솔리디티 언어 기초지식

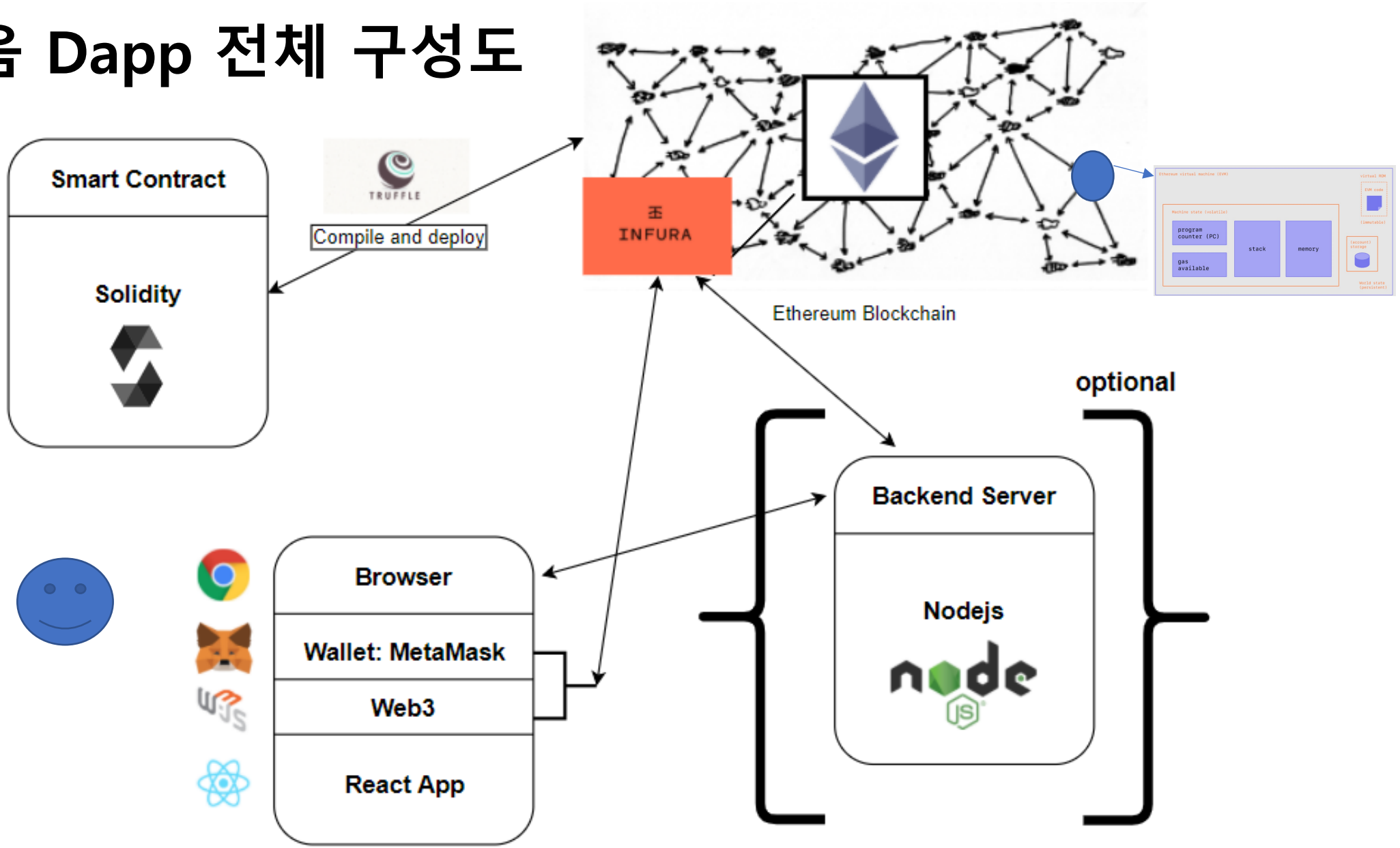
김동환

dannykim@kakao.com

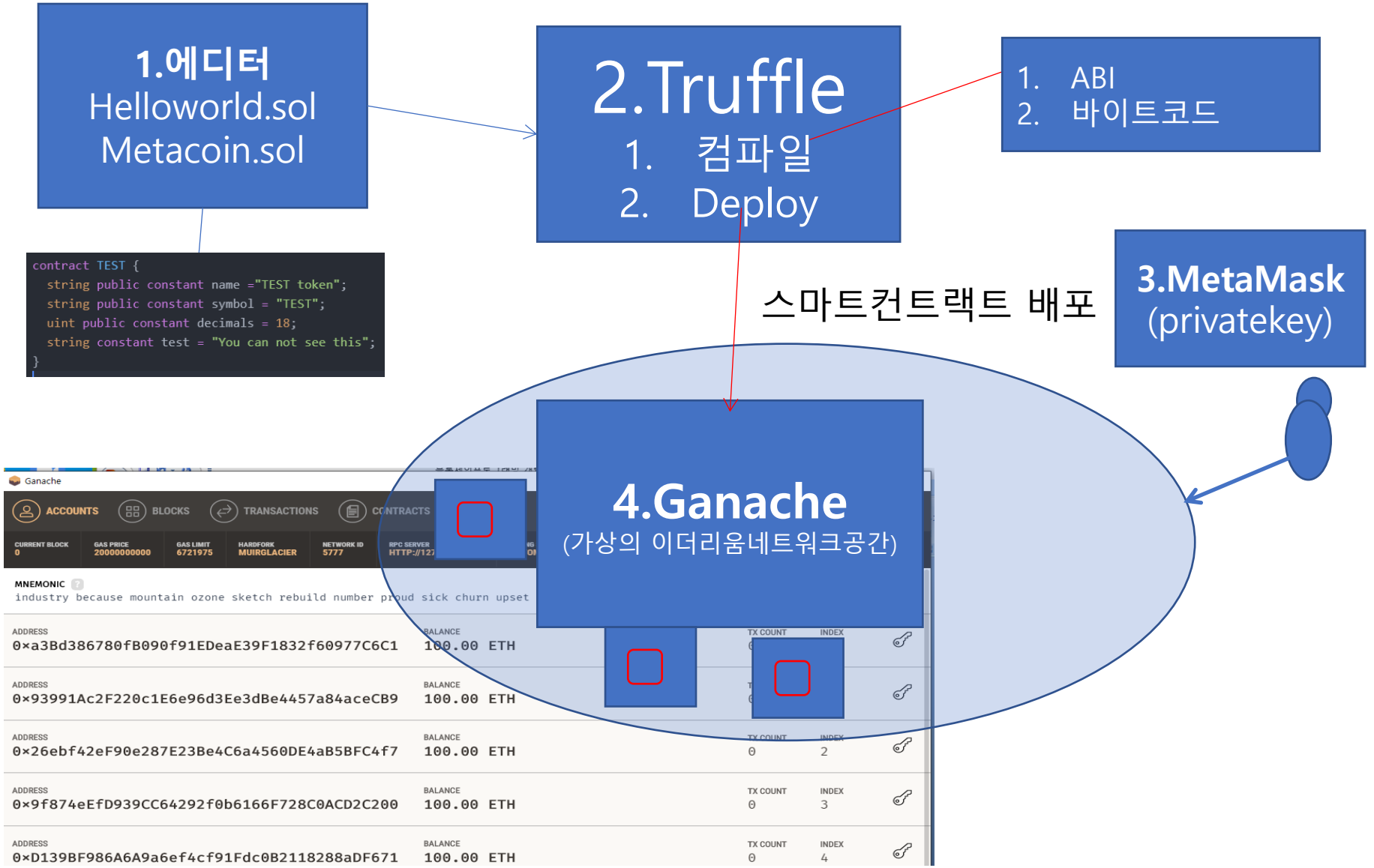
https://kimsfamily.kr/page/Profile

Dapp 개발환경

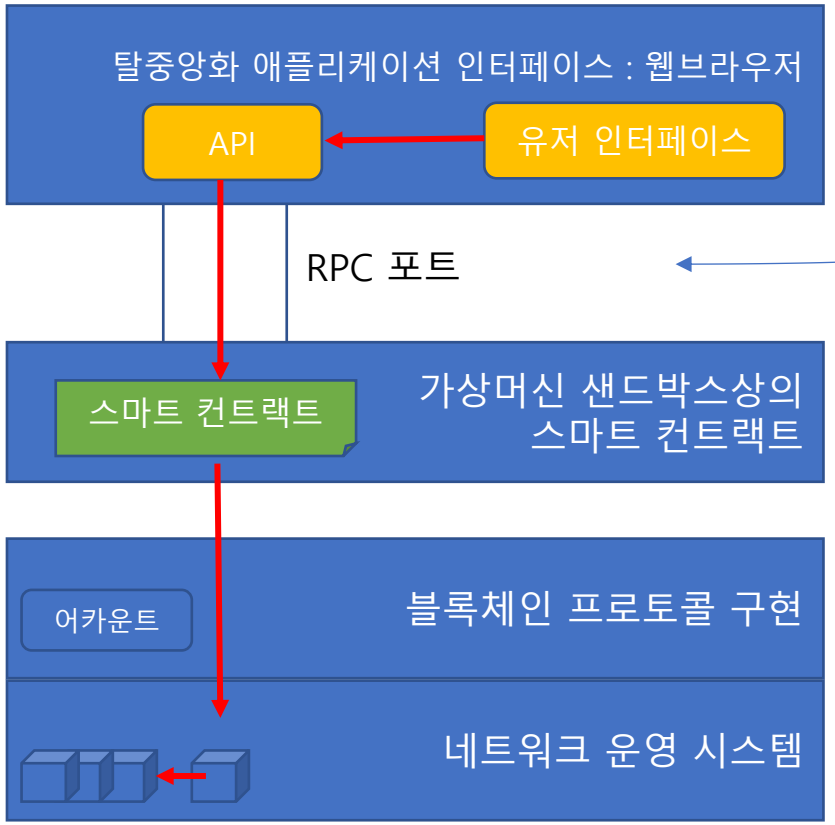
이더리움 Dapp 전체 구성도



스마트컨트랙트 개발환경 및 구성도



Dapp 네트워크 구성도

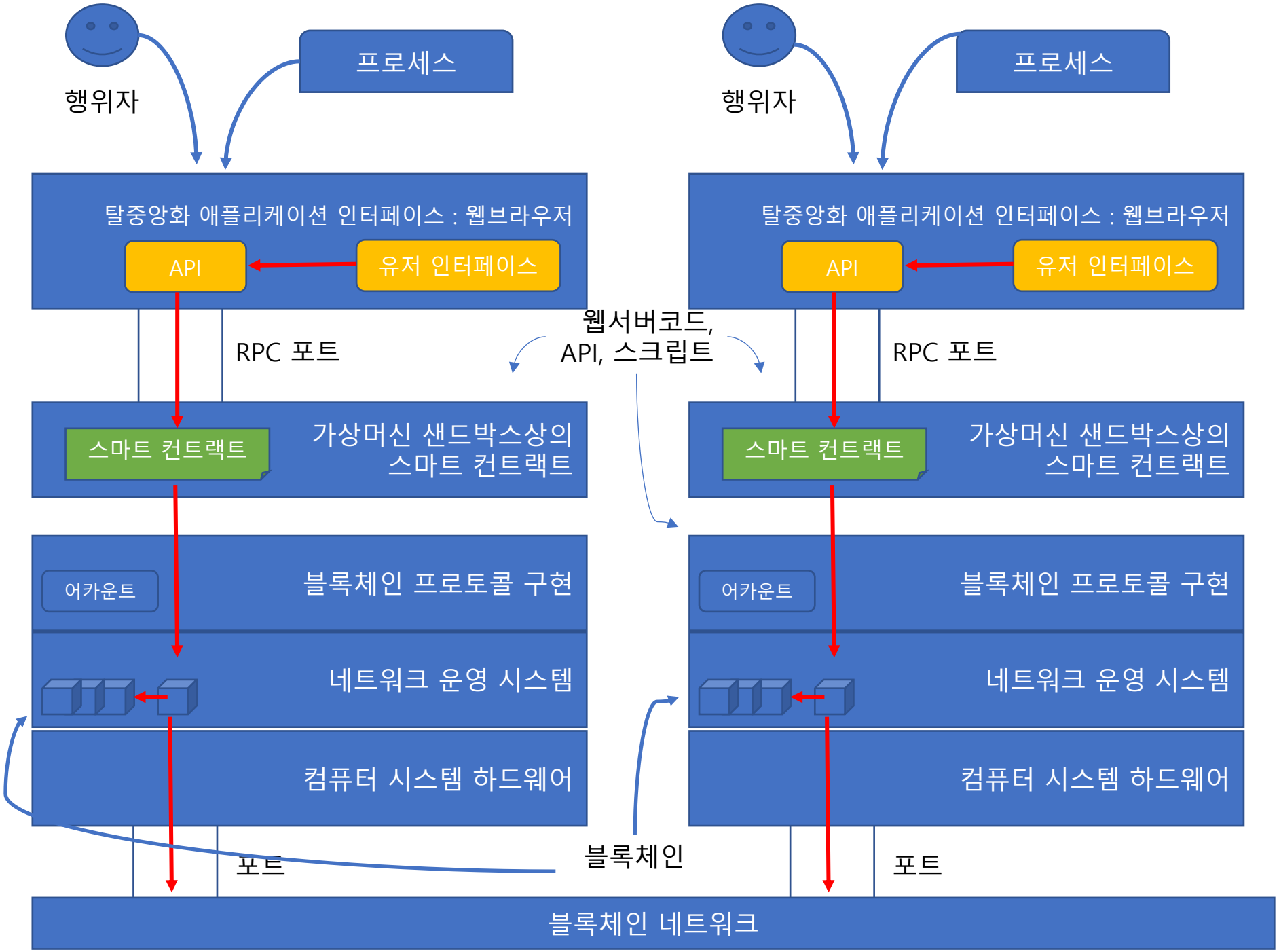


1. 웹 애플리케이션 프로그래밍

2. 서버 환경설정
Dapp UI를 스마트컨트랙트와
결합하는 코드

3. 스마트 컨트랙트 프로그래밍

4. 블록체인 클라이언트
/테스트 체인 : 가나쉬를 사용


행위자
프로세스
탈중앙화 애플리케이션 인터페이스 : 웹브라우저
API
유저 인터페이스
RPC 포트
스마트 컨트랙트
가상머신 샌드박스상의
스마트 컨트랙트
어카운트
블록체인 프로토콜 구현
네트워크 운영 시스템
컴퓨터 시스템 하드웨어
포트
웹서버코드,
API, 스크립트
블록체인
블록체인 네트워크

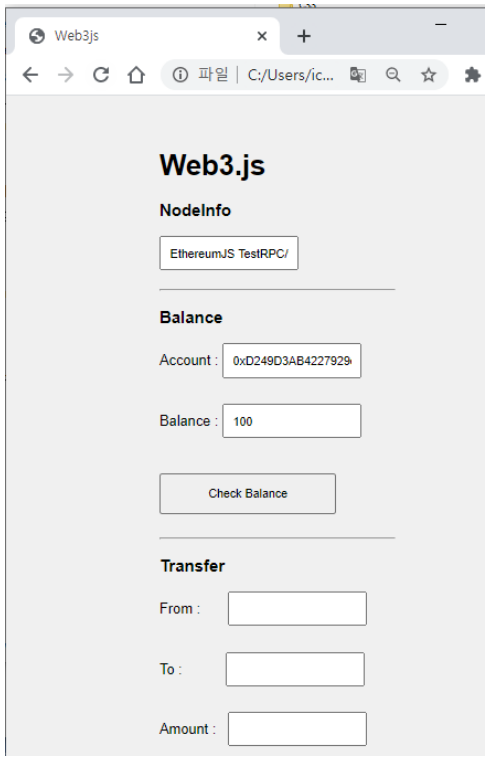
Dapp 개발자

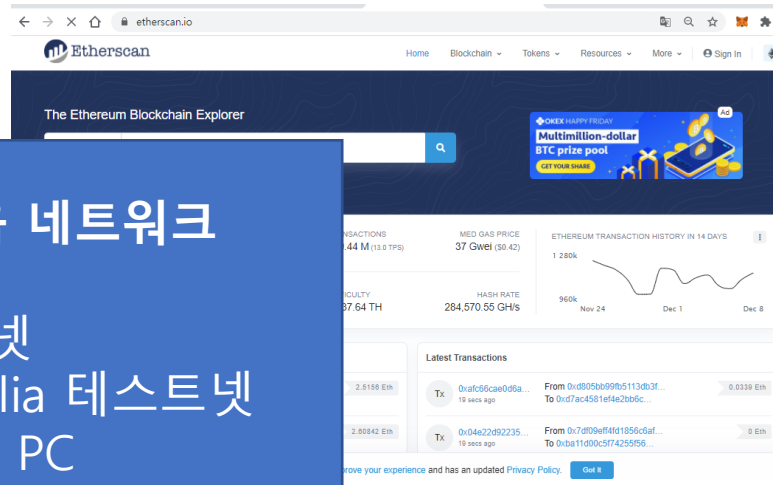
Dapp 개발 및 웹응용프로그램과의 활용

지갑앱

WEB3.js

HTML
CSS
JavaScript

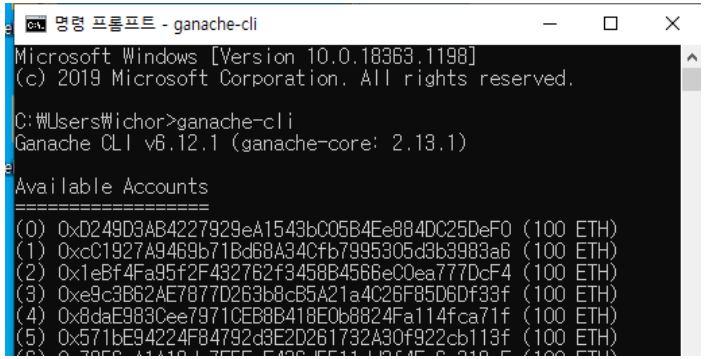




이더리움 네트워크

1. 메인넷
2. Sepolia 테스트넷
3. Local PC

스마트
컨트랙트
Solidity



솔리디티 개발자
<스마트컨트랙트 개발자>

솔리디티 언어

솔리디티 개념

- 솔리디티는 스마트 계약을 구현하기 위한 객체지향 고급언어
- 스마트계약은 이더리움 상태 내에서 계정의 동작을 관리하는 프로그램
- 솔리디티는 EVM을 대상으로 설계된 중괄호(Curly-bracket) 언어임
- C++, 파이썬, 자바스크립트의 영향을 받음
- 솔리디티는 정적으로 유형이 지정되며 다른 기능중에서 상속, 라이브러리 및 복잡한 사용자 정의 유형을 지원함
- 솔리디티를 사용해서 크라우드펀딩, 블라인드 경매, 다중서명지갑 같은 계약 생성할 수 있음

간단한 스마트컨트랙트

- 상태변수의 값을 설정하고 해당 상태변수를 다른 컨트랙트, 계정이 사용할 수 있도록 하는 예

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.16 <0.9.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {    22498 gas
        storedData = x;
    }

    function get() public view returns (uint) {    2437 gas
        return storedData;
    }
}
```

ch1_1.sol

블록체인 기초

트랜잭션(Transaction)

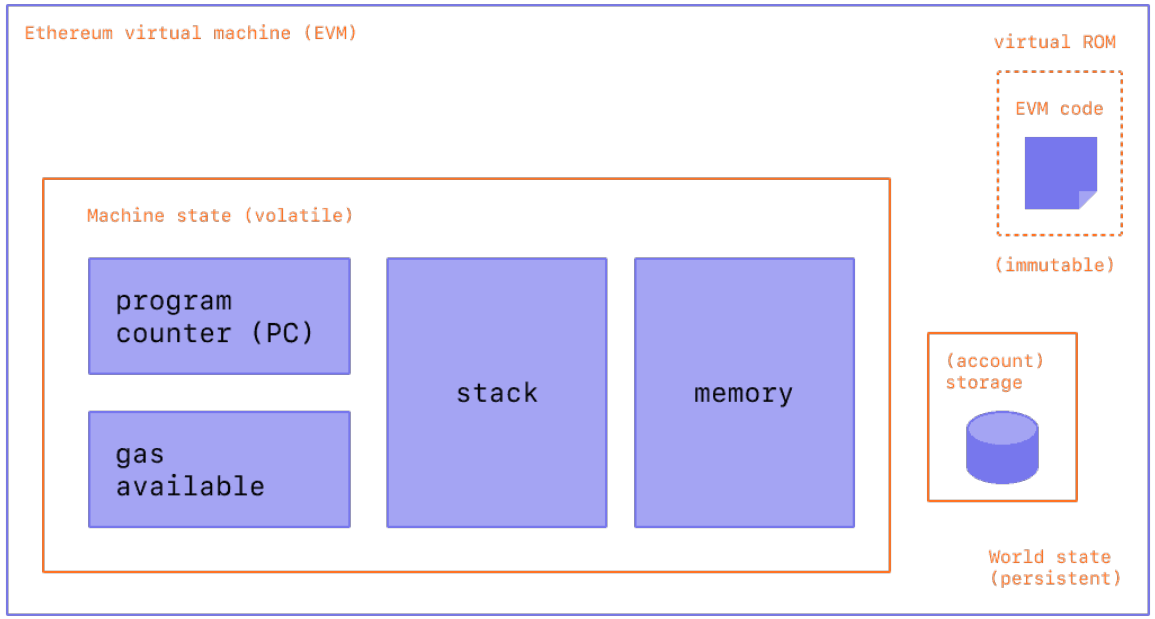
- 블록체인은 전 세계적으로 공유되는 트랜잭션 데이터베이스
  - 모든 사람이 네트워크에 참여하기만 하면 데이터베이스의 항목을 읽을 수 있다는 걸 의미
- 데이터베이스의 내용을 변경하려면 다른 모든 사람들이 수락해야하는 ' 트랜잭션'을 생성해야 함
- ' 트랜잭션'은 수행하려는 변경(동시에 두 값을 변경한다고 가정)이 전혀 수행되지 않거나 완전히 적용되었음을 의미
- ' 트랜잭션'이 데이터베이스에 적용되는 동안에는 다른 트랜잭션이 이를 변경할 수 없음
- 예) 전자화폐로 된 모든 계좌의 잔액을 나열하는 테이블
  - 한 계좌에서 다른 계좌로 이체하는 경우, 한 계좌에서 차감, 다른 계좌에서 추가(동시적으로)
- 거래는 보낸사람(생성자)에 의해 암호화된 방식으로 서명됨-> 엑세스에 대한 보호!!

블록(Block)

- 극복해야 할 장애물 중 하나는 '이중지불공격'
  - 네트워크에 계정을 비우려는 두개의 트랜잭션이 존재하는 경우, 하나만 유효함.
  - 일반적으로 먼저 승인되는 트랜잭션임, P2P에서는 '최초'가 객관적인 용어가 아님
  - 개발자는 이를 신경 쓰지 않아도 됨, 전 세계적으로 허용되는 거래 순서가 선택되어 충돌을 해결함
- 트랜잭션은 '블록'으로 묶인 다음 실행되어 모든 참여 노드에 분산됨
  - 두 거래가 서로 모순되면 두번째 거래가 거부되고 블록의 일부가 되지 않음
  - 시간에 따라서 선형적(linear) 순서를 형성함
- "주문 선택 매커니즘(마이닝)"의 일부로 블록이 때때로 되돌려지는(revert) 일 발생(체인의 끝에서)
  - 블록 위에 더 많은 블록이 쌓이면 이 가능성은 점차 줄어 듬
  - 거래가 다음블록, 미래에 특정블록에 포함된다는 보장은 없음(채굴자에 달려 있음)

이더리움 가상머신(EVM-Ethereum Virtual Machine)

- EVM은 스마트계약을 위한 이더리움 런타임 환경임
- 샌드박스 처리되며 실제로 완전히 격리되어 있음
- EVM 내부에서 실행되는 코드는 네트워크, 파일시스템, 기타 프로세스에 엑세스 할 수 없음
- 스마트계약은 다른 스마트계약에 대한 엑세스도 제한되어 있음



계정(Accounts)

- 외부계정(External Accounts) : Public-private 키에 의해 컨트롤 되는 계정(사람에 의해 관리)
- 계약계정(Contraact Accounts) : 코드에 이해 컨트롤되는 계정
- 외부계정은 공개키에 의해 결정, 계약계정은 계약이 생성된 시점에 결정됨
- 계정에 코드가 저장되어 있는지 여부와 관계없이 EVM에서 동일하게 취급됨
- 모든 계정에는 256비트단어를 256비트단어로 매핑하는 영구 key-value 저장소 (storage)가 있음
- 모든 계정에는 Ether(정확하게는 wei) 잔액이 있으며 Ether를 포함하는 거래를 전송하여 수정가능
- 1 ether = 10**18 wei

트랜잭션(transactions)

- 트랜잭션은 한 계정에서 다른 계정으로 전송되는 메세지임
- 바이너리 데이터('payload'라고 함)와 Ether가 포함될 수 있음
- 대상 계정에 코드가 포함되어 있으면 해당 코드가 실행되고 페이로드가 입력 데이터로 제공됨
- 대상 계약이 설정되지 않은 경우(수신자가 없거나 수신자가 null인 경우), 트랜잭션은 새계약을 생성함
- 계약 주소는 보낸사람과 전송된 트랜잭션 수("nonce")에서 파생된 주소
- 계약 생성 트랜잭션의 페이로드는 EVM 바이트코드로 간주되어 실행됨
- 실행의 계약코드는 영구적으로 저장됨

가스(Gas)

- 생성 시 각 트랜잭션에는 트랜잭션 창시자가 지불해야 할 일정량의 가스(tx.orgin)가 부여됨
- EVM이 실행되는 동안 가스는 특정규칙에 따라 점차적으로 고갈됨
- 어느 시점에서든 가스가 모두 소모되면 부족 예외 트리거가 실행되어 실행 종료, 현재 호출프레임의 상태에 대한 모든 수정사항이 되돌려짐(revert)
- EVM 실행시간의 경제적인 사용을 장려, EVM 실행자(채굴자/스테이커)의 작업에 대한 보상을 함
- 각 블록에는 최대 가스량이 있으므로 블록을 검증하는데 필요한 작업량도 제한됨
- 가스가격(gas price)은 EVM 실행자에게 선불로 지불해야 하는 거래 게시자가 설정한 값임
- 실행 후 가스가 남으면 거래 발신자에게 환불됨, 변경사항을 되돌리는 예외의 경우는 환불안됨
- EVM 실행자는 트랜잭션을 포함할지 여부를 선택 할 수 있음

스토리지, 메모리, 스택

- EVM에는 데이터를 저장할 수 있는 세가지 영역 존재(스토리지, 메모리, 스택)
- 각 계정에는 함수 호출과 트래잭션간에 지속되는 스토리지(storage)라는 데이터 영역 있음
- 스토리지는 256비트 단어를 256비트 단어로 매핑하는 key-value 값 저장소임
- 컨트랙트 내에서 스토리지를 열거하는 건 불가능, 읽고 초기화, 수정에 비용이 많이 듬
- 영구 저장소에 저장하는 항목을 계약실행에 필요한 항목으로 최소화 해야함

- 메모리 : 계약은 각 메세지 호출에 대한 새로워진 인스턴스를 얻음
- 메모리는 선형이며 바이트 수준에서 주소를 지정할 수 있지만 읽기는 256비트 너비로 제한됨
- 쓰기는 8비트 또는 256비트 너비가 될 수 있음
- 이전에 변경되지 않은 메모리 워드에 엑세스 할때 메모리는 워드(256비트)만큼 확장됨
- 증설시에는 가스비를 지불해야 함, 메모리는 크기가 커질수록 비용이 더 많이 듬(2차적으로 확장됨)

스토리지, 메모리, 스택

- EVM은 레지스터 머신이 아니라 스택머신이므로 모든 계산은 스택이라는 데이터 영역에서 수행됨
- 최대크기는 1024개 요소이며, 256비트 워드를 포함함
- 최상위 16개 요소중 하나를 스택 맨 위로 복사하거나 최상위 요소를 그 아래의 16개 요소 중 하나로 교체할 수 있음
- 다른 모든 작업은 스택에서 최상위 2개(또는 작업에 따라 하나 이상) 요소를 가져와 결과를 스택에 푸시함
- 스택 요소를 스토리지나 메모리로 이동할 수 있지만, 스택의 상단을 제거하지 않고는 스택의 더 깊은 임의 요소에 엑세스하는것은 불가능함