



Smart Contract on NexRes

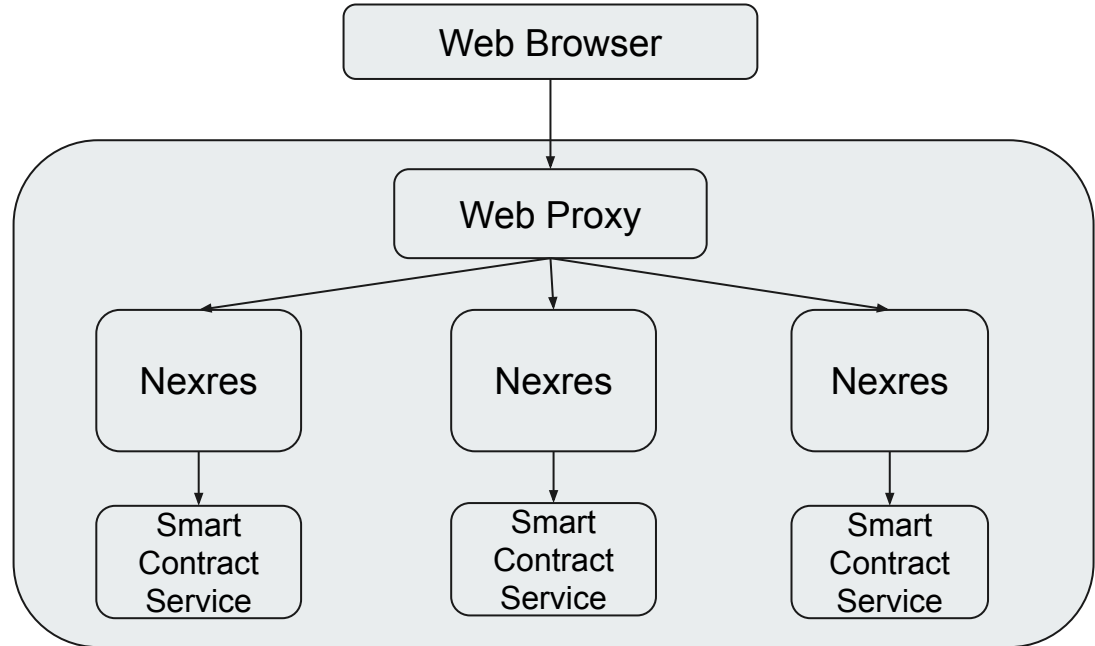
ECS 265 Final Project Mid term report

Team members:

Tong Zhu, Hongxiang Zhang, Siyuan Liu, Yifeng Shi, Junchao Chen

System Architecture

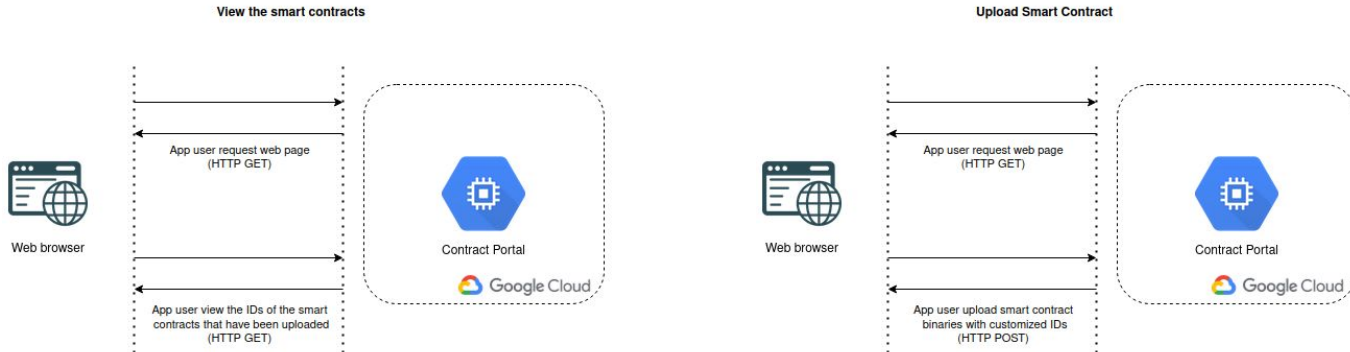
1. Users submit their smart contracts or transactions throughout their webbrowsers.
2. Web Proxy forward the smart contracts and transactions to Nexres to save them into each node.
3. Each smart contract and transaction will be executed by a local smart contract service once they are committed by Nexres



Web Browser

Web Browser is the interface used by the users. User can upload and review their contracts, and submit their transactions with a contract id.

We Use Contract Portal as our application platform. Contract Portal is a web application for users to upload, view and delete the smart contracts. The backend API of the web application is developed with .NET Core. The frontend is developed with React. The web application is deployed and running in docker containers hosted on Google Cloud Compute Engine service.





Upload and review smart contracts

Contract Portal

[View](#) [Upload](#)

Upload a smart contract binary

Choose file







Contract Name

Smart contract *contract2.sol* uploaded successfully ✕

Contract Portal

[View](#) [Upload](#)

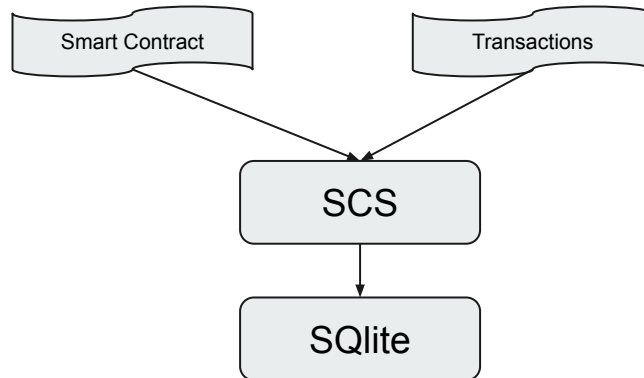
Smart Contracts

File Name	Actions
contract0.sol	 
contract1.sol	 
contract2.sol	 

Smart Contract Service (SCS)

Smart Contract Service(SCS) is a service handling the user contracts and transactions. All the contracts will be uploaded to the SCS to register and all the transactions will be executed inside the SCS.

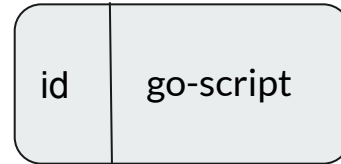
It uses SQLite as a backend database to storage the transaction results.



Smart Contract

Smart Contract contains a go-script written by the users. It contains multiple functions and a unique contract id.

Smart Contract will be assigned a unique id to identify itself.



```
pragma solidity >0.8.17;

contract EtherWallet {
    address payable public owner;

    constructor() {
        owner = payable(msg.sender);
    }

    receive() external payable {}

    function withdraw(uint _amount) external {
        require(msg.sender == owner, "caller is not owner");
        payable(msg.sender).transfer(_amount);
    }

    function getBalance() external view returns (uint) {
        return address(this).balance;
    }
}
```



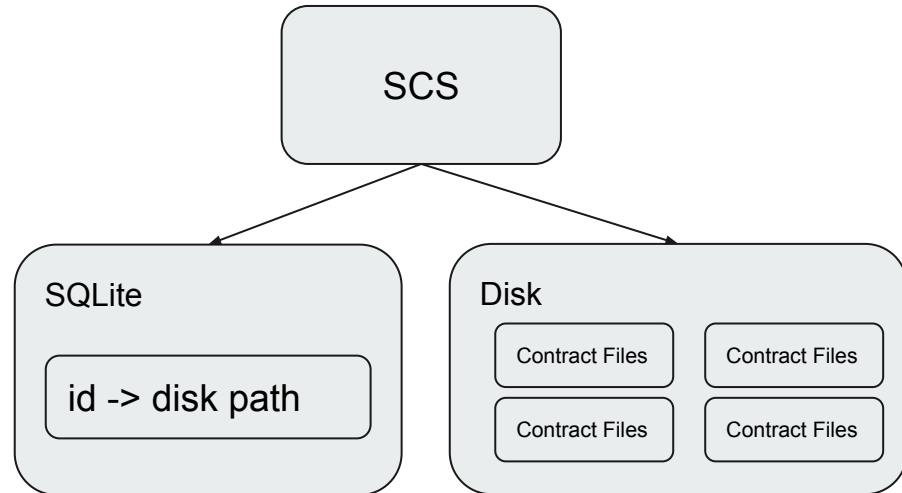
Compile the contract

We use Sodyly to compile contract to abi (application binary interface) when the contract is going to be uploaded.

```
solc --abi demo.sol -o build
```

Register a Smart Contract

1. SCS saves the smart contract file into local disk.
2. SCS saves the contract id with its saved path into SQLite.



Execute a Smart Contract

1. Parameters are sent by the user with the contract id.
2. Obtain the contract file path from SQLite by its id.
3. Execute the contract using the parameters.
4. Return the results to the user.

