M3102 : Services Réseaux

Bruno BEAUFILS

2021/2022

1. Web

- Fonctionnement
- Logiciels serveurs
- Apache
- Logiciels clients

1. Web

HTTP, HTML, URI/URL

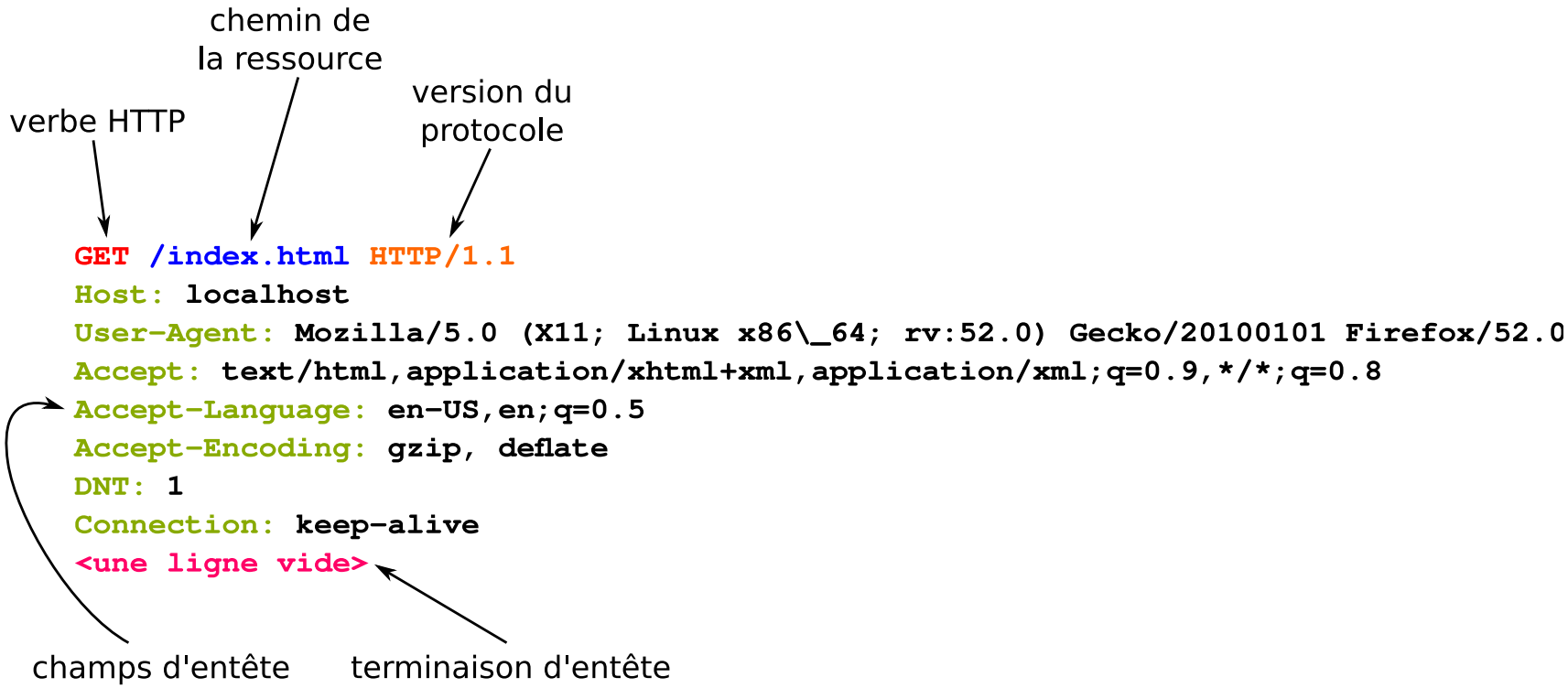
Web

Méthode universelle d'échange de données

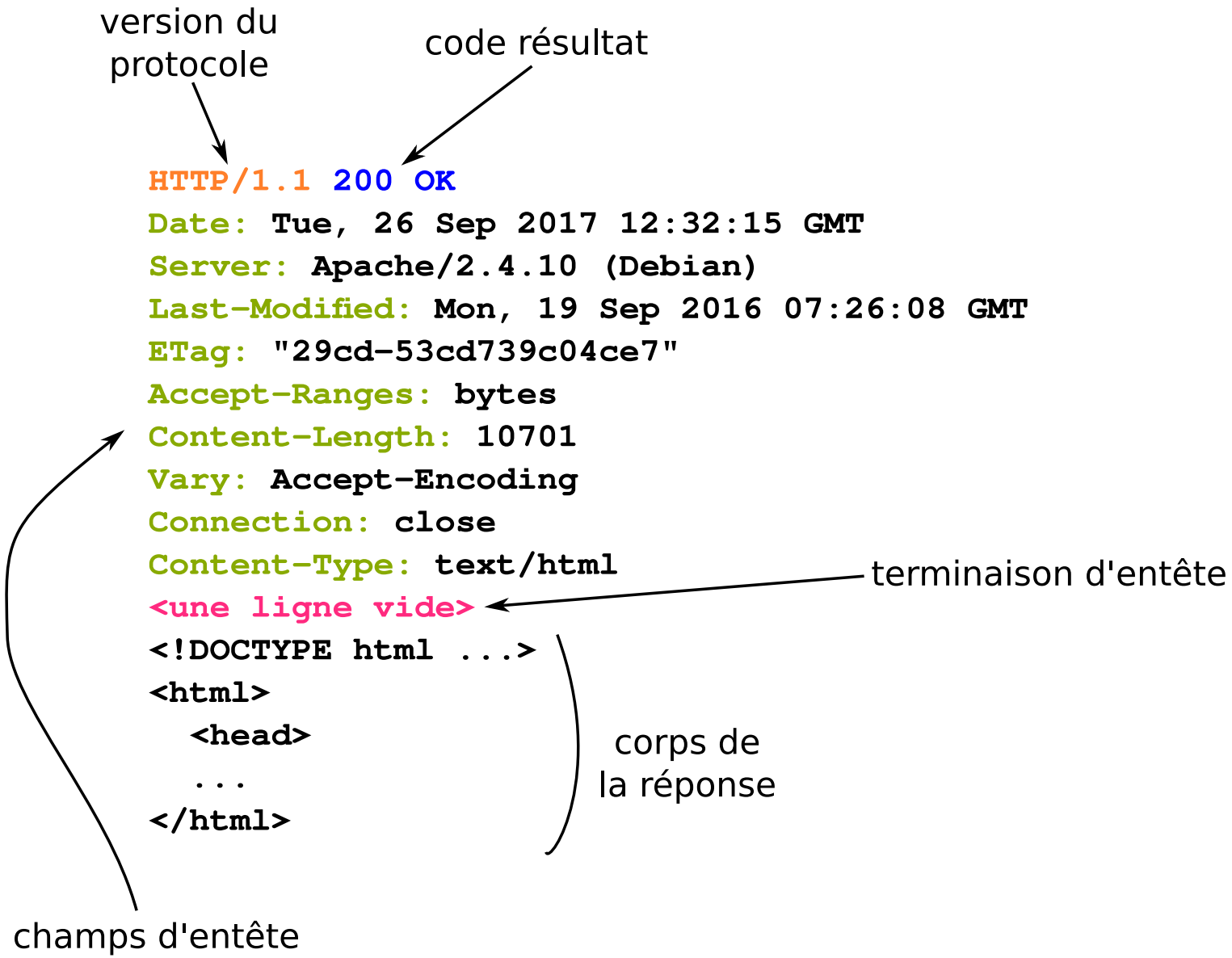
HTTP = *HyperText Transfer Protocol*

- un protocole de manipulation de ressources web
  - inventé au début des années 1990 par Tim BERNERS-LEE au CERN
  - initialement
    - les ressources sont des documents hypermédia (liens vers d'autres documents)
    - le protocole permettait uniquement de récupérer une ressource
- protocole de communication applicatif **déconnecté** *TCP, port 80*
  - utilise une architecture client/serveur
    1. établissement connexion
    2. échange requête
    3. échange réponse
    4. fermeture connexion
  - RFC 7230 et suivantes

Requête HTTP



Réponse HTTP



Code de statut HTTP

RFC 7231

- `1xx` Information (traitement en cours)
- `2xx` Réussite (traitement réussi)
- `3xx` Redirection (action supplémentaire nécessaire)
- `4xx` Erreur du client (requête incorrecte)
- `5xx` Erreur du serveur (requête impossible à réalisée)a

Cookies HTTP

- données…
  - **envoyées par le serveur** au client dans les entêtes d'une réponse…

    `Set-Cookie: nom=nouvelle_valeur; expires=date; path=/; domain=.exemple.org`
  - … et **retournées par le client** au serveur à **chaque** requête suivante
  - équivalent d'un fichier texte stocké par le client pour le serveur
- pour échapper à l'aspect **déconnecté** et **sans-état** (*stateless*) de HTTP
  1. le client envoi des informations au serveur
  2. le serveur les synthétise (sérialisation/hachage/etc.)
  3. le serveur les mets dans un cookie chez le client
  4. lors de la connexion suivante le serveur demande le cookie
- utilisation
  - authentification
  - session (un état de la communication entre serveur et client)
    - identification du client
    - panier dans le commerce électronique
  - stockage information spécifique (préférences sérialisées, etc.)
  - pistage

Cookies (exemple)

1. Requête du navigateur vers le serveur

```
GET /index.html HTTP/1.1
Host: www.facebook.com
```

1. Réponse du serveur au navigateur

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name=value
```

1. Toutes les requêtes suivantes faites au même serveur seront de la forme

```
GET /toto.html HTTP/1.1
Host: www.facebook.com
Cookie: name=value
Accept: */*
```

Généralement `Set-Cookie` ajouté par gestionnaire dynamique (CGI, PHP, etc.)

Cookies (exemple)

1. Requête du navigateur vers le serveur

```
GET /index.html HTTP/1.1
Host: www.facebook.com
```

1. Réponse du serveur au navigateur

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name=value
```

1. Toutes les requêtes suivantes faites au même serveur seront de la forme

```
GET /toto.html HTTP/1.1
Host: www.facebook.com
Cookie: name=value
Accept: */*
```

Généralement `Set-Cookie` ajouté par gestionnaire dynamique (CGI, PHP, etc.)

Cookies (exemple)

1. Requête du navigateur vers le serveur

```
GET /index.html HTTP/1.1
Host: www.facebook.com
```

1. Réponse du serveur au navigateur

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name=value
```

1. Toutes les requêtes suivantes faites au même serveur seront de la forme

```
GET /toto.html HTTP/1.1
Host: www.facebook.com
Cookie: name=value
Accept: */*
```

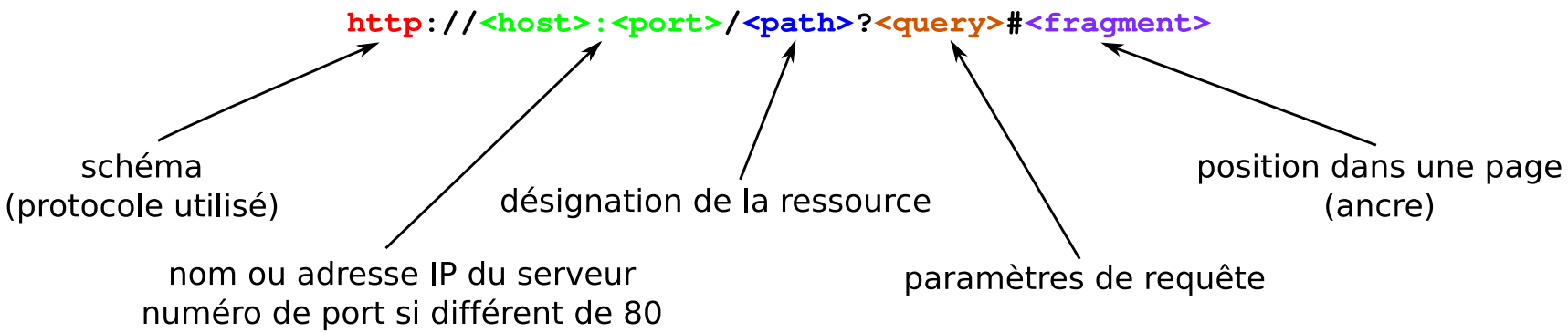
Généralement `Set-Cookie` ajouté par gestionnaire dynamique (CGI, PHP, etc.)

HTML = HyperText Markup Language

- langage de **description** de documents hypermédia à balises
- https://www.w3.org/TR/html53/

URI / URL

- être capable d'identifier des ressources **web** (pas locales au serveur)
  - pour les ancres, les images, etc.
- RFC 3986
  - URI (*Uniform Resource Identifier*) identification textuelle de ressource abstraite ou concrète
  - URL (*Uniform Resource Locator*) une URI permettant la localisation (accès) réseau



1. Web

Serveurs web

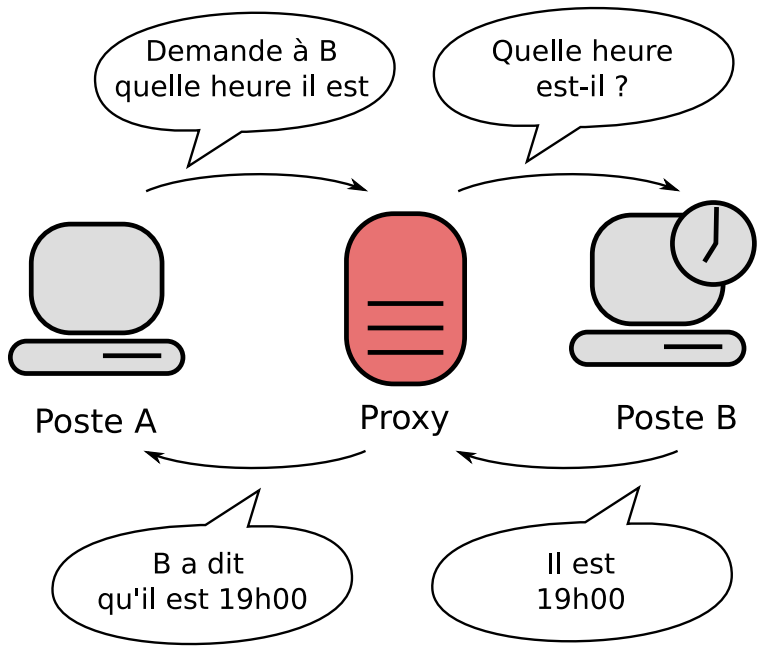
- Objectifs : répondre à des requêtes HTTP
- Le serveur a accès à tout un tas d'information
  - éléments de la **connexion** (IP origine, etc.)
  - éléments de la **requête** (Resource, Champs d'entêtes, etc.)
  - **historique** de connexion (journaux)
- Répond à plusieurs noms différents
  - notion de **Virtual Host**
- Exemples
  - Apache, lighthttpd, thttpd, nginx, IIS

Cas d'utilisation

- Utilisation **statique**
  - un appel identique fait plusieurs fois donne toujours la même réponse
  - projection d'un espace de nommage sur un système de fichier
    - renvoyer des fichiers
    - resources de la requête = chemin relatif du fichier à renvoyer
  - utilisation historique (échange de documents hypertextes)
- Utilisation **dynamique**
  - exécute une fonction/commande pour construire la réponse
  - interface CGI (RFC 3875) : transmission requête/paramètres au programme
  - langage adapté (perl, php, etc.)
  - mode d'exécution différent

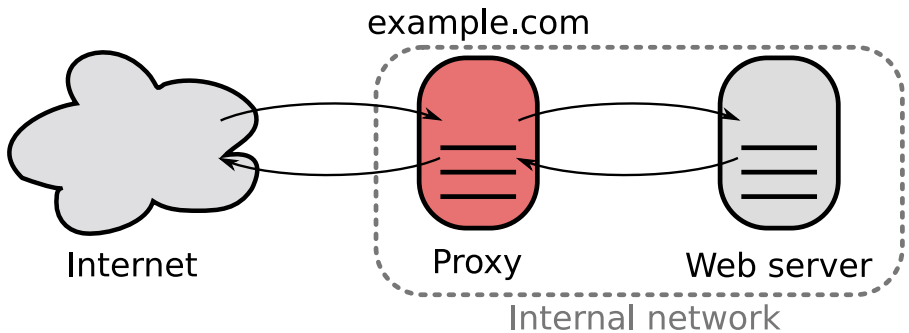
Cas d'utilisation

- Intermédiaire
  - reçoit la requête et la transmets à un autre serveur
  - **Proxy** : pour pouvoir accéder à l'Internet depuis un réseau privé
    - Utilisation : filtrage, cache
    - Exemple : Squid (`http://cache.univ-lille.fr:3128`)
  - **Reverse Proxy** : pour pouvoir accéder à un réseau privé depuis l'Internet
    - Utilisation : répartition de charge, protection, cache
    - Exemple : apache



Cas d'utilisation

- Intermédiaire
  - reçoit la requête et la transmets à un autre serveur
  - **Proxy** : pour pouvoir accéder à l'Internet depuis un réseau privé
    - Utilisation : filtrage, cache
    - Exemple : Squid (`http://cache.univ-lille.fr:3128`)
  - **Reverse Proxy** : pour pouvoir accéder à un réseau privé depuis l'Internet
    - Utilisation : répartition de charge, protection, cache
    - Exemple : apache



Fonctionnalités

En vrac :

- répertoire par utilisateur avec adresse particulière
- génération automatique d'index de répertoire
- règle de ré-écriture des ressources demandés
- redirection
- etc.

Configuration

- `/etc/apache2/apache2.conf`
  - contient des **directives** de configuration
  - une par ligne
  - commentaire introduit par #

Configuration

- `/etc/apache2/apache2.conf`
  - contient des **directives** de configuration
  - une par ligne
  - commentaire introduit par #

Directives simples

- paramétrage d'une fonctionnalité
- exemples

```
DocumentRoot /var/www/html
Include "ports.conf"
IncludeOptional conf-enabled/*.conf
```

Directives simples

- paramétrage d'une fonctionnalité
- exemples

```
DocumentRoot /var/www/html
Include "ports.conf"
IncludeOptional conf-enabled/*.conf
```

Directives structurées

- bloc de syntaxe *à la* XML
- **grouper des directives pour en limiter la portée**
- exemples

```
<Directory /var/www/>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
</Directory>
```

Limitations

- limitations à **un sous-ensemble des données servies**
  - `<Directory>` en fonction du dossier sur le système de fichier
  - `<Files>` en fonction des fichiers
- limitation à **un sous-ensemble des données demandées**
  - `<Location>` en fonction de l'URL
  - `<VirtualHost>` en fonction du nom/IP demandé

Configuration sous Debian

- déportée dans plusieurs fichiers...

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

- mécanisme de répertoire de préparation `-available` vs `-enabled`
  - activation d'un fragment de configuration par des liens symboliques
  - commandes
    - `a2en*` = **activation**
    - `a2dis*` = **désactivation**

Clients (Navigateurs)

- **interface entre l'utilisateur et le serveur web**
  - le plus simple c'est `nc` ou `telnet` (un seul rôle : la connexion)
- Rôles de base
  1. interpréter/préparer une URL pour **faire une requête** à un serveur
  2. interpréter et **présenter la réponse** à l'utilisateur
     - généralement afficher une page HTML
     - faire un rendu (graphique/sonore/braille)
     - séparation fond/forme (HTML/CSS)
- Rôles courants
  - préparer la requête
    - saisie de formulaire
    - raccourci (moteur de recherche)
    - authentification HTTP
  - aider l'utilisateur
    - historique
    - bookmarks
    - paramétrage de la connexion (proxy)
    - stockage local d'informations (mot de passe, etc.)
- Exemple :
  - Firefox (Gecko), Chrom(e/ium)/Edge/Safari/Opera (Webkit), IE (Trident)
  - w3m, lynx, links, **curl**, **wget**

Interface d'application

- Langage interprété sur le client (JavaScript)
  - langage de programmation complet
  - fonction de connexion directe au serveur (`XMLHttpRequest`)
    - interfaces directes au moteur de rendu
    - nombreuses librairies (stockage/multimédia/etc.)
- Gestion des cookies
- Plugins/Addons
- Les applications mobiles sont souvent de *simples* applications web
  - merci HTML5/CSS3/JS

Travail caché

- Lors de son rendu d'une page HTML le serveur peut faire **beaucoup** de requêtes sans en informer **explicitement** l'utilisateur
  Récupération nécessaire au rendu ou demandé par la présentation de la réponse
  - image
  - script
  - styles
- Chaque requête est en HTTP et peut-être accompagné d'un envoi de cookie
- Cookies tierce-partie