

INTERNET OF THINGS BASED SMART SYSTEMS

Ingegneria Informatica LM-32
A.A. 2022/2023

ETH DASHBOARD: Creazione di un token fungibile [ERC 20] associato ad un caso d'uso

Studente:

Cardillo Mattia

Matricola: 1000037133

1. Introduzione

Lo scopo dell'elaborato è quello di creare uno smart contract ERC-20 su rete **Ethereum** basato su testnet con l'obiettivo di mostrare una serie di transizioni semplici.

Partendo dalle basi, la **Blockchain** è un **sistema di pagamento decentralizzato peer to peer**. Si tratta di una tecnologia software che consente agli utenti di inviare e ricevere denaro e/o beni scrivendo le transazioni su un registro distribuito accessibile da chiunque in qualsiasi momento.

In un sistema Blockchain, il consenso su una transazione viene raggiunto utilizzando un **protocollo** specifico (il primo è stato **Bitcoin** e successivamente **Ethereum**). Ethereum è definito come **Blockchain 2.0**: è una Blockchain che consente agli sviluppatori di arricchire i metadati contenuti in una transazione per costruire applicazioni smart.

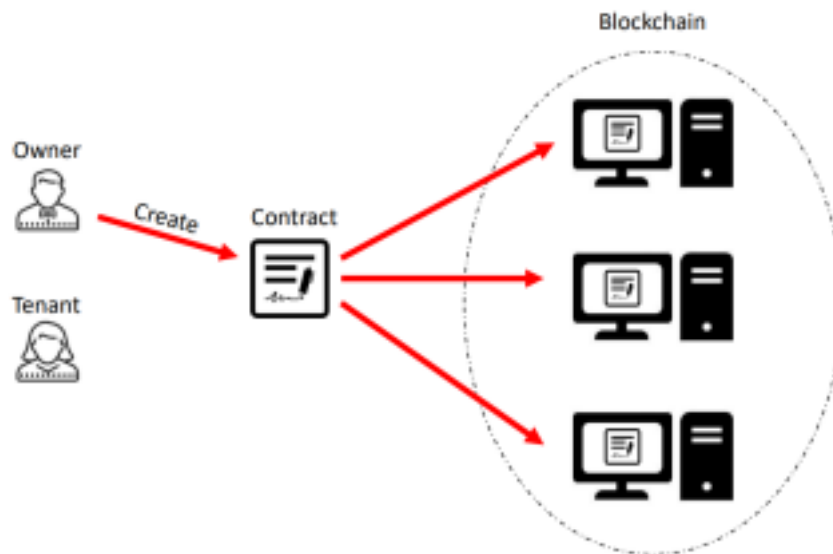
Ether (ETH) è la criptovaluta guadagnata come premio del processo di "*mining*" adesso sostituito con quello di "*staking*" e viene utilizzato come "*carburante*" dagli sviluppatori di applicazioni e dagli utenti per pagare le spese di elaborazione (*processing fees*) delle applicazioni decentralizzate.

Le merci e le risorse su una Blockchain sono identificate tramite indirizzi (**addresses**): stringhe di caratteri associate a una quantità di criptovaluta (**token**).

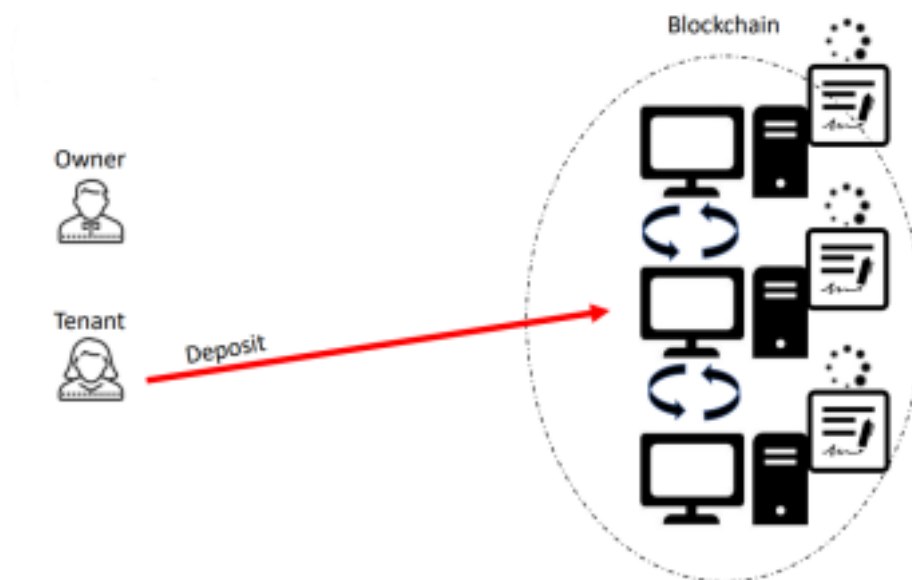
Uno **smart contract**, come qualsiasi altro contratto, regola i termini e le condizioni tra le parti. A differenza di quanto avviene per un contratto tradizionale, i termini di uno smart contract vengono eseguiti sulla base di un codice programmato su una blockchain come Ethereum. Gli smart contract consentono agli sviluppatori di creare app che sfruttano la sicurezza, l'affidabilità e l'accessibilità della blockchain, offrendo sofisticate funzionalità peer-to-peer che comprendono i servizi di richiesta e concessione prestiti e assicurativi, la logistica e i giochi.

Ad oggi, Ethereum è la Blockchain più popolare utilizzata per eseguire gli smart contract. Le figure che interagiscono in uno smart contract sono:

- **Owner/Landlord**: crea il contratto e lo replica nei vari nodi. Può anche controllare il balance e se lo stato del contratto è stato fetchato da un nodo;



- **Tenant:** deposita il contratto e ne modifica lo stato nei vari nodi.



2. Sviluppo dello smart contract

Il contratto presente nell'elaborato è progettato per consentire la creazione di nuovi coin (*minting*) e l'invio di tali coin a indirizzi appartenenti alla stessa blockchain (*send*). Per creazione di nuovi coin si intende la valuta che verrà utilizzata all'interno del contratto stesso, non ETH.

```
pragma solidity >= 0.7.1 < 0.9.0;

contract SampleCoin {
    address public minter;

    mapping (address => uint) public balances;

    event Sent(address from, address to, uint amount);

    constructor() public {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public returns (uint){
        require(msg.sender == minter);
        require(amount < 1e60);
        balances[receiver] += amount;
        return balances[receiver];
    }

    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender], "Insufficient balance detected.");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }

    function showBalances(address account) external view returns (uint){
        return balances[account];
    }
}
```

}

Il contratto crea un token **SampleCoin** inizializzando un coin miner (con il *costruttore pubblico*), ovvero un indirizzo che può creare nuovi coin. Solo il creatore del contratto può coniare nuovi coin.

Questo contratto modella anche il mapping di ogni indirizzo con il suo saldo, e la funzione **send** controlla semplicemente la fattibilità della transazione. Se il mittente ha meno saldo di quello che desidera inviare, il contratto restituirà un messaggio di saldo insufficiente e la transazione non verrà completata. Se la transazione è invece fattibile, la funzione emetterà un evento di invio. I client Ethereum (*wallets o app decentralizzate sul web*) possono ascoltare questi eventi emessi sulla blockchain. L'ascolto/lettura dei contratti e delle transazioni all'interno di una blockchain è gratuito, mentre si paga la creazione delle transazioni e le modifiche dei contratti già esistenti all'interno della chain. Non appena l'evento viene emesso, l'ascoltatore riceve gli argomenti from, to e amount, il che rende possibile tenere traccia delle transazioni.

Per ottenere il saldo di un indirizzo può essere utilizzata la funzione **showBalances**.

Per testare il corretto funzionamento del contratto, si è fatto uso di **Remix IDE**, un tool open source per scrivere, compilare e testare contratti in Solidity.

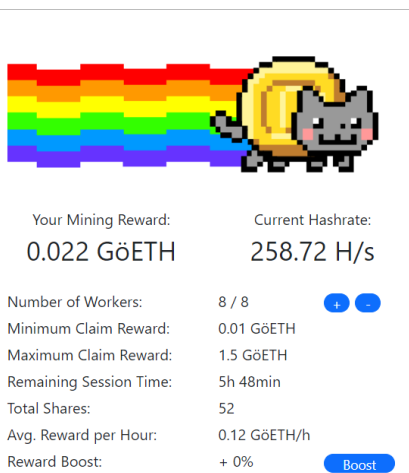
Dopo aver scritto il contratto, sono stati effettuati diversi deploy utilizzando la tab “*Deploy & run transactions*”:

- Per il primo deploy è stato impostato come **Environment** “*Remix VM (London)*”, impostato un valore di **Gas limit** a 3000000 e **Value** a 0, poiché la funzione chiamata deve essere *payable* per inserire un value diverso da quello indicato. Il *gas* si riferisce alla fee richiesta per condurre con successo una transazione o eseguire un contratto sulla piattaforma blockchain di Ethereum, poiché ogni transazione consuma risorse di calcolo;
- Per il secondo deploy è stato effettuato il collegamento a **Metamask**, dopo aver creato un account e ottenuto i fondi necessari da un faucet nella testnet goerli.

In particolare i fondi di test sono stati richiesti dal faucet “<https://goerli-faucet.pk910.de/>”

Goerli PoW Faucet

The goerli testnet is [deprecated](#) and will be shut down at the end of the year! Move over to the sepolia testnet for anything except validator testing.



The interface features a cartoon cat with a rainbow trail. Below it, various mining statistics are displayed in a two-column layout. On the right side, there are interactive buttons for adjusting the number of workers and a 'Boost' button.

Your Mining Reward:	0.022 GÖETH	Current Hashrate:	258.72 H/s
Number of Workers:	8 / 8		
Minimum Claim Reward:	0.01 GÖETH		
Maximum Claim Reward:	1.5 GÖETH		
Remaining Session Time:	5h 48min		
Total Shares:	52		
Avg. Reward per Hour:	0.12 GÖETH/h		
Reward Boost:	+ 0%		Boost

3. Creazione dell'interfaccia web dimostratore

Al fine di simulare le funzioni contenute nel contratto, è stata creata un'interfaccia web facendo uso di HTML e jQuery.

Il risultato ottenuto è il seguente:

Address

Amount

Confirm and Send

Address for balance check

Contract address for balance check

Confirm and Send

Tale pagina consente di:

- Richiamare la funzione di **mint** premendo sul primo bottone “Confirm and Send”, inserendo l’indirizzo del ricevitore e l’amount negli appositi input fields;
- Richiamare la funzione di **showBalances** premendo sul secondo bottone “Confirm and Send”, inserendo l’indirizzo da controllare e l’indirizzo del contratto precedentemente generato negli appositi input fields.

Al click sui bottoni, verranno eseguite delle chiamate ajax alle API esposte dal server scritto in Express.

4. Server Express (Node.js)

Il server è stato scritto in **Express**, un framework per **Node.js**, rispettando il **principio di singola responsabilità**; tale principio si applica alla programmazione orientata agli oggetti, ma può anche essere considerato un principio architetturale simile alla separazione dei concetti. Afferma che gli oggetti devono avere una sola responsabilità e un solo motivo per cambiare. Nello specifico, l'unica situazione in cui l'oggetto può cambiare è se il modo in cui svolge la sua unica responsabilità richiede di essere aggiornato. Seguendo questo principio, questo principio consente di produrre sistemi più modulari, poiché molti tipi di nuovo comportamento possono essere implementati come nuove classi, anziché aggiungendo ulteriore responsabilità alle classi esistenti. Questo principio può essere facilmente sfruttato per l'inserimento di nuove versioni alle API già esposte.

Gli elementi più importanti del codice sviluppato si trovano all'interno del path `services/v1/blockchain.js`:

- La funzione **generateTokens** permette di generare nuovi coin. A seguito del deploy del contratto viene invocata la funzione **send** al fine di poter modificare il contratto. La modifica del contratto richiede il pagamento di una fee, motivo per cui va specificato il valore del gas;
- La funzione **getBalance** permette di ottenere il balance dell'address specificato nel body della richiesta. Sul contratto viene invocata la funzione **call** al fine di interrogare il contratto stesso, senza effettuare modifiche.

Si precisa che in questo scambio non si ottiene nessun trasferimento di criptovalute, ma di un valore che può essere modificato solo dall'owner del contratto.

Tramite il debugger del server è possibile leggere informazioni aggiuntive circa il processamento delle richieste mediante il terminale.

N.B: Per la simulazione in locale è richiesta l'installazione di **Ganache** al fine di avere a disposizione una serie di address di test.