[BLOCKCHAIN] HYPERLEDGER FABRIC DEVELOPER GUIDE

Moo Je Kong

IBM | mjkong@kr.ibm.com

1. Hyperledger Fabric Overview

Hyperledger 프로젝트는 B2B (business-to-business) 및 B2C (B2B) 거래를 위한 블록체인을 만드는 오픈소스 협업의 산출물입니다. IBM 은 Hyperledger Project 의 창립 멤버 중 한 명으로, 44,000 줄의 블록체인 코드를 Hyperledger Fabric 에 기부하여 프로젝트화 시켰습니다.

Hyperledger Fabric 의 목표는 B2B 및 B2C 트랜잭션과 관련된 다양한 산업 사례에 적용 할 수 있는 공개 표준을 만드는 것입니다.

이 노력의 주요 목표는 다음과 같습니다:

- 다양한 요구 사항을 가진 다양한 산업에 활용 사례 지원
- 최근 존재하는 규제 체제 준수
- 확인된 ID, 개인 및 보안 트랜잭션 지원
- 허가된 공유 원장 지원
- 성능, 확장성, 감사 가능성, ID, 보안 및 개인 정보 보호 지원
- 작업 증명에 포함된 고비용의 처리비용의 감소

기능과 필요한 기능을 제공하기 위해 Hyperledger Fabric 은 다음의 개념을 기반으로 합니다:

- 스마트 계약
- 디지털 자산
- 기록 보관소 / 저장 시스템
- 분산된 합의 기반 네트워크
- 플러그인 기반의 합의(Consensus) 알고리즘/모델
- 암호 보안

1.1. 용어

다음은 Fabric 기반 블럭체인의 기술을 익히기 위해서 필요한 용어들입니다. 용어들은 가급적 한글화를 하지 않고 영문 용어를 그대로 사용했습니다. 그리고 대부분 시스템 용어라고 생각하시면 이해하는데 도움이 될 것 같습니다.

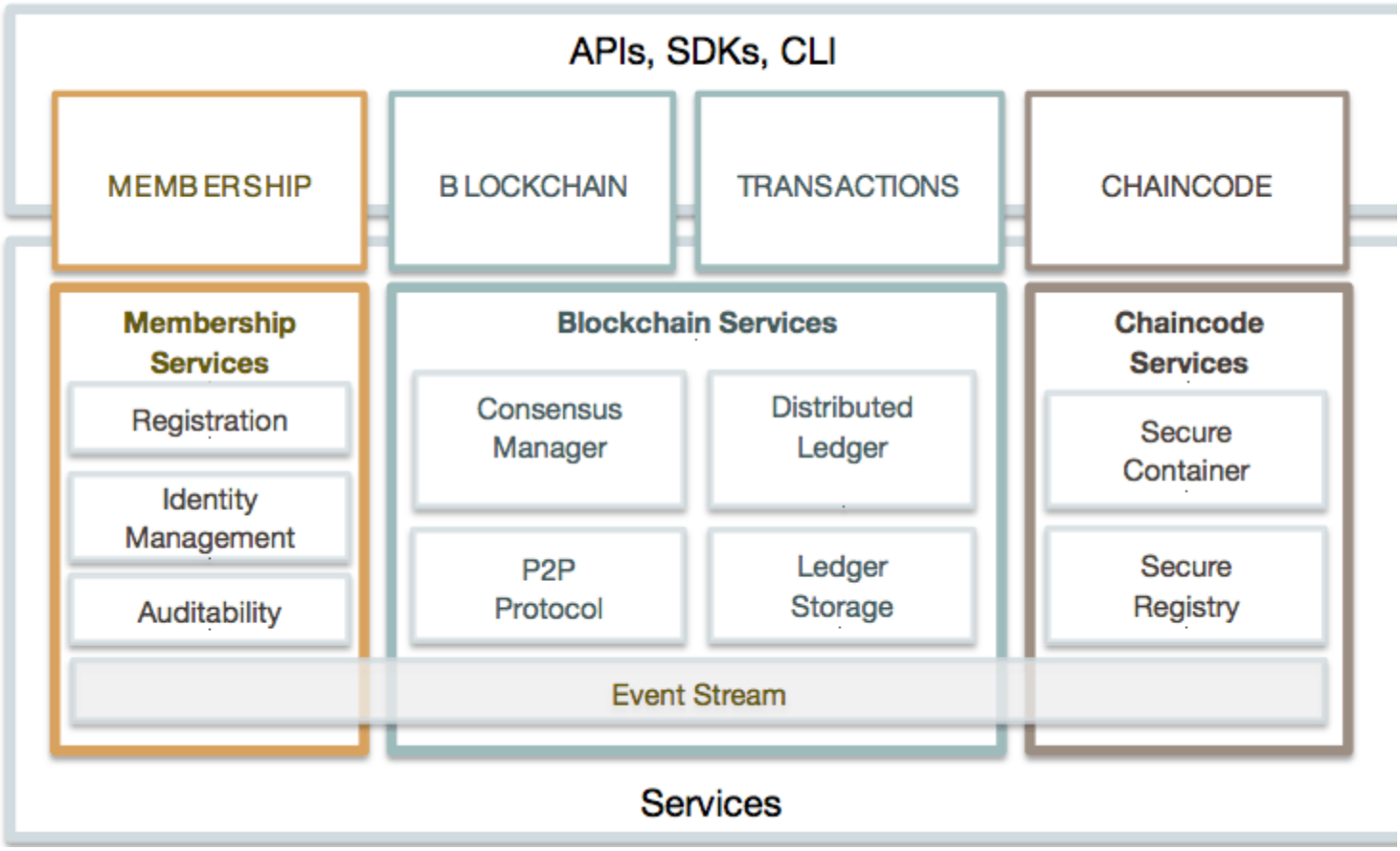
- **Transactor :** 트랜잭션(거래)을 일으키는 엔티티를 말합니다. 대표적으로 클라이언트 애플리케이션이 됩니다.
- **Transaction :** 트랜잭션은 블럭체인 네트워크에 대해서 비즈니스 로직을 수행하기 위한 요청입니다. 트랜잭션의 유형은 deploy, invoke 및 query dlau, 체인코드를 통해서 사전 정의된 인터페이스에 대한 함수를 구현합니다.
- **Ledger :** Ledger : 트랜잭션과 현재 세계 상태를 포함하는 일련의 암호 학적으로 링크 된 블록. 이전 거래의 데이터 외에도 원장에는 현재 실행중인 체인 코드 애플리케이션의 데이터가 포함되어 있습니다.
- World state : 트랜잭션에 의해서 체인코드가 호출될 때 상태 및 데이터 저장을 위한 Key-value 데이터베이스 입니다.
- **Chaincode :** 다양한 트랜잭션의 유형을 구현한 블록체인에 임베드되는 로직입니다. 개발자에 의해 체인코드가 작성되고 블록체인 네트워크로 디플로이 합니다. 최종 사용자는 블록체인 네트워크를 구성하는 피어 또는 노드와 인터페이스 되어 있는 클라이언트 애플리케이션을 통해서 체인코드를 실행시킵니다. 체인코드는 트랜잭션을 일으키고 유효성이 확인되면 공유원장에 추가하고 World state 를 수정합니다.
- **Validating peer (VP) :** 블록체인 네트워크에서 원장을 관리 유지하기 위해서 트랜잭션의 유효성을 검증하는 합의 프로토콜을 실행하는 노드입니다. 검증된 트랜잭션은 원장에 블록 단위로 추가됩니다. 트랜잭션이 합의에

실패하면 블록에서 제거되므로 장부에 기록되지 않습니다. Validating peer 는 체인코드를 deploy, invoke, query 할 권한을 가집니다.

- **Non-validating peer (NVP) :** Transactor 가 Validating peer 에 접속 할 수 있도록 프록시 역할을 하는 노드입니다. Non-validating peer (NVP) 는 호출된 요청을 Validating peer 로 전달하며, 이벤트 스트림, REST 서비스를 담당하는 노드입니다.
- **Consensus :** 블록체인 네트워크의 트랜잭션(deploy, invoke) 순서를 유지하는 프로토콜, Validating 노드들은 합의 프로토콜을 구현하여 트랜잭션을 승인하기 위해서 함께 동작합니다.
- **Permissioned network :** 각 노드는 블록체인 네트워크에서 접근 권한을 관리해야 하는 노드이며, 각 노드는 권한이 있는 사용자만 접근할 수 있습니다.
-

1.2. 아키텍처

블록체인의 엔진으로 볼수 있는 Fabric 은 크게 3 가지 서비스로 구성이 되어 있습니다.



1.2.1. Membership Services

멤버십 서비스는 블록체인 네트워크에서 인증 서비스를 제공합니다. Non-permissioned 블록체인의 경우 사용자 인증이 필요없으며, 모든 노드는 동등하게 트랜잭션 처리가 가능하고 블록에 트랜잭션 정보를 입력할 수 있습니다. 멤버쉽 서비스는 PKI(Public Key Infrastructure)와 분산화/합의 컴포넌트를 non-permissioned 블록체인에서 permissioned 블록체인으로 변환시킵니다. Permissioned 블록체인에서는 엔티티가 장기적인 인증서(enrollment certificates)를 획득하기 위해 등록절차를 거치게 되며, 엔티티 유형에 따라 구별될 수 있습니다. 사용자의 경우 TCA (Transaction Certificate Authority)가 인증서를 발급 할 수 있습니다. 여기서 획득한 인증서는 트랜잭션을 발생시킬때 인증하는데 사용됩니다.

1.2.2. Blockchain Services

블록체인 서비스는 HTTP/2 표준을 기반으로 P2P 프로토콜을 통해서 분산원장을 관리합니다. 데이터 구조는 해시 알고리즘을 통해 World state 를 복제하는 등 관리 하는데 가장 효율적으로 관리할 수 있도록 최적화되어 있습니다. 필요에 따라 다른 합의 알고리즘 플러그인(PBFT, Raft, PoW, PoS)을 연결하고 구성 할 수 있습니다.

1.2.3. Chaincode Services

체인코드 서비스는 Validating 노드에서 안전하고 가볍운 방법으로 체인코드가 실행되도록 보장합니다. 환경은 보안 OS 및 체인 코드 언어, Go, Java 및 Node.js 의 런타임 및 SDK 계층을 포함하는 일련의 서명 된 기본 이미지와 함께 "잠긴"보안 컨테이너입니다. 필요한 경우 다른 언어를 사용할 수 있습니다.

1.2.4. 그외 기능

1.2.4.1. Events

Validating peers 와 체인코드는 블록체인 네트워크에서 이벤트를 발생 할 수 있습니다. 이벤트 발생을 대기하고 있는 애플리케이션에 필요한 노티를 보냅니다. 이벤트는 미리 정의된 이벤트 혹은 커스텀 이벤트를 발생 할 수 있으며, 하나 이상의 어댑터를 통해 이벤트 수신 할 수 있습니다. 또한 Web hooks 나 Kafka 등을 이용하여 이벤트 수신도 가능합니다.

1.2.4.2. Application Programming Interface (API)

Fabric 에 대한 기본 인터페이스는 REST API 입니다. API 호출을 통해 사용자 등록, 블록체인에 대한 쿼리, 트랜잭션 발생 들을 할 수 있습니다. 특히 체인코드와 상호작용하기 위한 API 를 통해서 트랜잭션을 관리 할 수 있습니다.

1.2.4.3. Command Line Interface (CLI)

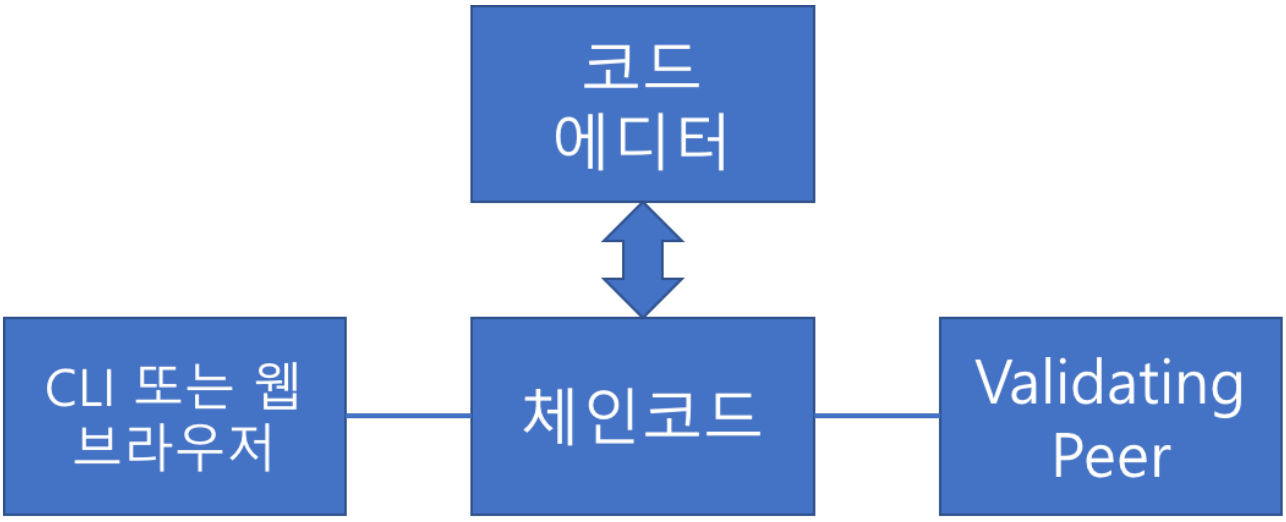
REST API 의 일부 기능을 지원하는 CLI 기능을 통해서 체인코드의 디플로이 및 트랜잭션 처리 등을 빠르게 진행할 수 있도록 합니다. CLI 는 Golang 으로 제작이 되었으며 다양한 OS 를 지원합니다.

1.3. 토폴로지

Fabric 을 통한 볼록체인 네트워크는 하나의 멤버쉽 서비스와 다수의 Validating peer 와 non-validating peer 들로 이루어 질 수 있습니다. 이 모든 컴포넌트를 통해 하나 또는 다수의 체인을 운영 할 수 있습니다.

1.3.1. Single Validating Peers

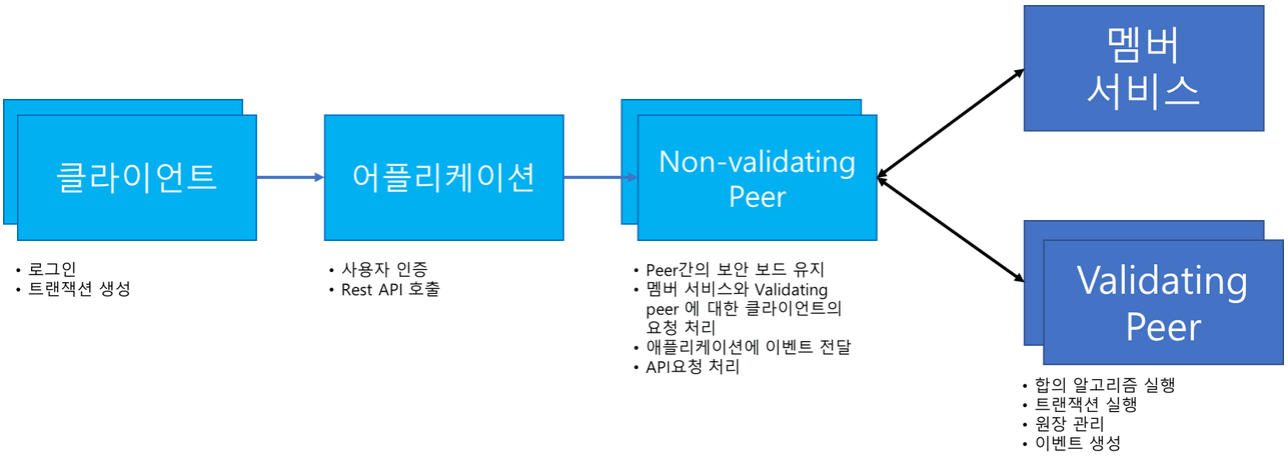
기능적으로 non-validating peer 는 validating peer 의 서브셋입니다. 그러므로 모든 non-validating peer 의 기능은 validating peer 로 사용이 가능합니다. 그래서 가장 간단한 블록체인의 네트워크는 하나의 validating peer 로 구성된 블록체인 네트워크입니다. 이 토폴로지는 보통 개발을 위한 환경으로 적합합니다.



이 토폴로지는 합의 알고리즘 사용을 할 수 없습니다. 그래서 기본으로 설정되어있는 합의 모듈인 noops 플러그인을 적용해서 사용합니다.

1.3.2. Multiple Validating Peers

운영환경이거나 개발 환경일 경우 다양한 validating peer 와 non-validating peer 를 이용하여 블록체인 네트워크를 구성해야 합니다. 이 구성에서 non-validating peer 는 이벤트 처리 및 REST API 서비스 관리등의 역할을 하게 되는 노드입니다.



Validating peers 들은 블록체인 네트워크 상에서 일어나는 모든 이벤트, 트랜잭션 등의 데이터를 공유하게 됩니다.

1.3.3. Multichain

각각 블록체인 네트워크는 validating peer 와 non-validating peer 로 이루어져 있습니다. 다양한 목적에 따라서 이와 같은 조합으로 다양한 블록체인을 구성 할 수 있습니다.

2. 개발환경 준비

Hyperledger Fabric 은 일차적으로 Docker 이미지 형태로 배포가 됩니다. 그래서 블록체인의 런타임 환경을 구성하기 위해서는 기본적으로 Docker engine 을 설치해야 하며, 체인코드는 기본으로 golang 으로 개발합니다.

2.1. Docker engine 설치

최신의 Docker engine 을 설치하게 되면 Docker compose, Docker machine 등 Docker 사용을 편리하게 하는 도구들이 함께 설치됩니다. 다만, 최신 버전 설치를 위해서는 OS 가 최소 요구사항에 맞아야 하므로 개발 PC 의 환경에 맞춰서 Docker engine 을 설치하시기 바랍니다. (현 문서에서는 docker-machine 명령어를 통해 진행 할 예정입니다.)

2.1.1. Windows

최신의 Docker engine 은 Windows 10 Pro 이상을 지원합니다. 개발 환경의 OS 가 최소 사양에 만족하면 다음의 경로에서 설치 정보를 확인해서 Docker engine 을 설치하시기 바랍니다.

- 최신 Docker engine 설치 정보 링크: https://docs.docker.com/docker-for-windows/#/download-docker-for-windows

만약, Windows 10 이전 버전(Windows 7 등) 을 사용하시면 Docker Toolbox 를 설치하셔야 합니다.

- Docker Toolbox 설치 링크 : https://www.docker.com/products/docker-toolbox

2.1.2. Mac

맥은 다음의 링크에 따라 설치하시기 바랍니다. 그리고 또한 VirtualBox 도 설치합니다.

- Mac Docker engine 설치 링크 : https://docs.docker.com/docker-for-mac/
- VirtualBox 다운로드 경로 : https://www.virtualbox.org/wiki/Downloads

2.2. Docker VM 환경 생성

앞서 Docker engine 이 설치되었으면 docker-machine 명령어를 통해 docker 사용을 위한 VM 을 생성합니다.

```
docker-machine create --driver virtualbox blockchain
```

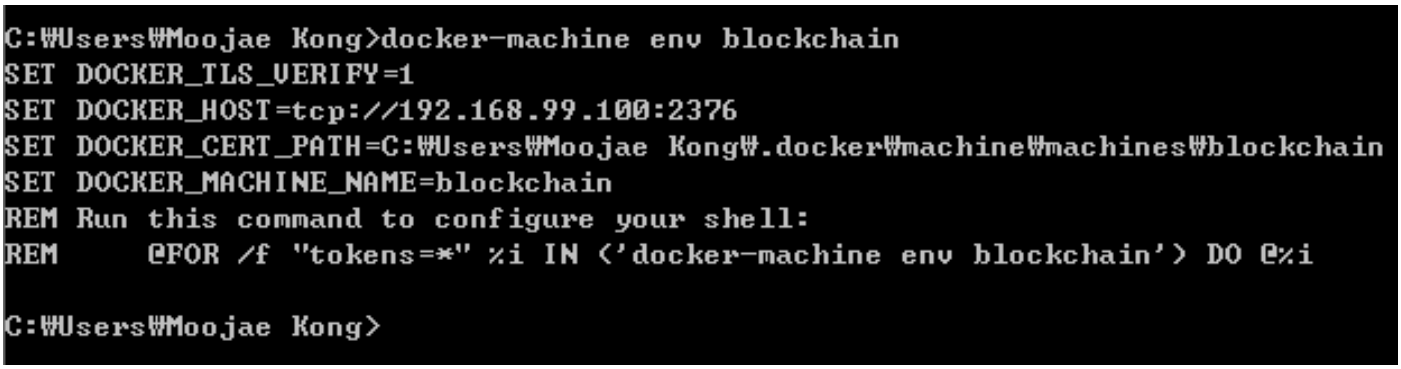
```
mjmac:~ mjkong$ docker-machine create --driver virtualbox blockchain
Running pre-create checks...
Creating machine...
(blockchain) Copying /Users/mjkong/.docker/machine/cache/boot2docker.iso to /Users/mjkong/.docker/machine/machines/blockchain/boot2docker.iso...
(blockchain) Creating VirtualBox VM...
(blockchain) Creating SSH key...
(blockchain) Starting the VM...
(blockchain) Check network to re-create if needed...
(blockchain) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docker-machine env blockchain
mjmac:~ mjkong$
```

다음으로 새로 생성된 VM 의 환경을 Host 에 적용하기 위해서 다음의 명령어를 실행합니다. (OS 별로 결과 내용이 틀립니다.)

```
docker-machine env blockchain
```

2.2.1. Windows

Windows 의 경우에 아래 그림과 같이 메세지가 나오며 이어서 다음의 명령를 입력하시면 됩니다.




```
@FOR /f "tokens=*" %i IN ('docker-machine env blockchain') DO @%i
```

2.2.2. Mac

Mac 의 경우 아래와 같이 화면에 출력되며, 다음의 명령어를 이어서 입력하시면 됩니다.


```
eval $(docker-machine env blockchain)
```

2.2.3. 설정 확인

다음의 명령어를 통해 아래 이미지와 같이 출력되면 정상적으로 설정 된 것입니다.

```
docker images
```

```
mjmac:~ mjkong$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mjmac:~ mjkong$
```

2.3. Hyperledger image 받기

Docker 환경이 설치가 되었으면 다음으로 Hyperledger images 를 각각 다운받습니다.
(참고로 네트워크 환경에 따라 다운로드 받는 시간이 길어질 수 있습니다.)

Image name	Tag
hyperledger/fabric-baseimage	x86_64-0.2.2
hyperledger/fabric-membersrvc	x86_64-0.6.1-preview
hyperledger/fabric-peer	x86_64-0.6.1-preview

```
docker pull hyperledger/fabric-baseimage:x86_64-0.2.2
docker pull hyperledger/fabric-membersrvc: x86_64-0.6.1-preview
docker pull hyperledger/fabric-peer: x86_64-0.6.1-preview
```

의의 모든 이미지를 다운로드 받고 다음의 명령어를 통해서 리스트를 확인합니다.

```
docker images
```

```
mjmac:~ mjkong$ docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
hyperledger/fabric-baseimage  x86_64-0.2.2        4ac07a26ca7a        6 weeks ago         1.24 GB
hyperledger/fabric-membersrvc x86_64-0.6.1-preview b3654d32e4f9        3 months ago        1.42 GB
hyperledger/fabric-peer       x86_64-0.6.1-preview 21cb00fb27f4        3 months ago        1.42 GB
```

2.3.1. 이미지 태깅

위에서 3 가지의 이미지를 모두 받았으면 baseimage 에 대해서 latest 로 태킹을 합니다.(향후 운영모드에서 체인코드 디플로이 시 hyplerledger/fabric-baseimage:latest 이미지를 참조합니다.)

```
docker tag hyperledger/fabric-baseimage:x86_64-0.2.2 hyperledger/fabric-baseimage
docker images
```

```
mjmac:~ mjkong$ docker tag hyperledger/fabric-baseimage:x86_64-0.2.2 hyperledger/fabric-baseimage:latest
mjmac:~ mjkong$ docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
hyperledger/fabric-baseimage  latest              4ac07a26ca7a        6 weeks ago         1.24 GB
hyperledger/fabric-baseimage  x86_64-0.2.2        4ac07a26ca7a        6 weeks ago         1.24 GB
hyperledger/fabric-membersrvc x86_64-0.6.1-preview b3654d32e4f9        3 months ago        1.42 GB
hyperledger/fabric-peer       x86_64-0.6.1-preview 21cb00fb27f4        3 months ago        1.42 GB
```

자, 여기까지 준비가 되었으면 개발을 위한 Docker 설치와 Hyperledger Fabric 이미지 준비가 끝났습니다.
다음으로 체인코드 개발을 위한 Golang 설치만 하시면 개발을 위한 기본 환경 셋팅이 완료됩니다

2.4. Golang 설치

Hyperledger Fabric 환경에서 체인코드 개발은 현재까지 Golang, Java 로 가능합니다. 이 문서에서는 기본 환경인 Golang 으로 체인코드 개발하는 방법에 대해서 설명합니다.

2.4.1. Golang 바이너리 설치

개발 환경의 OS 에 맞춰 해당하는 Golang 바이너리 다운 받아서 설치하면 되고, 다음의 링크에서 설치 바이너리 받으시기 바랍니다.
(현재 문서를 작성하는 순간의 최신 버전은 1.7.4 버전입니다.)
Golang 설치 바이너리 다운로드 경로 : https://golang.org/dl/

2.4.1.1. Windows

MSI 인스톨러 버전을 받아서 설치하면, 기본적인 환경 변수까지 셋팅이 됩니다. 설치 이후 다음의 명령어를 통해서 정상적으로 설치가 되었는지 확인합니다

```
> go
```

```
C:₩Users₩Moojae Kong>go
Go is a tool for managing Go source code.

Usage:

        go command [arguments]

The commands are:

        build       compile packages and dependencies
        clean       remove object files
        doc         show documentation for package or symbol
        env         print Go environment information
        fix         run go tool fix on packages
        fmt         run gofmt on package sources
        generate    generate Go files by processing source
        get         download and install packages and dependencies
        install     compile and install packages and dependencies
        list        list packages
        run         compile and run Go program
        test        test packages
        tool        run specified go tool
        version     print Go version
        vet         run go tool vet on packages

Use "go help [command]" for more information about a command.

Additional help topics:

        c           calling between Go and C
        buildmode   description of build modes
        filetype    file types
        gopath      GOPATH environment variable
        environment environment variables
        importpath  import path syntax
        packages    description of package lists
        testflag    description of testing flags
        testfunc    description of testing functions

Use "go help [topic]" for more information about that topic.
```

설치가 확인되었으면, Workspace 에 해당하는 gopath 를 설정합니다. 특정 위치에 폴더를 만들고 다음과 같이 GOPATH 를 환경변수로 설정합니다.

```
mkdir C:₩work
set GOPATH=C:₩work
```

2.4.1.2. Mac

Mac 에 golang 설치를 위해서는 .pkg 파일을 다운로드해서 설치합니다. 기본 설치 경로는 /usr/local/go 입니다.
Workspace 에 해당하는 디렉토리 생성후 각 사용자 계정에서 환경 변수를 다음과 같이 설정합니다.
저의 개인설정은 다음과 같으니 참조 바랍니다.

- 사용자 계정 mjkong 하위에 Dev/go 에 workspace 디렉토리 생성
- GOROOT, GOPATH 에 대한 환경 변수를 설정
- GOROOT 하위 bin 리렉토리를 PATH 로 추가합니다.

```
#### Go ####
export GOROOT=/usr/local/go
export GOPATH=/Users/mjkong/Dev/go
export PATH=$PATH:$GOROOT/bin
```

2.4.1.3. Linux

리눅스에서도 맥에서와 같이 동일한 방법으로 설정하시면 됩니다. 다만 tar.gz 압축파일 형태로 배포되므로 다음의 명령어를 통해서 압축을 해제합니다.
명령은 root 혹은 sudo 명령을 통해서 진행합니다.

```
tar –C /usr/local -xzf go1.7.4.linux-amd64.tar.gz
```

이후, 맥에서와 같이 GOROOT, GOPATH, PATH 에 대한 환경변수 설정을 합니다.

3. Validating Peer 구동

앞서 기본 환경설정이 완료되었으면 체인코드 개발 및 테스트를 위해 Docker Compose 기반으로 하나의 멤버쉽 서비스와 Validating Peer 를 구동해보도록 하겠습니다.

특정 디렉토리에서 다음의 내용으로 docker-compose.yml 파일을 생성합니다.

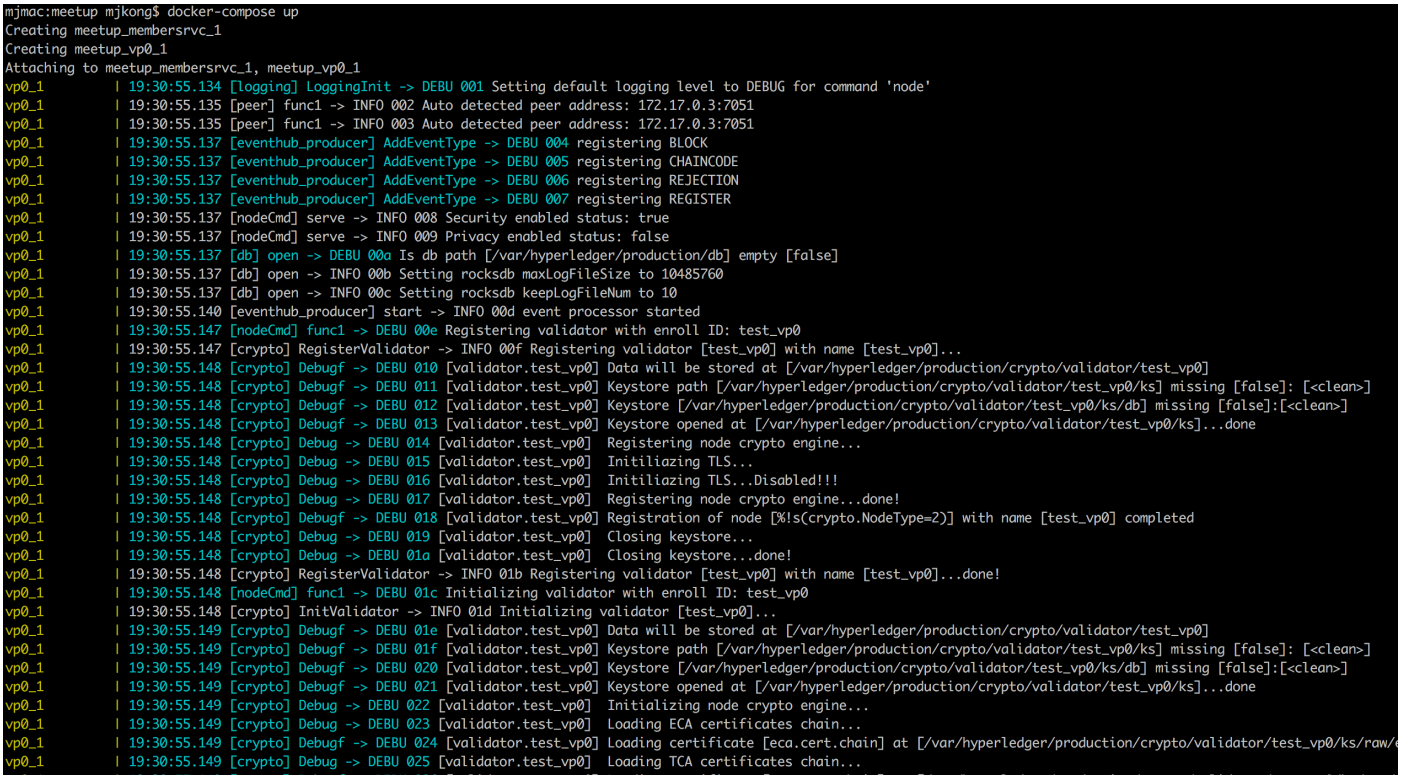
```
membersrvc:
  image: hyperledger/fabric-membersrvc:x86_64-0.6.1-preview
  ports:
      - "7054:7054"
  command: membersrvc
vp0:
  image: hyperledger/fabric-peer:x86_64-0.6.1-preview
  ports:
      - "7050:7050"
      - "7051:7051"
      - "7053:7053"
  environment:
    - CORE_PEER_ADDRESSAUTODETECT=true
    - CORE_LOGGING_LEVEL=DEBUG
    - CORE_PEER_ID=vp0
    - CORE_PEER_PKI_ECA_PADDR=membersrvc:7054
    - CORE_PEER_PKI_TCA_PADDR=membersrvc:7054
    - CORE_PEER_PKI_TLSCA_PADDR=membersrvc:7054
    - CORE_SECURITY_ENABLED=true
    - CORE_SECURITY_ENROLLID=test_vp0
    - CORE_SECURITY_ENROLLSECRET=MwYpmSRjupbT
  links:
    - membersrvc
  command: sh -c "sleep 5; peer node start --peer-chaincodedev"
```

yml 파일의 특성으로 잘못 작성될 경우(빈 공백에 탭문자등 스페이스가 아니 다른 문자가 들어갈 경우나 탭의 위치가 맞지 않을 경우 등) docker 컨테이너 시작이 되지 않으므로 적성된 파일을 공유합니다. 아래의 링크에서 받으시기 바립니다.

https://github.com/mjkong/blockchain_hyperledger/blob/master/docs/Setup/devmode/docker-compose.yml

docker-compose.yml 파일과 동일한 디렉토리에서 다음의 명령어를 실행합니다. 실행하면 다음의 그림에서와 같이 관련 로그가 올라가는 것을 확인 할 수가 있습니다.

```
docker-compose up
```



4. 개발 모드로 스마트 컨트랙 코드(체인코드) 작성

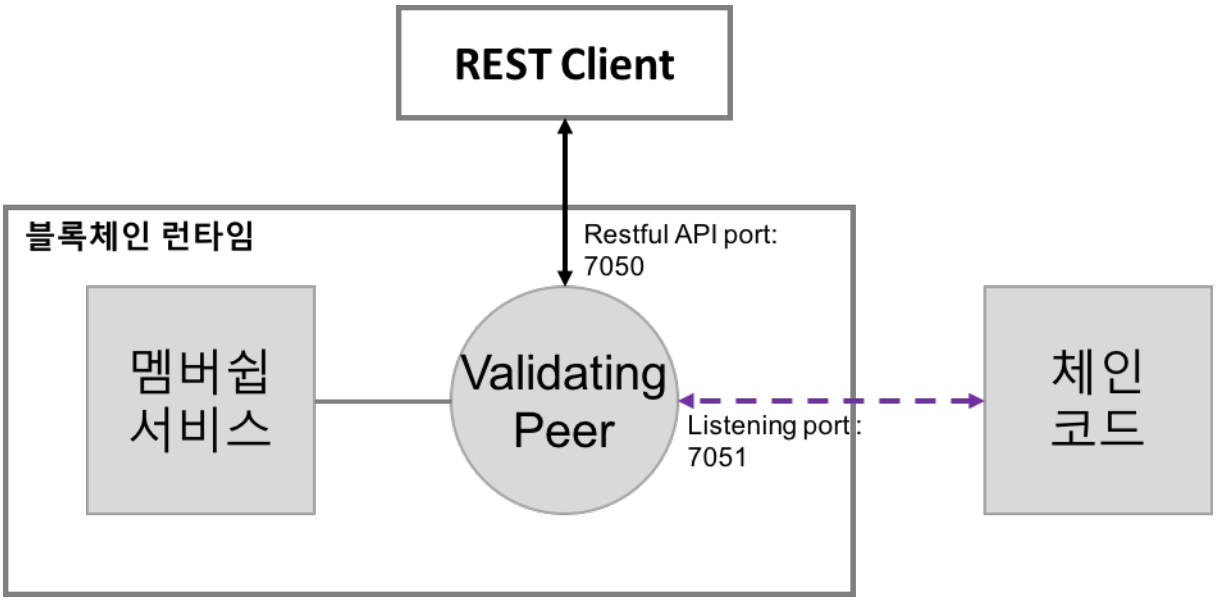
체인코드 개발을 위해서는 앞서 개발환경 준비에서 설정한 GOPATH 아래에 Fabric 의 소스코드를 넣고 빌드하는 과정을 거칩니다.

4.1. 개발모드 vs. 운영모드

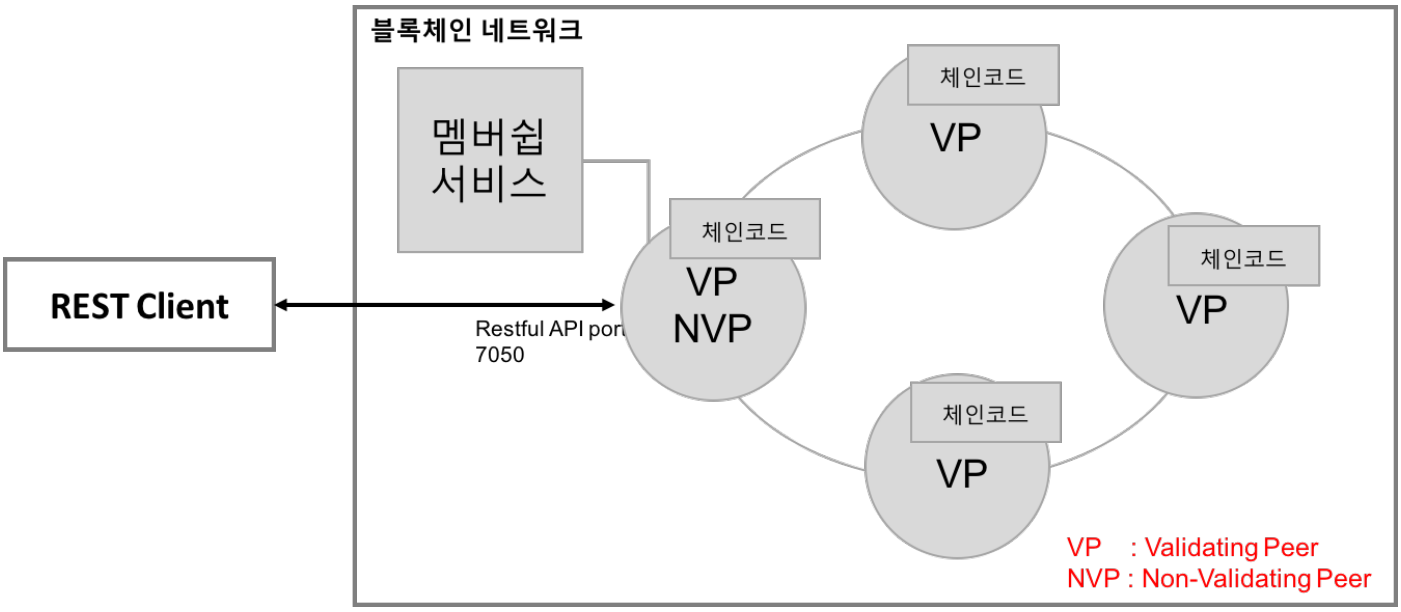
개발모드와 운영모드의 차이점을 설명하면 아래 두 그림을 비교해 보면서 이해하시기 바랍니다.
먼저, 큰 차이점은 개발모드에서는 체인코드가 블록체인 런타임과 별도의 프로세스로 실행된다는 것입니다.

- 운영모드에서는 체인코드를 디플로이하면 블록체인 네트워크에 있는 모든 피어들에게 복제가 되고 모든 피어들이 동일한 체인코드로 동작하게 되지만, 개발모드에서는 단일 피어에 체인코드가 붙어서 동작하는 구조입니다.
- 그래서 체인코드 개발은 체인코드 실행 –> 테스트 –> 디버깅 –> 체인코드 빌드 –> 체인코드 실행의 순환적인 방법으로 이루어집니다.

4.1.1. 개발모드



4.1.2. 운영모드



4.2. Fabric 소스 코드 다운로드

Fabric 의 소스 코드는 git 명령어를 통해 다운 받을 수 있습니다.

- Windows

```
mkdir %GOPATH%₩src₩github.com₩hyperledger
cd %GOPATH%₩src₩github.com₩hyperledger
git clone https://github.com/hyperledger/fabric.git
cd fabric
git checkout v0.6.1-preview
```

- Mac / Linux

```
mkdir -p $GOPATH/src/github.com/hyperledger
cd $GOPATH/src/github.com/hyperledger
```

```
git clone https://github.com/hyperledger/fabric.git
cd fabric
git checkout v0.6.1-preview
```

4.3. 체인코드 빌드

다음의 경로에서 체인코드를 빌드합니다.

- Windows

```
cd %GOPATH%₩src₩github.com₩hyperledger₩fabric₩examples₩chaincode₩go₩chaincode_example02
go build
```

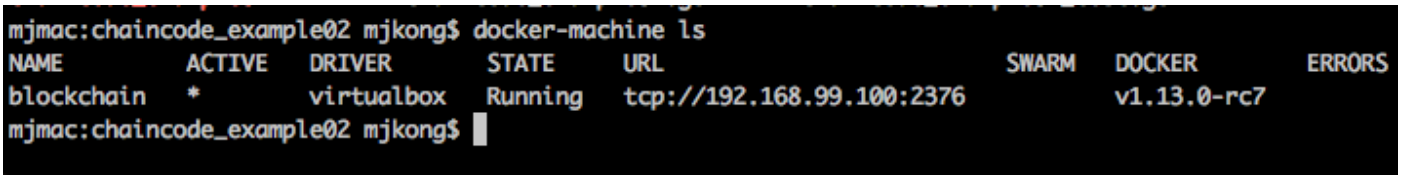
- Mac

```
cd $GOPATH/src/github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
go build
```

4.4. 체인코드 실행

다음의 명령을 통해서 vm 의 IP 를 확인합니다.

```
docker-machine ls
```

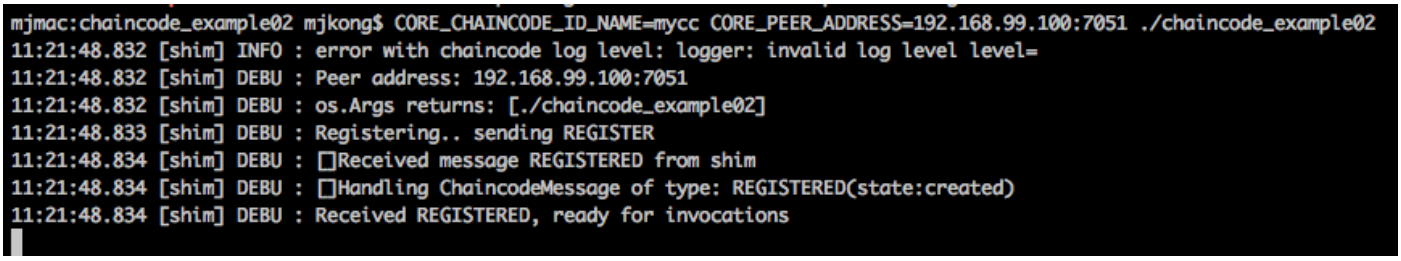


다음의 명령을 통해서 체인코드를 실행합니다.

```
CORE_CHAINCODE_ID_NAME=mycc CORE_PEER_ADDRESS=192.168.99.100:7051 ./chaincode_example02
```

아래와 같이 메세지가 나오면 정상적으로 체인코드가 실행된 것입니다.

(마지막에 "REGISTERED" 메세지 확인)

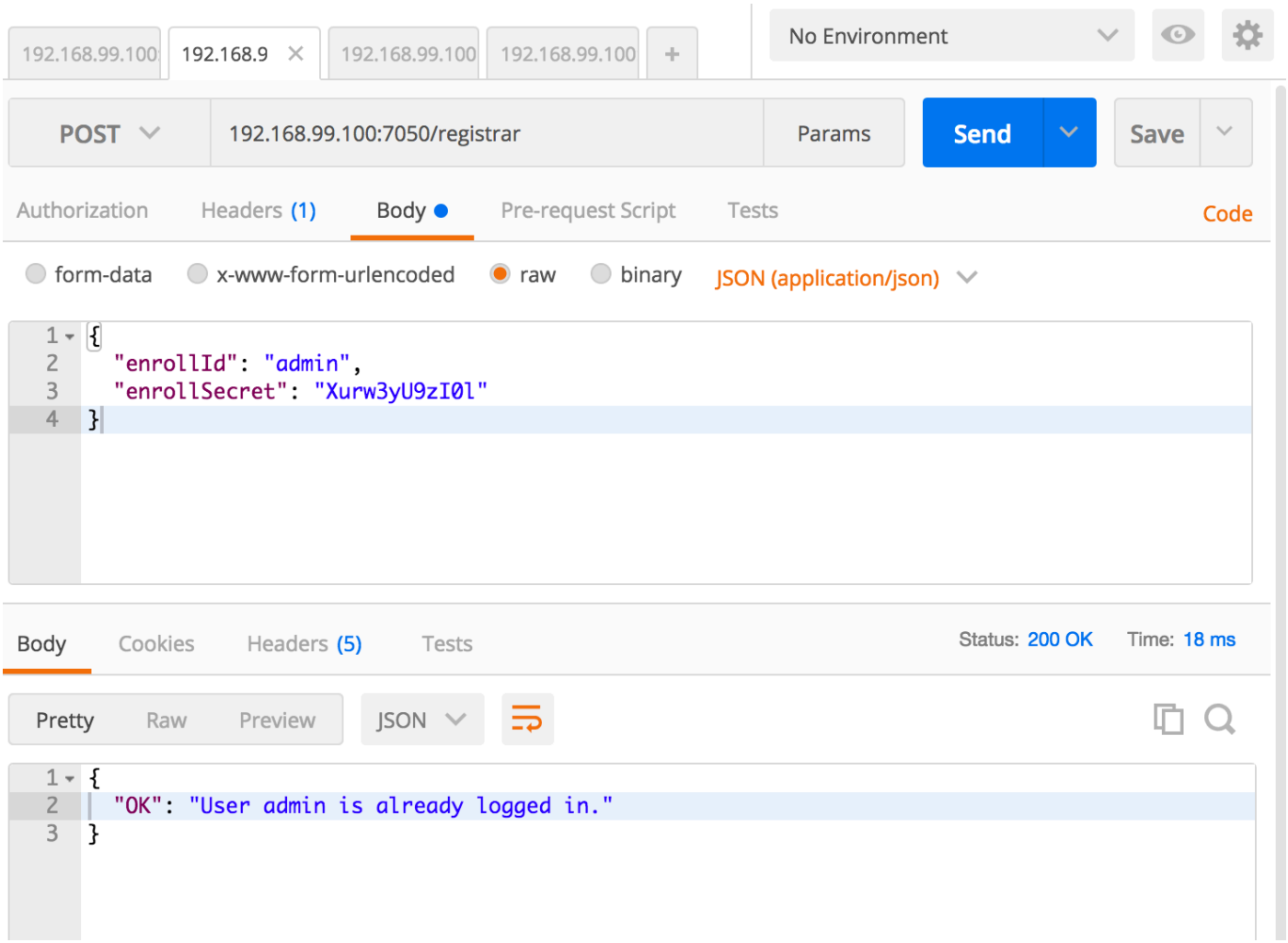


4.5. REST API 를 통한 테스트

4.5.1. 로그인

```
POST <host:port>/registrar
```

```
{
  "enrollId": "admin",
  "enrollSecret": "Xurw3yU9zI0l"
}
```

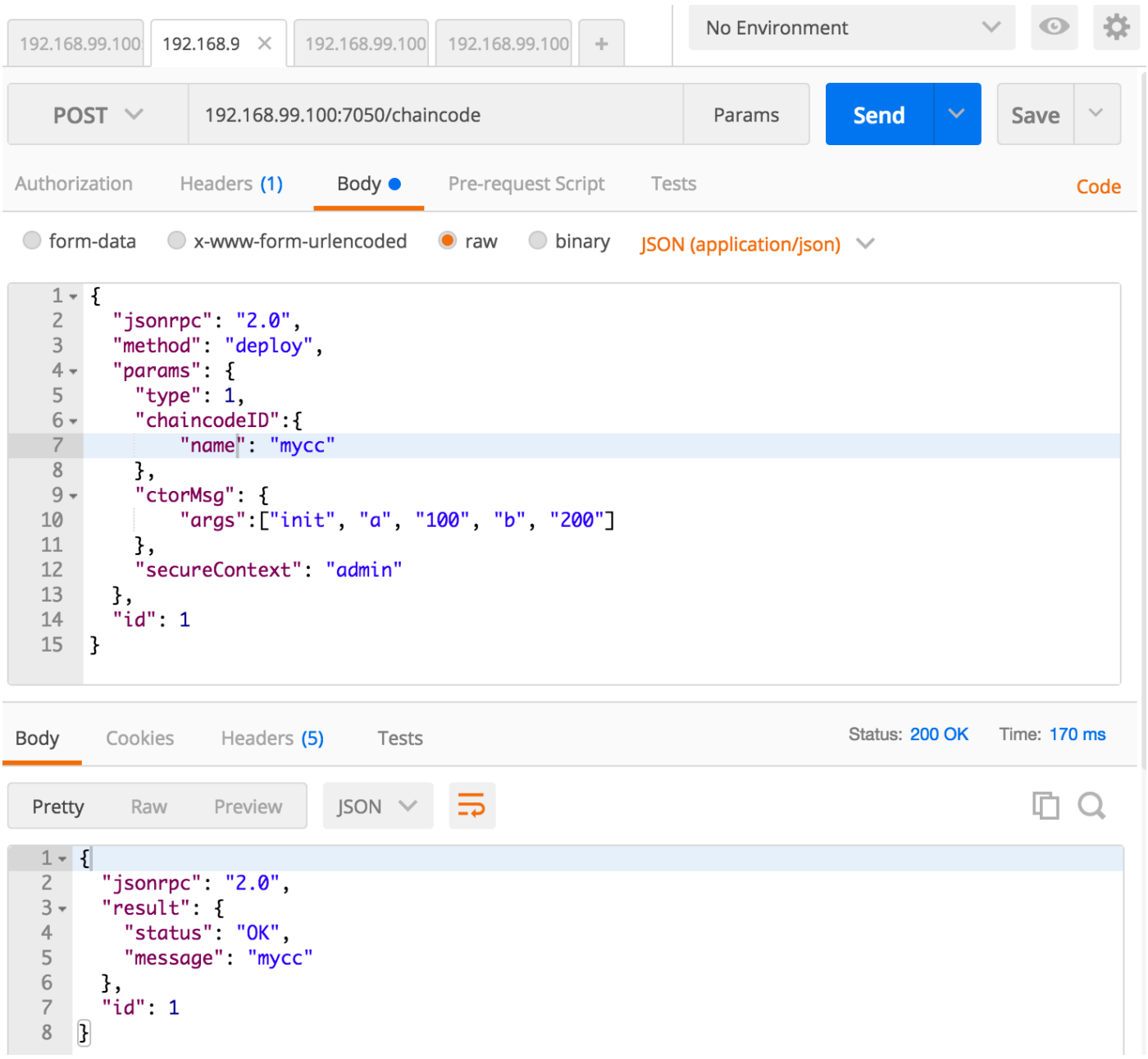


4.5.2. 체인코드 디플로이

```
POST <host:port>/chaincode


{
  "jsonrpc": "2.0",
  "method": "deploy",
  "params": {
    "type": 1,
    "chaincodeID":{
        "name": "mycc"
    },
    "ctorMsg": {
        "args":["init", "a", "100", "b", "200"]
    },
    "secureContext": "admin"
```

```
  },
  "id": 1
}
```



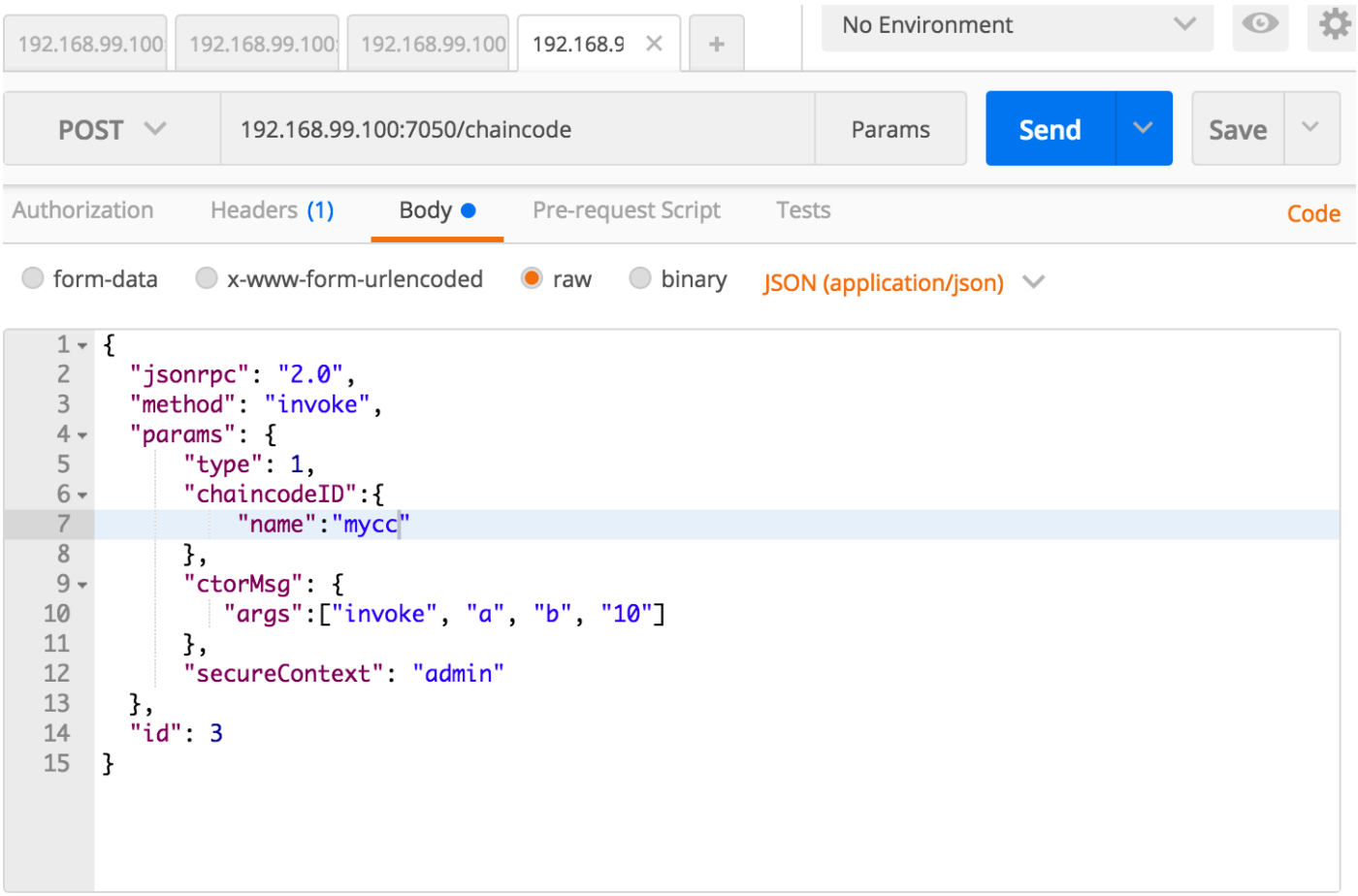
개발 모드에서 체인코드 디플로이는 "chaincodeID" 의 "name" 에 경우 4.3 항목에서 체인코드 실행시 입력한 "CORE_CHAINCODE_ID_NAME"의 값을 입력합니다.

4.5.3. Invoke

```
POST <host:port>/chaincode


{
  "jsonrpc": "2.0",
  "method": "invoke",
  "params": {
      "type": 1,
```

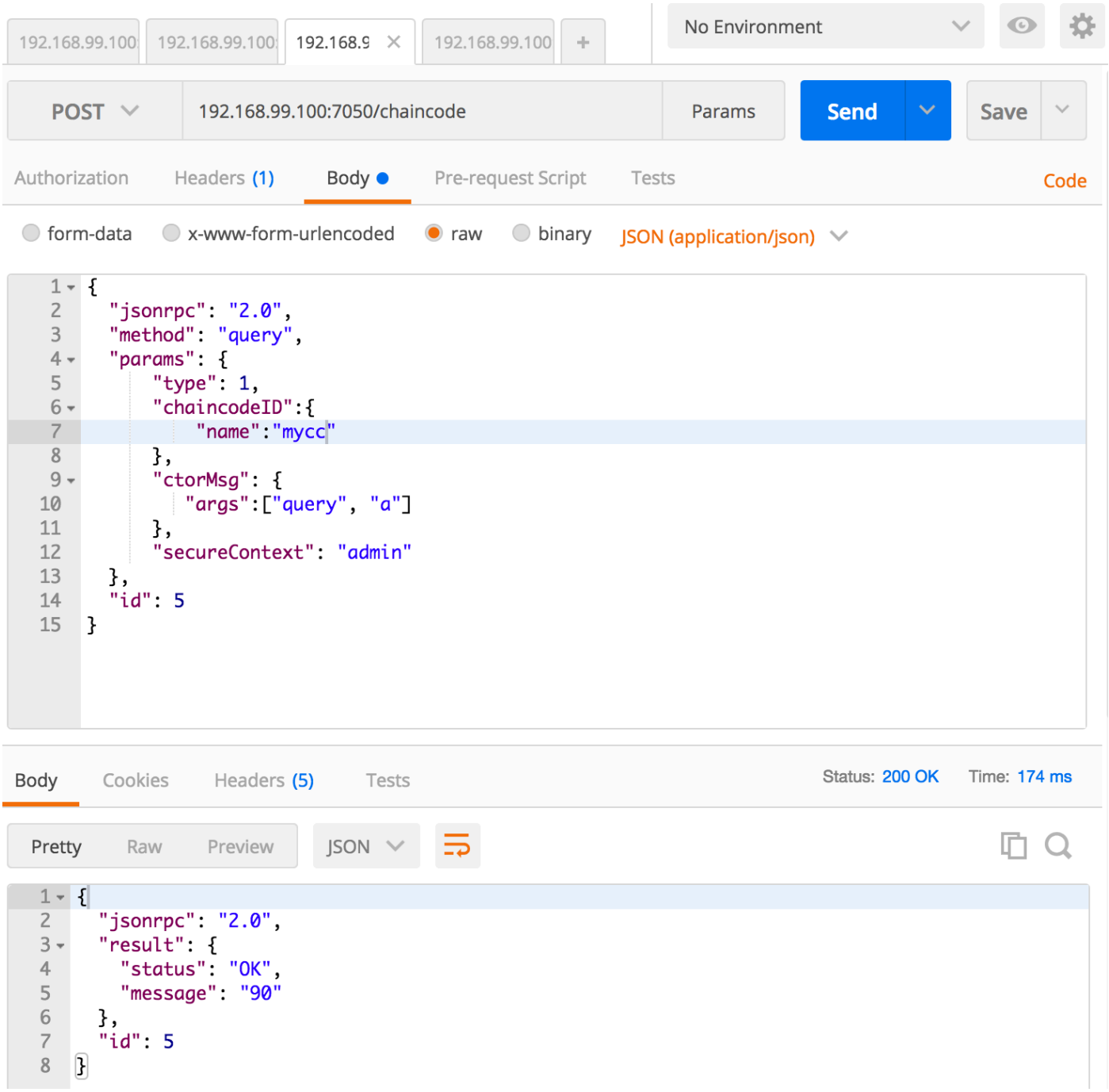
```
    "chaincodeID":{
        "name":"mycc"
    },
    "ctorMsg": {
       "args":["invoke", "a", "b", "10"]
    },
    "secureContext": "admin"
  },
  "id": 3
}
```



4.5.4. Query

```
POST <host:port>/chaincode

{
  "jsonrpc": "2.0",
  "method": "query",
  "params": {
      "type": 1,
      "chaincodeID":{
          "name":"mycc"
      },
      "ctorMsg": {
         "args":["query", "a"]
      },
      "secureContext": "admin"
  },
  "id": 5
}
```


192.168.99.100
192.168.99.100
192.168.9
192.168.99.100
No Environment
POST
192.168.99.100:7050/chaincode
Params
Send
Save
Authorization
Headers (1)
Body
Pre-request Script
Tests
Code
form-data
x-www-form-urlencoded
raw
binary
JSON (application/json)
{
"jsonrpc": "2.0",
"method": "query",
"params": {
"type": 1,
"chaincodeID":{
"name":"mycc"
},
"ctorMsg": {
"args":["query", "a"]
},
"secureContext": "admin"
},
"id": 5
}
Body
Cookies
Headers (5)
Tests
Status: 200 OK
Time: 174 ms
Pretty
Raw
Preview
JSON
{
"jsonrpc": "2.0",
"result": {
"status": "OK",
"message": "90"
},
"id": 5
}

5. 운영 모드로 스마트 컨트랙 코드(체인코드) 배치

5.1. Validating Peer 이미지 수정

운영모드에서 체인코드의 디플로이를 하게되면 validating peer 에서 Docker remote API 접근을 하여 hyperledger/fabric-baseimage 를 기반으로 디플로이된 체인코드의 Docker image 를 생성합니다.
Docker-machine 을 통해 Docker 를 실행하게 되면 기본으로 TLS 가 적용되어서 실행되므로 Validating Peer 에서 Docker remote API 접근을 위해서는 TLS 를 위한 인증서를 가지고 있어야 합니다.
이를 위해서 Peer 이미지에 대한 수정을 한 후 진행하도록 하겠습니다.

터미널 창을 두 개 정도 실행 한 뒤 각 터미널 창에서 개발 PC 의 OS 에 따라 3.2.1 또는 3.2.2 에서와 같이 docker 명령어를 쓸 수 있도록 합니다.

5.1.1. Hyperledger/fabric-peer 컨테이너 생성 후 접속

첫번째 터미널에서 다음의 명령을 실행하여 Docker container 를 생성합니다. 그럼 다음과 같이 프롬프트가 나옵니다.

```
docker run -it hyperledger/fabric-peer bash
```

```
mjmac:blockchain mjkong$ docker run -it hyperledger/fabric-peer bash
root@4435495b2a75:/opt/gopath/src/github.com/hyperledger/fabric#
```

이 동작은 Validating Peer 를 위한 Docker image 인 hyperledger/fabric-peer 를 하나 실행하여 가상환경을 하나 만들고 만들어진 가상환경에 터미널 접속을 한 것과 동일한 현상입니다.

5.1.2. Validating peer 설정파일 수정

vi 를 이용하여 Validating peer 의 설정을 위한 설정파일을 수정합니다.

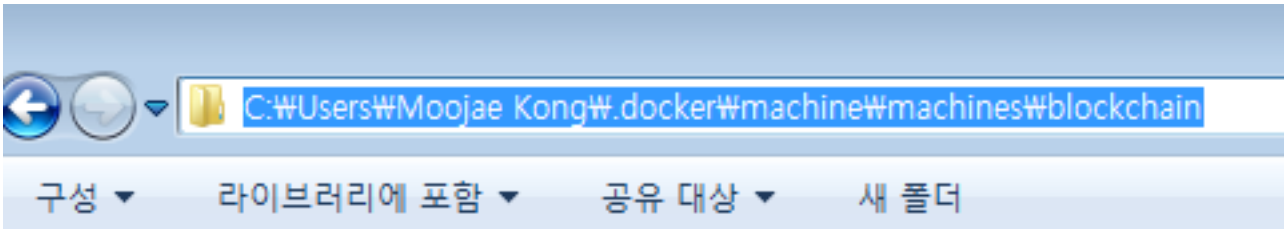
```
cd peer
vi core.yaml
```

```
238
239     # settings for docker vms
240     docker:
241         tls:
242             enabled: true
243             cert:
244                 file: /root/key/cert.pem
245             ca:
246                 file: /root/key/ca.pem
247             key:
248                 file: /root/key/key.pem
249         # Parameters of docker container creat
```

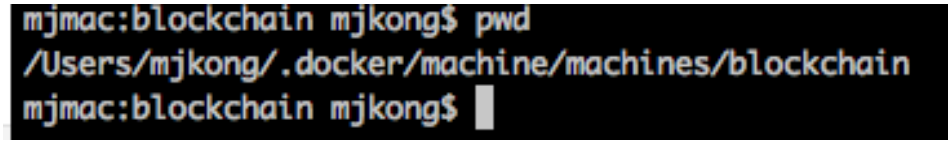
5.1.3. Docker TLS 인증서를 컨테이너로 복사 및 Docker commit 을 통해 새로운 이미지 생성

앞서 실행한 터미널 창 중 두번째 창에서 사용자 계정의 홈 디렉토리 아래의 docker-machine 인스턴스 디렉토리로 들어갑니다. 각 OS 별로 다음의 위치를 참조합니다. 경로의 제일 마지막 디렉토리는 docker-machine create 에서 생성된 인스턴스 명입니다.

5.1.3.1. Windows



5.1.3.2. Mac



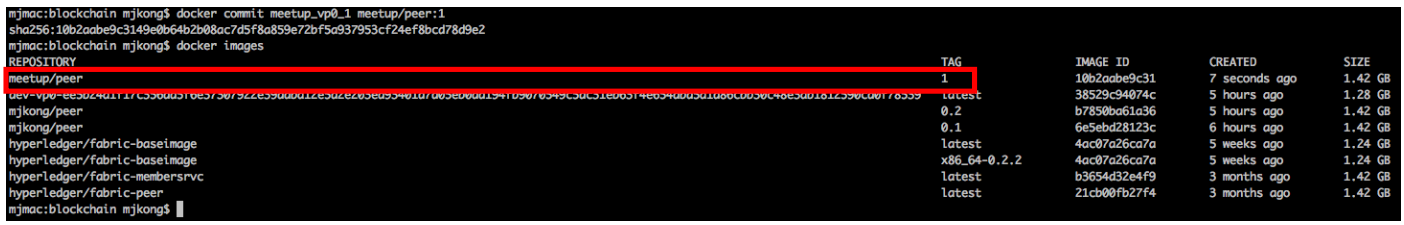
인스턴스 디렉토리에서 key 라는 이름의 디렉토리를 생성하고 ca.pem, cert.pem, key.pem 파일을 복사합니다.

그리고 다음의 명령을 통해서 복사해야 할 컨테이너 명을 확인 합니다.

```
docker ps
```

그 후 다음의 명령어를 통해서 key 디렉토리를 컨테이너 안으로 복사한 후 docker commit 명령을 통해서 새로운 이미지를 생성합니다.

```
docker cp ./key meetup_vp0_1:/root/key
docker commit meetup_vp0_1 meetup/peer:1
```



5.1.4. 새로운 이미지를 사용하여 컨테이너 실행

앞서 개발모드에서 생성한 docker-compose.yml 파일 수정 또는 새로운 디렉토리에 docker-compose.yml 파일을 생성해서 테스트 합니다.

운영모드 docker-compose.yml 파일

https://github.com/mjkong/blockchain_hyperledger/blob/master/docs/Setup/prodmode/docker-compose.yml

```
membersrvc:
  image: hyperledger/fabric-membersrvc:x86_64-0.6.1-preview
  ports:
      - "7054:7054"
  command: membersrvc
vp0:
  image: hyperledger/fabric-peer: :x86_64-0.6.1-preview
  ports:
      - "7050:7050"
      - "7051:7051"
      - "7053:7053"
  environment:
    - CORE_PEER_ADDRESSAUTODETECT=true
    - CORE_VM_ENDPOINT=https://192.168.99.100:2376
    - CORE_LOGGING_LEVEL=DEBUG
    - CORE_PEER_ID=vp0
    - CORE_PEER_PKI_ECA_PADDR=membersrvc:7054
    - CORE_PEER_PKI_TCA_PADDR=membersrvc:7054
    - CORE_PEER_PKI_TLSCA_PADDR=membersrvc:7054
    - CORE_SECURITY_ENABLED=true
    - CORE_SECURITY_ENROLLID=test_vp0
    - CORE_SECURITY_ENROLLSECRET=MwYpmSRjupbT
  links:
    - membersrvc
  command: sh -c "sleep 5; peer node start"
```

여기서 개발모드 때와의 차이점은 "CORE_VM_ENDPOINT"의 항목 추가와 제일 마지막 command 가 변경되어야 하는 점입니다. "CORE_VM_ENDPOINT" 항목은 docker-machine ls 명령어를 통해 확인 할 수 있습니다.

```
mjmac:blockchain mjkong$ docker-machine ls
NAME         ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER        ERRORS
blockchain   *        virtualbox   Running   tcp://192.168.99.100:2376           v1.13.0-rc7
mjmac:blockchain mjkong$
```

command 는 "--peer-chaincodedev" 항목을 삭제합니다.

다음의 명령을 통해서 컨테이너를 실행하고 로그에 에러가 없음을 확인합니다.

. (docker-compose.yml 파일 위치에서 실행합니다.)

```
docker-compose up
```

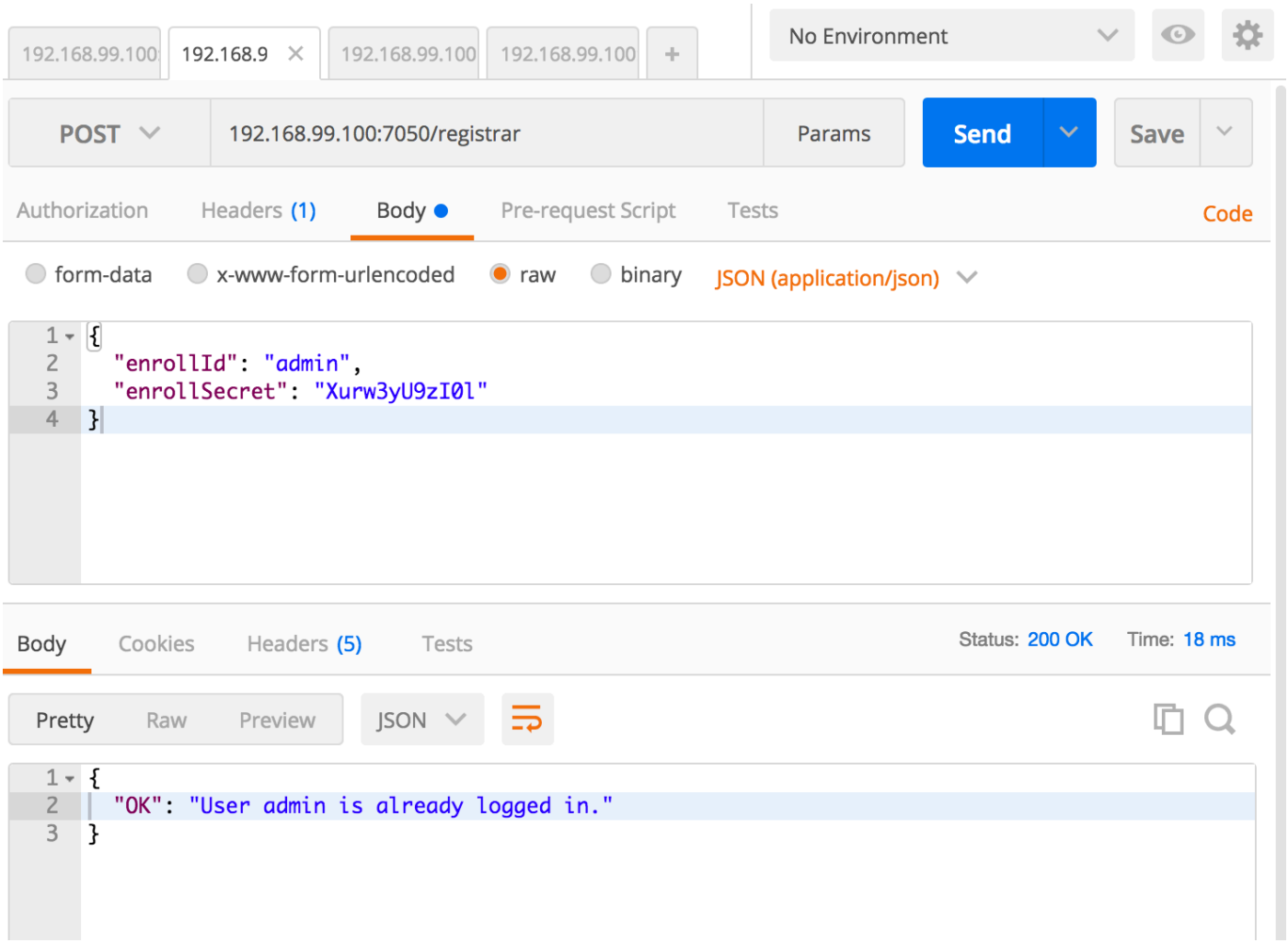
5.1.5. REST API 를 통한 테스트

정상적으로 컨테이너가 실행이 되었으면, REST API 를 통해서 샘플 체인코드를 디플로이하여 동작을 확인합니다. REST API 테스트는 개인별로 사용하기 편리한 클라이언트를 사용해도 무방합니다. 다음의 순서대로 테스트 합니다.

5.1.5.1. 로그인

```
POST <host:port>/registrar


{
  "enrollId": "admin",
  "enrollSecret": "Xurw3yU9zI0l"
}
```

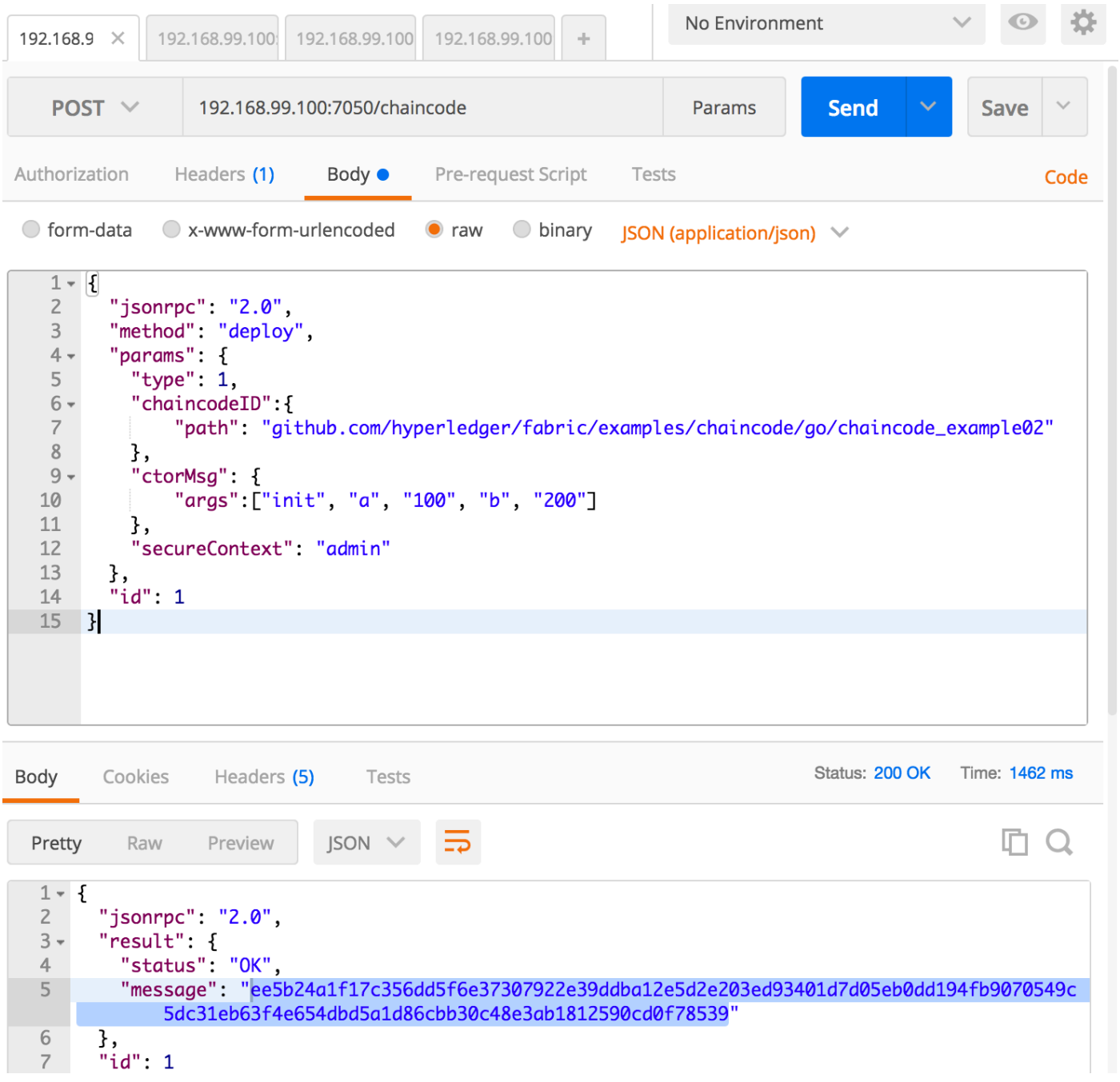


5.1.5.2. 체인코드 디플로이

```
POST <host:port>/chaincode


{
  "jsonrpc": "2.0",
  "method": "deploy",
  "params": {
    "type": 1,
```

```
    "chaincodeID":{
        "path": "github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02"
    },
    "ctorMsg": {
        "args":["init", "a", "100", "b", "200"]
    },
    "secureContext": "admin"
  },
  "id": 1
}
```



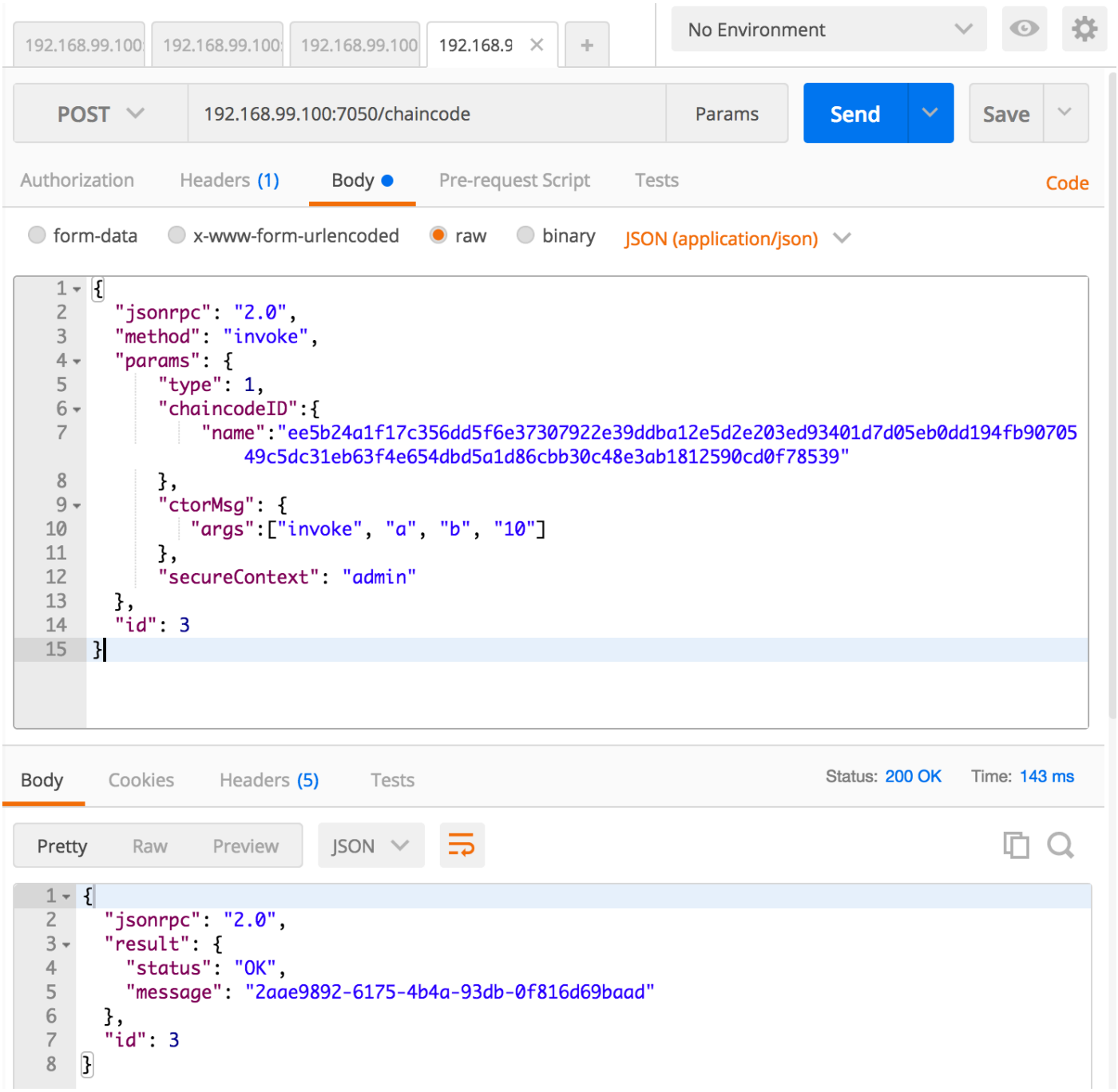
체인코드 디플로이의 경우 Response 에 있는 message 의 값을 기록해 두어야 합니다. 이 값은 chaincode id 라고 부르는 값이며 향후 체인코드에 대한 invoke, query API 콜에서 사용을 합니다.

5.1.5.3. Invoke

```
POST <host:port>/chaincode

{
  "jsonrpc": "2.0",
  "method": "invoke",
  "params": {
      "type": 1,
      "chaincodeID":{

"name":"ee5b24a1f17c356dd5f6e37307922e39ddba12e5d2e203ed93401d7d05eb0dd194fb9070549c5dc31eb63f
4e654dbd5a1d86cbb30c48e3ab1812590cd0f78539"
      },
      "ctorMsg": {
         "args":["invoke", "a", "b", "10"]
      },
      "secureContext": "admin"
  },
  "id": 3
}
```



5.1.5.4. Query

```
POST <host:port>/chaincode


{
  "jsonrpc": "2.0",
  "method": "query",
  "params": {
      "type": 1,
      "chaincodeID":{

"name":"ee5b24a1f17c356dd5f6e37307922e39ddba12e5d2e203ed93401d7d05eb0dd194fb9070549c5dc31eb63f4e654dbd5a1d86cbb30c48e3ab1812590cd0f78539"
```

```
      },
      "ctorMsg": {
         "args":["query", "a"]
      },
      "secureContext": "admin"
  },
  "id": 5
}
```