



Comment développer une Dapp ?

ÉTAPES A SUIVRE

Créer un nouveau projet sur Github

- Installer NodeJS , version > 7
- *Npm install -g truffle*
- *Npm install -g ganache-cli (ou utiliser Infura)*
- *Truffle init*
- *Npm init*

- Ecriture du nouveau smart contract (.sol)
- Modification du fichier de **migration** pour ajouter son smart contract
- Modification du fichier **truffle.js** pour ajouter sa configuration (réseau local, infura, testnet, mainnet, etc)
- (lancement de ganache-cli dans une fenêtre de commande attribué OU utilisation de Infura)

- Compilation avec truffle compile
- Déploiement avec truffle migrate

- Implémentation du Front
- Utilisation de web3.js sur le Front pour appeler des méthodes du smart contract

IDE CONSEILLÉ

- Conseillé : **Visual Studio Code avec le plugin Solidity**
- Atom avec les plugin Ethereum/Solidity
- Sublime Text avec plugin Ethereum/Solidity
- IntelliJ avec plugin solidity

TRUFFLE

Pour interagir avec le smart contract, on utilise un framework qui facilite le développement. **Truffle** permet de compiler, déployer et tester son smart contract.

- *truffle init* : Lors de l'**initialisation** du projet, permet de créer les dossiers contracts, migrations et test
- *truffle compile* : **Compile** le smart contract et affiche les warning/erreurs de la compilation
- *truffle migrate* : **Déploie** les fichiers compilés sur la blockchain
- *truffle test test/file.js* : Cette commande permet de lancer un **fichier de test**

De plus, Truffle contient des « boxes » qui permettent de démarrer une application rapidement en partant d'une base existante :

<https://truffleframework.com/boxes>

Exemple : *truffle unbox pet-shop*



CLIENT ETHEREUM

Afin de déployer nos smart contract , nous avons besoin d'une blockchain, on devrait utiliser le client officiel des nœuds Ethereum : **go-ethereum**

<https://github.com/ethereum/go-ethereum/wiki/geth>

<https://github.com/ethereum/go-ethereum/wiki/Command-Line-Options>

Il existe d'autre implémentation que celle en Go à savoir :

Parity : <https://www.parity.io/>

Pantheon : <https://pegasys.tech/developers/#besu-documentation>

(hyperledger besu) focused on privacy and entreprise features (hide transaction to some node)

```
3. Wallet
. create account
. backup
. check balance
. sendTransaction

pi@raspberrypi:~$ cd ./go-ethereum/build/bin/
pi@raspberrypi:~/go-ethereum/build/bin$ ls
./ ../ geth*
pi@raspberrypi:~/go-ethereum/build/bin$ ./geth attach
Welcome to the Geth JavaScript console!

instance: Geth/v1.6.7-unstable-78c04c92/linux-arm/go1.7.3
modules: admin:1.0 debug:1.0 eth:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

> eth.accounts
[]
> personal.newAccount("uber vandal floor")
"0x2feefa4a5db64004b1c5cb939d87bf3e9ab2cfe1"
> exit
pi@raspberrypi:~/go-ethereum/build/bin$ pushd ~/.ethereum/keystore/
~/.ethereum/keystore ~/go-ethereum/build/bin
pi@raspberrypi:~/.ethereum/keystore$ ls
./
../
UTC--2017-06-25T20-19-03.307825070Z--2feefa4a5db64004b1c5cb939d87bf3e9ab2cfe1
pi@raspberrypi:~/.ethereum/keystore$
```

GANACHE

Dans le cadre du développement, nous aurons besoin d'une blockchain, afin de faciliter son utilisation, on utilisera **Ganache** qui simulera une blockchain sur notre environnement.

Installation :

```
npm install -g ganache-cli
```

Utilisation

```
ganache-cli -a 40 -l 0x1C9C380
```

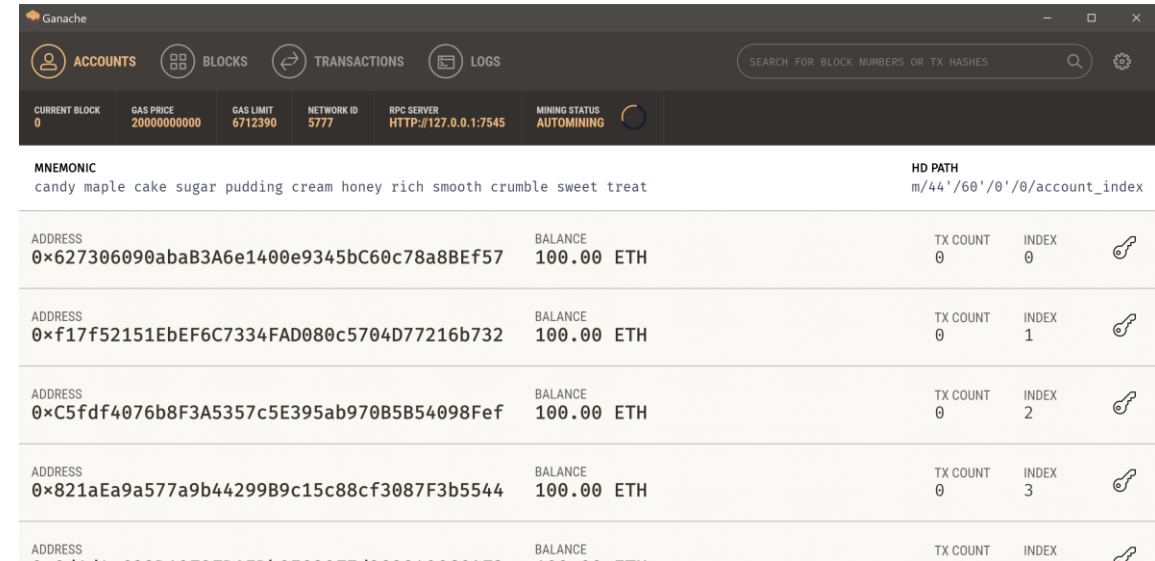
-a : nombre d'account

-l : gas limit

Ganache fait office de mise à jour de **TestRPC**

L'interface peut être télécharger ici :

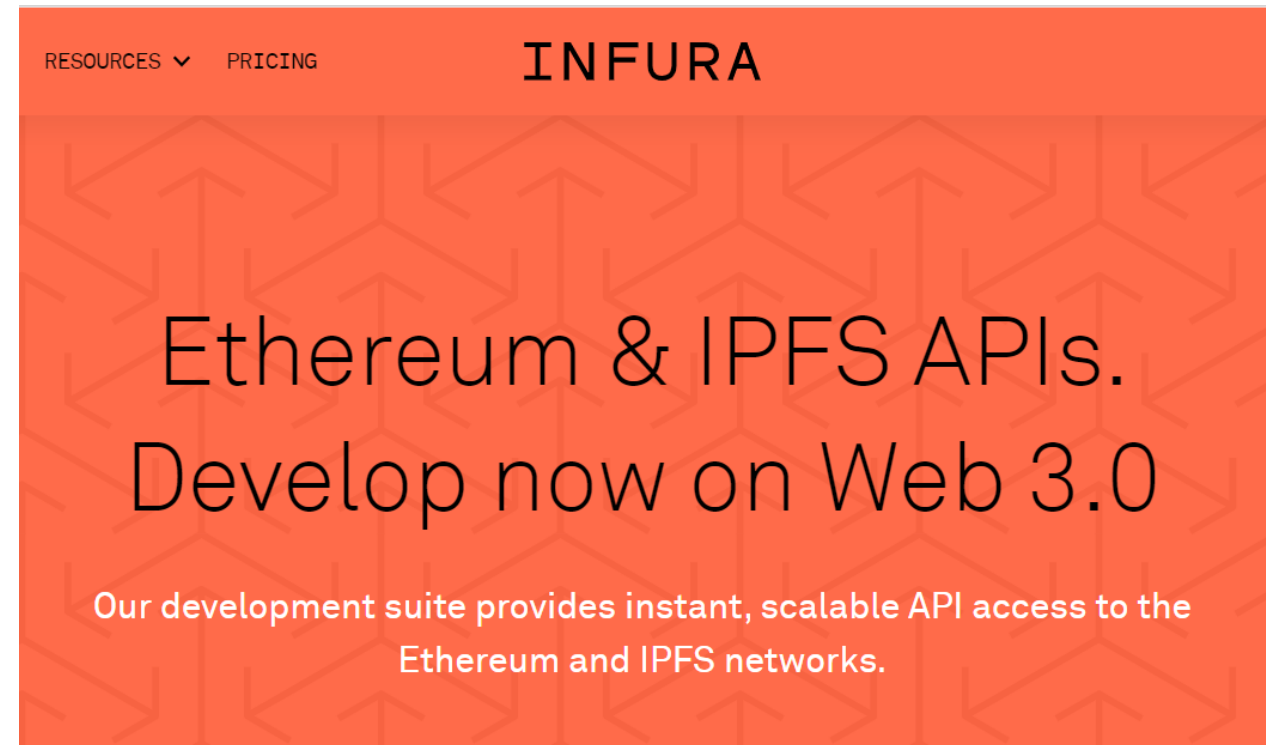
<http://truffleframework.com/ganache/>



INFURA

Pour éviter de télécharger toute la blockchain sur son PC, on peut aussi d'utiliser Infura, qui est un node provider et qui nous permet d'exécuter des fonctions comme avec ganache.
De plus, on peut accéder au testnet (comme Kovan ou Ropsten)

On peut créer un compte sur : <https://infura.io/>
Et la doc d'utilisation pour Truffle :
<https://www.trufflesuite.com/tutorials/using-infura-custom-provider>





TRUFFLE.JS

Afin d'utiliser notre blockchain local avec **testRPC**, il faut modifier les fichiers de configuration **truffle.js** en spécifiant le nouveau port de la blockchain :

```
module.exports = {  
  networks: {  
    development: {  
      host: 'localhost',  
      port: 8545,  
      network_id: '*' // Match any network id  
    }  
  }  
}
```


MIGRATION

Pour déployer les smart contract avec Truffle, on doit changer les fichiers migrations pour intégrer le smart contract développé (nomSC). Pour cela, créer un fichier **2_deploy_contracts.js** dans migrations/

Celui-ci doit contenir :

```
var nomSC = artifacts.require("./nomSC.sol");  
module.exports = function(deployer) {  
  deployer.deploy(nomSC);  
};
```





Comment implementer des tests unitaires sur son smart contract avec Mocha ?

Exemple :

<https://github.com/ConsenSys/Tokens/blob/master/test/eip20/eip20.js>

Execution des tests avec :
truffle test tests/test1.js

MOCHA.JS

```
var ListOfPublisher = artifacts.require("./ListOfPublisher.sol")

contract('ListOfPublisher', (accounts) => {
  var address = accounts[0]

  it('Add a new publisher and get this publisher', async function() {
    var instance = await ListOfPublisher.deployed()
    var publisher = {}
    publisher.name = 'lefigaro'
    publisher.address = accounts[1]

    await instance.addNewPublisher(publisher.address, publisher.name, {from: address})
    var result = await instance.getPublisher(publisher.name)

    assert.equal(publisher.name, web3.toUtf8(result[2])) ;
    assert.equal(publisher.address, result[1]);

  })
})
```

WEB3.JS

Web3.js est une librairie qui permet d'interagir avec un nœud Ethereum (l'API d'Ethereum). Cette librairie permet de **communiquer** depuis le front/back end avec la blockchain et plus particulièrement les smart contract.

Documentation :

<https://web3js.readthedocs.io/en/1.0/>

PROVIDER

*Qu'est ce qu'un **provider** ?*

Les provider sur Web3 sont des « addresses web » qui hébergent un nœud geth ou parity et qui communique avec le réseau Ethereum.

Lorsqu'on developpe on utilise le nœud sur notre **localhost**.

On peut utiliser **Metamask** ou **Infura** pour avoir des provider distants. Ainsi on pourra utiliser des réseaux Ethereum « publiques » de développement comme **Rinkeby** ou **Kovan** qui « reproduisent » les conditions du reseau mainet.

WEB3.JS (SIMPLE JS)

Installation

```
npm install web3
```

Initialisation

```
var Web3 = require('web3');  
  
window.addEventListener('load', function() {  
  if (typeof web3 !== 'undefined') {  
    web3js = new Web3(web3.currentProvider);  
  } else {  
    console.log(« Please consider using Metamask » );  
  }  
})
```

Chargement du smart contract (remplacer SC par son smart contract)

```
var SCaddress= "YOUR_CONTRACT_ADDRESS";  
SC= new web3js.eth.Contract(SCABI, SCaddress);
```

Script à déclarer au début du fichier

```
<script language="javascript" type="text/javascript" src="web3.min.js"></script>  
<script language="javascript" type="text/javascript" src=« SC_abi.js"></script>
```

```
import json
```

```
PATH_TRUFFLE_WK = '/home/myUserName/Projects/myEthereumProjet/'  
truffleFile = json.load(open(PATH_TRUFFLE_WK +  
'/build/contracts/myContractName.json'))
```

```
abi = truffleFile['abi']  
bytecode = truffleFile['bytecode']
```

WEB3.JS (VUE/REACT/ANGULAR)

Installation

```
npm install web3  
npm install truffle-contract
```

Initialisation

```
var Web3 = require('web3');  
var contract = require('truffle-contract');  
  
if (typeof web3 !== 'undefined') {  
  var web3 = new Web3(web3.currentProvider)  
} else {  
  var web3 = new Web3(new  
Web3.providers.HttpProvider('http://localhost:8545'))  
}
```

Chargement du smart contract (remplacer SC par son smart contract)

```
const SC_artifacts = require('./build/contracts/SC.json');  
const SC = contract(SC_artifacts);
```

Tweak à déclarer au début du fichier (pour avoir des appels await/async)

```
SC.setProvider(web3.currentProvider);  
if (typeof SC.currentProvider.sendAsync !== "function") {  
  SC.currentProvider.sendAsync = function() {  
    return SC.currentProvider.send.apply(  
      SC.currentProvider, arguments  
    );  
  };  
}
```

WEB3.JS

Appel de fonction du smart contract depuis son backend

```
async function getActor(actorName) {  
  let deployedContract = await SC.deployed()  
  let getActor = await deployedContract.getActor.call(actorName)  
  return getActor;  
}
```

Asynchrone !

WEB3.JS

Appel de fonction du smart contract depuis son backend

```
async function addActor(actorName, films) {  
  let deployedContract = await SC.deployed()  
  let addNewActor = await deployedContract.addActor(actorName, films, {from:addressFrom, gas:30000000})  
  return addNewActor;  
}
```


WEB3.JS

Exemple d'utilisation possible

- `web3.eth.getAccounts()`
- `web3.eth.getBlock(blockHashOrBlockNumber [, returnTransactionObjects] [, callback])`
- `web3.eth.getTransaction(transactionHash [, callback])`
- `web3.eth.getTransactionReceipt(hash [, callback])`
- `web3.eth.sendTransaction(transactionObject [, callback])`
- `web3.eth.signTransaction(transactionObject, address [, callback])`

TROUBLESHOOTING

- **Exceeds gaz limit**

Augmenter la gas limit lors du lancement de testrpc ou autre

- **Probleme de migration, smart contract non trouvé**

cd build\contract

*rm *.json*

truffle migrate --reset --compile-all

- **La commande s'exécute une fois sur deux, imprevisible, elle s'exécutait avant**

=> Async/await

- **Truffle migrate error**

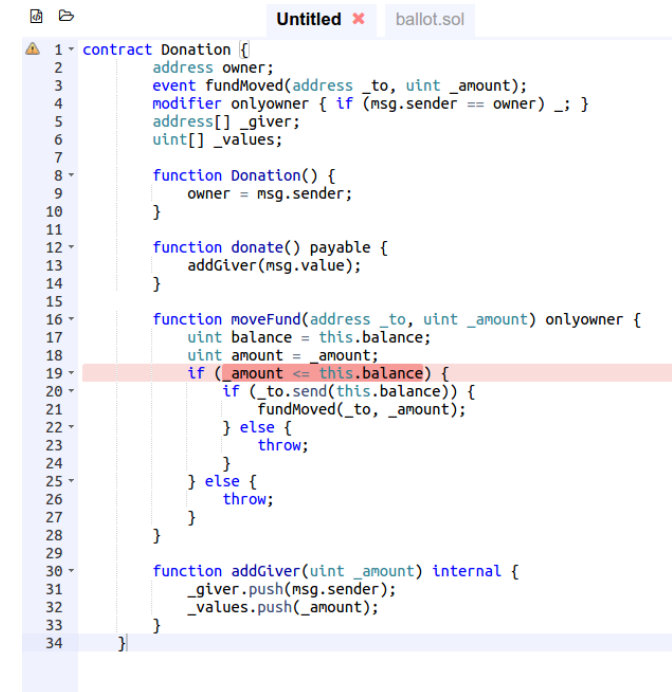
export NVM_DIR="\$HOME/.nvm«

[-s "\$NVM_DIR/nvm.sh"] && \. "\$NVM_DIR/nvm.sh"

REMIX

L'IDE le plus simple à utiliser est **Remix**, on peut :

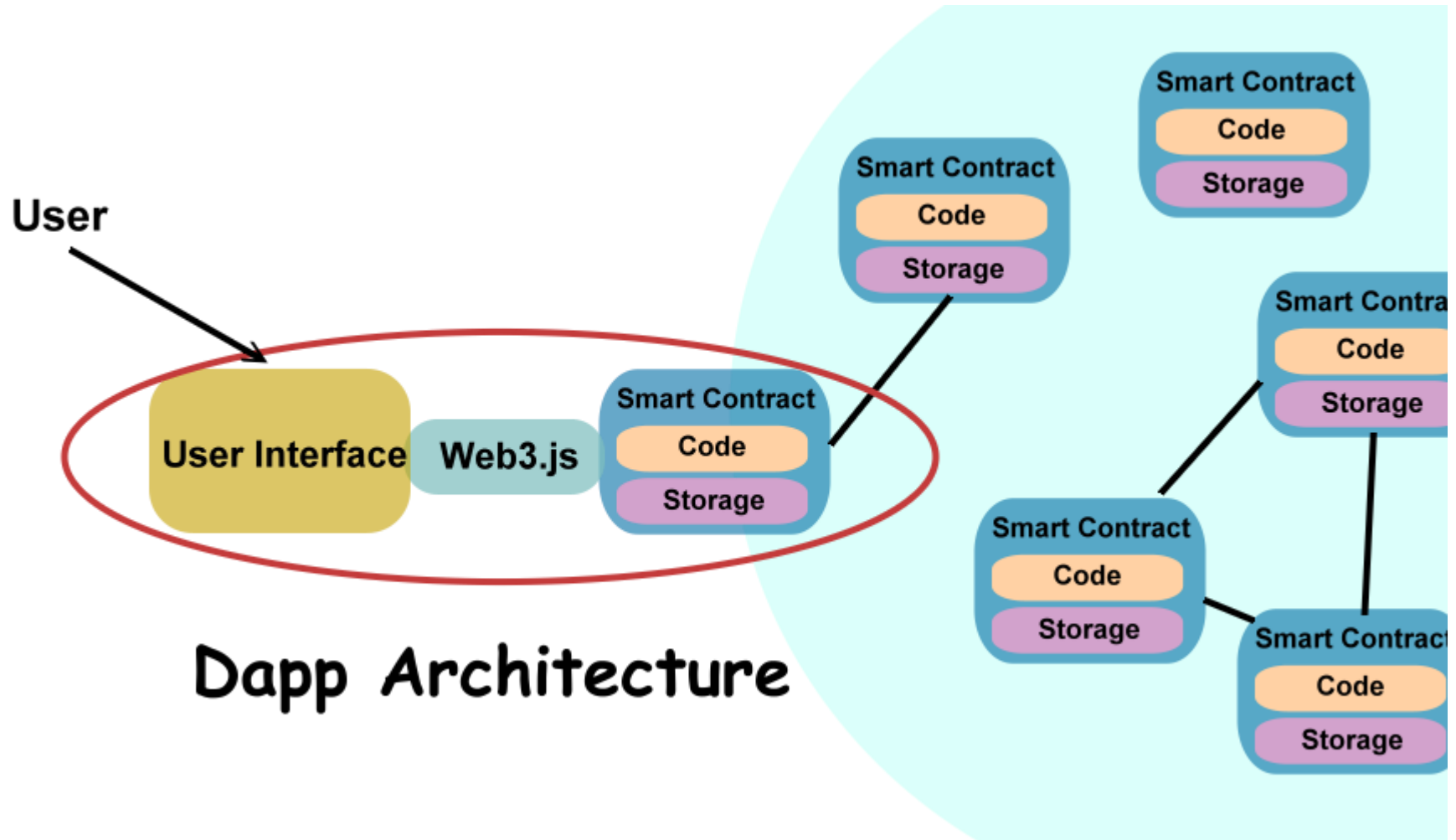
- compiler,
- changer son adresse,
- tester sur Rinkeby/Kovan,
- utiliser Web3 si besoin,
- lancer des fonctions du smart contract,
- débogé étape par étape le code



```

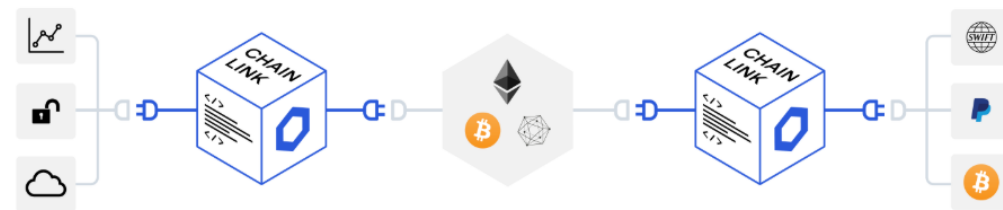
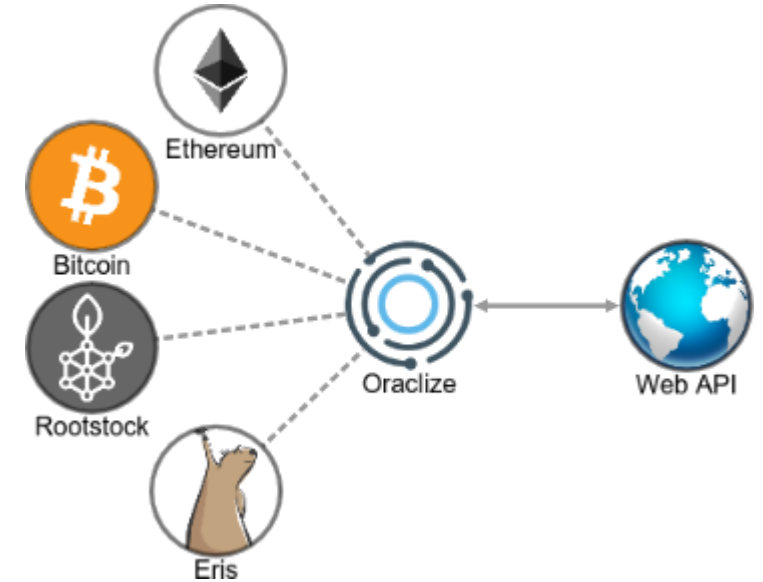
1 contract Donation {
2     address owner;
3     event fundMoved(address _to, uint _amount);
4     modifier onlyowner { if (msg.sender == owner) _; }
5     address[] _giver;
6     uint[] _values;
7
8     function Donation() {
9         owner = msg.sender;
10    }
11
12    function donate() payable {
13        addGiver(msg.value);
14    }
15
16    function moveFund(address _to, uint _amount) onlyowner {
17        uint balance = this.balance;
18        uint amount = _amount;
19        if (amount <= this.balance) {
20            if (_to.send(this.balance)) {
21                fundMoved(_to, _amount);
22            } else {
23                throw;
24            }
25        } else {
26            throw;
27        }
28    }
29
30    function addGiver(uint _amount) internal {
31        _giver.push(msg.sender);
32        _values.push(_amount);
33    }
34 }
  
```

ARCHITECTURE DAPP



REQUETE SUR UNE API

- **Provable(anciennement Oraclize) ou Chainlink permet de requêter une API depuis un smart contrat**
- <https://provable.xyz/>
- <https://chain.link/>



Connect to Any External API

Easily connect smart contracts to the data sources and APIs they need to function.

Send Payments Anywhere

Send Payments from your contract to bank accounts and payment networks.

BOILERPLATE ETH

- <https://github.com/austintgriffith/scaffold-eth>

The image displays the Scaffold-ETH development environment across four panels:

- Top Left (Terminal):** Shows the command `yarn run v1.22.4` and the output of `react-scripts build` and `react-scripts deploy`. It indicates that the contract was successfully compiled and deployed to the Ethereum testnet.
- Top Right (Web Browser):** Shows the Scaffold-ETH web interface. The contract name is "YourContract" and the balance is 0.0000 ETH. The purpose is "Programming Unstoppable Money".
- Bottom Left (Terminal):** Shows a React Hook warning: "React Hook useEffect has a missing dependency: 'refresh'. Either include it or remove the dependency array".
- Bottom Right (File Explorer):** Shows the project structure, including the `contracts` directory, the `react-app` directory, and the `src` directory.