Architecture(s) et application(s) Web

Télécom SudParis

05/09/2024

CSC4101 - Concepts fondamentaux, architecture appli Web 3 couches

Table des matières

Objectifs de cette séquence

Cette séquence de cours magistral présente les grands éléments du contexte du cours :

1. le *World Wide Web*
2. le dialogue entre clients et serveurs Web
3. les éléments principaux de l'architecture d'application à 3 couches

1 Le World Wide Web

On va balayer rapidement les concepts informatiques principaux qui caractérisent le Web.
Ils seront revus et détaillés dans les prochaines séquences.

1.1 *Web* = Toile

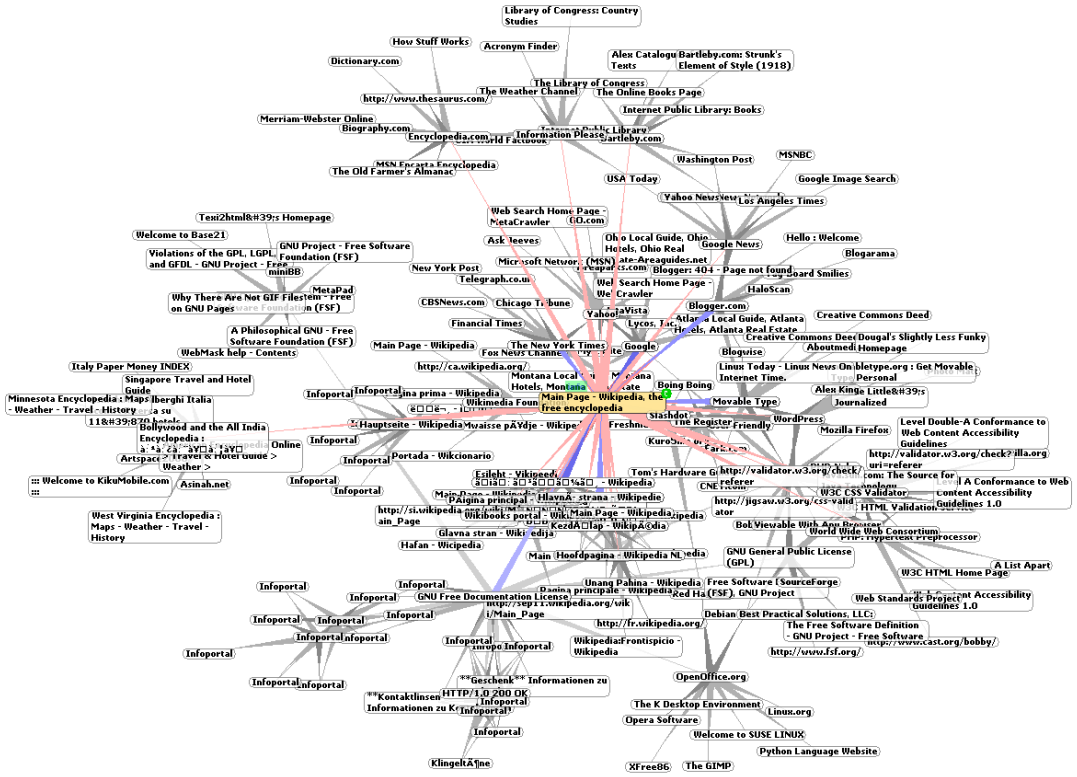
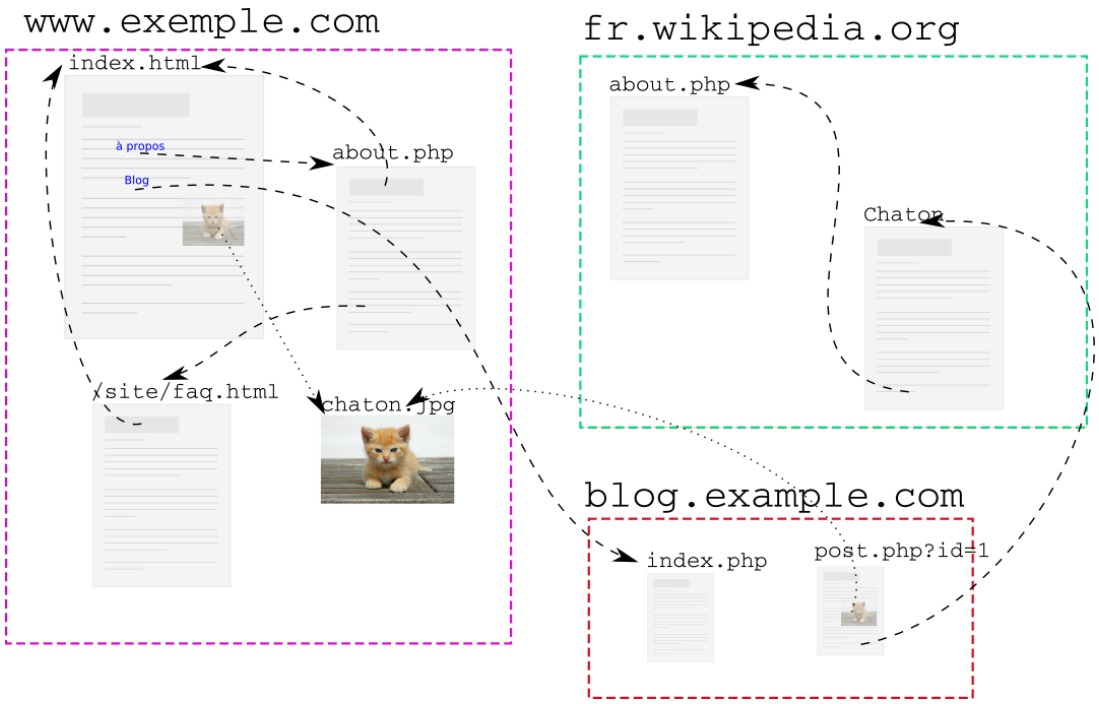


Figure 1 – « WorldWideWeb Around Wikipedia – Wikipedia as part of the world wide web »

Source : Chris 73 / Wikimedia Commons

Le *Web* signifie littéralement une toile (d'araignée), en anglais.
Chaque lien pointant d'un site à un autre dessigne une toile mondiale (*world wide web* : `www`).
Ce diagramme illustre le graphe de sites accessibles depus une page de Wikipedia.

1.2 Graphe de ressources liées décentralisé



Le Web est par essence **décentralisé**.
Les liens sont entre différents documents, du même site ou de sites différents.
Certains liens sont navigables, d'autres correspondent à l'inclusion de ressources externes (images).
Note : les liens sont uni-directionnels, du point de vue des serveurs : les documents ne « savent » pas qui leur pointe dessus. Aucune coordination : pas de permission à demander avant de tisser des liens, mais pas de garantie de disponibilité des ressources distantes.

1.2.1 Ressources

- Documents (statiques)
- Consultation dynamique :
  - ressources issues d'applications (dynamiques)
  - représentation de ces ressources dans les navigateurs Web
- Éléments « virtuels » décrivants des « faits » (Web sémantique/des données)

Pour être précis, on parle de **ressources**, sur le Web. Cette notion est très générique. Elle englobe des documents ou des pages de site, sans exclure des modèles plus avancés d'un Web des données par exemple.
Pour une discussion du terme, d'un point de vue historique, voir par exemple A Short History of « Resource » in web architecture rédigé en 2009 par Tim Berners-Lee.

1.2.2 Ressources liées

- Identification des ressources
- Localisation des ressources
- Agréger des ressources et leurs donner des propriétés pour créer de la connaissance

L'intelligence du Web réside justement dans les liens établis entre les ressources.
Cela permet de présenter des documents riches, mais aussi de naviguer vers d'autres sites.

1.2.3 Décentralisé

— Lier entre-elles des ressources localisées physiquement n'importe où sur Internet
— Confiance, provenance ?

C'est le travail des outils clients, comme le navigateur Web, de construire de l'intelligence en rendant accessible du contenu distribué, mis en ligne de façon pas nécessairement coordonnée.
Rien ne garantit alors que le contenu est fiable, pérenne dans le temps, ou qu'on puisse y faire confiance.
Contrairement au cas d'applications développées dans des environnements contraints et monolithiques, les applications déployées sur le Web doivent ainsi prendre en compte ces contraintes (on y reviendra).

1.3 Fonctionnement simplifié de l'accès aux pages d'une applica

Cette section décrit les grands principes architecturaux qui sous-tendent le fait d'utiliser le Web et HTTP pour faire fonctionner des applications.

1.3.1 Structure générale

1. Client + Serveur

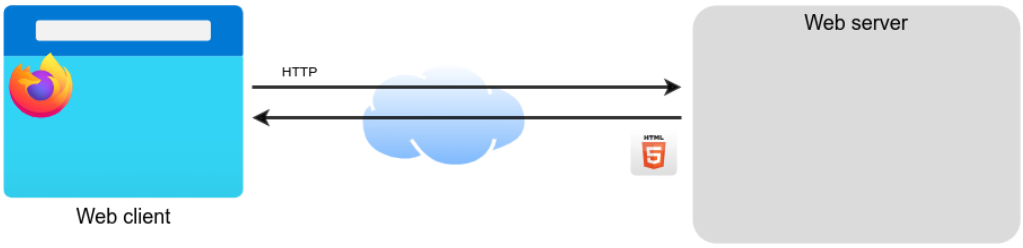
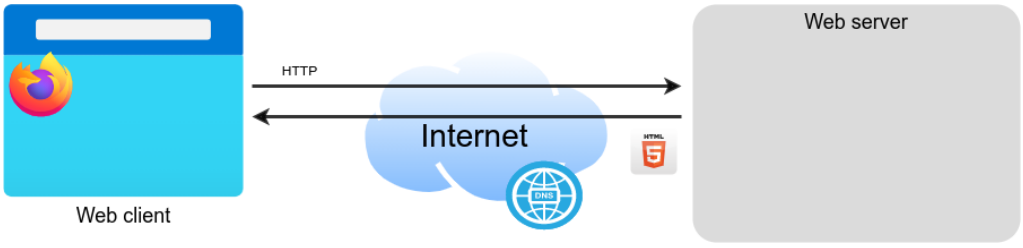


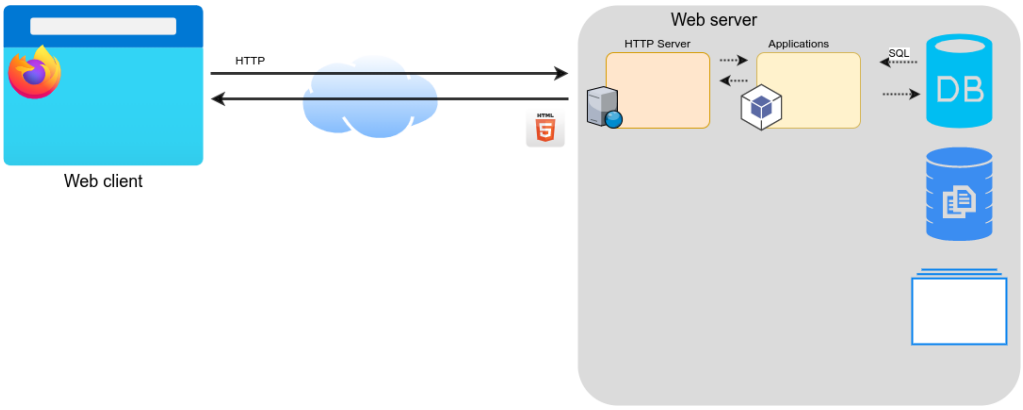
Illustration du navigateur (*Web browser*) Firefox, comme client Web, qui communique à travers l'Internet avec un serveur Web.
Il transmet des requêtes grâce au protocole HTTP, et récupère des documents HTML dans le contenu des réponses HTTP.
Regardons maintenant ce dont on a besoin au niveau d'Internet.

2. Internet sous-jacent



Le navigateur s'appuie notamment sur le système des noms de domaines (DNS) pour effectuer les requêtes auprès du serveur.
Ainsi, n'importe quel possesseur d'un nom de domaine peut mettre en ligne un serveur Web, qui sera immédiatement disponible pour que des clients s'y connectent, sans autre forme d'autorisation.
Aucune autorité centrale du Web n'entre en jeu pour délivrer une quelconque autorisation.
Attention cependant aux contraintes légagles en vigueur (LCEN, en France, par exemple).
Examinons maintenant ce qu'on trouve du côté du serveur

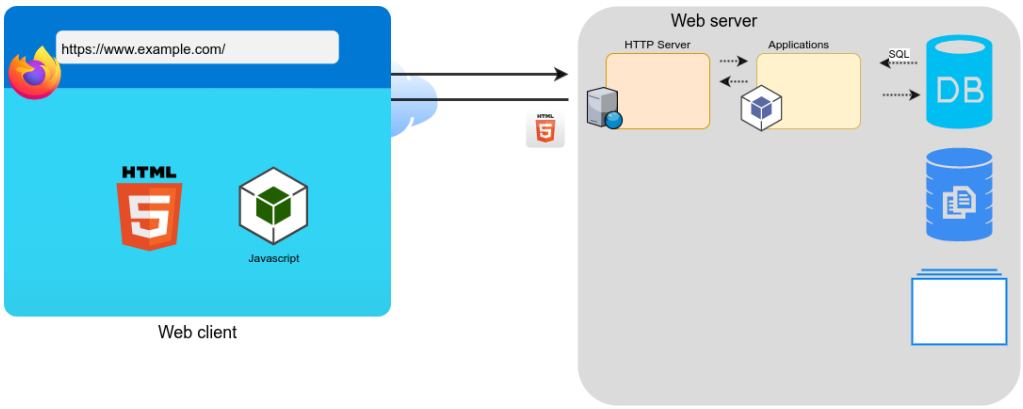
3. Côté serveur



Dans un premier temps, on distingue 3 ensembles d'applications mises en œuvre côté serveur :
- le serveur HTTP « frontal » qui écoute les requêtes transmises par les clients. Il en gère en direct un certain nombre (contenu « statique »)
- les applications, à qui le serveur HTTP passe la main, quand il s'agit de contenu dynamique, qui nécessite l'exécution d'une application pour obtenir la réponse attendue.
- différents composants additionnels, utilisés par les applications. En général, on peut trouver :
  - une base de données (*DB*), souvent relationnelle, dans laquelle l'application peut effectuer des requêtes avec le langage SQL
  - du stockage de fichiers, pour permettre la consultation de documents transmis par l'application, mais aussi pour gérer des stockages internes au serveur Web (sessions, etc.)
  - tout autre élément d'interfaçage avec le Système d'Informations (SI) d'entreprise

Revenons maintenant sur ce qu'on trouve dans le navigateur.

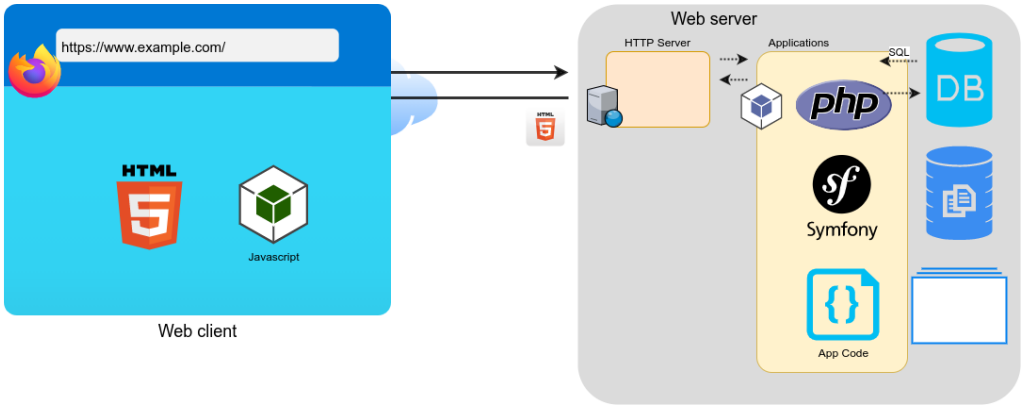
4. Dans le navigateur



Dans le navigateur, on distingue pour l'instant deux grands composants :
- le moteur de rendu HTML, qui va afficher les documents HTML transmis par le serveur sous forme consultable par l'utilisateur.
- une « machine virtuelle » Javascript capable de faire tourner des programmes dans le navigateur, pour disposer de plus d'intelligence du côté client.

Enfin, examinons le cœur de l'intelligence applicative, côté serveur

5. Code des applications



Finalement, concluons ce panorama rapide en regardant ce qui peut constituer une application qui fonctionne derrière le serveur HTTP frontal, côté serveur :
- le code spécifique de l'application est installé, et est exécuté lorsque certaines portions du site Web seront consultées, sur demande du frontal HTTP. Ce code est typiquement, dans ce cours, du code PHP, interprété
- on doit donc disposer d'un interpréteur du langage PHP
- mais il faut aussi disposer de toutes les bibliothèques (en général via un *framework*) nécessaires à l'exécution du code, comme par exemple avec Symfony, dans ce cours.

Le code s'appuiera sur le *framework* pour générer des pages HTML, en réponse aux requêtes, pour les rendre au frontal HTTP, afin que celui-ci les renvoie vers le client.

1.4 Clients et serveurs HTTP

Le cœur de l'architecture technique du Web est le protocole HTTP, qui repose sur un modèle client-serveur.

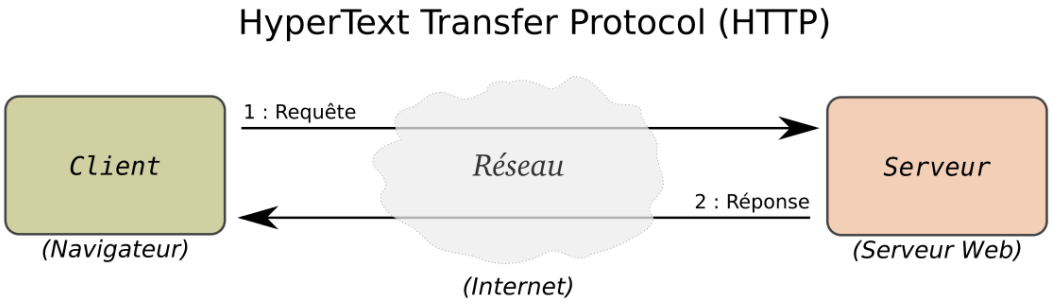
1.4.1 Architecture Client-Serveur

Protocole très simple :

1. requête
2. réponse

Cette architecture est très classique et n'est pas née avec le Web.
Ici, on doit plus précisément parler d'un serveur et de multiples clients qui s'y connectent. Sur le Web, et dans HTTP, les serveurs sont conçus pour communiquer systématiquement avec de multiples clients.
On rappelle qu'il est très fréquent qu'un même client parle avec différents serveurs pour des tâches très courantes, comme la consultation de pages Web dans un navigateur, qui nécessite de charger des contenus sur différents serveurs HTTP indépendants.

1.4.2 Client et serveur Web



En fait : 1 client - n serveurs (modèle distribué)

Le protocole de communication entre clients et serveurs est HTTP (*HyperText Transfer Protocol*).
Le client peut être tout type de programme (navigateur, robot, etc.).
Précision terminologique : dans les spécifications on parle de :
- Client appelé *user agent* dans les spécs
- Serveur appelé *origin server* dans les spécs

La notion d'*origin server* (serveur d'origine), permet de distinguer le serveur d'un éventuel *proxy* (serveurs mandataire) présent entre ce serveur et le client.
Les détails d'HTTP concernant les *proxies* ne seront pas abordés dans ce cours.
On étudiera le protocole HTTP plus en détails dans la prochaine séquence de cours magistral.

1.4.3 Dialogue entre client et serveur

- Communication en 3 étapes simples :
  1. Le **client** (navigateur) fait une **requête** d'accès à une **ressource** auprès d'un serveur Web selon le protocole **HTTP**
  2. Le **serveur** vérifie la demande, les autorisations et transmet éventuellement l'information demandée
  3. Le client interprète la **réponse** reçue (et l'affiche)
- On recommence pour des ressource complémentaires (images, …).

1.4.4 Concepts de base d'HTTP

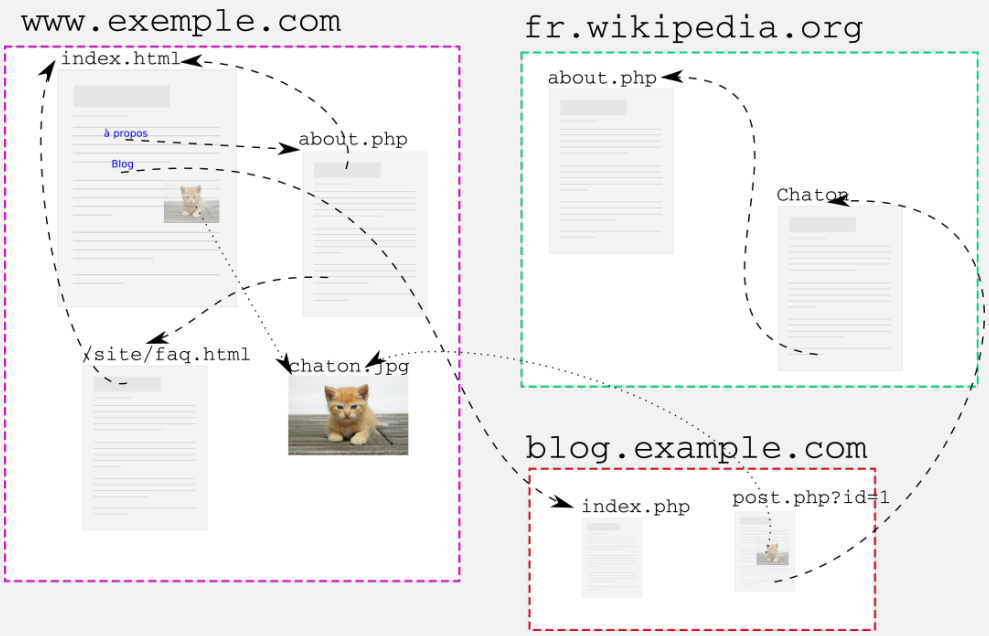
- **Interaction** avec des ressources hypertexte :
  - Quoi : **Action** : verbe (CRUD)
  - *Qui* : **Ressources** identifiées par des **URI** (données d'une application, …)
  - *Où* : **URL** pour **localiser** les représentations informatiques de ces ressources (documents hypertextes, images, etc.) sur des serveurs Internet
  - *Comment* : Requêtes protocole de communication **HTTP**

L'acronyme CRUD correspond aux opérations classiques qu'on peut effectuer sur des données : *Create*, *Retrieve*, *Update*, *Delete*.
On parle en fait plutôt d'URI que d'URL dans les spécifications, même si pour l'instant cette subtilité a peu d'importance.

1.4.5 Construction de la toile (*Web*)

— **Graphe** de ressources hypertexte/hypermedia **décentralisé**
— Les ressources référencent d'autres ressources (attribut `href` dans HTML)
— Les liens sont unidirectionnels
— Aucune garantie de disponibilité, cohérence
— Parcourir la toile :

1. trouver les liens,
2. accéder aux ressources liées
3. enrichir le modèle applicatif
4. *recommencer* …



Hyper-reference, *hyper-link*
Historiquement, **HTML** comme langage de mise en forme pour la visualisation des ressources de type « document hypertexte »
Le navigateur permet de suivre les liens dans le graphe : le graphe n'existe pas (à part dans un cache) tant qu'on n'a pas essayer de naviguer dessus.
Seul point « stable » : le système DNS, comme pour tout Internet.

- Composants de ce graphe :
  - 3 sites/serveurs Web : `www.example.com`, `blog.example.com`, `fr.wikipedia.org`
    - (peut-être sur 2 serveurs (2 adresses IP) : `example.com` et `fr.wikipedia.org` ?)
  - Documents/ressources :
    - `http://www.example.com/index.html` (HTML)
    - `http://www.example.com/site/faq.html` (HTML)
    - `http://www.example.com/about.php` (HTML)
    - `http://www.example.com/chaton.jpg` (image)
    - `http://blog.example.com/index.php` (HTML)
    - `http://blog.example.com/post.php?id=1` (HTML)
    - `http://fr.wikipedia.org/about.php` (HTML)
    - `http://fr.wikipedia.org/Chaton` (HTML)
  - Liens :
    - Inclusions images
      - Même site
      - Autre site
    - Hyper-liens navigables
      - Document du même site
      - Document sur un autre site
      - Fragment à l'intérieur d'un document

1.4.6 Localisation des ressources

- Objectif : nommer, localiser et accéder à l'information
- Solution : **URL (*Uniform Resource Locator*)** : identification universelle de ressource composée de 3 parties :
  1. le **protocole** (*comment*)
  2. l'identification du serveur Web, le **nom DNS** (*où*)
  3. l'**emplacement de la ressource** sur le serveur (*quoi*)

Plus tard, on raffine avec le concept d'URI : cf. RFC 3986 - Uniform Resource Identifier (URI) : Generic Syntax et standards associées.

1.5 Deploiement sur le Web

Allons-y, lançons-nous : pouvons-nous déployer nos sites Web ?

1.5.1 Interopérabilité ?

— Qui peut créer des clients Web ?
— Qui peut opérer des serveurs ?

Historiquement, le Web est conçu comme un bien commun, où tout le monde peut opérer des clients et serveurs, pour autant qu'on a accès à Internet, et qu'on respecte les standards élaborés par la communauté.
C'est encore vrai dans une certaine mesure, mais pour offrir des garanties de disponibilité, de sécurité, de performances, ça devient une affaire de professionnels, et certains prestataires deviennent incontournables.
Et pour que les différents clients et serveurs puissent se comprendre il faut assurer le respect des standards. Malheureusement, ce n'est pas toujours le cas, et les opérateurs ayant une part dominante sur le marché ne sont pas toujours des bons élèves.

1.5.2 Déploiement DIY

— Raspberry Pi
— Fibre
— Debian GNU/Linux
— Apache / Nginx
— …

1.5.3 Déploiement Cloud

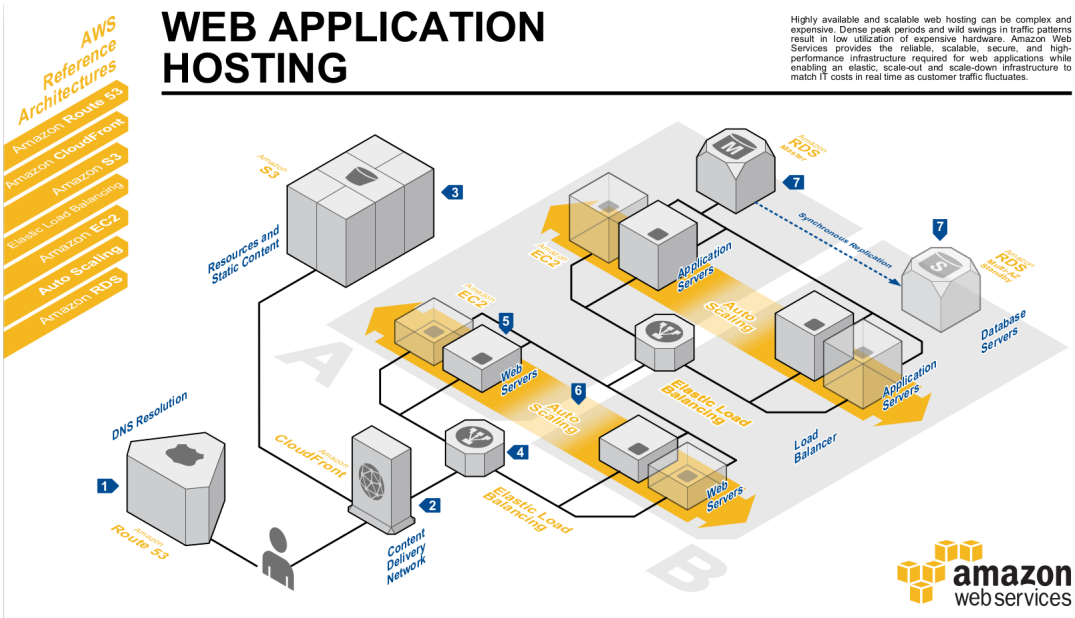


Figure 2 – Diagramme d'architecture d'*Amazon Web Services*

À titre d'information, l'hébergement et la conception des applications Web passe aujourd'hui par des plate-formes dédiées, sur le Cloud, comme celle d'Amazon illustrée ci-dessus.
Cependant, nous n'étudierons pas les propriétés de telles plate-formes dans le cadre de ce cours d'introduction. Nous observerons des plate-formes plutôt plus traditionnelles comme l'hébergement PHP sur un serveur Web « classique ».

2 Architecture 3 couches

Cette section présente les principes d'architecture des applications Web classiques, et notamment l'architecture dite 3 tiers.
Les applications Web sont ubiquitaires aujourd'hui, mais on n'a pas forcément toujours des idées très claires sur ce qui permet de faire fonctionner ces applications.
Cette séquence va nous permettre d'examiner les mécanismes et protocoles nécessaires au donctionnement d'une application Web.

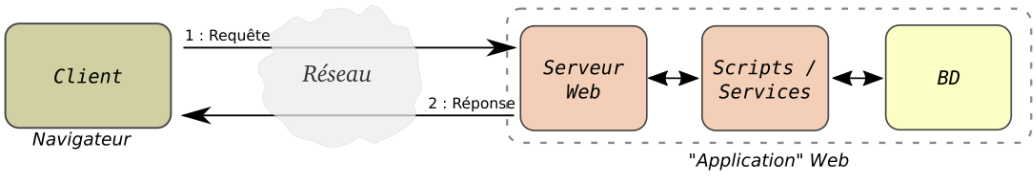
2.1 Architecture ?

- Structure générale d'un système informatique
  - matériel
  - **logiciel**
  - humain / responsabilités
- Conception
  - Normes ?
  - Standards ?
  - Bonnes pratiques ?

L'intitulé du cours mentionne ce mot « architecture ». Il est donc important de se pencher sur sa signification.
En informatique, comme dans d'autres domaines (bâtiment, …) on peut s'intéresser à des principes d'architecture qui permettent d'apprendre à construire des applications, de la « bonne façon ».
Nous n'étudierons pas en détail toutes les acceptions du terme, mais on verra par exemple, dans la suite du cours, un ensemble de bonnes pratiques consistant à s'appuyer sur un *framework* pour bénéficier d'un ensemble de bonnes pratiques, plutôt que d'avoir à réinventer la roue.

2.2 À l'origine, applications BD + Web

Les applications produisent des pages Web **dynamiques**, à partir de données issues de bases de données.



On décore le SGBD avec des pages qui génèrent des requêtes.

BD : Bases de Données. Typiquement via l'interrogation d'un SGBDR
Note *: on ne revient pas sur les technologies de bases de données dans ce cours, que l'on considère comme acquises (programme première année).*
Les programmes qui produisent les pages peuvent aussi utiliser d'autres sources de données que les bases de données

2.2.1 Succès historique des applications BD + Web

Des **applications** sont réalisables facilement, dès que des programmes génèrent des pages HTML en fonction des requêtes HTTP du client
Pour « M / Mme Michu », quand même plus convivial…
… que SQL !
Avantages
- Base de données gère l'intégrité des données (transactions, accès multiples, intégrité référentielle, …)
- Tient la charge (tant qu'on arrive à dupliquer l'exécution des scripts)
- Large gamme de choix du langage de programmation

On ne raffine pas trop le modèle de couches avec l'adaptateur de données qui est parfois introduit, pour ne pas trop complexifier
Les programmes sont appelés *scripts* sur le diagramme, car les langages de scripts (PHP, Perl, Python, etc.) furent très populaire dans l'essor de cette technologie de pages Web dynamiques. Des programmes dans un langage compilé sont aussi possibles (Java par ex.) sans que cela change le modèle.
Différentes techniques d'invocation des programmes par les serveurs Webs (qui étaient initialement dédiés aux pages statiques) ont vu le jour, comme CGI.
Un souci général est les performances pour des consultations simultanées, sachant que tout n'est pas complètement dynamique dans une page Web, mais que si tout est regénéré à chaque consultation, le serveur peut vite s'écrouler.
L'efficacité du mécanisme de déclencement de l'exécution du bon programme, et de récupération du contenu des pages généré est critique. Afin d'optimiser les performances, différentes générations de technologies ont émergé, qu'on verra plus loin.

2.3 Architecture 3 *tiers*

Architecture *logicielle* en couches (*tiers*) :
- couche de **présentation** ;
- couche de **traitement** ;
- couche d'**accès aux données**.

Simplification de l'approche générale d'une architecture à plusieurs niveaux (*multi tier*)

L'objectif de cette décomposition en couches est de mieux comprendre la logique des interactions entre différents composants logiciels mis en œuvre pour faire fonctionner une application. Il s'agit ici d'une architecture logicielle/système, indépendant des fonctionnalités fournies par telle ou telle application (on parlerait d'architecture fonctionnelle).
On parle aussi d'**architecture à trois niveaux** ou **architecture à trois couches**.
Le mot *tiers* est une traduction approximative de l'anglais *tier* signifiant étage ou niveau.

2.3.1 Variante ligne de commande

Application Symfony `Todo` (cf. TP), en ligne de commande :

présentation affichage sortie standard

traitement `App\Command\ListTodosCommand`

accès aux données Doctrine + SQLite (SQL)

Une application démarre pour une exécution pour un utilisateur unique, dans le contexte d'un shell, en ligne de commande dans un terminal.
L'application est un processus de l'interpréteur PHP qui démarre le programme `bin/console` du *framework* Symfony.
L'application est « monolithique », fonctionnant dans la mémoire d'un seul système d'exploitation, en local, en un seul processus.
Les 3 couches sont des couches logiques aidant à comprendre l'architecture fonctionnelle de l'application.

2.3.2 Variante « classique » sur le Web

Classique : éléments logiciels principalement côté serveur (y compris présentation)

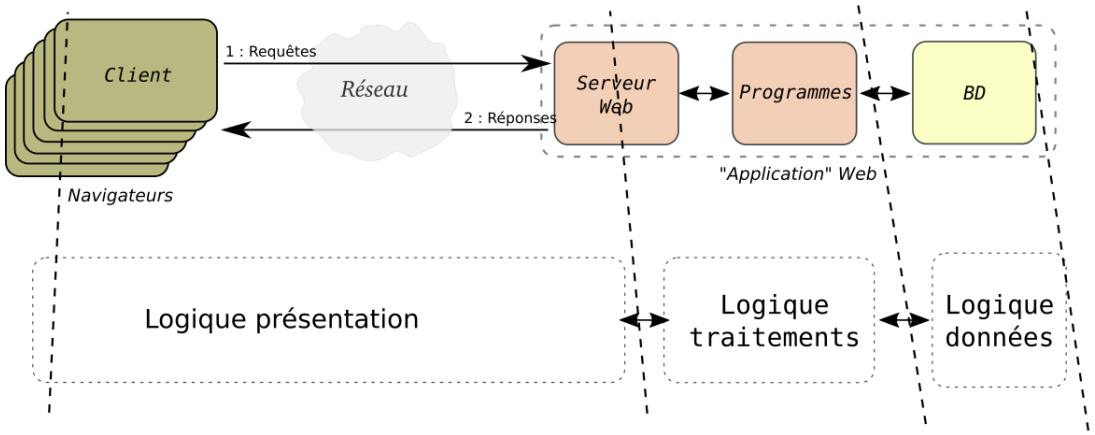


Figure 3 – Architecture 3 couches

L'application fonctionne sur un serveur accessible depuis le réseau via le protocole HTTP.
Un navigateur Web permet de s'y connecter et de consulter des pages Web présentant le résultat de l'exécution d'un programme fonctionnant dans la mémoire du serveur.
Attention, ici, plusieurs clients Web peuvent accéder simultanément à la même application fonctionnant sur le serveur Web.
La qualification de « classique » correspond à un positionnement des éléments logiciels principalement côté serveur (y compris pour une bonne part de la couche de présentation)

2.3.3 3 couches des applications Web

- **Présentation** : *pages HTML* :
  - *serveur* les construit (classique)
  - *client* (navigateur) les charge et affiche
- **Traitements** : programmes :
  - serveur Web (dialogue HTTP, invocation des applications)
  - serveur d'applications (exécute des programmes PHP, par ex.)
- **Accès aux données** : SGBD

Les multiples clients Web (de chacun des utilisateurs) communiquent via HTTP avec un serveur Web unique.
Le navigateur Web qui fait le rendu des pages HTML participe à l'application (présentation).
Les clients Web ne sont pas seulement les navigateurs des utilisateurs, mais peuvent aussi être des programmes se connectant à des APIs via HTTP.
Dans la variante classique, les traitements se font selon les programmes écrits par les développeurs (par exemple en PHP) fonctionnent sur / derrière ce serveur Web. Dans d'autres architectures plus récentes, les traitements peuvent s'exécuter de façon très importante sur le client.
Ils peuvent s'exécuter pour effectuer des traitements grâce aux services fournis par des serveurs d'applications sous-jacents (journeaux, envoi de mails, etc.).
Dans cette déclinaison en 3 couches, chaque couche est plus ou moins indépendante des autres. On peut par exemple imaginer que d'autres clients que les clients Web peuvent accéder aux mêmes serveurs d'applications pour interagir avec les programmes (via des protocoles autres que HTTP).
De même, les données stockées dans la base de données peuvent être manipulées par d'autres applications que ces applications Web, ou directement par des utilisateurs via les interfaces du SGBD (cf. CSC3601).

3 Développer aujourd'hui une appli « classique » ?

Dans ce cours, on concevra l'application de façon traditionnelle, en 3 couches.
On utilisera le cadriciel (*framework*) **Symfony** pour son approche **orienté objet**.
Dans le cours, on va se concentrer principalement sur l'apprentissage de la conception des couches :

1. présentation (*frontend*)
2. traitements et accès aux données (*backend*)

L'accès au données est considéré comme déjà acquis (SGBDR, SQL).

3.1 Architectures modernes, pédagogie...

On pourrait apprendre d'autres architectures plus modernes laissant une plus grande place aux composants sur le client (Javascript, serverless, etc.). Apprentissages complexes vue le temps imparti.
Objectif pédagogie ! Faciliter les apprentissages, donc architecture plus classique facilite la tâche.

4 Web comme plate-forme

Les technos du Web ne concernent plus seulement les « sites » vus dans un navigateur sur un PC
Importance du côté client :

- **Navigateur**
  - Moteur de rendu HTML
  - Machine virtuelle Javascript
- **Applications natives** programmées HTML5 + CSS + **Javascript** sur mobile
- Client HTTP + (micro) services REST

Native apps vs *Web apps*

L'architecture d'applications présentée dans le cours est très classique et repose sur un modèle un peu ancien, donc éprouvé.
De nouvelles architectures d'applications basées sur les technologies du Web ont vu le jour, mais qui reprennent des concepts similaire. Nous n'aurons pas le temps de les étudier en détails, mais elles reposent sur des variantes des concepts de base que nous aurons étudiés.
Ce que vous aurez appris vous servira (si vous continuez dans le développement d'applications), même avec d'autres architectures apparues plus récemment.

Take away

- Structure de la toile
  - Ressources
  - Ressources liées
  - Décentralisé
- Protocole client-serveur, HTTP, URL
- Applications générant des pages Web
- Modèle de couches (3 et +)
  - présentation
  - traitement
  - accès aux données

Aller plus loin

Le lecteur intéressé pourra consulter le document Web Architecture from 50,000 feet rédigé par Tim Berners Lee il y a 20 ans. Il date un peu pour certains aspects, mais l'essentiel est toujours applicable, étonamment.

Postface

Crédits illustrations et vidéos

— Illustration plate-forme Web Amazon AWS : http://media.amazonwebservices.com/architecturecenter/AWS_ac_ra_web_01.pdf
— Diagramme « Perdu sur le Web » : #Geekscottes par *Nojhan* https://botsin.space/@geekscottes/101748180337263915
— Illustration « chaton » : memegenerator.net
— « WorldWideWeb Around Wikipedia – Wikipedia as part of the world wide web » Chris 73 / Wikimedia Commons GFDL 1.3 or CC BY-SA 3.0

Figures interactives

On utilise un export HTML d'une illustration réalisés avec https://www.diagrams.net/ (moteur OpenSource de draw.io).

Annexes

Structure des URLs

— URL type :

— Composantes :

1. protocole : en majorité `http` ou `https`
2. adresse / nom du serveur : `www.monsite.fr`
3. chemin d'accès à la ressource / document : `/projet/doc.html` (ne garantit pas que le résultat est un document HTML)

URLs plus détaillées

Exemple d'URL plus complexe :

1. protocole : `http` ou `https` (mais aussi `ftp`, `file`, `mailto`, `gopher`, `news`,…)
2. autorité : nom du serveur : `www.monsite.fr` + port : `8000`
3. chemin d'accès à la ressource : `/projet/doc`
4. requête : deux paramètres : `id` et `f`
5. fragment : `num42` à l'intérieur du document

Il existe aussi un sur-ensemble des URLs, les URI (*Uniform Resource Identifier*). On se préoccupe plus d'identifier une ressource que de savoir comment y accéder. En pratique, cette subtilité ne change pas grand chose dans le cadre de ce cours.
Pour plus de détails, voir la spécification générale des URIs (RFC 3986).

URLs absolues ou relatives

— Les URL permettent d'établir des **liens** entre documents
  — sur le même serveur
  — entre différents serveurs
— Liens sur le même serveur :
  — chemin absolu / chemin relatif
  — analogie avec chemin de fichier Unix : `.`, `..`, `/`
  — lien intra-document : fragments/ancres : `doc.html#conclusion`
  — convention : lien dans l'espace d'un utilisateur : `~taconet/menu.html`

Exemples

Document source	Ressource liée	Emplacement réel
`http://w.e.c/index.html`	`about.php`	`http://w.e.c/about.php`
`http://w.e.c/about.php`	`/`	`http://w.e.c/` (contenu de `index.html`)
`http://w.e.c/site/faq.html`	`../index.html`	`http://w.e.c/index.html`
`http://w.e.c/index.html`	`./chaton.jpg`	`http://w.e.c/chaton.jpg`
`http://w.e.c/index.html`	`http://b.e.c/`	`http://b.e.c/` (contenu de `index.php`)
`http://w.e.c/about.php`	`/site/faq.html`	`http://w.e.c/site/faq.html`
`http://b.e.c/post.php?id=1`	`http://w.e.c/chaton.jpg`	`http://w.e.c/chaton.jpg`
`http://b.e.c/post.php?id=1`	`http://f.w.o/Chaton`	`http://f.w.o/Chaton`
`http://f.w.o/Chaton`	`/about.php`	`http://f.w.o/about.php`
`http://f.w.o/Chaton`	`/about.php#terms`	`http://f.w.o/about.php` (`<div id`« terms »>)

Copyright