



UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Laboratorio 4 – Oracoli in Ethereum & Chainlink

Prof. Esposito Christian

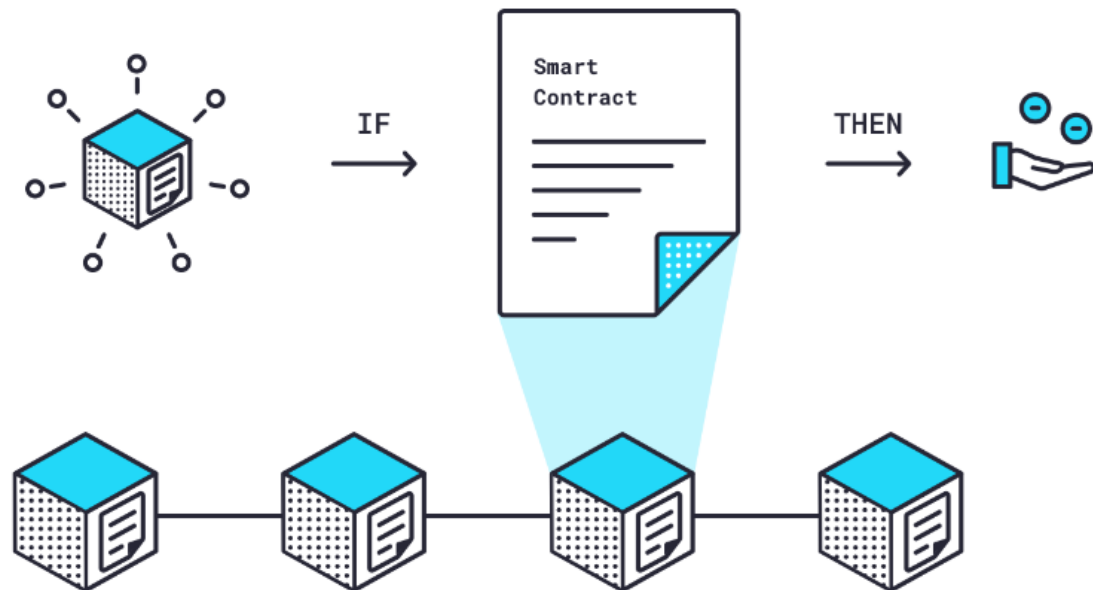
Corso di Sicurezza dei Dati



::... Smart Contract

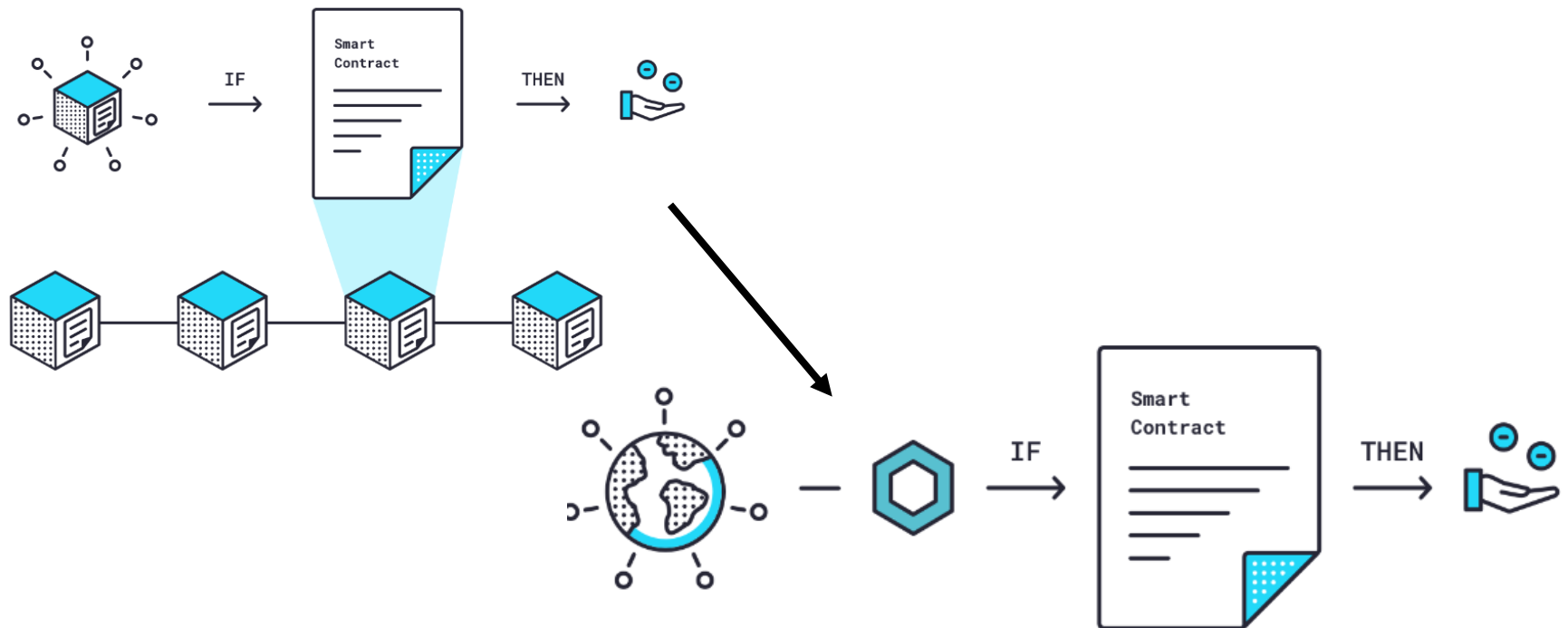
Gli smart contract ci permettono di avere un accordo sulla blockchain che valuta informazioni e viene automaticamente eseguito quando alcune condizioni sono soddisfatte.

Essi sono immutabili (non possono essere cambiati) e verificabili (tutti ne possono vedere il contenuto), creando così un alto livello di fiducia tra tutte le parti.



::... Smart Contract

In alcuni casi, essi necessitano di dati off-chain (disponibili al di fuori della blockchain) in un formato che sia utilizzabile on-chain. La difficoltà nel collegare le fonti di informazione esterne agli smart contract della blockchain in un linguaggio che entrambi possano comprendere è una delle principali limitazioni all'ampia diffusione di essi.



::... Smart Contract

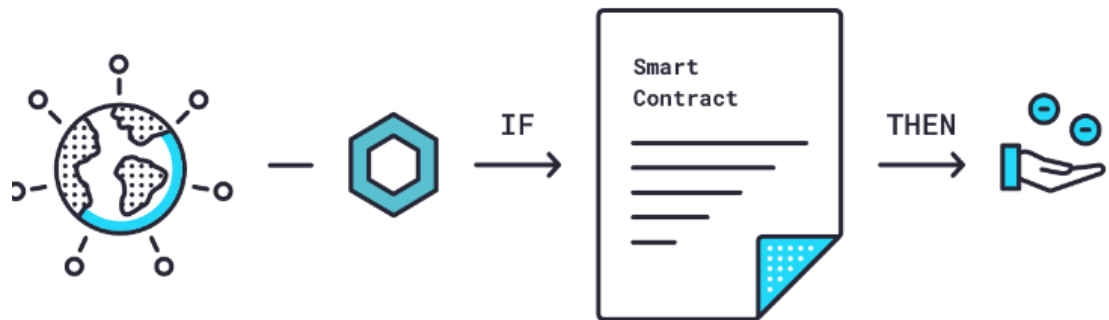
In alcuni casi, essi necessitano di dati off-chain (disponibili al di fuori della blockchain) in un formato che sia utilizzabile on-chain. La difficoltà nel collegare le fonti di informazione esterne agli smart contract della blockchain in un linguaggio che entrambi possano comprendere è una delle principali limitazioni all'ampia diffusione di essi.



::... Oracolo

È qui che entrano in gioco gli oracoli. Un oracolo è un software noto come ***middleware*** che funge da intermediario, traducendo i dati dal mondo reale agli smart contract sulla blockchain e viceversa.

Gli oracoli si comportano come delle vere e proprie API che gli smart contract possono interrogare per accedere ai dati off-chain. In questo modo uno smart contract può conoscere qualsiasi tipo di informazione (dalla quotazione di un'azione al tempo previsto per domani) e impiegarla nelle sue logiche computazionali.



::... Oracolo

```
import "github.com/oraclize/ethereum-api/oraclizeAPI.sol";

contract OracleExample is usingOraclize {

    string public EURUSD;

    function updatePrice() public payable {
        if (oraclize_getPrice("URL") > this.balance) {
            //Handle out of funds error
        } else {
            oraclize_query("URL", "json(http://api.fixer.io/latest?symbols=USD).rates.USD");
        }
    }

    function __callback(bytes32 myid, string result) public {
        require(msg.sender == oraclize_cbAddress());
        EURUSD = result;
    }
}
```

::... Oracolo

```
import "github.com/oraclize/ethereum-api/oraclizeAPI.sol";

contract OracleExample is usingOraclize {

    string public EURUSD;

    function updatePrice() public payable {
        if (oraclize_getPrice("URL") > this.balance) {
            //Handle out of funds error
        } else {
            oraclize_query("URL", "json(http://api.fixer.io/latest?symbols=USD).rates.USD");
        }
    }

    function __callback(bytes32 myid, string result) public {
        require(msg.sender == oraclize_cbAddress());
        EURUSD = result;
    }
}
```

Il servizio Oraclize viene utilizzato in quasi tutti i casi in cui è necessario un oracolo, il seguente esempio interagisce con l'oracolo Oraclize per ricevere l'attuale tasso di cambio da EUR a USD.

::... Oracolo

```
import "github.com/oraclize/ethereum-api/oraclizeAPI.sol";

contract OracleExample is usingOraclize {

    string public EURUSD;

    function updatePrice() public payable {
        if (oraclize_getPrice("URL") > this.balance) {
            //Handle out of funds error
        } else {
            oraclize_query("URL", "json(http://api.fixer.io/latest?symbols=USD).rates.USD");
        }
    }

    function __callback(bytes32 myid, string result) public {
        require(msg.sender == oraclize_thisChainId);
        EURUSD = result;
    }
}
```

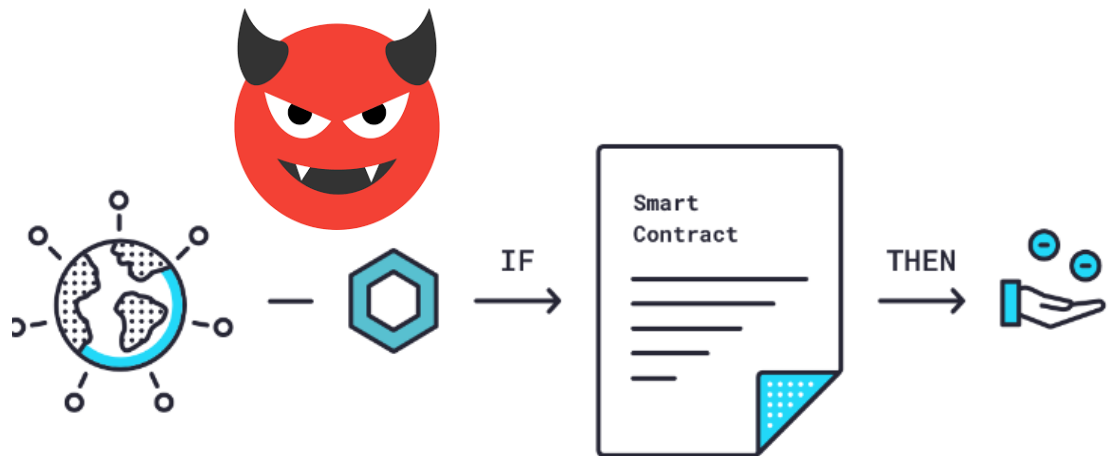
La funzione `updatePrice()` invia la query all'oracolo. Viene utilizzato il modificatore `payable`, che consente alla transazione di avere un valore (una certa quantità di Ether) allegato. Ciò è necessario perché l'utilizzo del servizio Oraclize non è gratuito.

Il servizio Oraclize viene utilizzato nel seguente esempio interagisce con l'oracolo Oraclize per ricevere l'attuale tasso di cambio da EUR a USD.

::... Oracolo

Tuttavia, un singolo oracolo centralizzato crea proprio il problema che un contratto intelligente sicuro e decentralizzato sulla blockchain mira a risolvere: ***un punto centrale di debolezza***.

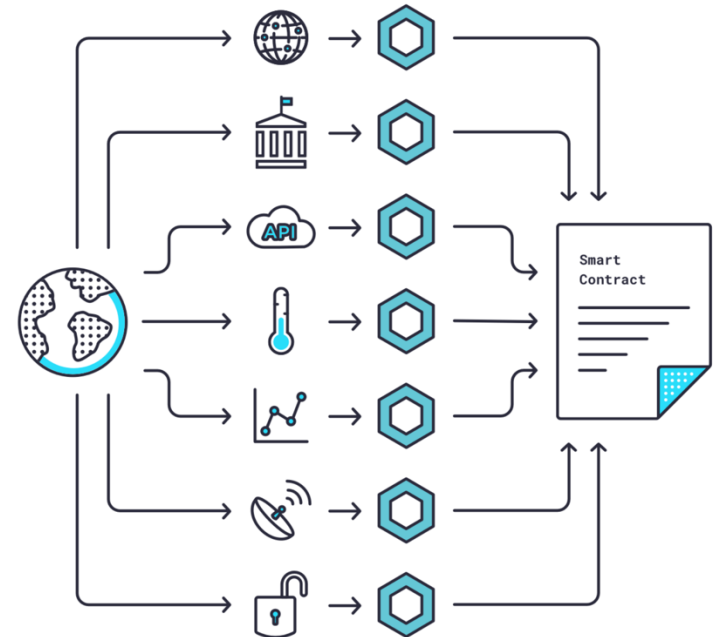
Se l'oracolo è difettoso o ***compromesso***, come si potrebbe sapere se i dati sono accurati? A cosa serve un contratto intelligente sicuro e affidabile sulla blockchain se i dati che lo alimentano sono in dubbio?



::... Oracolo

Per ovviare a queste limitazioni è possibile far uso di soluzioni esistenti come Chainlink, che è una rete decentralizzata di nodi che fornisce dati e informazioni da fonti esterne alla blockchain agli smart contract sulla blockchain tramite oracoli.

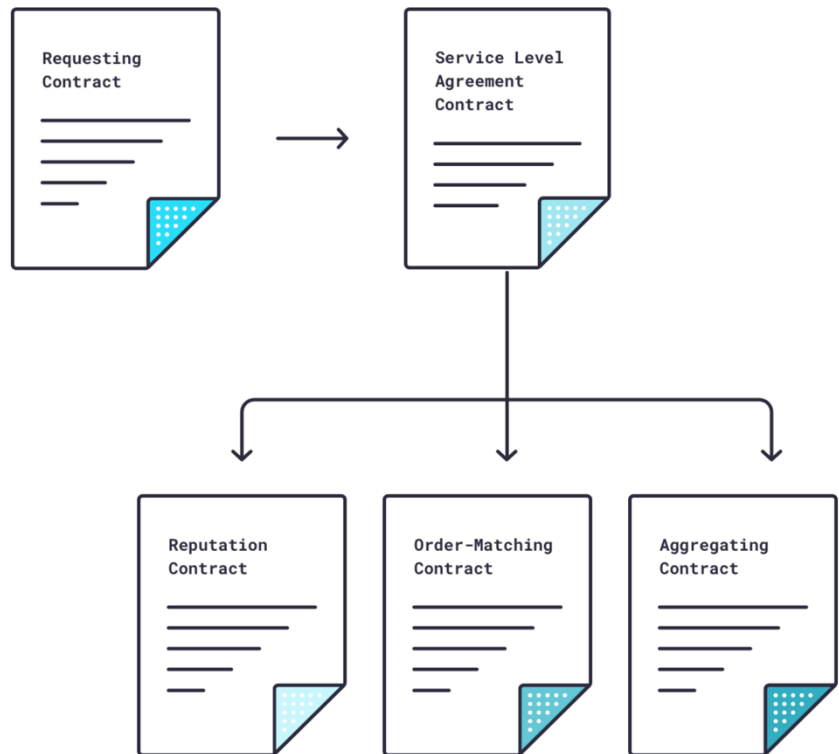
Questo processo elimina i problemi di affidabilità che potrebbero verificarsi utilizzando una sola fonte centralizzata.



::... Chainlink

Il protocollo Chainlink registra questa richiesta come un "evento" e, a sua volta, crea un contratto intelligente corrispondente (Contratto di Accordo sul Livello di Servizio di Chainlink, o SLA) anche sulla blockchain, per ottenere questi dati off-chain.

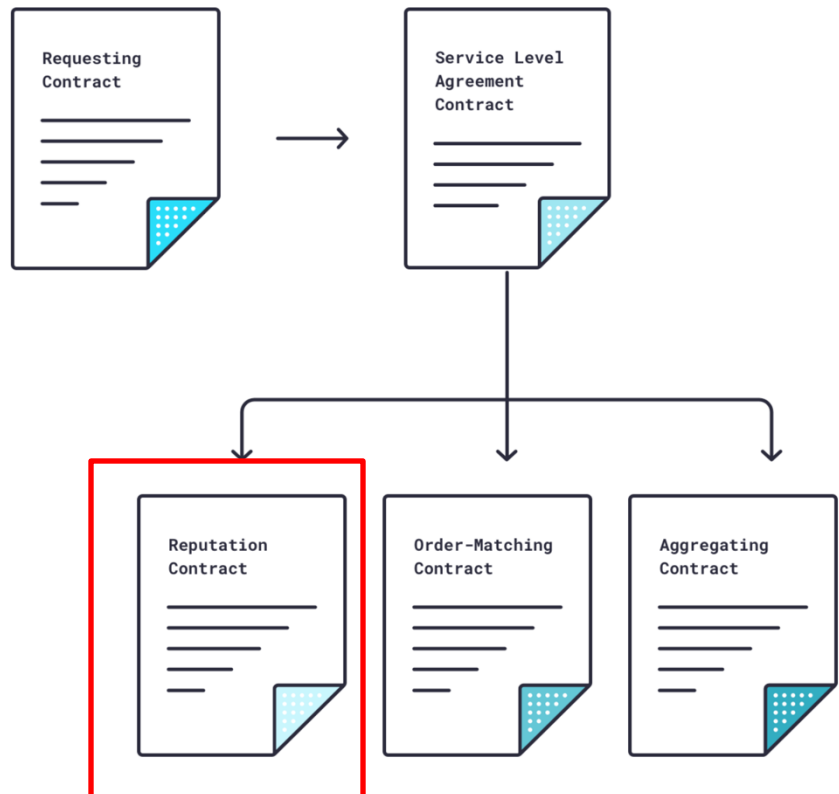
Il Contratto SLA di Chainlink genera tre sotto-contratti: un Contratto di Reputazione di Chainlink, un Contratto di Abbinamento Ordini di Chainlink e un Contratto di Aggregazione di Chainlink.



::... Chainlink Reputation Contract

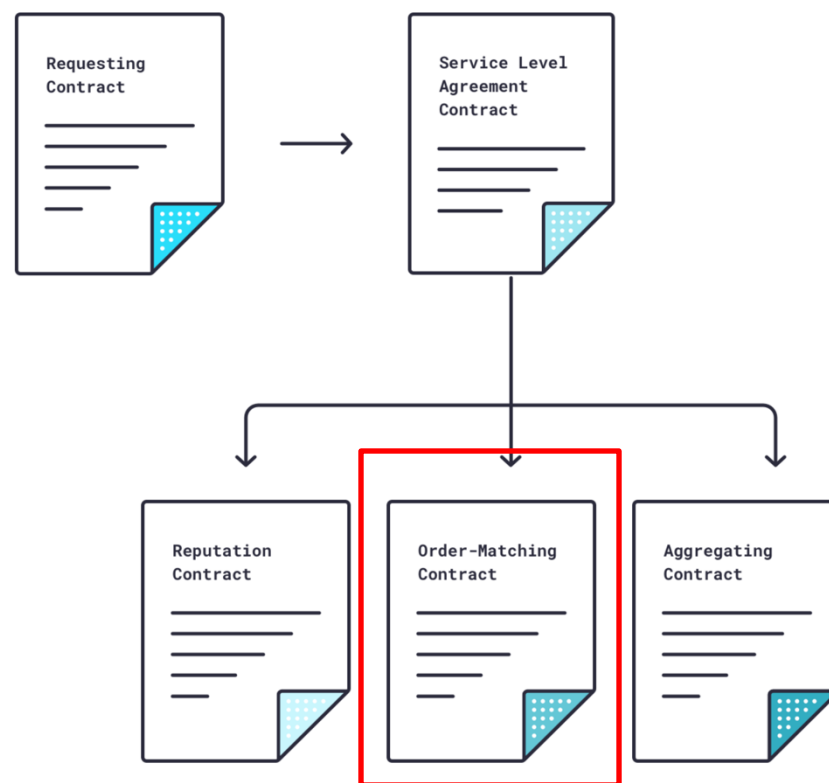
Il Contratto di Reputazione di Chainlink controlla lo storico di un fornitore di oracoli per verificare la sua autenticità e il suo rendimento passato, valutando ed eliminando i nodi inaffidabili o di dubbia reputazione.

Tale contratto permette di eliminare gli oracoli che non rispondono in maniera autentica, basandosi su un meccanismo di reputazione.



::... Chainlink Order-Matching Contract

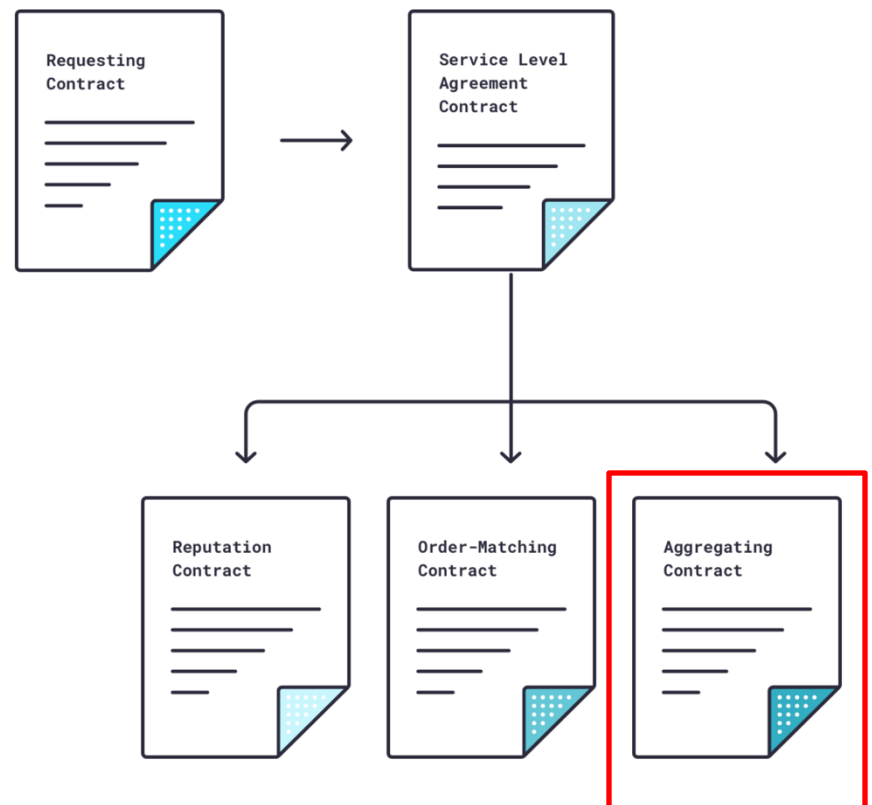
Il Contratto di Abbinamento Ordini di Chainlink invia la richiesta del Contratto Richiedente ai nodi Chainlink e raccoglie le loro offerte sulla richiesta (quando il Contratto Richiedente non sceglie un insieme specifico di nodi), selezionando poi il numero e il tipo di nodi più adatti per soddisfare la richiesta.



::... Chainlink Aggregating Contract

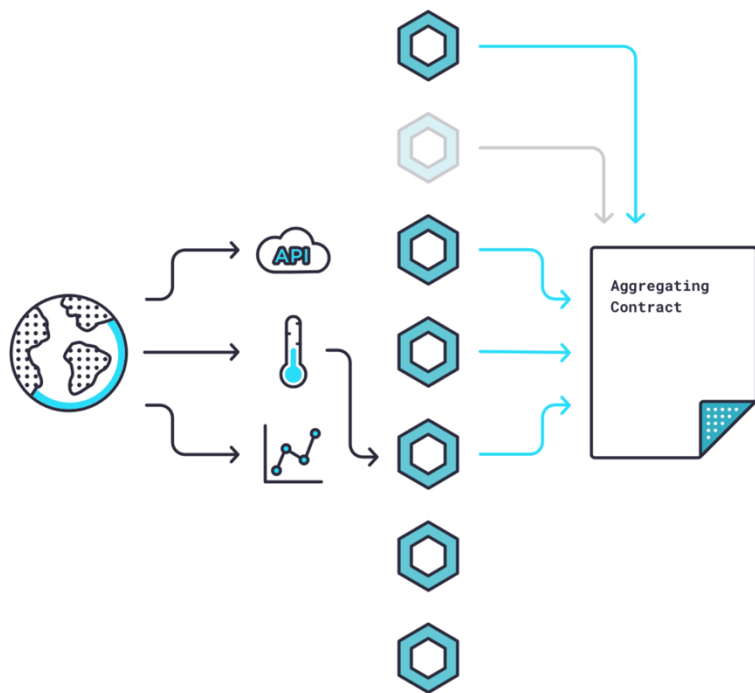
Il Contratto di Aggregazione di Chainlink raccoglie tutti i dati provenienti dagli oracoli scelti e li valida e/o riconcilia per ottenere un risultato accurato.

Il Contratto di Aggregazione di Chainlink può validare i dati provenienti da una singola fonte o da più fonti, e può anche riconciliare i dati provenienti da diverse sorgenti.



::: Chainlink Data Validation

I nodi Chainlink prendono quindi la richiesta di dati del Contratto Richiedente e utilizzano il software **Chainlink Core** per tradurre tale richiesta dal linguaggio di programmazione on-chain a un linguaggio off-chain comprensibile per una fonte di dati del mondo reale. Questa versione tradotta della richiesta viene poi inoltrata a un'API esterna che raccoglie i dati da quella fonte.

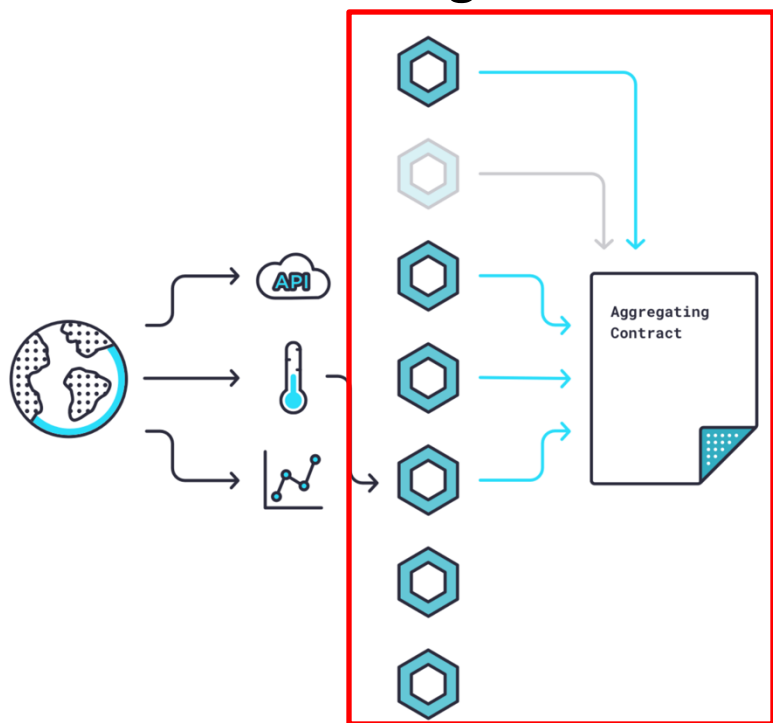


Una volta raccolti i dati, vengono nuovamente tradotti nel linguaggio on-chain tramite **Chainlink Core** e inviati al Contratto di Aggregazione di Chainlink.

::... Chainlink Data Validation

I nodi Chainlink prendono quindi la richiesta di dati del Contratto Richiedente e utilizzano il software **Chainlink Core** per tradurre tale richiesta dal linguaggio di programma off-chain comprensibile per un oracolo. Questa versione tradotta della richiesta viene inviata a una API esterna che raccoglie i dati da una

Se cinque nodi forniscono una risposta da un sensore meteorologico e altri due nodi offrono una risposta diversa, il Contratto di Aggregazione di Chainlink saprà che quei due nodi sono difettosi (o disonesti) e scarterà le loro risposte. In questo modo, i nodi Chainlink possono validare i dati provenienti da una singola fonte



Una volta raccolti i dati, vengono nuovamente tradotti nel linguaggio on-chain tramite **Chainlink Core** e inviati al Contratto di Aggregazione di Chainlink.

::... Chainlink Aggregating Contract

Il Contratto di Aggregazione di Chainlink può ripetere questo processo di validazione per più fonti, riconciliando poi tutti i dati validati mediandoli in un unico dato. In determinate circostanze, non tutte le risposte possono essere mediate, ma per semplicità non ci addentreremo ulteriormente in questo aspetto.



::... Chainlink Token (LINK)

I detentori del Contratto Richiedente utilizzano LINK per pagare gli operatori dei nodi Chainlink per il loro lavoro. I prezzi sono stabiliti dall'operatore del nodo Chainlink in base alla domanda per i dati che possono fornire e al mercato attuale per tali dati.

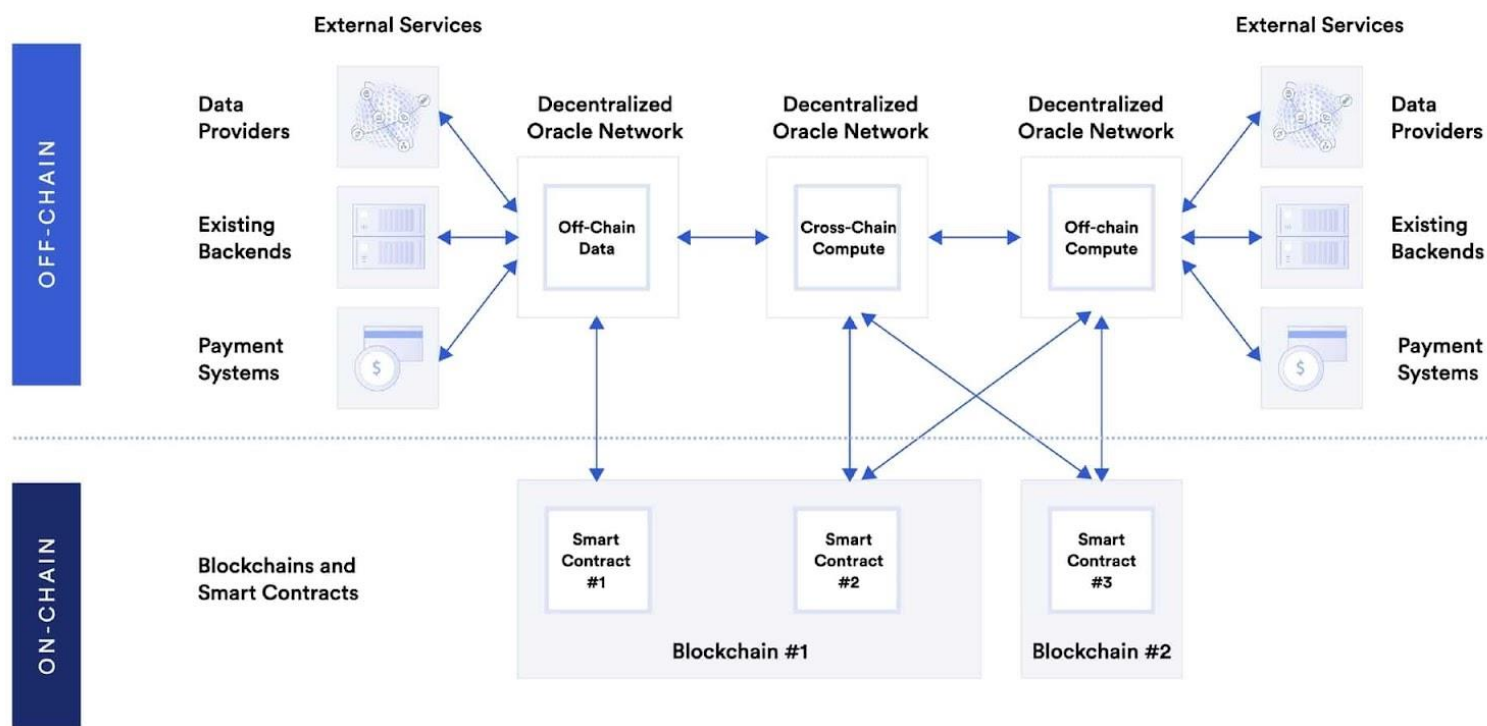
LINK è costruito su Ethereum in conformità con lo standard ERC-20 per i token. Può essere acquistato e venduto per valuta fiat o altre valute digitali.

 **Chainlink** LINK #15
€10.03 ▼ 2.16% (1g)

0x514910771af9ca656af840dff83e8264ecf986ca

::... Tipologie di Oracoli

Considerando l'ampia gamma di risorse off-chain, gli oracoli blockchain si presentano in molte forme.



::... Getting Start with Any API

L'obiettivo di questo tutorial è quello di apprendere come richiedere dati da un'API pubblica in uno smart contract.

Il primo step è capire cosa siano i **Task** e gli **External Adapter** e come gli **Oracle Jobs** li utilizzino.

Task: Attività integrate in ogni nodo (esempi: http, ethabidecode, ecc.).

External Adapters: Questi sono adattatori personalizzati che possono richiamare un determinato endpoint con un insieme specifico di parametri (come segreti di autenticazione che non dovrebbero essere visibili pubblicamente).

::... Getting Start with Any API

L'obiettivo di questo tutorial è quello di apprendere come richiedere dati da un'API pubblica in uno smart contract.

Il primo step è capire cosa siano i **Task** e gli **External Adapter** e come gli **Oracle Jobs** li utilizzino.

Task: Attività integrate in ogni nodo (esempi: http, ethabidecode, ecc.).

External Adapters: Questi sono adattatori personalizzati che possono richiamare un determinato endpoint con un insieme specifico di parametri (come segreti di autenticazione che non dovrebbero essere visibili pubblicamente).

Job: Insieme di Task eseguiti da un singolo nodo.

::... Job #1

Un esempio di Job è quello di effettuare una richiesta GET a un'API, recuperare un campo intero senza segno specifico dalla risposta JSON e poi inviarlo nuovamente al contratto richiedente. Questo richiede i seguenti step:

1. **HTTP** chiama l'API. Il metodo deve essere impostato su GET.
2. **JSON Parse** analizza il JSON ed estrae un valore a un determinato percorso chiave.
3. **Multiply** moltiplica l'input per un moltiplicatore, usato per rimuovere i decimali.
4. **ETH ABI Encode** converte i dati in un payload di byte secondo la codifica ABI di Ethereum.
5. **ETH Tx** invia la transazione alla blockchain, completando il ciclo.

::... ChainlinkClient Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

import {Chainlink, ChainlinkClient} from "@chainlink/contracts/src/v0.8/ChainlinkClient.sol";
import {ConfirmedOwner} from "@chainlink/contracts/src/v0.8/shared/access/ConfirmedOwner.sol";
import {LinkTokenInterface} from "@chainlink/contracts/src/v0.8/shared/interfaces/LinkTokenInterface.sol";

contract APIConsumer is ChainlinkClient, ConfirmedOwner {
    using Chainlink for Chainlink.Request;

    uint256 public volume;
    bytes32 private jobId;
    uint256 private fee;

    event RequestVolume(bytes32 indexed requestId, uint256 volume);

    /**
     * @notice Initialize the link token and target oracle
     *
     * Sepolia Testnet details:
     * Link Token: 0x779877A7B0D9E8603169DdbD7836e478b4624789
     * Oracle: 0x6090149792dAAeE9D1D568c9f9a6F6B46AA29eFD (Chainlink DevRel)
     * jobId: ca98366cc7314957b8c012c72f05aeeb
     */
    constructor() ConfirmedOwner(msg.sender) {
        _setChainlinkToken(0x779877A7B0D9E8603169DdbD7836e478b4624789);
        _setChainlinkOracle(0x6090149792dAAeE9D1D568c9f9a6F6B46AA29eFD);
        jobId = "ca98366cc7314957b8c012c72f05aeeb";
        fee = (1 * LINK_DIVISIBILITY) / 10; // 0,1 * 10**18 (Varies by network and job)
    }
}
```

```
function requestVolumeData() public returns (bytes32 requestId) {
    Chainlink.Request memory req = _buildChainlinkRequest(
        jobId,
        address(this),
        this.fulfill.selector
    );

    // Set the URL to perform the GET request on
    req._add(
        "get",
        "https://min-api.cryptocompare.com/data/pricemultifull?fsyms=ETH&tsyms=USD"
    );

    // Set the path to find the desired data in the API response, where the response format is:
    // {"RAW":
    //   {"ETH":
    //     {"USD":
    //       {
    //         "VOLUME24HOUR": xxx.xxx,
    //       }
    //     }
    //   }
    // }
    // request.add("path", "RAW.ETH.USD.VOLUME24HOUR");
    req._add("path", "RAW.ETH.USD.VOLUME24HOUR");
    // Multiply the result by 1000000000000000000 to remove decimals
    int256 timesAmount = 10 ** 18;
    req._addInt("times", timesAmount);

    // Sends the request
    return _sendChainlinkRequest(req, fee);
}

/**
 * Receive the response in the form of uint256
 */
function fulfill(
    bytes32 _requestId,
    uint256 _volume
) public recordChainlinkFulfillment(_requestId) {
    emit RequestVolume(_requestId, _volume);
    volume = _volume;
}

/**
 * Allow withdraw of Link tokens from the contract
 */
function withdrawLink() public onlyOwner {
    LinkTokenInterface link = LinkTokenInterface(_chainlinkTokenAddress());
    require(
        link.transfer(msg.sender, link.balanceOf(address(this))),
        "Unable to transfer"
    );
}
```

Source: <https://docs.chain.link/any-api/getting-started>

::... ChainlinkClient Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

import {Chainlink, ChainlinkClient} from "@chainlink/contracts/src/v0.8/ChainlinkClient.sol";
import {ConfirmedOwner} from "@chainlink/contracts/src/v0.8/ConfirmedOwner.sol";
import {LinkTokenInterface} from "@chainlink/contracts/src/v0.8/LinkTokenInterface.sol";
```

```
contract APIConsumer is ChainlinkClient, ConfirmedOwner {
    using Chainlink for Chainlink.Request;
```

```
    uint256 public volume;
    bytes32 private jobId;
    uint256 private fee;
```

```
    event RequestVolume(bytes32 indexed requestId, uint256 volume);
```

```
    /**
     * @notice Initialize the link token and target oracle
     *
     * Sepolia Testnet details:
     * Link Token: 0x779877A7B0D9E8603169DdbD7836e478b4624789
     * Oracle: 0x6090149792dAAeE9D1D568c9f9a6F6B46AA29eFD (Chainlink DevRel)
     * jobId: ca98366cc7314957b8c012c72f05aeeb
     *
     */
    constructor() ConfirmedOwner(msg.sender) {
        _setChainlinkToken(0x779877A7B0D9E8603169DdbD7836e478b4624789);
        _setChainlinkOracle(0x6090149792dAAeE9D1D568c9f9a6F6B46AA29eFD);
        jobId = "ca98366cc7314957b8c012c72f05aeeb";
        fee = (1 * LINK_DIVISIBILITY) / 10; // 0,1 * 10**18 (Varies by network and job)
    }
}
```

Inizializza il contratto indicando quali sono i contratti del Link Token e dell'Operatore. Anche il Job ID va definito in questa fase.

```
    /**
     * @notice Send a request to the oracle
     *
     * The request is a JSON object with the following fields:
     * - path: The path to the data source (e.g. "RAW.ETH.USD.VOLUME24HOUR")
     * - timesAmount: The number of times to request the data (e.g. 10)
     * - times: The time interval between requests (e.g. 18)
     *
     * The response is a JSON object with the following fields:
     * - volume: The volume of the request (e.g. 1000000000000000000)
     *
     * The request is sent to the oracle via a Chainlink request.
     *
     */
    function _sendChainlinkRequest(Chainlink.Request req, uint256 fee) private {
        // {
        //   "VOLUME24HOUR": xxx.xxx,
        // }
        // }
        // }
        // request.add("path", "RAW.ETH.USD.VOLUME24HOUR");
        req._add("path", "RAW.ETH.USD.VOLUME24HOUR");
        // Multiply the result by 1000000000000000000 to remove decimals
        int256 timesAmount = 10 ** 18;
        req._addInt("times", timesAmount);

        // Sends the request
        return _sendChainlinkRequest(req, fee);
    }

    /**
     * @notice Receive the response in the form of uint256
     */
    function fulfill(
        bytes32 _requestId,
        uint256 _volume
    ) public recordChainlinkFulfillment(_requestId) {
        emit RequestVolume(_requestId, _volume);
        volume = _volume;
    }

    /**
     * @notice Allow withdraw of Link tokens from the contract
     */
    function withdrawLink() public onlyOwner {
        LinkTokenInterface link = LinkTokenInterface(_chainlinkTokenAddress());
        require(
            link.transfer(msg.sender, link.balanceOf(address(this))),
            "Unable to transfer"
        );
    }
}
```


::... ChainlinkClient Contract

Funzione che richiama l'API di cryptocompare.com per prendere i prezzi di scambio ETH/USD

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

import {Chainlink, ChainlinkClient} from "@chainlink/contracts/src/v0.8/ChainlinkClient.sol";
import {ConfirmedOwner} from "@chainlink/contracts/src/v0.8/ConfirmedOwner.sol";
import {LinkTokenInterface} from "@chainlink/contracts/src/v0.8/shared/interfaces/LinkTokenInterface.sol";

contract APIConsumer is ChainlinkClient, ConfirmedOwner {
    using Chainlink for Chainlink.Request;

    uint256 public volume;
    bytes32 private jobId;
    uint256 private fee;

    event RequestVolume(bytes32 indexed requestId, uint256 volume);

    /**
     * @notice Initialize the link token and target oracle
     *
     * * Sepolia Testnet details:
     * * Link Token: 0x779877A7B0D9E8603169DdbD7836e478b4624789
     * * Oracle: 0x6090149792dAAeE9D1D568c9f9a6F6B46AA29eFD (Chainlink DevRel)
     * * jobId: ca98366cc7314957b8c012c72f05aeeb
     *
     */
    constructor() ConfirmedOwner(msg.sender) {
        _setChainlinkToken(0x779877A7B0D9E8603169DdbD7836e478b4624789);
        _setChainlinkOracle(0x6090149792dAAeE9D1D568c9f9a6F6B46AA29eFD);
        jobId = "ca98366cc7314957b8c012c72f05aeeb";
        fee = (1 * LINK_DIVISIBILITY) / 10; // 0,1 * 10**18 (Varies by network and job)
    }
}
```

```
function requestVolumeData() public returns (bytes32 requestId) {
    Chainlink.Request memory req = _buildChainlinkRequest(
        jobId,
        address(this),
        this.fulfill.selector
    );

    // Set the URL to perform the GET request on
    req._add(
        "get",
        "https://min-api.cryptocompare.com/data/pricemultifull?fsyms=ETH&tsyms=USD"
    );

    // Set the path to find the desired data in the API response, where the response format is:
    // { "RAW":
    //   { "ETH":
    //     { "USD":
    //       {
    //         "VOLUME24HOUR": xxx.xxx,
    //       }
    //     }
    //   }
    // }
    // request.add("path", "RAW.ETH.USD.VOLUME24HOUR");
    req._add("path", "RAW.ETH.USD.VOLUME24HOUR");
    // Multiply the result by 1000000000000000000 to remove decimals
    int256 timesAmount = 10 ** 18;
    req._addInt("times", timesAmount);

    // Sends the request
    return _sendChainlinkRequest(req, fee);
}

/**
 * Receive the response in the form of uint256
 */
function fulfill(
    bytes32 _requestId,
    uint256 _volume
) public recordChainlinkFulfillment(_requestId) {
    emit RequestVolume(_requestId, _volume);
    volume = _volume;
}

/**
 * Allow withdraw of Link tokens from the contract
 */
function withdrawLink() public onlyOwner {
    LinkTokenInterface link = LinkTokenInterface(_chainlinkTokenAddress());
    require(
        link.transfer(msg.sender, link.balanceOf(address(this))),
        "Unable to transfer"
    );
}
```

::... ChainlinkClient Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

import {Chainlink, ChainlinkClient} from "@chainlink/contracts/src/v0.8/ChainlinkClient.sol";
import {ConfirmedOwner} from "@chainlink/contracts/src/v0.8/shared/access/ConfirmedOwner.sol";
import {LinkTokenInterface} from "@chainlink/contracts/src/v0.8/shared/interfaces/LinkTokenInterface.sol";

contract APIConsumer is ChainlinkClient, ConfirmedOwner {
    using Chainlink for Chainlink.Request;

    uint256 public volume;
    bytes32 private jobId;
    uint256 private fee;

    event RequestVolume(bytes32 indexed requestId, uint256 volume);

    /**
     * @notice Initialize the link token and target oracle
     *
     * Sepolia Testnet details:
     * Link Token: 0x779877A7B0D9E8603169DdbD7836e478b4624789
     * Oracle: 0x6090149792dAAeE9D1D568c9f9a6F6B46AA29eFD (Chainlink DevRel)
     * jobId: ca98366cc7314957b8c012c72f05aeeb
     */
    constructor() ConfirmedOwner(msg.sender) {
        _setChainlinkToken(0x779877A7B0D9E8603169DdbD7836e478b4624789);
        _setChainlinkOracle(0x6090149792dAAeE9D1D568c9f9a6F6B46AA29eFD);
        jobId = "ca98366cc7314957b8c012c72f05aeeb";
        fee = (1 * LINK_DIVISIBILITY) / 10; // 0,1 * 10**18 (Varies by network and job)
    }
}
```

Chiamata al contratto
Operator di Chainlink con i
parametri richiesti dall'API.

```
function requestVolumeData() public returns (bytes32 requestId) {
    Chainlink.Request memory req = _buildChainlinkRequest(
        jobId,
        address(this),
        "GET https://api.chainlink.com/v1/price?fsyms=ETH&tsyms=USD"
    );
    // request.add("path", "RAW");
    req.add("path", "RAW");
    req.add("path", "RAW");
    // Multiply the result by 1000000000000000000 to remove decimals
    int256 timesAmount = 1000000000000000000;
    req.addInt("times", timesAmount);

    // Sends the request
    return _sendChainlinkRequest(req, fee);
}

/**
 * Receive the response in the form of uint256
 */
function fulfill(
    bytes32 _requestId,
    uint256 _volume
) public recordChainlinkFulfillment(_requestId) {
    emit RequestVolume(_requestId, _volume);
    volume = _volume;
}

/**
 * Allow withdraw of Link tokens from the contract
 */
function withdrawLink() public onlyOwner {
    LinkTokenInterface link = LinkTokenInterface(_chainlinkTokenAddress());
    require(
        link.transfer(msg.sender, link.balanceOf(address(this))),
        "Unable to transfer"
    );
}
```

::... ChainlinkClient Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

import {Chainlink, ChainlinkClient} from "@chainlink/contracts/src/v0.8/ChainlinkClient.sol";
import {ConfirmedOwner} from "@chainlink/contracts/src/v0.8/shared/access/ConfirmedOwner.sol";
import {LinkTokenInterface} from "@chainlink/contracts/src/v0.8/shared/interfaces/LinkTokenInterface.sol";

contract APIConsumer is ChainlinkClient, ConfirmedOwner {
    using Chainlink for Chainlink.Request;

    uint256 public volume;
    bytes32 private jobId;
    uint256 private fee;

    event RequestVolume(bytes32 indexed requestId, uint256 indexed volume);

    /**
     * @notice Initialize the link token and ta
     *
     * Sepolia Testnet details:
     * Link Token: 0x779877A7B0D9E8603169DdbD7836e478b4624789
     * Oracle: 0x6090149792dAAeE9D1D568c9f9a6F6B46AA29eFD (Chainlink DevRel)
     * jobId: ca98366cc7314957b8c012c72f05aeeb
     */
    constructor() ConfirmedOwner(msg.sender) {
        _setChainlinkToken(0x779877A7B0D9E8603169DdbD7836e478b4624789);
        _setChainlinkOracle(0x6090149792dAAeE9D1D568c9f9a6F6B46AA29eFD);
        jobId = "ca98366cc7314957b8c012c72f05aeeb";
        fee = (1 * LINK_DIVISIBILITY) / 10; // 0,1 * 10**18 (Varies by network and job)
    }
}
```

Funzione per gestire il valore di ritorno della chiamata dell'API.

```
function requestVolumeData() public returns (bytes32 requestId) {
    Chainlink.Request memory req = _buildChainlinkRequest(
        jobId,
        address(this),
        this.fulfill.selector
    );

    // Set the URL to perform the GET request on
    req._add(
        "get",
        "https://min-api.cryptocompare.com/data/pricemultifull?fsyms=ETH&tsyms=USD"
    );

    // Set the path to find the desired data in the API response, where the response format is:
    // {"RAW":
    //   {"ETH":
    //     {"USD":
    //       {
    //         "VOLUME24HOUR": xxx.xxx,
    //       }
    //     }
    //   }
    // }
    // request.add("path", "RAW.ETH.USD.VOLUME24HOUR");
    req._add("path", "RAW.ETH.USD.VOLUME24HOUR");
    // Multiply the result by 1000000000000000000 to remove decimals
    int256 timesAmount = 10 ** 18;
    req._addInt("times", timesAmount);

    // Sends the request
    return _sendChainlinkRequest(req, fee);
}

/**
 * Receive the response in the form of uint256
 */
function fulfill(
    bytes32 _requestId,
    uint256 _volume
) public recordChainlinkFulfillment(_requestId) {
    emit RequestVolume(_requestId, _volume);
    volume = _volume;
}

/**
 * Allow withdraw of Link tokens from the contract
 */
function withdrawLink() public onlyOwner {
    LinkTokenInterface link = LinkTokenInterface(_chainlinkTokenAddress());
    require(
        link.transfer(msg.sender, link.balanceOf(address(this))),
        "Unable to transfer"
    );
}
```

::... So, HardHat?

Una volta capito il funzionamento di Chainlink dobbiamo implementare l'intera infrastruttura in locale tramite HardHat.

Nel contratto del Client (construct) vengono richiamati:

- L'indirizzo del contratto **Operator.sol**
- L'indirizzo del contratto **LinkToken.sol**
- Il Job ID associato al nodo che esegue il Job.

L'idea è quella di andare a deployare i due contratti .sol in ambiente locale ed eseguire un Nodo Chainlink tramite Docker

::... HardHat Setup

1. Creiamo un nuovo progetto npm
2. Importiamo hardhat.
3. Inizializziamo un progetto hardhat.
4. Importiamo la dipendenza alla libreria Chainlink (v. 0.6.1)
5. Creiamo due smart contract:



```
npm init
npm i hardhat
npm i hardhat init
npm i @chainlink/contracts@0.6.1
```

Operator.sol

```
contracts > Operator.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.7.6;
3 import "@chainlink/contracts/src/v0.7/Operator.sol";
```

LinkToken.sol

```
contracts > LinkToken.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.4.16;
3 import "@chainlink/contracts/src/v0.4/LinkToken.sol";
```

::... HardHat Setup

1. Creiamo un nuovo progetto npm
2. Importiamo hardhat.
3. Inizializziamo un progetto hardhat
4. Importiamo la dipendenza libreria Chainlink (v. 0.6.1)
5. Creiamo due smart contract:

Gli smart contract usano versioni di solc diverse. Pertanto è necessario modificare il file `hardhat.config.js`

ts@0.6.1

Operator.sol

```
contracts > Operator.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.7.6;
3 import "chainlink/contracts/src/v
```

```
require("@nomicfoundation/hardhat-toolbox");

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: {
    compilers: [
      {
        version: "0.7.6",
      },
      {
        version: "0.4.16",
      },
    ],
  },
};
```

LinkToken.sol

```
LinkToken.sol
License-Identifier: MIT
solidity ^0.4.16;
chainlink/contracts/src/v0.4/LinkToken.sol";|
```

::... HardHat Setup

Prima di deployare i due smart contract mettiamo i due .sol nella cartella /contracts ed osserviamo i costruttori.

```
function LinkToken()  
    public  
{  
    balances[msg.sender] = totalSupply;  
}
```

LinkToken.sol

Operator.sol

```
/**  
 * @notice Deploy with the address of the LINK token  
 * @dev Sets the LinkToken address for the imported LinkTokenInterface  
 * @param link The address of the LINK token  
 * @param owner The address of the owner  
 */  
constructor(address link, address owner) ConfirmedOwner(owner) {  
    linkToken = LinkTokenInterface(link); // external but already deployed and unalterable  
}
```

::... HardHat Setup

Prima di deployare i due smart contract mettiamo i due .sol nella cartella /contracts ed osserviamo i costruttori.

```
function LinkToken()  
    public  
{  
    balances[msg.sender] = total  
}
```

LinkToken.sol

L'Operator.sol richiede due parametri, ovvero l'indirizzo del Token e dell'owner.

Operator.sol

```
/**  
 * @notice Deploy with the address of the LINK token  
 * @dev Sets the LinkToken address for the imported LinkTokenInterface  
 * @param link The address of the LINK token  
 * @param owner The address of the owner  
 */  
constructor(address link, address owner) ConfirmedOwner(owner) {  
    linkToken = LinkTokenInterface(link); // external but already deployed and unalterable  
}
```


::... HardHat Setup

Dunque, creiamo il file `deploy.js` e assicuriamoci di passare i parametri corretti per il deployment. Ricordiamoci di stampare gli indirizzi in quanto ci serviranno per il deployment del `ChainlinkClient`.

```
const LinkToken = await ethers.getContractFactory("LinkToken");
const Operator = await ethers.getContractFactory("Operator");
const token = await LinkToken.deploy();
const operator = await Operator.deploy(token.target, address);

console.log("Token address:", token.target);
console.log("Operator address:", operator.target);
```

Il deployment può essere effettuato invocando il seguente comando

```
npx hardhat run ./deploy.js --network local
```

Assicurandoci di
aver eseguito il
nodo locale con

```
npx hardhat node
```

::... Chainlink Setup

Una volta capito il funzionamento di Chainlink dobbiamo implementare l'intera infrastruttura in locale tramite HardHat.

Nel contratto del Client (construct) vengono richiamati:

- ~~L'indirizzo del contratto Operator.sol~~
- ~~L'indirizzo del contratto LinkToken.sol~~
- Il Job ID associato al nodo che esegue il Job.

Dobbiamo ora avviare il docker container per eseguire un nodo Chainlink

1. Avviamo un'istanza di Postgresql
2. Settiamo i file di configurazione di Chainlink Node
3. Eseguiamo l'istanza di Chainlink Node

```
docker run --name cl-postgres -e POSTGRES_PASSWORD=mysecretpassword -p 5432:5432 -d postgres
```

Source: <https://docs.chain.link/chainlink-nodes/v1/running-a-chainlink-node>

::... Chainlink Setup

```
mkdir ~/.chainlink-sepolia  
echo "[Log]  
Level = 'warn'
```

```
[WebServer]  
AllowOrigins = '*'  
SecureCookies = false
```

```
[WebServer.TLS]  
HTTPSPort = 0
```

```
[[EVM]]  
ChainID = '31337'
```

```
[[EVM.Nodes]]  
Name = 'HardHat'  
WSURL = 'ws://host.docker.internal:8545'  
HTTPURL = 'http://host.docker.internal:8545'  
" > ~/.chainlink-sepolia/config.toml
```

secrets.toml

```
echo "[Password]  
Keystore = 'mysecretkeystorepassword'  
[Database]  
URL = 'postgresql://postgres:mysecretpassword@host.docker.internal:5432/postgres?sslmode=disable'  
" > ~/.chainlink-sepolia/secrets.toml
```


config.toml

::... Chainlink Setup

Una volta salvati i file di configurazione è possibile eseguire l'istanza del Chainlink Node assicurandosi di indicare correttamente il path in cui sono contenuti i file nel parametro -v

```
cd ~/.chainlink-sepolia && docker run --platform linux/x86_64/v8 --name chainlink  
-v ~/.chainlink-sepolia:/chainlink -it -p 6688:6688 smartcontract/chainlink:2.17.0  
node -config /chainlink/config.toml -secrets /chainlink/secrets.toml start
```



 Chainlink Operator

JobsRunsChainsNodesBridgesTransactionsJob Distributors

Activity

New Job

No recent activity

Account Balances

Chain ID 31337

Address

0x7F719DaF4F71e22F07846F393A41750abFAfaca0

Native Token Balance

LINK Balance

0.000000000000000000--

Recent Jobs

No recently created jobs

::... Job Setup

Cliccando su «New Job» è possibile creare un nuovo Job per questo nodo. Ricordiamo che il job è composto da più task con l'obiettivo finale di interrogare un'API. Tornando all'esempio potremmo scrivere un Job come segue.

```
type = "directrequest"
schemaVersion = 1
name = "Fetch ETH Price"
forwardingAllowed = false
maxTaskDuration = "20s"
contractAddress = "YOUR_ORACLE_ADDRESS"
evmChainID = "31337"
minContractPaymentLinkJuels = "0"
observationSource = ""

decode_log  [type="ethabiencode" abi="OracleRequest(bytes32 indexed specId, address requester, bytes32 requestId,
uint256 payment, address callbackAddr, bytes4 callbackFunctionId, uint256 cancelExpiration, uint256 dataVersion, bytes data)"
data="${jobRun.logData}" topics="${jobRun.logTopics}"]
decode_cbor [type="cborparse" data="${decode_log.data}"]
fetch [type="http" method="GET" url="http://https://min-api.cryptocompare.com/data/price?fsym=ETH&tsyms=BTC,USD,EUR" allowUnrestrictedNetworkAccess=true]
encode_data [type="ethabiencode" abi="(bytes32 _requestID)" data="{ \"_requestID\": ${decode_log.requestId}}"]
encode_tx   [type="ethabiencode"
abi="fulfillOracleRequest2(bytes32 requestId, uint256 payment, address callbackAddress,
bytes4 callbackFunctionId, uint256 expiration, bytes calldata data)"
data="{\"requestID\": ${decode_log.requestId}, \"payment\": ${decode_log.payment},
\"callbackAddress\": ${decode_log.callbackAddr}, \"callbackFunctionId\": ${decode_log.callbackFunctionId},
\"expiration\": ${decode_log.cancelExpiration}, \"data\": ${encode_data}}"]
]
submit_tx   [type="eth_tx" to="YOUR_ORACLE_ADDRESS" data="${encode_tx}" gasLimit="500000"]

decode_log → decode_cbor → fetch → encode_data → encode_tx → submit_tx
""
```

::... Job Setup

Cliccando su «New Job» è possibile creare un nuovo Job per questo nodo. Ricordiamo che il job è composto da più task con l'obiettivo finale di interrogare un'API. Tornando all'esempio potremmo scrivere un Job come segue.

The screenshot shows the Chainlink job setup interface for a job named "Fetch ETH Price". The interface includes a table with job details, tabs for "Overview", "Definition", "Errors", and "Runs", and a section for "Recent job runs".

ID	Type	External Job ID	Created
1	Direct Request	c02dd0da-22a6-48e3-a1f2-63ab3747cc14	just now

Below the table, there are tabs for "Overview", "Definition", "Errors", and "Runs". The "Overview" tab is currently selected. Under the "Overview" tab, there is a section titled "Recent job runs" which states "No jobs have been run yet".

A red box highlights the "External Job ID" field, which contains the value "c02dd0da-22a6-48e3-a1f2-63ab3747cc14". A red arrow points from this box to a red callout box on the right.

The callout box contains the text: "Ecco il Job ID da utilizzare all'interno del ClientChainlink".

Below the callout box, there is a diagram showing two tasks: "decode_log" and "decode_cbor", connected by a line.

::... Technical Steps

A questo punto è necessario trasferire degli ETH al nodo Chainlink usando il suo indirizzo e i task di HardHat.

Account 3

→


0x7F719...fac...

SENDING ETH

T 50

\$122,105.00

Estimated fee

 \$0.08

0.00003284 ETH

Max fee: 0.00003284 ETH

Totale

\$122,105.08

50.00003284 ETH

Amount + gas fee

Max amount: 50.00003284 ETH

Account Balances

Chain ID 31337

Address

0x7F719DaF4F71e22F07846F393A41750abFAfaca0

Native Token Balance

LINK Balance

0.000000000000000000

--

Account Balances

Chain ID 31337

Address

0x7F719DaF4F71e22F07846F393A41750abFAfaca0

Native Token Balance

LINK Balance

50.000000000000000000

--

::... Technical Steps

Ora è necessario :

1. Invocare la funzione `setAuthorizedSenders` (con parametro [indirizzo del nodo]) presente nel contratto `Operator.sol` per autorizzare il nodo a partecipare alla rete.
2. Inviare dei LINK al contratto `ClientChainlink` necessari per effettuare la richiesta alla rete Chainlink.
3. Deployare il contratto `ClientChainlink` con i parametri dell'`Operator`, `LinkToken` e `Job ID` ottenuti seguendo la guida.