DNS

Struktúrája:

* **Domain Name Space**: Maga a DNS hierarchia, tartománynevek szervezésének az alapja, a tartománynevek pontokkal elválasztott részletek sorozata melyek az internet egyes szolgáltatásait vagy számítógépeket, szerverek azonosítanak.
* **Domain Name Server**: Névszerverek, ezek tárolják a tartományokat és azokhoz tartozó IP címeket. (lehetővé teszi a név alapján való keresést (IP cím ismerete nélkül))
* **DNS-Resolver**: Szoftver, mely megkeresi egy tartomány név alapján az ahhoz tartozó IP címet. (alkalmazások által küldött kéréseket kezel)
* **Root DNS Server**: Tartalmazzák a teljes tartománynév-területet lefedő DNS-rekordokat és irányítják a DNS-kéréseket. (DNS hierarchia legtetején van)
* **Top-Level Domain (TLD) Servers**: Ezen szerverek kezelik a TDL-eket (pl.: .com,.hu,.org) minden szerver a saját specifikus tartományát kezeli. (előzőleg felsorolt tartományok)
* **Authoritative Name Servers**: Ezen szerverek felelősek egy adott tartománynévhez rendelt IP-címek vagy egyéb DNS-rekordok tárolásáért és kezeléséért, például az "example.com" tartománynevéhez tartozó IP-címet adják vissza DNS-kérések esetén.

Működése:

1. Amikor egy felhasználó a böngészőjében vagy más alkalmazásban egy tartománynevet (pl. [www.vanenet.](http://www.vanenet.)hu) ír be, a DNS-feloldó a rendszerbe küld egy DNS-kérést.
2. A DNS-feloldó elkezdi a DNS-kérés feldolgozását, kezdve a helyi cache-ek ellenőrzésével. Ha a keresett rekordot megtalálja a cache-ben, akkor azonnal visszaadja az IP-címet, különben folytatja a keresést.
3. Ha a cache nem tartalmazza a keresett rekordot, akkor a DNS-feloldó továbbítja a kérést a gyökér DNS-szerverekhez.
4. A gyökér DNS-szerverek átirányítják a kérést a megfelelő TLD-szerverekhez.
5. A TLD-szerverek továbbítják a kérést az adott tartományt kezelő szerződéses névszerverekhez.
6. A szerződéses névszerverek visszaküldik az IP-címet vagy a kért DNS-rekordot a DNS-feloldónak.
7. A DNS-feloldó visszaadja az IP-címet a böngészőnek vagy alkalmazásnak, és ezzel lehetővé teszi az internetes erőforrás elérését.

REST és SOAP API

|  |  |
| --- | --- |
| SOAP | REST |
| A SOAP web provisioning szolgáltatások megváltoztatása gyakran bonyolult kódváltozással jár a kliens oldalon. | A REST webprovisioningban a szolgáltatások megváltoztatása nem igényel változtatást a kliens oldali kódban. |
| Nagyobb a terhelése | Könnyűsúlyú, kifejezetten könnyű adatátvitelt biztosít egy a legtöbb esetben ismert felületen, - az URI-n keresztül. |
| Bináris csatolmány elemzést igényel. | A REST közvetlenül támogatja az összes adattípust. |
| A SOAP nem igazán kompatibilis a vezeték nélküli infrastruktúrához. | A REST kompatibilis a vezeték nélküli infrastruktúrához. |
| A SOAP web szolgáltatások mindig XML adatot adnak vissza. | Míg a REST webes szolgáltatások rugalmasságot biztosítanak a visszaadott adattípusok tekintetében. |
| SOAP válaszként sokkal több sávszélességet használ fel, mivel egy SOAP válasz akár több mint tízszer annyi bájtot is igényelhet, mint a REST. | Kevesebb sávszélességet használ fel, mert a válasz könnyűsúlyú. |
| A SOAP kérések POST módszert használnak, és egy összetett XML kérést igényelnek a létrehozáshoz, ami megnehezíti a válasz gyorsítótárazását. | A RESTful API-k egyszerű GET kérésekkel fogyaszthatók, köztes proxy szerverek / fordított proxy-k könnyen gyorsítótárba helyezhetik válaszaikat. |
| A SOAP HTTP-alapú API-kat használ, amelyeket egy vagy több HTTP URI-ként tesznek elérhetővé, és a tipikus válaszok XML / JSON formátumban érkeznek. A válasz sémák egyediek az adott objektumhoz. | A REST másrészről a standardizált URI-k használatának elemeit is hozzáadja, és fontosnak tartja az alkalmazott HTTP műveletet (pl. GET / POST / PUT stb.). |
| Nyelv-, platform- és szállításfüggetlen. | Nyelv- és platformfüggetlen. |
| A distributed computing környezetek kezelésére tervezve. | Egy ponttól-pontig kommunikációs modellt feltételez - nem alkalmas elosztott számítási környezetre, ahol az üzenet áthaladhat egy vagy több köztes csatornán. |
| Nehéz fejleszteni, eszközök szükségesek hozzá. | A webes szolgáltatások fejlesztése sokkal egyszerűbb, mint a SOAP esetében. |
| A webes szolgáltatások terén uralkodó szabvány, és ezért jobb támogatást kap más szabványoktól (WSDL, WS) és eszközöktől a szolgáltatóktól. | A biztonság, a politika, az üzenetküldés megbízhatósága stb. terén hiányoznak a szabványos támogatások, így azok a szolgáltatások, amelyeknek kifinomultabb követelményeik vannak, nehezebben fejleszthetők. |

Szoftverfejlesztéshez használatos hálózati protokollok:

**SSH (Secure SHell):**

Az SSH (Secure Shell) egy kriptografikus hálózati protokoll, mely biztosítja a biztonságos kommunikációt két távoli számítógép között. A főbb tulajdonságai közé tartozik a titkosított adatkommunikáció, széleskörű hitelesítési lehetőségek biztosítása (mint például jelszóalapú vagy kulcs-alapú hitelesítés), távoli hozzáférés lehetősége. Az SSH segítségével lehetővé válik a szerverek távoli kezelése, fájlok biztonságos átvitele és tárolása, például a man-in-the-middle támadásokkal szemben. Az SSH a TCP, UDP, STCP portokat használhatja.

**FTP (File Transfer Protocol):**

Az FTP két módban működhet: aktív és passzív. Aktív módban a szerver a 20-as portról kezdeményezi az adatkapcsolatot a klienssel, míg passzív módban a kliens a PASV parancs segítségével kap szerver IP-címet és portszámot, amire kapcsolódik. Ez a módszer segít megoldani a tűzfalak és NAT által okozott kapcsolati problémákat, mivel lehetővé teszi a bejövő kapcsolatok fogadását anélkül, hogy közvetlen kapcsolatot kellene kezdeményezni a kliens felől.

**SMTP (Simple Mail Transfer Protocol):**

E-mailek küldéséhez a levelezőkliens (MUA, Mail User Agent) SMTP (Simple Mail Transfer Protocol) protokollon keresztül, a TCP (Transmission Control Protocol) 587-es portját használva küldi az üzenetet egy levelezőszerverre (MSA, Mail Submission Agent), amely továbbítja azt a levelezőküldő ügynökhöz (MTA, Mail Transfer Agent). Az MTA-k feladata az üzenetek továbbítása lehet egyszerű, egy gépen történő feldolgozás, vagy több gépen szétosztott munka, ahol az MTA-k SMTP-t használva kommunikálnak egymással. A határ MTA a címzett domainjének MX (Mail Exchanger) rekordját DNS (Domain Name System) segítségével keresi meg, és az alapján létesít kapcsolatot a címzett szervereivel az üzenet továbbítása érdekében. Az SMTP protokoll csak az üzenetátvitelt definiálja, nem a tartalmat, és parancsok és válaszok sorozatával működik egy megbízható TCP kapcsolaton keresztül.

**HTTP**:

HTTP (Hypertext Transfer Protocol) a kliens-szerver modellben működik, ahol a kliens, például egy webböngésző, HTTP kérésüzeneteket küld a szervernek. A szerver, amely weboldalakat vagy egyéb tartalmakat szolgáltat, válaszüzeneteket küld vissza. Az HTTP lehetővé teszi az intermediális hálózati elemek, mint például gyorsítótár-szerverek és proxyk használatát, amelyek javíthatják a kommunikációt és csökkenthetik a hálózati forgalmat. Az HTTP fejléceket különböző módon kezelik (hop-by-hop vagy végponttól-végpontig), és a protokoll az alkalmazási rétegben helyezkedik el az internetes protokollhalmazban.

**HTTPS**:

HTTPS (HTTP Secure) hasonló szintaktikát használ, mint az HTTP, de SSL/TLS (Secure Socket Layer/Transport Layer Security) titkosítási réteget ad hozzá a kommunikáció biztonságának növelése érdekében. Az HTTPS biztonságos csatornát biztosít a kliensek és szerverek közötti adatátvitel során, különösen fontos a bizonytalan hálózati környezetekben, mint például nyilvános Wi-Fi hálózatokon. Az HTTPS használata növeli a weboldalak biztonságát és lehetővé teszi az újabb HTTP verziók, mint az HTTP/2 és HTTP/3 használatát, amelyek javítják a lapbetöltési sebességet és csökkentik a késleltetést. Az HTTPS szerverek megbízhatóságát előre telepített tanúsítványok garantálják, és az HTTP Strict Transport Security (HSTS) használatát ajánlják a közbeékeléses támadások elleni védelem érdekében.

**Http hibakódok**

* (1xx – tájékoztatás)
* 2xx – sikeres kérés
* 3xx – átirányítás
* 4xx – klienshiba
* 5xx – szerverhiba

**2xx**

* 200 – OK – sikeres kérés
* 201 – Created – sikeres kérés, tartalom létrehozásra került
* 202 – Accepted – kérés elfogadva, de a feldolgozás nem fejeződött be
* 204 – No Content – kérés sikeresen feldolgozva, nem küld vissza a szerver tartalmat

**3xx**

* 300 - Multiple Choices –több opció felkínálása, amiből a kliens választhat
* 301 – Moved Permanently – véglegesen megváltozott a tartalom URL-je
* 302 – Found – átmenetileg megváltozott a tartalom URL-je

**4xx**

* 400 – Bad Request – kérés nem feldolgozható kliensoldali hiba miatt, pl. szintaktikai hiba, túl nagy méret
* 401 – Unauthorized – a szerver értelmezte a kérést, de megtagadta a hozzáférést, mert hitelesítés szükséges
* 403 – Forbidden – a szerver értelmezte a kérést, és megtagadta a hozzáférést
* 404 – Not Found – a tartalom nem található
* 405 – Method Not Allowed – nem támogatott http metódussal érkezett a kérés

**5xx**

* 500 – Internal Server Error – váratlan hiba lépett fel, ezért nem teljesíthető a kérés
* 501 – Not Implemented – a szerver nem ismerte fel a http metódust, vagy pedig nem képes teljesíteni a kérést
* 502 – Bad Gateway – a szerver átjáróként vagy proxy-ként működött, és érvénytelen választ kapott az upstream szervertől
* 503 – Service Unavailable – a szerver nem tud kéréseket fogadni, pl. túlterhelés, karbantartás miatt
* 504 – Gateway Timeout - a szerver átjáróként vagy proxy-ként működött, és nem kapott időn belül választ az upstream szervertől
* 505 – Http Version Not Supported

**Http metódusok**

Leggyakoribb metódusok a GET és POST, ezen kívül még létezik PUT, HEAD, DELETE, PATCH, OPTIONS, CONNECT, TRACE

* GET: Reprezentációt kér a meghatározott tartalomból. Csak adatok kérésére használják. Cache-elhető, a böngésző előzmények között maradnak, hosszuk korlátozott.
* POST: Adatot küld a szervernek, hogy tartalom kerüljön létrehozásra, frissítésre. Nem cache-elhető, nem marad az előzmények között és nincs korlátozás a hosszukra.
* PUT: A cél összes reprezentációját lecseréli a tartalmára (felülír). Adatot küld a szervernek a POST-hoz hasonlóan. A különbség az, hogy ugyanannak a PUT requestnek többszöri meghívására egyszer lesz a tartalom létrehozva. A POST ezzel szemben többször is létrehozza.
* HEAD: A GET-tel majdnem megegyezik a funkciója azzal a különbséggel, hogy a válasz nem tartalmaz response body-t.
* DELETE: Törli a megadott tartalmat.
* OPTIONS: A szerver által támogatott metódusok listáját adja vissza.
* PATCH: Részleges változtatásokat végez a tartalmon.
* CONNECT: Tunnelt hoz létre a szerverrel.
* TRACE: Visszaküldi a kapott kérést (loopback) a tartalom elérési útjával.

**HTTP fejlécek szerepe**

A http fejlécek lehetővé teszik, hogy további információkat adjon át a szerver és a kliens egymásnak egy http kéréssel vagy válasszal. Csoportosításuk: request, response, representation, payload.

* User-Agent: Request fejléc, ami felismeri a kérést küldő user agentről\* a következő információkat: alkalmazás neve, verziószám, operációs rendszer, egyéb információk  
  pl. User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36   
  \*user agent: egy személy reprezentálására használt program
* Authorization: Request típusú, általában akkor használják, amikor a user agent\* megpróbál elérni hitelesítést igénylő védett tartalmat. Ez a fejléc tartalmazza az autentikációhoz szükséges hitelesítő adatokat (felhasználónév, jelszó).  
  pl. Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=
* Set-Cookie: Http-response fejléc, amit a szerver használ egy cookie elküldésére a user agenthez. Tartalmazza a nevet, értékét, és egyéb információt pl. lejárati dátum. A cookie-t a válasz fogadása után lokálisan tárolják, és automatikusan csatolva lesz a további kérésekhez ugyanarra a domainre.  
  pl. Set-Cookie: sessionId=abc123; Expires=Wed, 09 Jun 2024 10:18:14 GMT; Path=/
* Accept: Request típusú. A kliens használja annak jelzésére a szerver felé, hogy milyen formátumú és típusú tartalmat fogad el a szervertől. Opcionálisan tartalmazhat paraméterként karakterkészletet, nyelvet stb.  
  pl. Accept: text/html
* Content-Type: A szerver használja arra, hogy jelezze a küldött média típusát a kliens felé. Használják pl. PUT, POST requesteknél.  
  pl. Content-Type: application/x-www-form-urlencoded
* Content-Disposition: Response header, ami jelzi, hogy a tartalmat inline (weboldalként vagy weboldal részeként) kell megjeleníteni vagy pedig csatolni, és letölteni és menteni lokálisan.  
  pl. Content-Disposition: attachment; filename="example.txt",  
  Content-Disposition: inline
* Location: 3xx kódú válaszokban, átirányításkor használják, vagy pedig 201-nél, létrehozásnál.   
  pl. Location: https://example.com/new-page

Példa: Könyvtárnyilvántartó programhoz API készíteni.

1. Adatmodell kialakítása: Könyv:
   1. ID: Egyei azonosító
   2. author: szerző
   3. title: cím
   4. genre: műfajok
   5. status: kivan-e kölcsönözve
2. Felhasznált technológiák:
   1. Backend: Java
   2. Adatbázis: PostgreSQL
3. API specifikáció
   1. Könyv adatai:
      * Könyvek keresése
      * URI: /books/{ID}/data
      * Módszer: GET
      * Paraméterek:
        1. title
        2. author
        3. genre
   2. Könyv hozzáadása:
      * URI: /books/add
      * Paraméter: ID
      * Módszer: POST
   3. Könyv törlése
      * URI: /books/{ID}/delete
      * Módszer: DELETE
      * Válasz: Törlési státusz.
   4. Könyv frissítése
      * URI: /books/{ID}/update
      * Módszer: PUT
   5. Hiba kezelés
      * 404 Not Found: Ha a keresett erőforrás nem található.
      * 400 Bad Request: Ha a kérés formátuma helytelen.
      * 401 Unauthorized: Ha a hitelesítési token érvénytelen vagy lejárt.
      * 500 Internal Server Error: Szerver oldali hiba esetén.

(Forrás: https://developer.mozilla.org/)