"디렉토리에 설치해야 한다.
- ✓ "package.json"파일을 검사하여 "issue.js"가 다운로드 할 패키지 및 해당 버전을 식별하는 방법을 확인한다: (문서 참조)

Application dependencies

- ✓ npm 버전 관리는 매우 강력하다. "here"에서 더 자세히 읽을 수 있다.
- ✓ "npm install"명령으로 이러한 패키지를 설치한다. 완료하는 데 최대 1 분이 소요될 수 있다.
- ✓ (실습)
- ✓ 이 명령이 디렉토리를 어떻게 업데이트 했는지 확인한다:
- ✓ (실습)
- ✓ "node_modules"디렉토리를 검사하여 설치된 패키지를 확인한다. "js-yaml"과 "fabric-network"는 다른 npm 패키지에 내장되어 있기 때문에 많은 것이 있다! "package-lock.json" "file"은 설치된 정확한 버전을 식별하므로 환경을 정확하게 재현하려는 경우 매우 중요하다. 예를 들어, 테스트, 문제 진단 또는 입증 된 응용 프로그램을 제공한다.

Wallet

- ✓ Isabella는 MagnetoCorp 상용 용지 "00001"을 발행하기 위해 "issue.js"를 실행할 준비가 거의 되었다. 수행해야 할 작업은 하나뿐이다! "issue.js"가 Isabella를 대신하여 작동하므로 MagnetoCorp는 이러한 사실을 반영하는 "wallet"의 ID를 사용한다. 이제 지갑에 적절한 X.509 자격 증명을 추가하는 이 일회성 작업을 수행해야한다.
- ✓ Isabella의 터미널 창에서 "addToWallet.js"프로그램을 실행하여 지갑에 신원 정보를 추가한다.
- ✓ (실습)
- ✓ Isabella는 지갑에 여러 ID를 저장할 수 있지만 이 예제에서는 "User1@org.example.com"만 사용한다. 이 ID는 현재보다 현실적인 PaperNet 구성이 아니라 기본 네트워크와 연결되어 있다. 이 튜토리얼은 곧 업데이트 될 예정이다.
- ✓ "addToWallet.js"는 여가 시간에 검사 할 수 있는 간단한 파일 복사 프로그램이다. 기본 네트워크 샘플에서 Isabella의 지갑으로 ID를 이동한다. "PaperNet"에 거래를 제출하는 데 사용될 지갑의 내용인 이 프로그램의 결과에 중점을 둔다: (실습)

Wallet

✓ 디렉토리 구조가 "User1@org1.example.com"아이디를 매핑하는 방법을 참조한다. Isabella가 사용하는 다른 아이디에는 자체 폴더가 있다. 이 디렉토리에는 "isabella"을 대신하여 "issue.js"가 사용할 ID 정보가 있다. (실습)

✓ 주의: • 개인 키 "c75bd6911a ...- priv"는 Isabella를 대신하여 거래에 서명하는 데 사용되었지만 즉시 통제 할 수 없는 범위에는 배포되지 않았다.

• Isabella의 개인 키와 암호로 연결된 공개 키 "c75bd6911a ...- pub" 이것은 Isabella의 X.509 인증서에 전부 포함되어 있다.

• 인증서 생성시 인증 기관에서 추가 한 Isabella의 공개 키 및 기타 X.509 속성이 포함 된 인증서 "User1@org.example.com" 이 인증서는 다른 시간에 다른 행위자가 Isabella의 개인 키로 생성 된 정보를 암호화 적으로 확인할 수 있도록 네트워크에 배포된다.

✓ 인증서 "here"에서 자세히 알아본다. 실제로 인증서 파일에는 Isabella의 조직 및 역할과 같은 패브릭 별 메타 데이터도 포함되어 있다. "wallet"항목에서 자세히 읽어본다.

Issue application

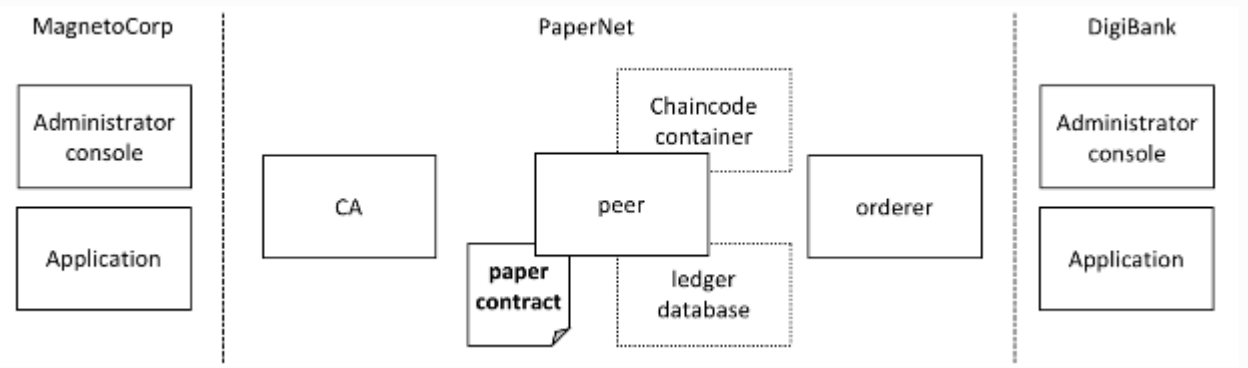
- ✓ Isabella는 이제 "issue.js"를 사용하여 MagnetoCorp 기업어음 "00001"을 발행하는 거래를 제출할 수 있다. (실습)
- ✓ "node"명령은 node.js 환경을 초기화하고 "issue.js"를 실행한다. 프로그램 출력에서 MagnetoCorp 기업어음 00001이 액면가 5M USD로 발행되었음을 알 수 있다.
- ✓ 위에서 본 바와 같이, 이를 달성하기 위해 애플리케이션은 "papercontract.js"내의 "CommercialPaper"스마트 컨트랙트에 정의 된 "issue"트랜잭션을 호출한다. 이것은 MagnetoCorp 관리자에 의해 네트워크에 설치되고 인스턴스화 되었다. 이는 "putState ()"및 "getState ()"와 같이 Fabric API를 통해 원장과 상호 작용하여 새로운 기업어음을 world state 내에서 벡터 상태로 나타내는 스마트 컨트랙트이다. 스마트 컨트랙트 내에서 정의 된 "buy"및 "redeem"거래를 통해 이 벡터 상태가 어떻게 조작되는지 살펴 본다.

Issue application

- ✓ 기본 Fabric SDK는 항상 거래 승인, 오더링 및 알림 프로세스를 처리하여 애플리케이션의 논리를 간단하게 만든다. SDK는 "gateway"를 사용하여 네트워크 세부 정보를 추상화하고 "connectionOptions"를 사용하여 트랜잭션 재시도와 같은 고급 처리 전략을 선언한다.
- ✓ 이제 기업어음을 구매할 DigiBank로 전환하여 MagnetoCorp "00001"의 라이프 사이클을 살펴본다.

Working as DigiBank

✓ 이제 MagnetoCorp가 기업어음 "00001"을 발행 했으므로 DigiBank의 직원으로서 PaperNet과 상호 작용하도록 컨텍스트를 전환 해본다. 먼저, PaperNet과 상호 작용하도록 구성된 콘솔을 생성할 관리자 역할을 한다. 그런 다음 최종 사용자인 Balaji는 Digibank의 "buy"응용 프로그램을 사용하여 기업어음 "00001"을 구입하여 수명주기의 다음 단계로 옮긴다.



✓ 튜토리얼은 현재 PaperNet의 기본 네트워크를 사용하므로 네트워크 구성은 매우 간단하다. 관리자는 MagnetoCorp와 유사한 콘솔을 사용하지만 Digibank의 파일 시스템에 맞게 구성되었다. 마찬가지로 Digibank 최종 사용자는 Digibank 고유의 논리 및 구성을 포함하지만 MagnetoCorp 응용 프로그램과 동일한 스마트 컨트랙트를 호출하는 응용 프로그램을 사용한다.

Working as DigiBank

- ✓ 공유 비즈니스 프로세스를 캡처하는 스마트 컨트랙트와 어떤 애플리케이션이 호출하든 상관없이 공유 비즈니스 데이터를 보유하는 원장이다.
- ✓ DigiBank 관리자가 PaperNet과 상호 작용할 수 있도록 "fabric-samples"에서 별도의 터미널을 연다: (실습)
- ✓ 이 도커 컨테이너는 이제 Digibank 관리자 네트워크와 상호 작용할 수 있다.
- ✓ 이 튜토리얼에서는 "cliDigiBank"라는 명령 줄 컨테이너를 사용하여 DigiBank를 대신하여 네트워크와 상호 작용한다. 우리는 모든 도커 컨테이너를 표시하지 않았으며 실제로 DigiBank 사용자는 액세스 할 수 있는 네트워크 구성 요소 (피어, 오더러, CA) 만 볼 수 있다.
- ✓ PaperNet 네트워크 구성이 매우 간단하기 때문에 Digibank의 관리자는 지금 튜토리얼에서 할 일이 많지 않다. Balaji에게 관심을 돌리자.

Digibank applications

- ✓ Balaji는 DigiBank의 "buy"애플리케이션을 사용하여 거래처를 원장에게 제출한다. 이 거래는 MagnetoCorp에서 DigiBank로 상업용 용지 "00001"의 소유권을 이전합니다.
- ✓ "CommercialPaper"스마트 컨트랙트는 MagnetoCorp의 응용 프로그램에서 사용하는 것과 동일하지만 이번에는 거래가 다르다. "redeem"보다는 "buy"다. DigiBank의 응용 프로그램 작동 방식을 살펴 본다.
- ✓ Balaji에 대한 별도의 터미널 창을 연다. 패브릭 샘플에서, buy.js 애플리케이션이 포함 된 DigiBank 애플리케이션 디렉토리로 변경하고 편집기에서 이를연다.
- ✓ 보다시피 이 디렉토리에는 Balaji가 사용할 “buy” 및 “redeem” 응용프로그램이 모두 포함되어 있다.

Digibank applications

✓ DigiBank의 "buy.js"응용 프로그램은 MagnetoCorp의 "issue.js"와 구조가 매우 유사하며 두 가지 중요한 차이점이 있다:

• ID : 사용자는 MagnetoCorp의 "Isabella"가 아닌 DigiBank 사용자 "Balaji"이다.

<const wallet = new FileSystemWallet ('../ identity / user / balaji / wallet');>

응용 프로그램이 PaperNet 네트워크 채널에 연결될 때 "balaji"지갑을 사용하는 방법을 본다. "buy.js"는 "balaji"지갑에서 특정 ID를 선택합니다.

• Transaction : 호출 된 트랜잭션은 "issue"가 아닌 "buy"

<const buyResponse=await contract.submitTransaction('buy', 'MagnetoCorp', '00001'...); >

기업어음 "00001"의 소유권을 DigiBank로 이전하기 위해 "CommercialPaper"스마트 컨트랙트 클래스에서 사용하는 "MagnetoCorp", "00001"… 값으로 "buy"트랜잭션이 제출된다.

✓ "application"디렉토리의 다른 파일을 검사하여 응용 프로그램의 작동 방식을 이해하고 "topic"응용 프로그램에서 "buy.js"가 어떻게 구현되는지 자세히 읽는다.

Run as DigiBank

- ✓ 기업어음을 구매하고 상환하는 DigiBank 응용 프로그램은 MagnetoCorp의 issue 응용 프로그램과 매우 유사한 구조를 가지고 있다. 따라서 의존성을 설치하고 Balaji의 지갑을 설정하여 이러한 응용 프로그램을 사용하여 기업어음을 구입하고 사용할 수 있도록 한다.
- ✓ MagnetoCorp와 마찬가지로 Digibank는 "npm install"명령을 사용하여 필요한 응용 프로그램 패키지를 설치해야 하며, 이 과정을 완료하는 데 약간의 시간이 걸린다.
- ✓ DigiBank 관리자 창에서 애플리케이션 종속성을 설치한다: (실습)
- ✓ Balaji의 터미널 창에서 "addToWallet.js"프로그램을 실행하여 지갑에 신원 정보를 추가한다. (실습)

Run as DigiBank

- ✓ "addToWallet.js"프로그램은 "balaji"에 대한 신원 정보를 지갑에 추가했으며, "buy.js"및 "redeem.js"가 "PaperNet"에 거래를 제출하는데 사용한다.
- ✓ Isabella와 마찬가지로 Balaji는 지갑에 여러 ID를 저장할 수 있지만 이 예에서는 "Admin@org.example.com"만 사용한다.
  "digibank/identity/user/balaji/wallet/Admin@org1.example.com"에 포함 된 지갑 구조는 Isabella와 매우 유사하므로 자유롭게 확인한다.

Buy application

- ✓ Balaji는 이제 "buy.js"를 사용하여 MagnetoCorp 기업어음 "00001"의 소유권을 DigiBank로 이전하는 거래를 제출할 수 있다.
- ✓ Balaji의 창에서 "buy"응용 프로그램을 실행한다. (실습)
- ✓ Balaji가 DigiBank를 대신하여 MagnetoCorp 기업어음 00001을 성공적으로 구매 한 프로그램 출력을 볼 수 있다. "buy.js"는 "putState ()"및 "getState ()"Fabric API를 사용하여 world state에서 기업어음 "00001"을 업데이트 한 "CommercialPaper"스마트 컨트랙트에 정의 된 "buy"트랜잭션을 호출했다. 보다시피, 기업어음을 사고 발행하는 애플리케이션 로직은 스마트 컨트랙트 로직과 매우 유사하다.

Redeem application

- ✓ 기업어음 "00001"의 수명주기에서 최종 거래는 DigiBank가 MagnetoCorp로 이를 사용하는 것이다. Balaji는 "redeem.js"를 사용하여 스마트 컨트랙트 내에서 상환 로직을 수행 할 트랜잭션을 제출한다.
- ✓ Balaji의 창에서 "redeem"거래를 실행한다: (실습)
- ✓ "redeem.js가"CommercialPaper "에 정의 된 "redeem "트랜잭션을 호출했을 때 기업어음 00001이 어떻게 사용되었는지 다시 확인한다. 다시 소유권이 어음 발행자인 MagnetoCorp로 반환되었음을 반영하기 위해 world state 내에서 기업어음 " 00001 " 을 업데이트했다.

Further reading

✓ 이 튜토리얼에 표시된 애플리케이션 및 스마트 컨트랙트가 어떻게 작동하는지 이해하려면 " Developing Applications"을 읽는 것이 도움이 된다. 이 주제에서는 기업어음 시나리오, "PaperNet"비즈니스 네트워크, 해당 참여자 및 이들이 사용하는 응용 프로그램 및 스마트 컨트랙트에 대한 자세한 설명을 제공한다.

✓또한 이 샘플을 사용하여 자체 애플리케이션 및 스마트 컨트랙트 작성을 시작한다!

감사합니다