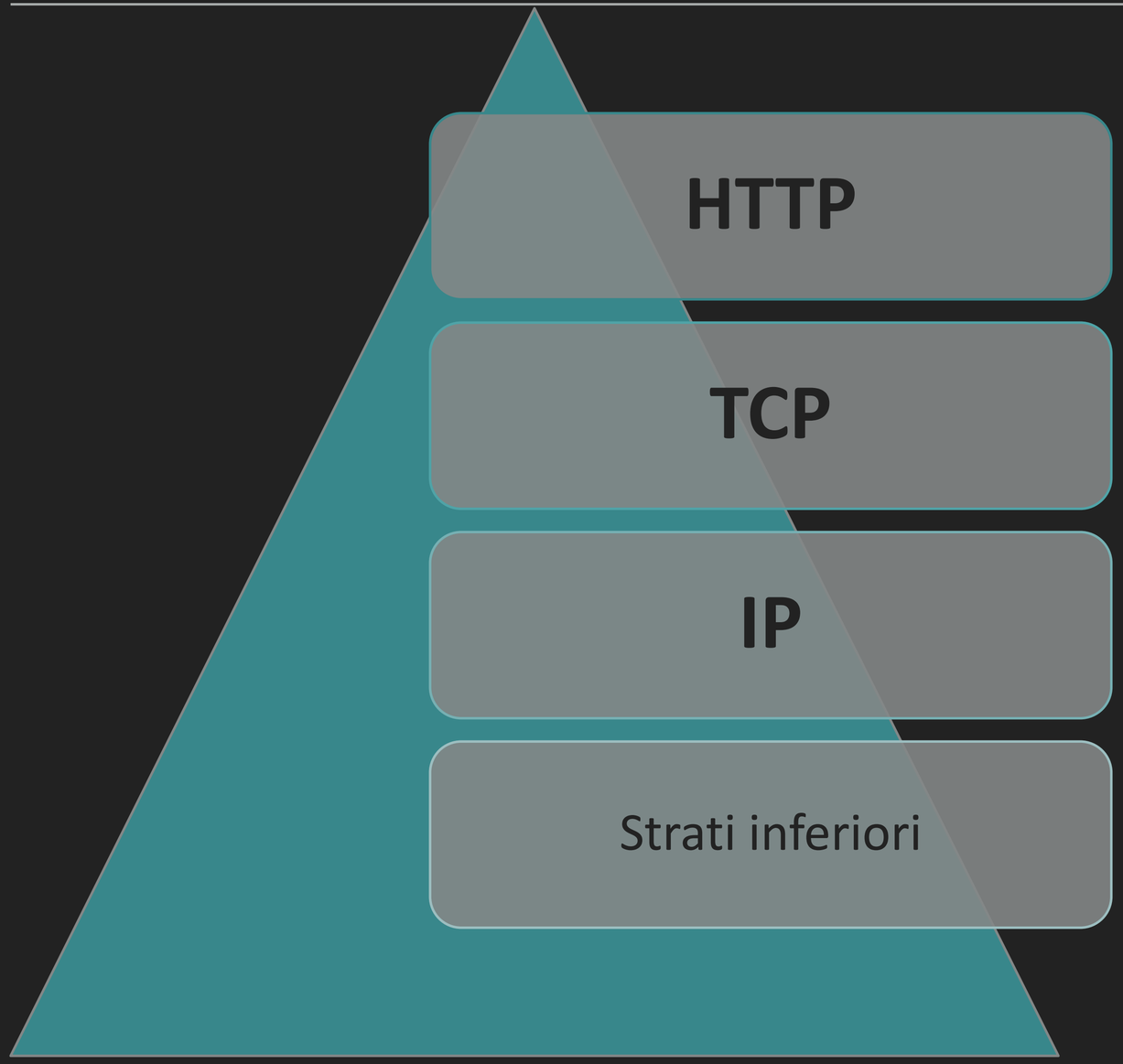
NODEJS STANDARD LIBRARY

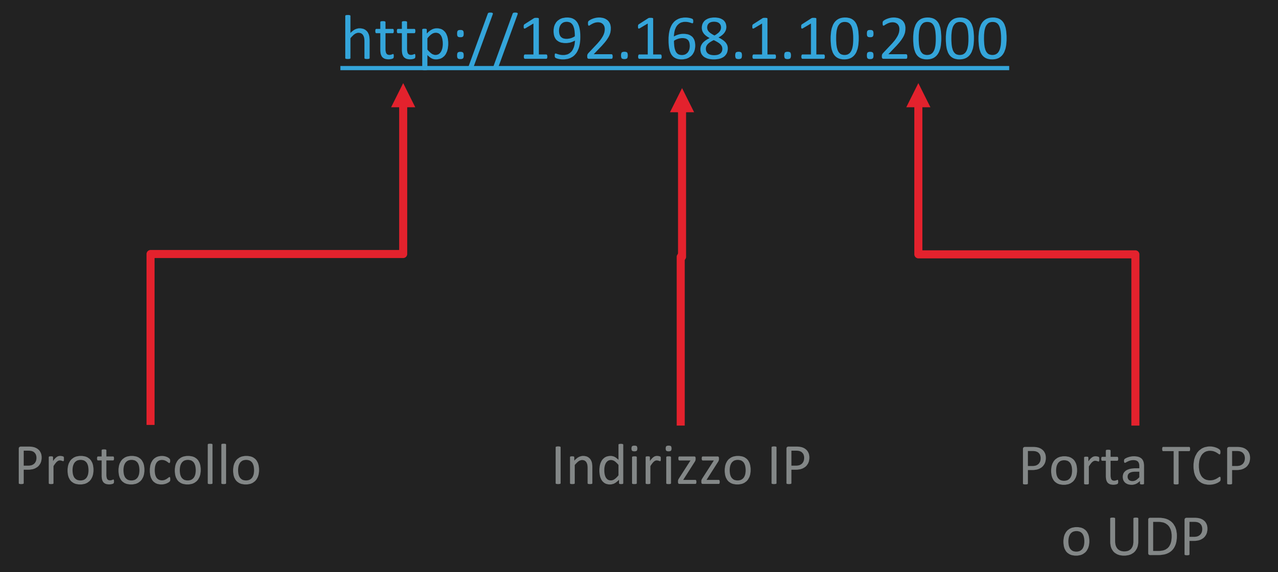
MODULO HTTP(S)

NODEJS STANDARD LIBRARY – PROTOCOLLO HTTP



- HTTP (HyperText Transfer Protocol) è un protocollo nato per il trasferimento di dati sul web (prevede quindi un'architettura client-server).
- E' un protocollo a strato applicativo, ovvero serve per il funzionamento di applicazioni ad alto livello (pagine web).
- Con il tempo si è evoluto fino agli utilizzi moderni (API REST) e ha implementato la comunicazione criptata (HTTPS).
- Solitamente quando si ha a che fare con HTTP (o di reti in generale) si notano gli *indirizzi* del tipo
  http://192.168.10.1:2000
- In questi indirizzi abbiamo un riferimento univoco all'interno di una rete: in particolare troviamo il protocollo utilizzato, un indirizzo IP e una porta TCP.

Analizziamo un indirizzo



- HTTP è solo uno dei tanti protocolli comunemente utilizzati nell'ambito di internet e delle reti.
- Alcuni esempi di altri protocolli sono SMTP, POP3 e IMAP per l'invio e la ricezione di email, FTP per il trasferimento file, DNS per la conversione da nome di un server a indirizzo IP, etc.

Cos’è un indirizzo IP?

- IP (Internet Protocol) è il protocollo fondante di internet. Permette a reti realizzate in maniera eterogenea di comunicare mediante un protocollo comune. Per permettere la comunicazione tra ogni coppia di terminali che si affacciano sulla rete è necessario poterli identificare in maniera univoca.
- L’indirizzo IP(v4) è una stringa che identifica univocamente un dispositivo in una rete. E’ composta da 4 cifre, variabili da 0 a 255, separate da un punto.
- In realtà alcuni indirizzi IP sono considerati privati, cioè utilizzabili in reti che non si affacciano su internet. In questo modo si evitano sprechi di indirizzi poiché in queste reti isolate non è necessario usare indirizzi diversi.

Cos’è una porta TCP?

- TCP (Transfer Control Protocol) è il protocollo che rende internet affidabile. Si preoccupa di controllare che i dati arrivino correttamente a destinazione ed eventualmente ritrasmetterli e di regolare inoltre il flusso sulla base della banda disponibile.
- Permette inoltre di instaurare più di una connessiona da/verso un indirizzo IP permettendo comunicazioni parallele.
- Per discriminare le singole connessioni attive su un indirizzo IP si usa l’indirizzo TCP (normalmente chiamato porta). Il numero di porta può variare da 0 a 65535.
- Le porte da 0 a 1023 sono riservate (*well known ports*) ed utilizzate per protocolli comuni; ad esempio la porta 80 è la standard del protocollo HTTP.

HTTP

- Il protocollo HTTP è client/server, ovvero un client (e.g. browser) invia una richiesta ad un server e quest'ultimo restituisce una risposta.
- Esistono quindi due tipi di messaggio HTTP: richieste e risposte.
- Entrambe le tipologie di messaggio sono molto simili:

- Metodo di richiesta
- Intestazioni di richiesta (*HEADERS*)
- Corpo della richiesta (*BODY*)

- Codice di risposta
- Intestazioni di risposta (*HEADERS*)
- Corpo della risposta (*BODY*)

HTTP Request

- Il primo campo di una richiesta HTTP è il metodo. Il metodo serve per indicare al server quale tipo di servizio si intende richiedere. Alcuni metodi sono:
  - GET
  - HEAD
  - POST
  - PUT
  - DELETE
- Solitamente le richieste GET si usano per richiedere dati al server e non ammettono body. La HEAD è come la GET ma richiede solo gli HEADERS. POST, PUT e DELETE inviano dati al server, solitamente per scopi diversi.

HTTP Request

- Gli Headers sono una serie di coppie chiave-valore utilizzate per inviare metadati sulla richiesta. Ad esempio tutti i browser, quando effettuano richieste, inseriscono un header chiamato *User-Agent* che specifica il tipo di browser usato dall'utente.
- Alcuni headers comuni sono:
  - User-Agent
  - Content-Type
  - Accept-Language
  - Authorization
  - Host

HTTP Request

- Il body è invece l'eventuale insieme di dati che il client invia al server.
- E' possibile utilizzare vari formati (testuale, JSON, binario).
- Solitamente il formato del body è specificato dall'header *Content-Type*
  - *Content-Type : application/json*

Indica che il body è un testo
formattato come JSON

HTTP Response

- La risposta HTTP è, nei campi Headers e Body, esattamente equivalente ad una richiesta. Differisce però perché al posto del metodo c'è il cosiddetto *status code*, un codice numerico che specifica l'esito della richiesta.
- 2xx. Richiesta eseguita con successo (e.g. codice 200)
- 4xx. Errore nella richiesta lato client (e.g codice 404 Not Found, o codice 401 Unauthorized)
- 5xx. Errore lato server (e.g. codice 500)
- Per una documentazione completa sui codici di risposta è possibile consultare la documentazione Mozzilla al link HTTP response status codes - HTTP | MDN (mozilla.org)

Postman

- Per aiutare gli sviluppatori nella programmazione di richieste HTTP è stato creato il tool Postman (scritto in NodeJS).
- L'app è scaricabile al seguente indirizzo Download Postman | Try Postman for Free, e permette di eseguire con facilità richieste HTTP.

Modulo HTTP

- Il modulo http di node fornisce strumenti utili per agire da client HTTP o da server HTTP.
- Esistono framework che aiutano lo sviluppatore con ulteriori astrazioni (e.g. express, axios), sia per agire da client che per agire da server.
- Tuttavia per effettuare richieste i metodi esposti da Node sono abbastanza semplici, mentre è più complesso creare un server HTTP.

- Importiamo il modulo http


```
const http = require('http');
```


```
import * as http from 'http';

import {request} from 'http';
```

- Ci sono tre oggetti fondamentali nel modulo http, *ClientRequest, ServerResponse,* e *Server.*
- ClientRequest e ServerResponse sono create internamente e resistuite all'utente. Contengono i riferimenti a header, body, status code, e metodo http.
- Entrambe sono stream, in particolare *ClientRequest* è un Writable (perché una richiesta può inviare dati al server) e *ServerResponse* è un Readable (perché il client riceve dati dal server).
- La classe *Server* invece si usa per creare un server HTTP.

- http.request()

- Input:
  - options — oggetto con i parametri di richiesta
  - callback — funzione con l'oggetto ServerResponse da gestire
- Restituisce
  - Oggetto di tipo ClientRequest

- http.request(), l'oggetto options

```
{
    auth: stringa di autorizzazione (solitamente nel formato user:password),
    headers: oggetto contenente gli headers,
    host: indirizzo IP o nome del server,
    method: stringa con il metodo HTTP,
    path: percorso nel server remoto,
    port: porta (default 80)
}
```

- http.request(), esempio

```
import * as http from ‘http’;

let options = {
        hostname: ‘google.it',
        port: 443,
        path: ‘/’,
        method: 'GET'
}

let req = http.request(options, res => {
        console.log('statusCode: ' + res.statusCode);
        res.on('data', chunk => {
                        console.log(chunk);
        });
});

req.on('error', error => {
        console.error(error)
});

req.end();
```

- http.ServerResponse
  - Campi fondamentali:
    - statusCode — codice della risposta
  - Metodi fondamentali
    - getHeaders() — restituisce tutti gli headers

- http.ClientRequest

- Campi fondamentali:
  - host, path, method… tutte le opzioni viste nel metodo http.request()
- Metodi fondamentali
  - setHeader(name, value) imposta un header
  - removeHeader(name) rimuove un header

- http.createServer()

- Campi fondamentali:
  - requestListener — funzione che specifica come rispondere ad una richiesta

- http.createServer(), esempio

```
import * as http from ‘http’;

let server = http.createServer((req, res) => {
        res.statusCode = 200;
        res.setHeader(‘Content-Type’, ‘application/json’);
        res.end();
});

server.listen();
```