PWA- Pré-requis

Y. Stroppa
2020

Sommaire

Références et sites

Applications PWA :

https://pwa.rocks

Introduction

Pour des raisons de clarification, il semble important de préciser certaines choses dans l'environnement informatique dans lequel nous allons progresser. En effet, une mise au point sur l'utilisation et la variété des langages à utiliser dans le développement applicatif ainsi qu'une clarification de la notion de service, port et persistance semble indispensable pour poursuivre dans des concepts PWA qui devront s'appuyer sur ces fondamentaux.

Les langages

Point sur la variété des langages. On pourrait commencer par une pseudo classification de ces langages en trois grandes familles : langages interprétés, compilés, et pseudo-compilés.

Les langages
interprétés

Parmi les langages interprétés, on peut citer PHP, JavaScript, Perl, python, Ruby,shell .... ce sont principalement des langages de scripting permettant de décrire des instructions des plus complexes. Ils s'appuient chacun sur un interpréteur qui permettra de traduire le langage en instructions machines pour être exécuté. Le principe d'un langage de ce type est d'être interprété, donc nécessité d'installer les éléments indispensables pour effectuer cette opération. Sinon incompréhension de ces fichiers scripts par votre système. Pour Javascript l'interpréteur le plus classique est directement le navigateur à partir de la console. Pour ceux qui dispose de Nodejs on pourra l'utiliser également.Pour tous ces langages il est nécessaire d'installer l'interpréteur sur les différentes machines qui seront susceptibles de l'utiliser.

Les langages
langages compilés

On peut distinguer d'autres langages comme C, Fortran et C++ principalement dédiés et os dépendant. Ces langages très utilisés dans le domaine de traitement et du calculs offrent des possibilités de performance très intéressantes du fait qu'ils sont compilés. C'est à dire qu'à partir des sources on utilise un compilateur qui vérifie et traduit les instructions source en langage machine en passant par différentes étapes pour obtenir un fichier binaire directement exploitable et interprétable par la CPU de votre machine. Mais qui est spécifique pour un environnement et un système dédié.

Les langages
langages pseudo compilés

L'avantage des langages compilés est leur performance mais présentent des inconvénients dans leur distribution, car on effet lors de l'utilisation de plusieurs systèmes de type Windows, Unix, Linux ...il est nécessaire de produire des livrables pour chaque type de système car les versions ne sont pas compatibles. Donc pour pallier à ce problème de distribution, ce que l'on ne connaissait pas avec les langages interprétés, de nouveau concept et paradigme ont vu le jour. On a essayer de prendre les bons côtés des deux solutions l'interprété et le compilé pour bâtir des langages comme Java et par la suite C#, F#, ...Ces langage présentent comme avantage d'être pseudo-compilés dans un langage intermédiaire (en Byte-code pour Java et en CIL pour C#) et d'être ensuite interprété par une machine virtuelle VM pour Java et Dot.Net pour C#. De ce fait, le livrable est compatible sur tous les systèmes et nécessite l'installation de la couche nécessaire pour permettre son exécution. (JRE, JDK, DotNet)

Commutateurs de navigateurs

Chrome/Chromium

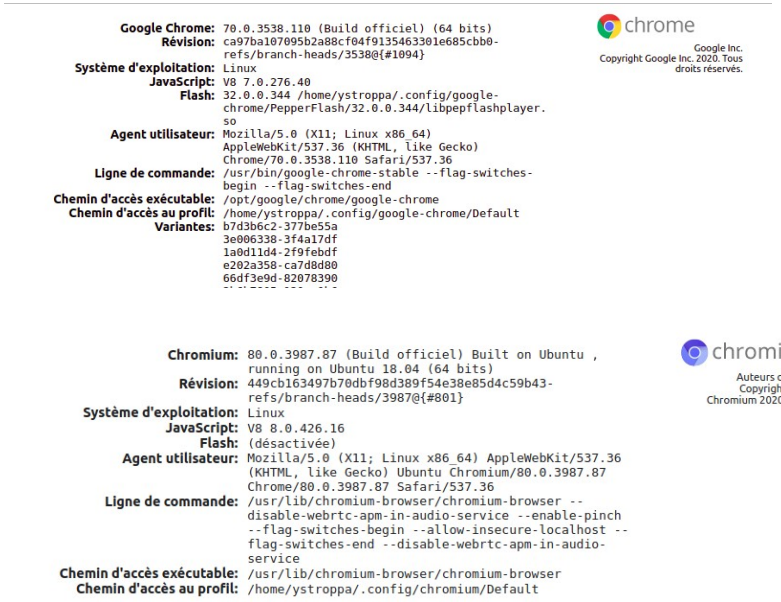
chrome://chrome-urls/ Liste des options possibles d'utiliser

chrome://flags permet de modifier certains paramètres de chrome

chrome://apps

chrome://settings

chrome://version



Firefox

about:serviceworkers

about:config

Frameworks de développement Web

Ionic

Angular.js

Quazar

React.js

Vue.js

Meteor

Ember.js

Mithril

Nodejs

Polymer

Aurelia

Backbone.js

Architecture

Une architecture est organisée d'un ensemble des machines physiques (Serveurs) dotés d'une ensemble de service le tout interconnecté via un réseau. Chaque ressource (serveur) est attaché au moins sur un Lan (réseau) qui leur permet de communiquer les uns avec les autres pour offrir des services . Parmi les services on peut citer :

Services d'authentification centralisé tels que des annuaires (Active Directory ou Ldap) : leur rôle est de permettre le contrôle des accès aux différentes ressources de l'établissement ;

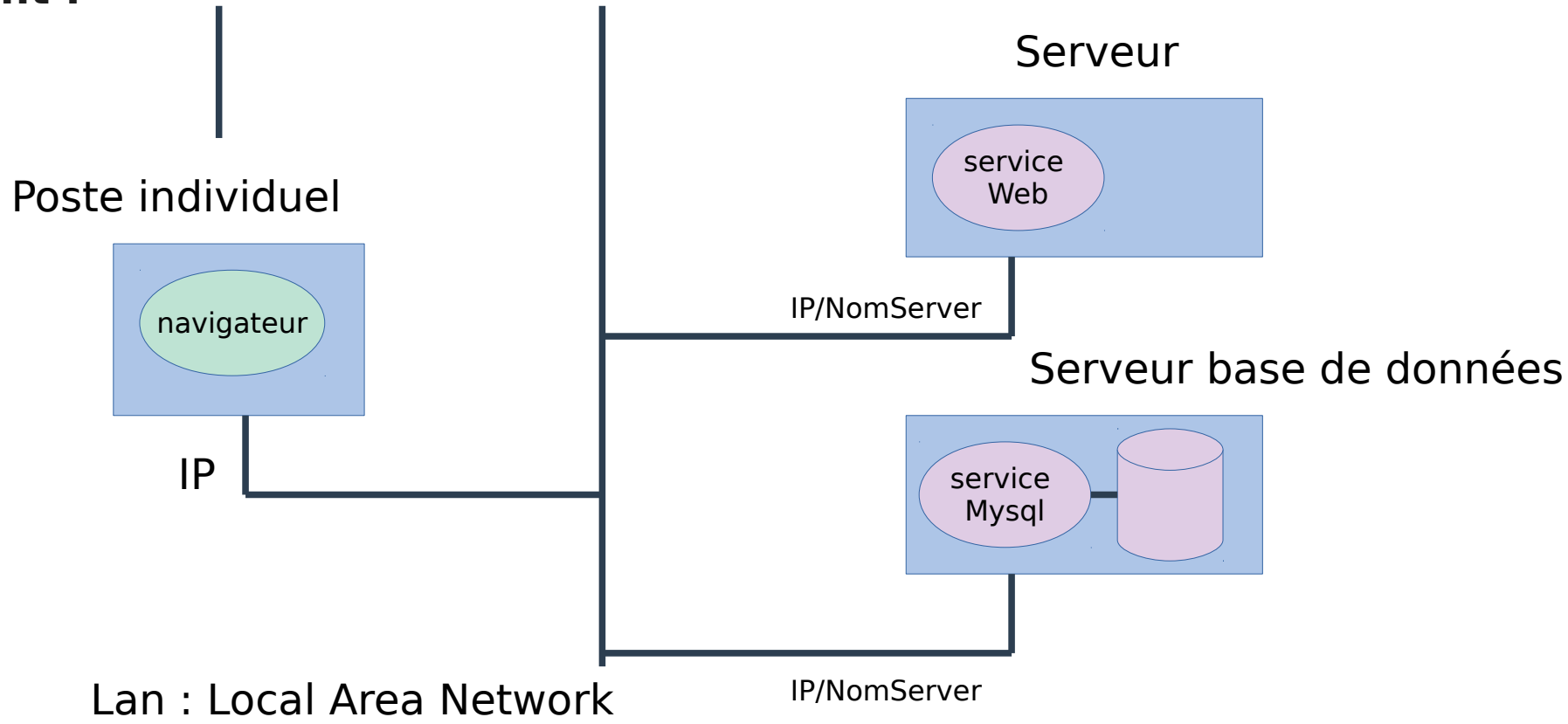
Services de messagerie dans lesquels on peut distinguer les services d'envoi de mail de type SMTP, de réception de mail POP, IMAP … ;

Services d'impression permettant de gérer des ressources d'impressions ;

Services Métiers qui offrent des accès à des applications en ligne de type SAP ou autres applications métiers de vos structures ; qui peuvent être composés de plusieurs parties Serveur et client ; et pour finir afin de rester dans notre périmètre on peut parler de service web (Intranet, Extranet etc …)

Architecture

Si on examine d'un peu plus près ce type de service pour essayer de situer les différentes parties et leur composition. Soit le schéma suivant :



Architecture

Voici la composition un peu plus classique de ce que l'on rencontre dans un établissement : des services Web (Apache, IIS,Nodejs, autres python), des services de bases de données (Mysql, Mariadb , Postgresql, Oracle, SqlServer ...) ... le tout formant des applications accessibles à partir d'un navigateur. C'est le service Web qui expose à l'aide de différentes pages Web

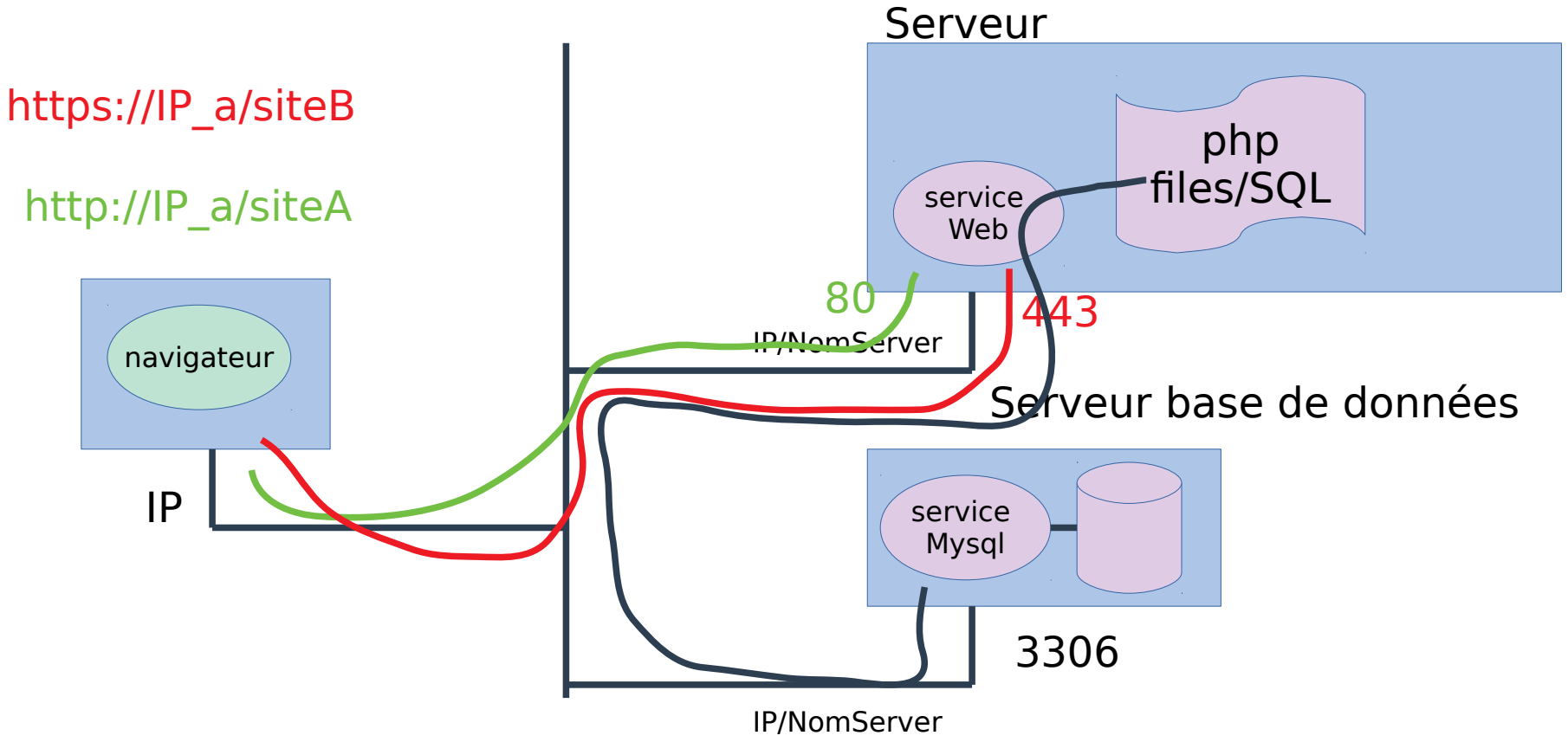
Etant donné qu'un serveur peut abriter un ensemble de services, il est nécessaire qu'il puisse distribuer les trames réseaux qu'il reçoit sur la même interface (carte réseau) aux différentes applications qu'il héberge.

Ceci est possible, car chaque service installé de type Réseau va "handler-- va accrocher" une socket sur un numéro de **port** donné (convention).

Par exemple le service apache va écouter sur le port 80 ou/et 443 en fonction du protocole HTTP ou HTTPS, le service de base de données Mysql écoute par défaut sur le port 3306 ... et ainsi de suite chaque application (Service -- démon) réseau s'alloue un numéro de port et pourra ainsi gérer les demandes clients.

Architecture

Ce qui nous donne sur ce schéma



Architecture

Vos pages php pour accéder à la base de données doivent tout d'abord établir une connexion.

Pour ce faire il est nécessaire de définir les paramètres de connexions suivants :

- Localisation du service (Nom de la machine ou IP)
- Le login/password : permet de contrôler qui se connecte et d'affecter les autorisations associées
- La database sur laquelle vous voulez travailler

D'inclure tous ces paramètres dans la demande de connexion, et de vérifier que le service est bien accessible (Firewall à configurer).

Architecture -- SQL

Une fois tous ces paramètres définis, il nous reste à inclure les différentes instructions SQL que l'on encapsule dans les pages PHP pour produire des pages html. Les instructions que l'on peut utiliser peuvent être catégorisées en deux familles : LMD et LDD.

LMD : langage de manipulation de données qui regroupe les instructions : insert into, update, select, delete

LDD : langage de définition de données regroupe pour sa part les instructions SQL qui permettent de modifier et de créer les éléments contenus dans une base de données tels que create, drop et alter ...

Bien sur l'accès à ces instructions va dépendre de l'autorisation que l'on vous donne et qui sera conditionnée par la DBA DataBase Abdministrator ; Pour des raisons de sécurité.

Architecture -- Relationnel

Essayons de situer la base de données dans un SI et de son rôle prépondérant dans l'architecture informatique. Les applications développées au sein d'infrastructure ont généralement comme objectif de présenter des interfaces (IHM) aux utilisateurs qui leur permettent d'effectuer leurs missions. L'application peut être considéré comme un accès facilité à la manipulation de données (d'informations) de l'entreprise que ce soit pour la gestion du personnel, de clients, ou de fournisseurs en passant pas la gestion des produits. Donc toutes les données manipulées à travers ces applications finissent à un moment ou à un autre dans une base de données. Le rôle de la base de données est donc multiple : conserver les données, les mettre à disposition d'un ensemble d'applications ; garantir leur cohérence et contrôler leur intégrité, sécuriser les accès et être performante.

Architecture -- Relationnel

Donc comme on peut le voir, son rôle est crucial pour les SI. Ce rôle est possible car pour construire une telle base de données, vous avez suivi une certaine démarche de conception qui est le fruit d'une longue expérience et qui tourne autour du principe relationnel. Mais qu'est ce qu'une base de données relationnelle telles que Mysql, PostgresSQL, Oracle, SqlServer ... et bien d'autres. Voir le concept base de données relationnelles et la notion de relation qui s'appuie sur les notions opérateurs relationnels.

Reprenons simplement juste pour clarifier ce concept la notion de relation. Une relation est définie comme l'expression suivante :<Attrib1:Dom1, Attrib2:dom2 ...>. Où chaque attribut est défini dans un domaine de validité.

Architecture -- Relationnel

La traduction d'une relation dans une base de données est la table. Qui est composée de colonnes où chaque colonne est défini et appartient à un type. Le type fait office de domaine de définition, qui sera à chaque modification de la table contrôlé et vérifié. Ce sont les premières contraintes d'intégrité qu'effectue la base de données. De plus dans la relation nous allons définir la notion de PK : primary Key qui va nous apporter une garantie d'unicité des enregistrements stockés dans chaque table ... pour éviter les redondances. Et la notion de FK Foreign Key qui va nous amener des contrôles de référence entre les tables sachant que suite aux méthodes de conceptions nous sommes amenés à répartir certaines données reliées (au sens métier) entre elles dans plusieurs tables. Cette deuxième partie fait office de ce que l'on appelle contrainte d'intégrité référentielle. Autre contrainte que va nous garantir la cohérence de la base de donnée à chaque modification d'enregistrement.

Architecture -- Relationnel

Sur ces mécanismes de PK, le SGBD associe de façon systématique de l'indexation. Car de part la conception, nous pouvons sans nous tromper beaucoup faire le pari que beaucoup de requêtes vont utiliser la clé primaire des relations. Donc on a tout intérêt à associer à ces clés primaires des index qui vont permettre au moteur de requêtes d'optimiser ses traitements. Bien entendu pour des raisons de performance, vous pouvez dans ce type de structure vous appuyer sur ces constructions d'index pour améliorer vos propres traitements. Les index utilisent des mécanismes de B-Tree associées à la table qui permettent de trouver plus rapidement les éléments recherchés. Ces mécanismes sont également utiliser dans certains langages dans les notions de collections de type (Key,value).

Architecture -- Relationnel -- transaction

Les transactions dans une base de données relationnelle ont des propriétés ACID qui amènent encore une fois une garantie d'un état cohérent de la base.

ACID : Atomicité, Cohérence,Isolation et Durabilité.

Tous les éléments compris entre les instructions par exemple en PHP du type beginTrasaction() sont interprétés comme un tout soit tout est validé à l'aide de commit soit tout est annuler à l'aide de rollback.

```
<?php
/* Démarre une transaction, désactivation de l'auto-commit */
$dbh->beginTransaction();
/* Modification du schéma de la base ainsi que des données */
/* Instructions SQL  */
/* Instructions SQL  */
/* Instructions SQL  */

/* test de validité soit tou c'est bien passé et on utilise à la finde la transactionl'autocommit
/* soit on s'aperçoit d'une erreur et on annule les modifications */
$dbh->rollBack();
/* Le connexion à la base de données est maintenant de retour en mode auto-commit */
?>
```

Architecture -- Relationnel

Pour finir sur ces bases de données relationnelles afin de bien apprécier ce que l'on utilise et pourquoi on l'utilise, il faut évoquer les notions complémentaires que l'on a dans une base de données ce sont les notions de :

Vues et de procédures stockées.

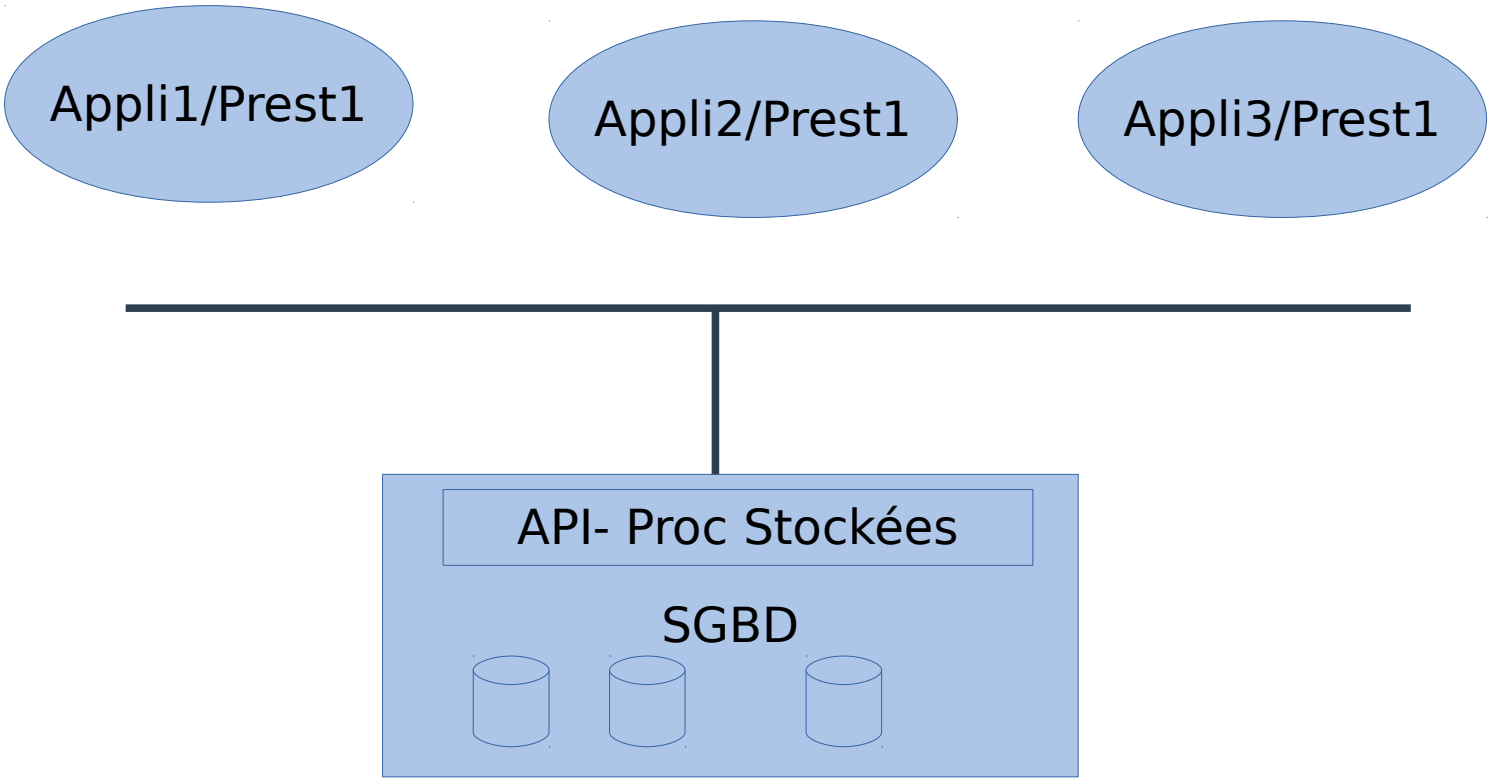
La vue est ce que l'on appelle une relation dérivée qui se traduit dans la base de données par une requête SQL que l'on préserve sous cette dénomination et que l'on peut utiliser dans des requêtes. Le but de la vue est multiple aussi, alléger les requêtes en terme d'écriture, et d'ajouter un niveau d'abstraction supplémentaire. Ne pas dévoiler la structure réelle de votre base mais permettre l'accès à vos données par l'intermédiaire des vues.

Les procédures stockées sont des mécanismes très intéressants lorsqu' on souhaite apporter également un niveau d'abstraction supplémentaire et des performances accrues. Car en effet les procédures stockées sont au cœur du SGBD et disposent de tous les moyens pour être exécuté le plus rapidement possible. Elle sont déjà pré-compilées, plan d'exécution est déjà préparé par le moteur de requêtes

Architecture -- Relationnel

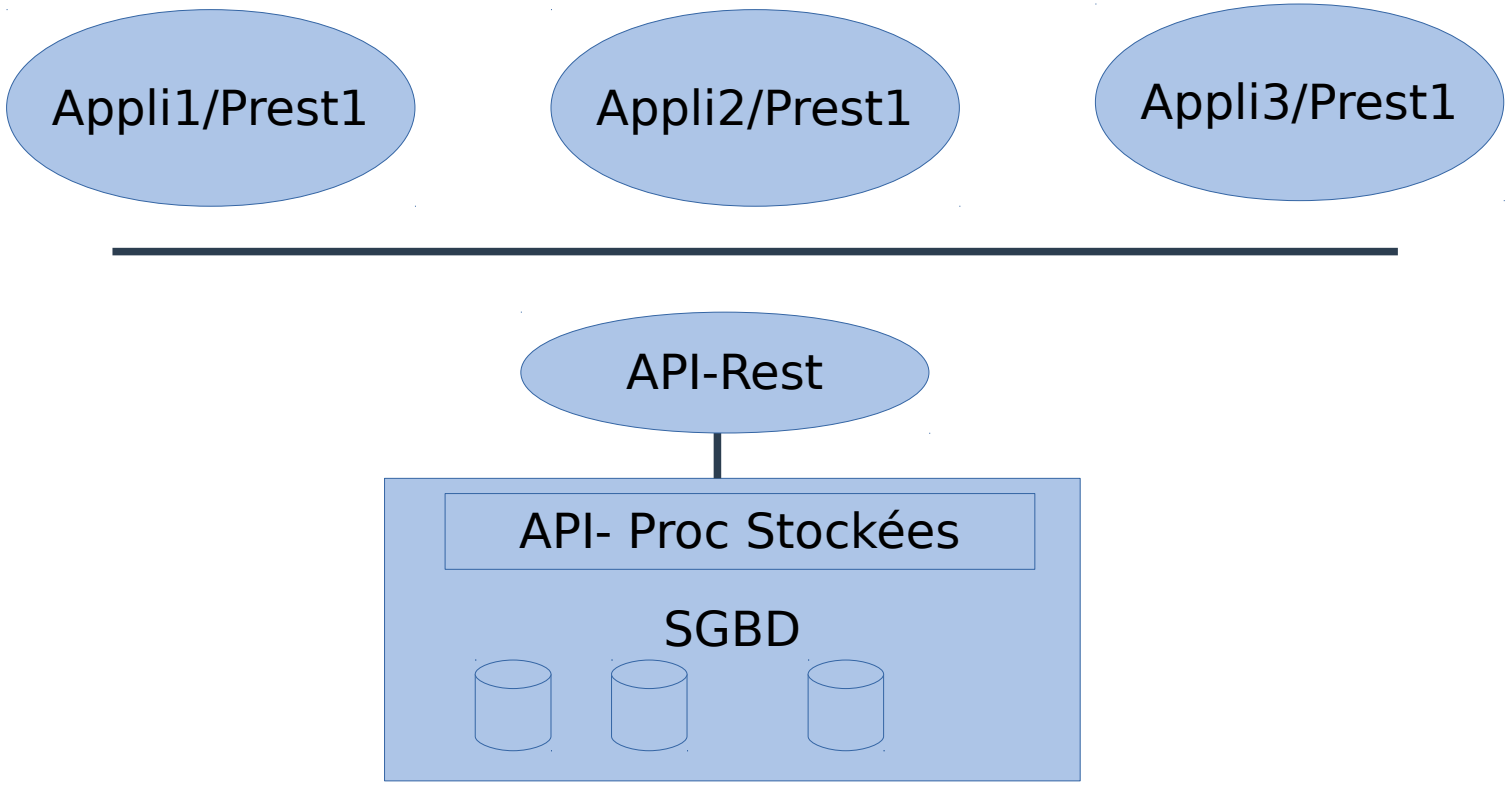
Du coup, on, pourrait pour des raisons de sécurité et de performance plutôt que de donner l'accès à tous (prestataires ...) à la base de données, même avec un contrôle de profil, de donner l'accès à une niveau d'abstraction qui pourrait être une composition de vue et de procédures stockées. ce qui vous permet de conserver et d'isoler le cœur de la base de données et de préserver une indépendance entre les différents niveaux applicatifs de votre architecture. Et vous permet de préserver une meilleur cohérence de votre système d'informations.

Architecture -- Relationnel



Architecture -- Relationnel

On peut pour des raisons techniques si besoin ajouter un niveau de Web service via de APi REST

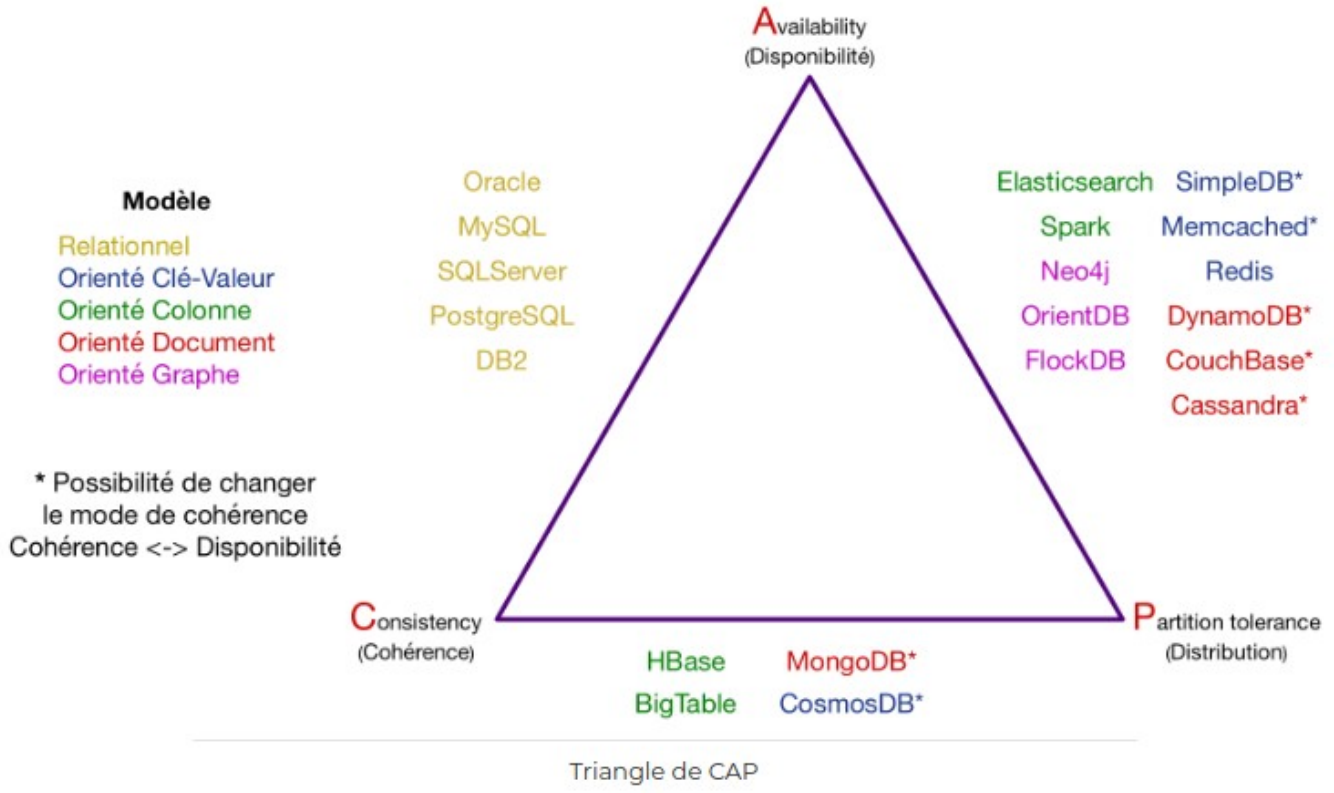


Base de données NoSQL

Point sur les solutions NoSQL disponibles dans nos environnements surtout lors de l'utilisation de cache du côté du navigateur à l'aide d'indexedDB. En effet, les solutions NOSQL amènent une approche et des fonctionnalités différentes. Ces solutions ont été bâties par les grosses sociétés du GAFA pour répondre à leur besoin de gestion de données très volumineuses. Les solutions Relationnelles étant arrivées à leur limite (problématique de réplication et de lourdeur associée au modèle lui-même). Des simplifications s'imposaient pour permettre aux différentes applications et à l'utilisation d'architecture distribuée de fonctionner et de permettre aux business de prospérer. On a changé de paradigme et on est passé du modèle relationnel (modèle Intègre, avec son lot de contraintes et ses mécanismes de contrôle) à un paradigme de CAP. En partant du principe qu'une base de données ne pouvait pas remplir toutes les caractéristiques de C : Consistancy cohérence , et A : Availability -- Disponibilité et de P :Partition Tolerance (distribution)

Base de données NoSQL

Ce qui nous donne le schéma suivant pour positionner les différentes solutions existantes :



Triangle de CAP

Base de données NoSQL

Les modèles de bases de données NoSQL foisonnent et nous permettent de monter des solutions très adaptés et flexible pour nos applications.

Virtualisation

De nombreuses solutions sont possibles et accessibles dans nos environnements de développement. Et quelques fois deviennent inévitables pour tester et déployer nos livrables sur les infrastructures en production. Sur nos postes de dévelopement, il est très habituel d'avoir installer un hyperviseur de type Hyper V sous Windows, VirtualBox ou VMware Workstation. Ces hyperviseurs nous permettent d'adapter une certaines démarches dans le cadre de développement sur nos différentes machines. Plutôt que d'installer sur vos postes infor, en fonction du cours ou du projet sur lequel vous travaillez les solutions applicatives directement en natif sur votre OS, ce qui aura comme effet au bout d'un certains temps suite à une gestion difficile des effets de bords plutôt désagréables. De type instabilité de l'OS , incompatibilité de différentes versions d'une même bibliothèque, d'une même application, des problèmes d'accès à des ressources identiques etc ..

Virtualisation

Donc la méthode la plus sage est de toujours essayer de préserver votre environnement de base ... et de venir ajouter des éléments regrouper et isole en fonction de la nature de l'activité et facilement gérable. D'où l'intérêt d'utiliser des machines virtuelles ? L'avantage que l'on a c'est de pouvoir associer des operating system les uns avec les autres. monter un Windows sur un Linux ou sur un Mac OS .... deuxième intérêt est de pouvoir installer dans cette VM tous les éléments nécessaires pour votre cours ou projet. De conserver une isolation entre les différentes installations effectuées dans les VMs. De pouvoir échanger entre collègues ces VMs avec toutes leurs dépendances et de n pas oublier une installation éventuelle et qui provoquerait des effets désagréable. De pouvoir après utilisation soit archiver la VM sur un média donné ou supprimer entièrement la VM. A l'aide des snapshots vous pouvez sauvegarder une image à un instant donné de votre VM et ainsi pouvoir la restaurer après des manipulations hasardeuses.

Virtualisation

De plus, les systèmes et les infrastructures de production disposent pour la plus part d'entre eux au moins de systèmes de virtualisation, les clouds utilisent bcp ces notions d'où l'intérêt de ce projeter dans configurations. Sachez juste que les Hyperviseurs sur les machines en production sont : VMWARE-ESX, Proxmox, KVM , HyperV ....

L'utilisation des VMs en production permet de garantir les isolations entre les différentes applicatifs et évite les problématiques de incompatibilité, deplus comme chaque VM fonctionne avec ses ressources, il y a un meilleur controle au niveau système Hôte et ainsi on évité les débordements d'une application sur les autres car chacune reste dans son environnement délimité.

Conteneurisation

Quelques mots sur la conteneurisation, est un moyen de simplifier et de disposer d'applicatif à partir d'une image. C'est docker qui a permis en définissant le contenu d'une image de distribuer et d'accélérer l'usage de ce type de solution(connu sous LXC). Beaucoup, plus léger que la VM, il devient un compagnon très proche du développeur, car il va lui permettre de pouvoir sans installation complexe bénéficier de la mise ne œuvre de service lui permettant de tester de développer des applicatifs plus facilement. Par exemple, lors de tests de plusieurs versions d'une même application sur la même machine sans installation sans avoir de conflit ... la distribution et le déploiement est rendu plus simple car le conteneur est déployé avec ses dépendances ... ce qui réduit les problèmes d'installation.

Conteneurisation

Quelques petits exemples d'utilisation de docker

On souhaite par exemple sur une machine démarrer un service apache et un service mongodb. Pour ce faire il vous suffit après avoir installation docker sur votre machine et démarre lé service exécutez les commandes suivantes :

#docker run -p 80:80 -d httpd

#docker run -p 27017:27017 -d mongo

Avecc ces deux commandes vois allez télécharger deux images à partir du dépôt DockerHub et démarrez deux applications sur votre propre machine.

Gulp // Nodejs

Gulp est un gestionnaire de tâches qui sera piloté à partir d'un fichier gulpfile.js. Les tâches que l'on peut automatiser à l'aide de cet outil sont :

- tâche de remise en forme de fichiers sources de type css
- de recharger l'ensemble des plugins de nodejs à partir du fichier package.json
- permet également de faire des copies de fichiers entre sources destinations pour préparer des livrables généralement sous dist.
- Fonctionne un peu comme Makefile avec différents points d'entrée et des commandes à exécuteur en fonction du point d'entrée sélectionné.

Gulp // Nodejs

ex : /home/ystroppa/PSB/M2/pwa-ecommerce-demo/pwa-ecommerce-demo/project

Les actions sont décrites en Javascript sous forme de fonction que l'on associé à la commande task.

Exemple de tâches gulp :

```
gulp.task('images', gulp.parallel(images, thirdPartyImages));

gulp.task('copy', copy);
```