Chapitre 1: Introduction à PHP

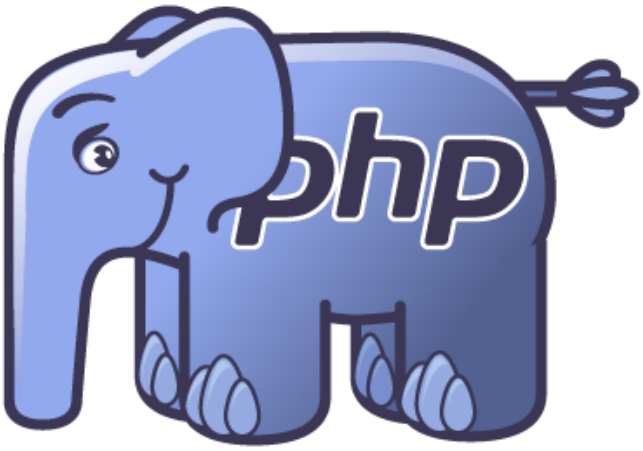
Prérequis

Pour suivre cette formation sur PHP il faudra au préalable avoir des **bases** sur **HTML** & **CSS** hormis cela la formation part du principe que vous n'avez aucune connaissance en **PHP**

▼ Qu'est-ce que PHP?

PHP, qui signifie **PHP Hypertext Preprocessor**, est un **langage de scripts** conçu pour le développement d'**applications Web**.

Sa mascotte est un éléphant, car les lettres **php** ressemblent à un … **éléphant**



C'est un **langage très accessible** qui permet de commencer à développer des applications **simples** en **quelques minutes**.

Il est généralement perçu comme un langage pour les **débutants** en programmation Web car il est **bien plus simple que les autres**.

Voici un exemple basique :

```
<!DOCTYPE html>
 <html>
  <head>
  <title>Hello PHP</title>
  </head>
  <body>

  <?php
      echo "Hello PHP!";
  ?>

  </body>
</html>
```

La balise `<!DOCTYPE html>` indique que vous utilisez la version HTML5.

Le code PHP est inséré entre les balises `<?php` et `?>`. Lorsque le serveur web traite cette page, il exécute le code PHP et affiche "Hello PHP!" dans le corps de la page.

La sortie du code PHP est affichée à l'intérieur de la balise `<body>`, de sorte que le texte "Hello PHP!" est visible lorsque vous ouvrez cette page dans un navigateur.

Si vous chargez ce fichier dans un serveur web avec **PHP activé**, vous verrez le texte "Hello PHP!" affiché dans le navigateur. C'est un exemple simple pour montrer comment PHP peut être intégré dans une page web. Vous pouvez ensuite construire des applications web plus complexes en utilisant PHP pour gérer les données et la logique de votre site.

L'histoire de PHP

Pour comprendre PHP il faut comprendre comment fonctionnait le Web et les premiers sites Internet.

Les premiers sites Web étaient **totalement statiques**. Les serveurs Web ne faisaient qu'envoyer des fichiers HTML, CSS et éventuellement JavaScript au navigateur qui les interprétait pour rendre la page.

Les serveurs Web n'effectuaient aucune autre tâche que d'envoyer des fichiers statiques lorsqu'ils recevaient des requêtes HTTP.

Une seconde génération de sites, devenus un peu dynamiques, utilisaient le langage C puis Perl pour générer dynamiquement du contenu HTML en fonction des requêtes et de données externes. Il a donc fallu inventer un moyen pour faire communiquer ces programmes et le serveur Web qui recevait la requête HTTP.

C'est ce qui a donné lieu au protocole **CGI** en **1993** qui **permet de faire communiquer un serveur HTTP et un programme pour générer d'une manière dynamique des documents HTML**. L'illustration la plus commune en est l'accès à une base de données, celui-ci étant de plus en plus fait à l'aide de PHP ou ASP.

A mesure que les sites devenaient plus dynamiques, d'autres langages plus spécifiques ont fait leur apparition dont **PHP** en **1994**. PHP permet ainsi de **mixer du code HTML et du code PHP** améliorant grandement la productivité.

Créé par **Rasmus Lerdorf** en 1994, PHP était à l'origine une librairie en **C** qui servait à garder une trace de toutes les personnes qui consultaient son **CV** sur son site personnel.

Il a continué à développer sa librairie et a fini par obtenir un ensemble de fonctionnalités (connexion à des bases de données, gestion des formulaires etc). Il a donc décidé de publier son code en 1995 sous le nom **Personal Home Page Tools/Form Interpreter** (**PHP/FI**).

Deux étudiants, **Andi Gutmans** et **Zeev Suraski** ont amélioré la librairie (**PHP 3**) et ont réécrit le moteur interne de PHP (**PHP 4**) leur donnant ainsi leur nom : **Zend Engine** (soit les deux premières lettres du prénom Zeev et les deux avant-dernières du prénom Andi).

Après que son utilisation ait explosé, des entreprises comme **Facebook** ou **Yahoo** se mirent à l'utiliser pour créer leurs plateformes.

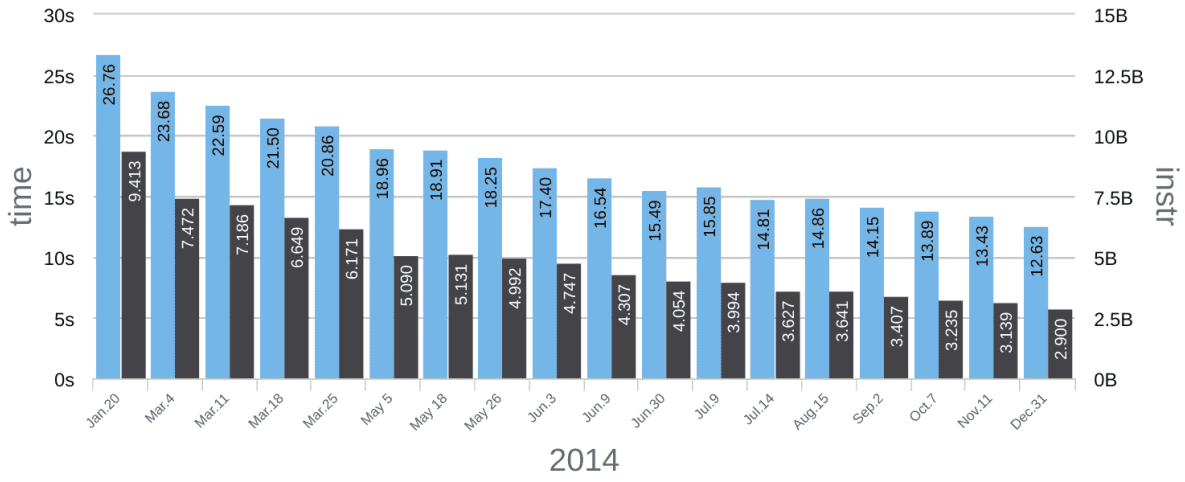
Le volume de requêtes à traiter étant bien supérieur aux petits sites personnels, Facebook a développé un ensemble d'outils pour améliorer les performances de PHP. D'abord, **HipHop for PHP** et **HPHPc** pour compiler le code source **PHP** en **C++** optimisé, puis **HHVM** qui est une machine virtuelle permettant de compiler le **PHP** en un langage intermédiaire (**bytecode**) et de l'exécuter à la volée.

En réaction, la **version 7 de PHP** a été développée avec une nouvelle version de son moteur interne en 2015. Les performances ont été drastiquement améliorées

par rapport à **PHP 5** et les **benchmarks** montrent des performances au moins égales, (et parfois supérieures), à **HHVM**.

Les résultats sont impressionnants, avec plus de 2 fois moins de ressources utilisées, pour deux fois plus de performances.

Voici par exemple, les tests réalisés par l'équipe PHP pendant le développement de PHP 7, au fur et à mesure de leurs progrès :



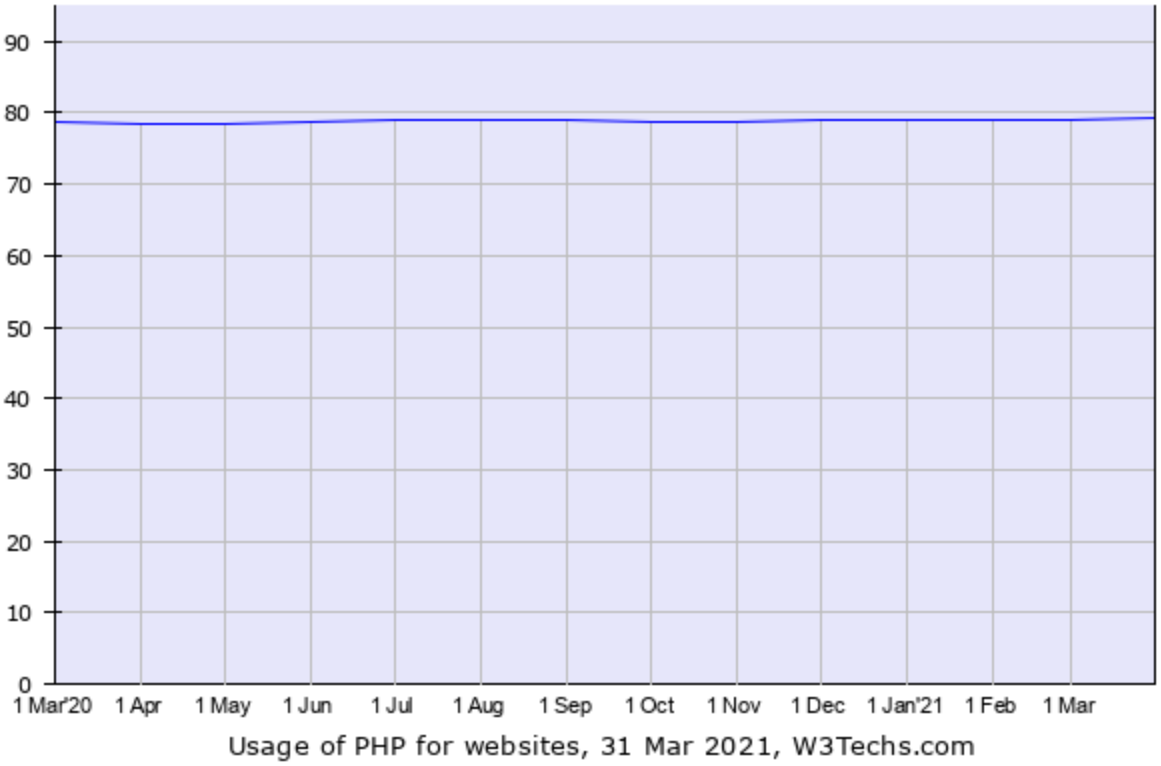
Légende : tests du nombre d'instructions et du temps nécessaire pour 100 requêtes sur la page d'accueil Wordpress au fur et à mesure du développement de PHP 7.

D'autres améliorations des performances notables ont été la **version 7.4** avec la possibilité de pré-charger tout le code PHP, augmentant les performances au démarrage et la **version 8** avec l'introduction de la **compilation à la volée** (**JIT ou Just-in-time compilation**) qui va désormais compiler des parties du code **pendant son exécution**. Il agira ainsi de la même manière qu'une version en **cache du code**, ce qui devrait nettement améliorer les performances..

Utilisation du PHP

Si **PHP**, étant un langage de scripts, peut être utilisé pour toutes ces utilisations : scripts, traitements d'images, de fichiers audios ou vidéos etc, son utilisation première, et son objectif, est de créer des sites Web.

PHP est le langage le plus utilisé par les **sites Web** :

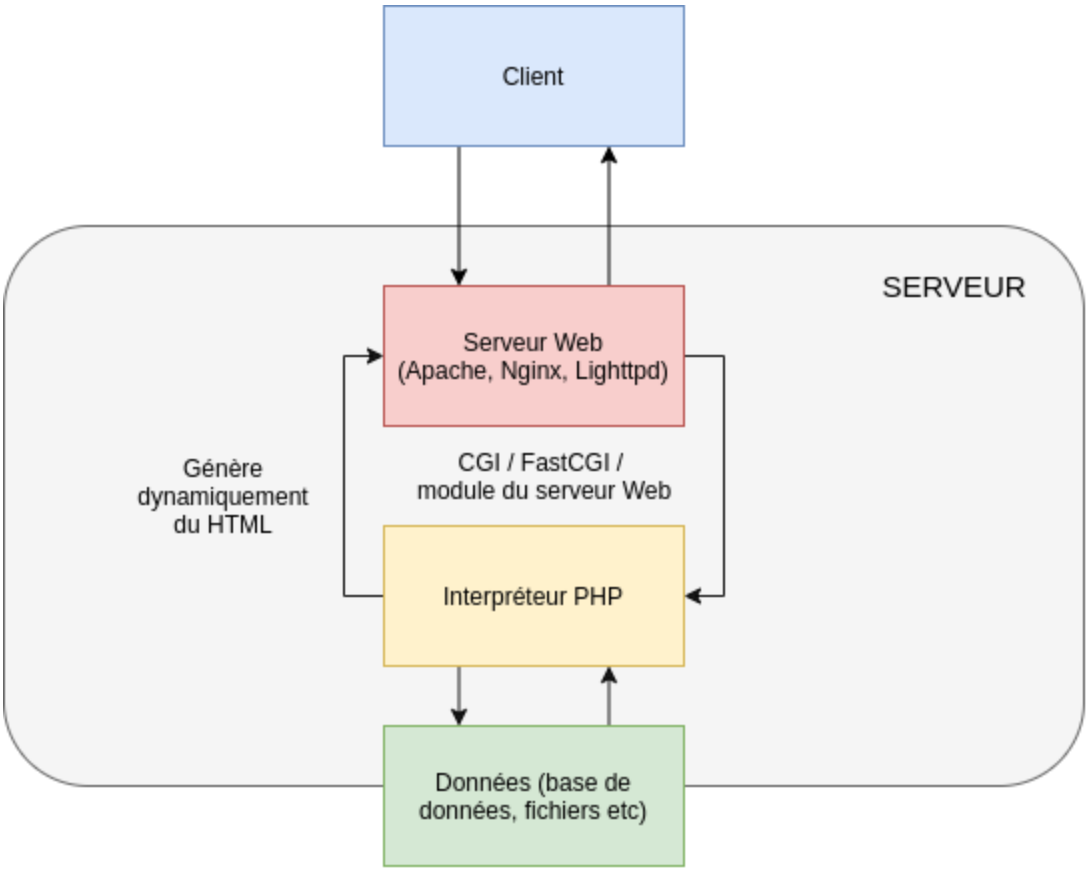


Environ 80% des sites disponibles sur le Web l'utilisent en 2021.

Des sites connus utilisant PHP sont Wikipédia, Tumblr, Etsy...

PHP pour le Web : modèle client / serveur

Nous allons voir ensemble l'utilisation du PHP dans le contexte du Web.



Les étapes sont toujours les mêmes, bien que certaines modalités ont évolué suivant l'apparition de nouvelles technologies, comme nous le verrons dans la leçon suivante.

- Premièrement, **un client envoie une requête HTTP** à un serveur. Un client est un navigateur Web (Chrome, Firefox, Safari etc), sur un appareil (ordinateur, mobile, tablette, smart TV etc).
- Deuxièmement, **le serveur Web reçoit cette requête**. Un serveur Web est situé sur un serveur physique, qui est une machine tournant en continu et connectée au réseau Internet. Un serveur Web est est un programme conçu pour recevoir des requêtes HTTP et y répondre. Les trois plus utilisés avec PHP sont Apache, Nginx et Lighttpd mais il en existe de nombreux autres.
- Ensuite, **le serveur Web transmet les données de la requête à un interpréteur PHP**. Il y a plusieurs possibilités : soit en utilisant une interface **CGI**, soit une interface **FastCGI**, soit directement à un module interne du

serveur Web contenant l'interpréteur. Nous détaillerons ces possibilités durant le cours.

- Enfin, l'interpréteur PHP va charger un ou plusieurs scripts PHP correspondant à la requête. Ces scripts vont utiliser les données de la requête, et le plus souvent des données externes (base de données, fichiers ou toute autre source), pour **générer dynamiquement du contenu HTML qui sera retourné par le serveur Web au client**.

Une requête HTTP, ou Hypertext Transfer Protocol request (en français, une "requête HTTP"), est une demande envoyée par un client (généralement un navigateur web) à un serveur web pour obtenir des informations ou effectuer une action sur le serveur. Les requêtes HTTP sont utilisées pour communiquer entre le client et le serveur dans le contexte du World Wide Web. Voici quelques éléments clés d'une requête HTTP :

1. **Méthode HTTP :** La méthode indique quelle action le client souhaite effectuer sur le serveur. Les méthodes courantes incluent GET (pour récupérer des données), POST (pour envoyer des données au serveur), PUT (pour mettre à jour des données), DELETE (pour supprimer des données), etc.
2. **URI (Uniform Resource Identifier) :** L'URI spécifie l'emplacement de la ressource sur le serveur vers laquelle la requête est dirigée. Il peut s'agir d'une URL (Uniform Resource Locator) ou d'une URN (Uniform Resource Name).
3. **En-têtes HTTP :** Les en-têtes fournissent des informations supplémentaires sur la requête, telles que les informations sur le client, le type de contenu accepté, les informations d'authentification, etc.
4. **Corps de la requête :** Dans le cas des méthodes POST, PUT, etc., des données peuvent être envoyées dans le corps de la requête. Par exemple, lorsqu'un formulaire est soumis, les données du formulaire sont incluses dans le corps de la requête.

Une fois que le serveur reçoit la requête HTTP, il la traite en fonction de la méthode et de l'URI, effectue les actions requises, puis renvoie une réponse au client. La réponse HTTP comprend généralement un code de statut (comme 200 OK pour une réponse réussie), des en-têtes de réponse et, le cas échéant, un corps de réponse contenant les données demandées.

Les requêtes HTTP sont essentielles pour le fonctionnement du web, car elles permettent aux navigateurs de récupérer des pages web, des images, des vidéos, des données JSON, et d'interagir avec des services web, entre autres. C'est le protocole fondamental qui sous-tend la navigation web et les interactions entre clients et serveurs sur Internet

Écosystème PHP

Nous allons maintenant survoler l'écosystème de PHP pour vous donner une idée des principaux outils.

1. **Les éditeurs de code**

PhpStorm est l'éditeur de code le plus utilisé pour PHP. C'est un éditeur offrant énormément de fonctionnalités (vérification du code, navigation intelligente, autocomplétion, refactorisation, débogage etc). Cet éditeur nécessite une licence (environ 200€ la première année, 160€ la seconde et 120€ les suivantes).

Visual Studio Code est l'éditeur de code le plus utilisé au monde. Il n'est pas spécifique à PHP mais plusieurs extensions permettent de bénéficier de nombreuses fonctionnalités retrouvées dans PhpStorm. L'avantage est qu'il est gratuit et Open Source, c'est pour cette raison que nous l'utiliserons dans la formation.

2. **Les CMS**

Les CMS (**content management system**) sont des systèmes de gestion de contenu.

Ce sont des logiciels permettant la création de sites Web sans avoir à les développer mais simplement à les configurer.

Le plus connu et le plus utilisé est **Wordpress** qui est utilisé par environ **40%** des sites internet dans le monde.

Il existe plusieurs centaines de CMS comme **Joomla**, **Drupal**, **PrestaShop**, **Magento** etc.

En tant que développeur, vous n'utiliserez pas de CMS sauf pour développer ou modifier des plugins.

3. ### Les frameworks

   Un framework est un ensemble de bibliothèques et d'outils permettant d'architecturer un projet et d'augmenter la productivité des développeurs.

   Les principaux avantages sont d'avoir **beaucoup moins de code à écrire** (plein de fonctionnalités sont déjà codées de manière optimisées), d'avoir **un cadre solide** pour travailler à plusieurs grâce à une architecture recommandée (chaque développeur ne fait pas son architecture dans son coin au risque d'avoir de graves problèmes de maintenabilité) et d'avoir une **meilleure sécurité** (de nombreuses configurations empêchant diverses attaques sont déjà incluses).

   Il existe environ **40 frameworks PHP**.

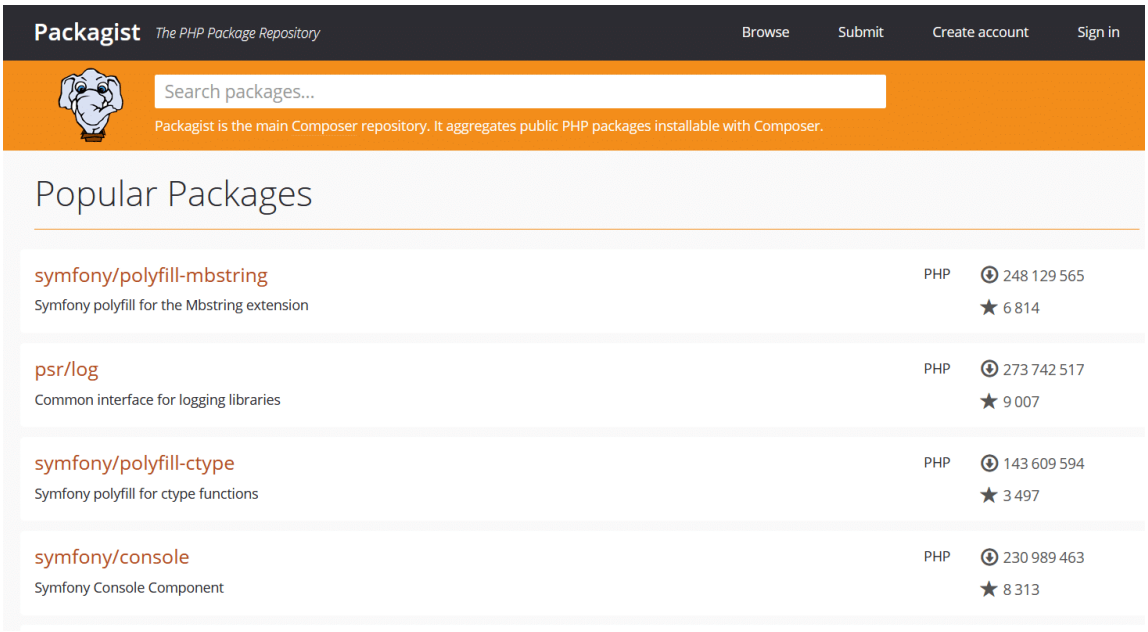
   Les trois principaux frameworks **PHP** sont **Symfony** (2005), **Laravel** (2011) et **CodeIgniter** (2006).

Symfony Laravel CodeIgniter

4. **Gestionnaire et répertoire de dépendances**

   Composer est le principal gestionnaire de dépendances (aussi appelés packages ou paquets) pour PHP. Il permet de gérer toutes les installations et toutes les mises à jour des libriairies utilisées par votre projet.

   Packagist principal répertoire pour Composer. Il contient des milliers de librairies utilisables (appelés packages).



▼ Comment fonctionne PHP ?

L'évolution de l'exécution du PHP sur un serveur

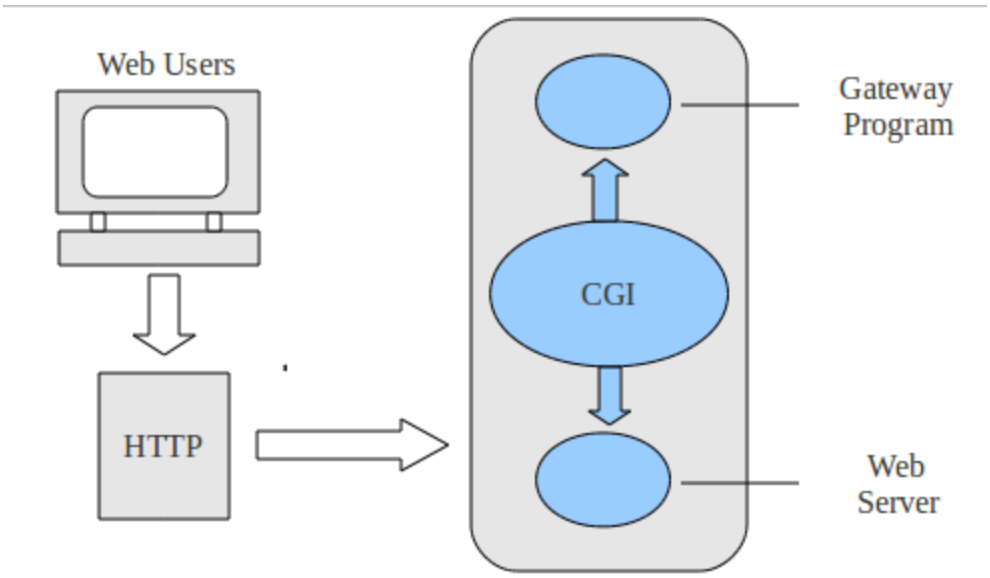
Il y a de nombreux moyens d'exécuter du PHP sur un serveur.

Nous allons étudier les principaux, dans une approche historique.

La Common Gateway Interface (CGI)

CGI est une interface normalisée (c'est-à-dire qu'il existe des spécifications officielles, appelées **RFC**), utilisée par les serveurs Web pour **exécuter un programme spécifique lorsqu'ils reçoivent une requête** HTTP.

Autrement dit, au lieu de directement retourner du HTML, du CSS et des images par exemple, CGI **permet au serveur de demander à un programme de générer du contenu dynamiquement puis de le retourner**.



CGI est totalement indépendant du langage de programmation utilisé comme interpréteur, cette interface est utilisable en C, Python, PHP, Java ou tout autre langage pouvant tourner sur un serveur.

L'inconvénient de CGI est qu'il faut lancer un **nouveau processus** pour chaque requête HTTP.

Chaque requête HTTP lance une instance de CGI qui va ensuite appeler le programme à exécuter pour générer le contenu dynamique, à savoir PHP.

Tous les modules PHP et les configurations (notamment le fichier d'initialisation **php.ini**) doivent être chargés à chaque requête.

Cela conduit à de très mauvaises performances, qui empire si un nombre relativement important de requêtes doivent être traitées.

Cette technologie a permis la première génération de sites dynamiques. Elle est aujourd'hui très fortement déconseillée, au vu de ses performances déplorables.

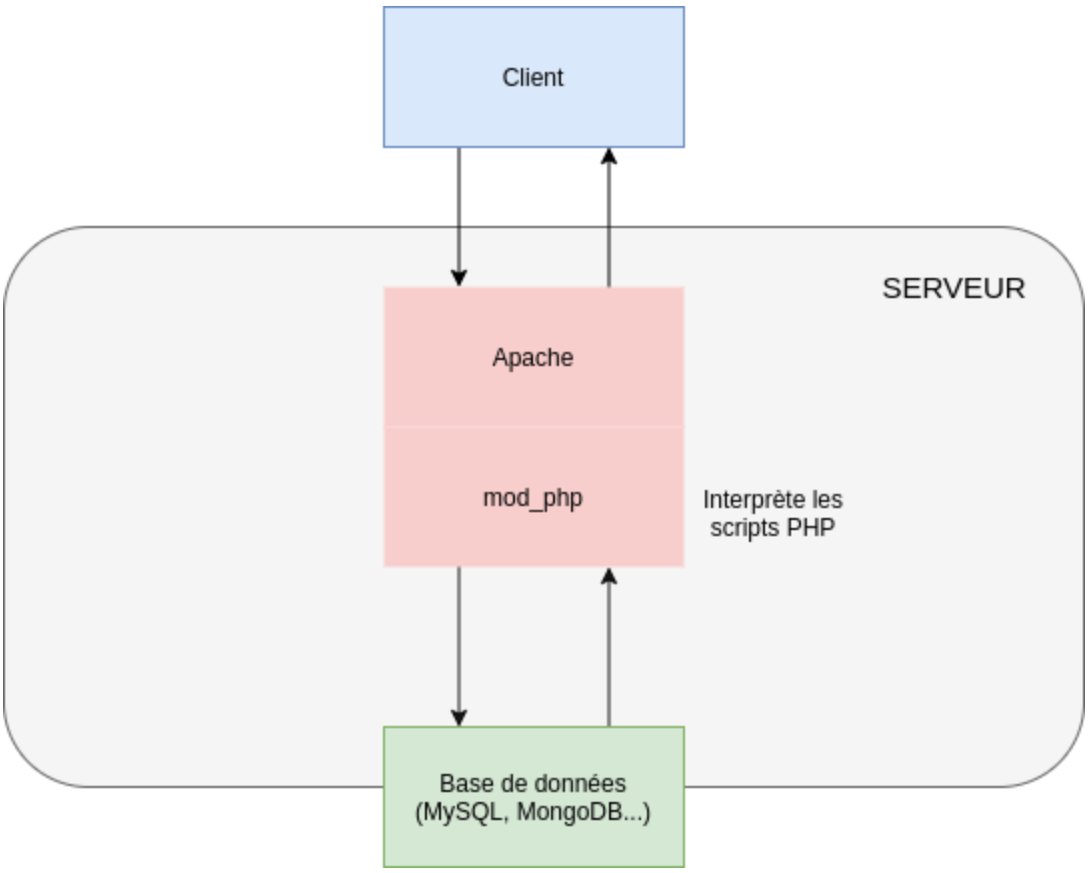
Les modules serveurs

mod_php est un module pour le serveur **Apache**. **PHP** est intégré directement dans la partie serveur Web. C'est d'ailleurs d'où vient son nom, qui signifie "*PHP en tant que module".*

Contrairement aux interfaces CGI, il n'y a pas d'autres processus ni de protocole de communication entre l'interpréteur PHP et le serveur Web. Tout est géré par le

processus Apache qui contient dans un module l'interpréteur PHP.

La plupart des benchmarks montrent une performances entre 300% à 500% plus rapide qu'en utilisant l'interface CGI.



L'évolution FastCGI

Une évolution de la technologie **CGI**, appelée **FastCGI** a fait son apparition et permet de remédier aux problèmes de performance initiaux.

Une seule instance de l'interpréteur PHP, **avec tous les modules et les configurations est chargée**. Vous pouvez le voir comme une sorte de serveur.

Elle ne reçoit que les fichiers PHP à traiter suivant la requête. Ainsi, les ressources (RAM, disque, CPU) peuvent être réutilisées pour toutes les requêtes HTTP, elles sont partagées et elles ne sont plus dédiées à chaque requête.

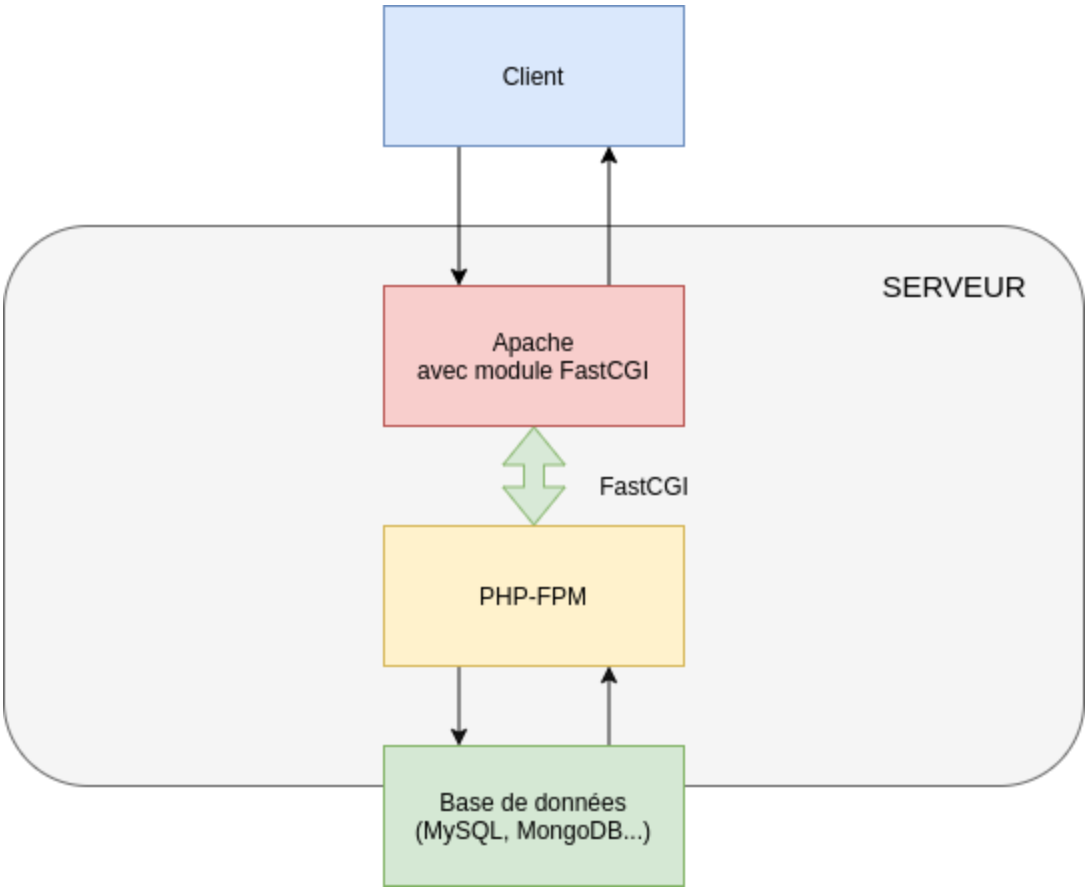
L'implémentation la plus connue de ce protocole, pour PHP, est PHP-FPM qui signifie PHP - FastCGI Process Manager. Il permet d'utiliser le protocole **FastCGI**

pour la communication entre un serveur Web et PHP. Sa première version est sortie en 2004.

Depuis 2010, **PHP-FPM** est intégré à la base de code PHP.

Le serveur Web communique avec **PHP-FPM**, soit directement (en utilisant **NGINX**), soit avec un module spécifique pour **Apache** (comme par exemple mod_fcgid, mod_fastcgi, proxy_fcgi, mpm_event etc).

PHP-FPM est un service, c'est-à-dire un processus exécuté en arrière plan, qui va gérer des processus enfants. Ces processus enfants vont traiter les requêtes passées par le serveur Web.



L'approche la plus moderne avec l'apparition de NGINX

Apache n'est plus tout jeune. Cette technologie de serveur Web date de 1995.

Son modèle est que chaque connexion est gérée par un processus distinct. Cela entraîne que pour 100 connexions, il faut 100 processus qui consomment chacun de

la mémoire.

Cela provoque également de très nombreuses failles de sécurité (recherchez des vidéos youtube sur les attaques slowlorris, slowhttprequest, apache DDoS etc si vous êtes intéressé).

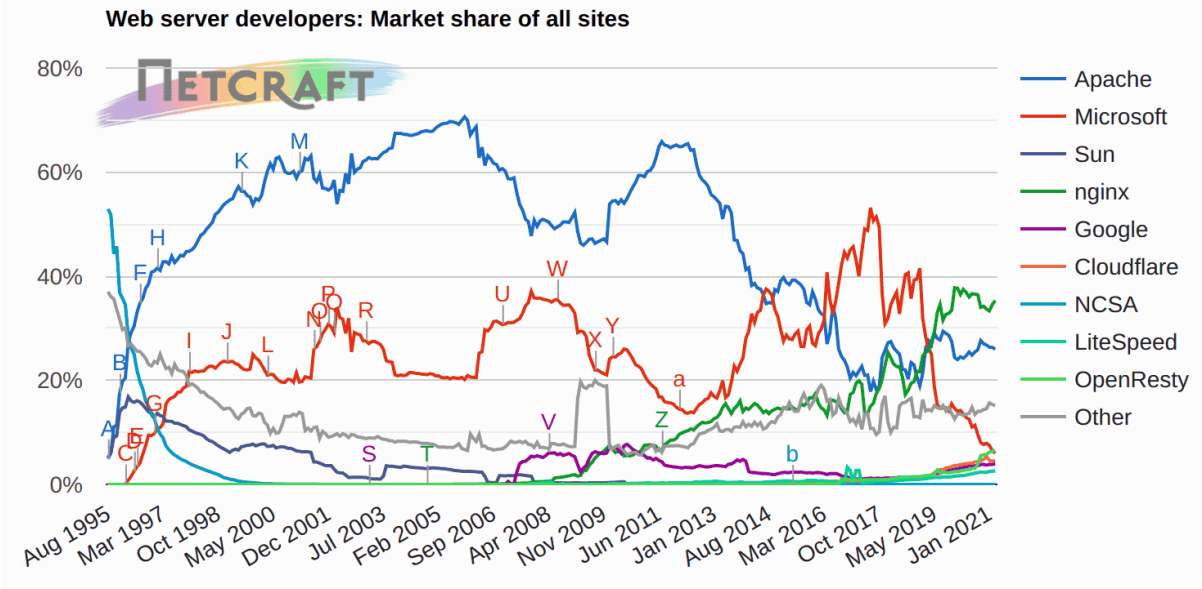
NGINX a été créé spécifiquement en 2002 pour "pour répondre aux limitations de performance des serveurs Web Apache". (https://www.nginx.com/blog/nginx-vs-apache-our-view/)

Son modèle est **event driven**, c'est-à-dire **orienté événement**. Un même processus peut gérer plusieurs milliers de connexions simultanées.

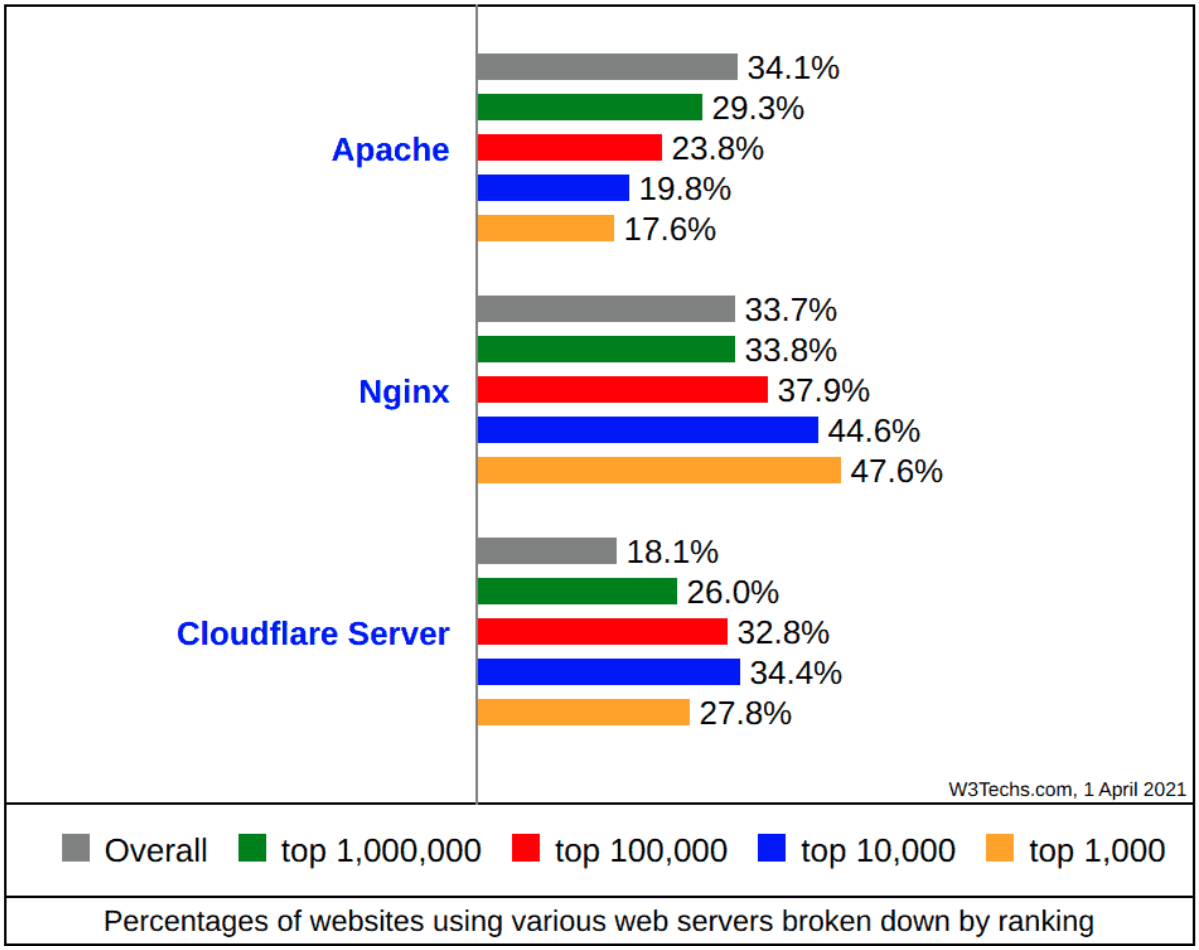
L'utilisation d'**NGINX** a complétement explosé, par exemple pour le directeur des services d'information de **Salesforce**, **Chris Lea** :

> "Apache est comme Microsoft Word. Il a un million d'options mais vous n'en avez besoin que de 6. NGINX fait 6 choses, il fait 5 d'entre elles 50 fois mieux qu'Apache".

Aujourd'hui **Apache** (**25%**) s'est c**omplètement effondré** au profit **d'NGINX** (**35%**), et cela continuera pour les années à venir :



C'est encore plus vrai si l'on considère les 1000 sites les plus fréquentés au monde, 47.6% pour NGINX contre 17.6% pour Apache :

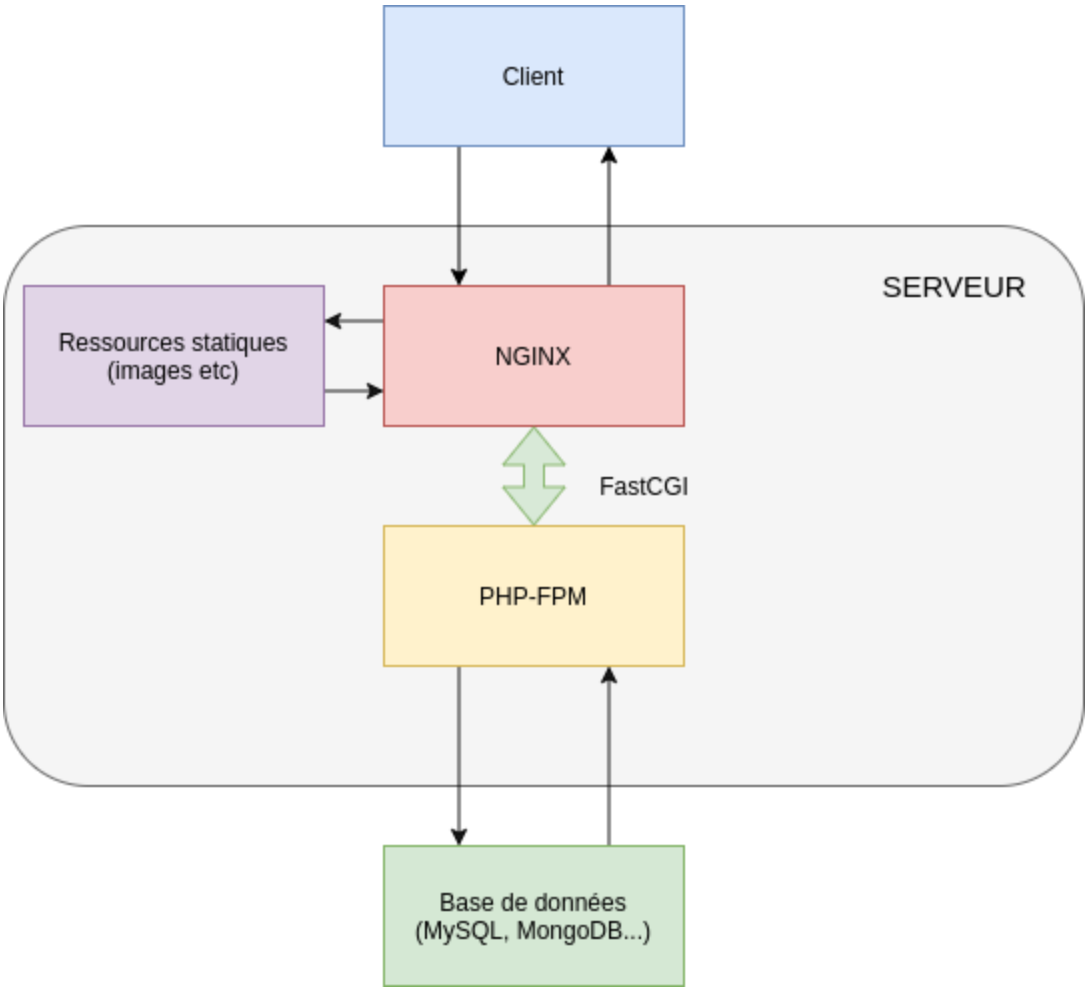


Enfin, ce changement se fait également car le Web a énormément évolué : adoption de l'architecture en microservices, protocole REST, WebSockets, conteneurisation, livraison continue (CI / CD) etc.

NGINX est plus apte à adopter le Web moderne de part son architecture.

D'ailleurs, le plus grand utilisateur de PHP au monde, à savoir Wordpress, utilise uniquement NGINX et a totalement abandonné Apache. Ils utilisent plus de 1000 serveurs avec 36 load balancers NGINX et bien sûr, uniquement PHP-FPM. Ils utilisent NGINX également pour servir tous les fichiers statiques et dynamiques (https://www.nginx.com/success-stories/nginx-wordpress-com/ et https://barry.blog/2012/06/16/nginx-spdy-and-automattic/)

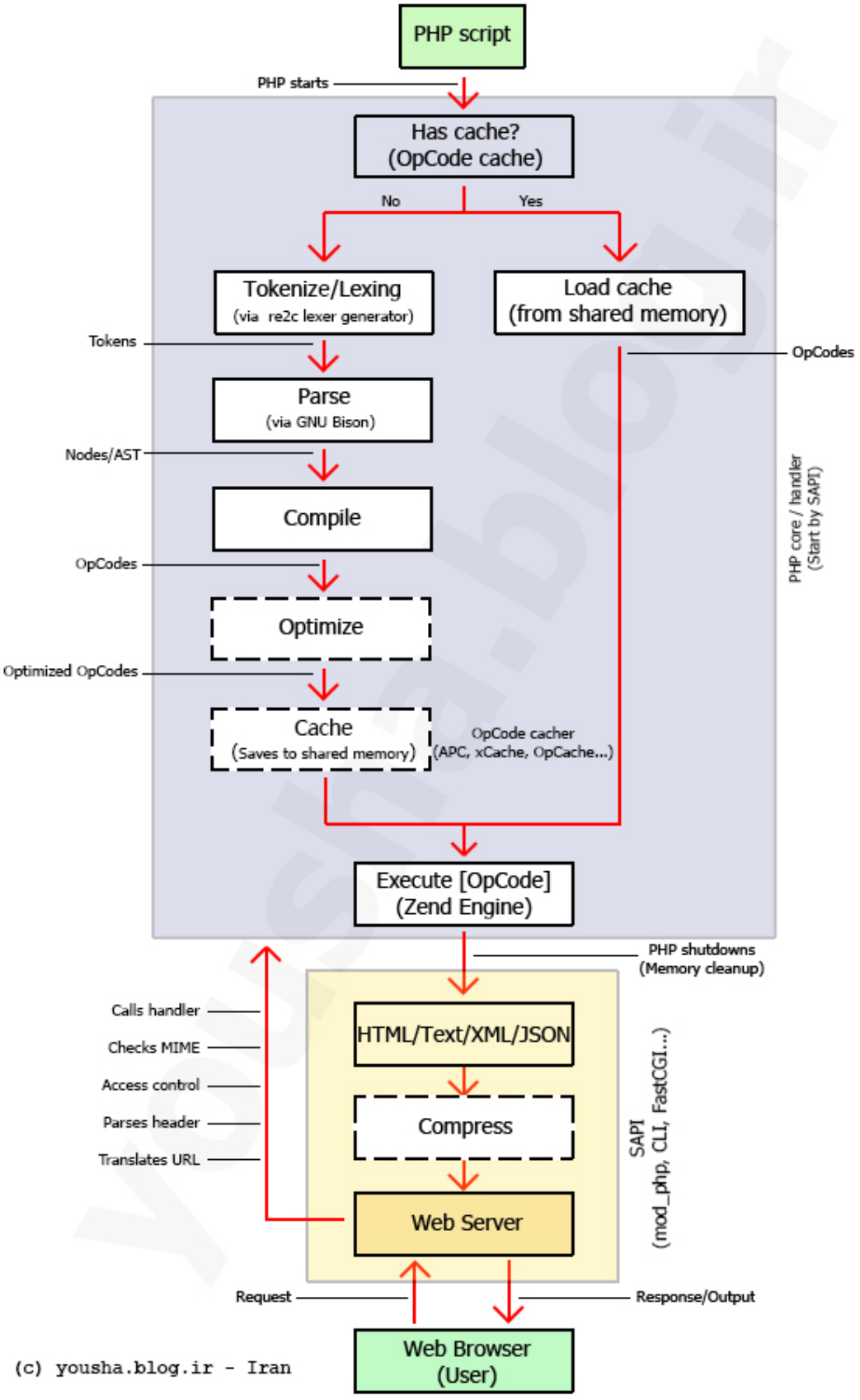
Aujourd'hui on utilise donc la combinaison NGINX / PHP-FPM pour servir ses applications PHP en production :



Le fonctionnement bas niveau du PHP

Nous avons vu les différentes manières d'exécuter l'interpréteur PHP sur un serveur et quel serveur Web utiliser. Nous allons maintenant passer à l'interpréteur en lui-même.

Voici un excellent schéma résumant ce que fait l'interpréteur PHP :


PHP script
PHP starts
Has cache?
(OpCode cache)
No
Yes
Tokenize/Lexing
(via re2c lexer generator)
Load cache
(from shared memory)
Tokens
OpCodes
Parse
(via GNU Bison)
Nodes/AST
Compile
OpCodes
Optimize
Optimized OpCodes
Cache
(Saves to shared memory)
OpCode cacher
(APC, xCache, OpCache...)
PHP core / handler
(Start by SAPI)
Execute [OpCode]
(Zend Engine)
PHP shutdowns
(Memory cleanup)
Calls handler
Checks MIME
Access control
Parses header
Translates URL
HTML/Text/XML/JSON
Compress
Web Server
SAPI
(mod_php, CLI, FastCGI...)
Request
Response/Output
Web Browser
(User)
(c) yousha.blog.ir - Iran

Le moteur **PHP** va d'abord charger le ou les scripts nécessaires contenant du code PHP en mémoire.

Il va ensuite effectuer une **tokenization** : c'est-à-dire lire le code PHP et le découper en unités utilisables par un programme. Par exemple, <?php va être transformé en T_OPEN_TAG. Vous pouvez retrouver tous les tokens à cette adresse (https://www.php.net/manual/en/tokens.php).

L'étape suivante est le **parsing** : les tokens sont lus et organisés en une structure en arbre appelée **AST** (**Abstract syntax tree**). Cela permet d'organiser les tokens en une succession d'opérations logiques.

L'étape suivante est la **compilation** : l'AST est transformé en code exécutable par la machine virtuelle **Zend VM**. On appelle cela l'**Opcode**, pour code opération, c'est-à-dire des instructions en langage machine qui spécifient les opérations à effectuer.

Cet **Opcode** est mis en cache pour ne pas devoir à chaque fois effectuer toutes les opérations précédentes pour chaque fichier PHP. Le code est ainsi compilé qu'une seule fois.

Il est à noter que depuis la **version 7.4** de PHP il est possible de précharger tous ses scripts PHP dans le moteur PHP.

Ils seront alors tous mis en **opcache** et tous les symboles (fonctions, classes etc) seront disponibles pour toutes les requêtes à venir sans avoir à effectuer les opérations précédentes.

Les architectures communes

Vous retrouverez souvent plusieurs architectures historiques, organisée avec un système d'exploitation, un langage serveur et un système de gestion de base de données (***SGBD***).

L'architecture **LAMP** (**Linux**, **Apache**, **MySQL**, **PHP**) est l'architecture la plus commune. Elle utilise une distribution Linux, le serveur Web Apache, le SGBD MySQL et l'interpréteur PHP. Les variantes sont **WAMP** pour Windows et **MAMP** pour MacOS.

Un autre exemple est le logiciel **XAMPP** qui installe et gère Apache MySQL Perl PHP pour le développement local. Son usage ayant toujours été réservé aux tests et non à la production.

Ces architectures, bien qu'encore communes, sont dépassées pour de nombreuses raisons, mais la principale est qu'Apache est en forte perte de vitesse au profit ***d'NGINX***.

▼ Installation de l’environnement

1. Installation de VS Code

   Pour installer l'éditeur VS Code, il suffit de le télécharger et de l'installer. Il fonctionne sur tous les environnements : Linux, Windows et MacOS.

   https://code.visualstudio.com/

2. Installation d'extensions VS Code

   Allez dans l'onglet extension sur VS Code.

   Recherchez et installez **Bracket Pair Colorizer** et **PHP Intelephense**.

   Bracket Pair Colorizer permet de changer les couleurs des crochets, parenthèses et accolades (()， []， {}) pour mieux identifier les paires ouvrantes / fermantes.

   PHP Intelephense permet d'apporter plein de fonctionnalités pour le traitement du langage PHP dans VS Code.

3. Installation de Git

   Par défaut, sur Windows, le terminal est Powershell.

   Dans tous les cours nous utilisons bash, qui est le terminal le plus utilisé et que vous retrouverez sur les serveurs et sur la plupart des environnements de développement.

   Nous allons donc installer git pour pouvoir l'utiliser : téléchargez le ici.

   Git - Downloads

   The entire Pro Git book written
   by Scott Chacon and Ben Straub is available to read online for free.
   Dead tree versions are available on Amazon.com.

   https://git-scm.com/downloads

4. Installation de PHP

   Vous avez 2 possibilités pour installer PHP sur votre machine:

   - La première manière c'est de l'installer via son **exécutable** ou via `Xamp`, `Wamp` etc, mais cette méthode vous impose à avoir une version spécifique de PHP installer si plus tard vous avez besoin d'une autre vous serez obligé de refaire la même procédure et ceci peut entrainer des conflits de versions suivant vos différents projets en PHP.

     Pour installer l'éditeur PHP, téléchargez l'exécutable ici.

     > PHP For Windows: Binaries and sources Releases
     >
     > This site is dedicated to supporting PHP on Microsoft Windows.
     > It also supports ports of PHP extensions or features as well as providing special builds for the various Windows architectures.
     >
     > https://windows.php.net/download#php-8.2

     Cliquez sur le lien de téléchargement Zip de la **version x64 Non Thread Safe.**

     Dézippez le dossier téléchargé par exemple dans `C > Programmes > php`.

     Il faut maintenant indiquer au système où trouver le programme php lorsque nous tapons dans le terminal php.

     Pour ce faire, nous allons ajouter le chemin vers le programme dans nos **variables d'environnement**.

     Dans la barre de recherche Windows tapez **environnement** puis sélectionnez **Modifier Variables d'environnement** puis **Variables d'environnement**, cherchez la variable **Path** et cliquez sur **modifier**.

     Ajoutez le chemin vers les exécutables **php**. Dans notre exemple, `C > Programmes > php.`

     Quittez complètement VS Code et tous les terminaux éventuellement ouverts.

     Relancez VS Code et ouvrez un terminal.

     Vérifiez l'installation de PHP :

```
php --version
```

- La deuxième méthode que je vous recommande fortement consiste à installer **Laragon**.

  Laragon est un environnement de développement web rapide, flexible, intuitif, productif et puissant qui d'adresse à tous. Laragon regroupe en un seul installeur tous les outils dont vous pourriez avoir besoin pour votre développement :

  - `Apache`
  - `Nginx`
  - `PHP`
  - `MySQL`
  - `Cmder`
  - `Git`
  - `WinSCP`
  - `NodeJS`
  - `Putty`

  Il est aussi extensible afin de permettre l'ajout de nouvelle fonctionnalités comme de nouvelles versions de PHP ou d'autres outils (PostgreSQL, MongoDB, Ruby, Python...).

  Nous allons donc installer `Laragon` pour pouvoir l'utiliser : téléchargez le ici.

  > Download
  >
  > Laragon is a universal development environment. It has many features to make you more productive:Benefits of Laragon After downloading, You can add git, phpmyadmin, Node.js/MongoDB, Python/Django/Fla
  >
  > https://laragon.org/download/