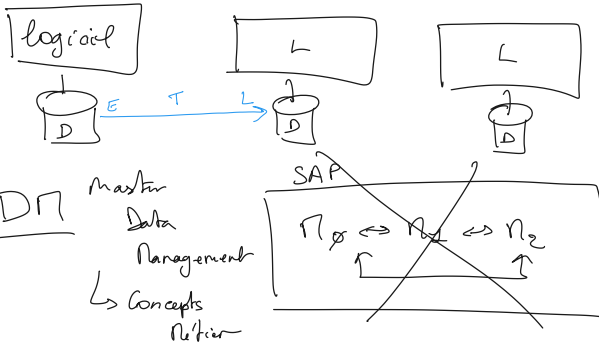
EAI

→ Voir Page 3

BUS → système de messagerie



MDM

Master Data Management

↳ Concepts Métier

→ Où se trouvent les données de référence ?

→ Les identifiants, ... ?

Flux = Temps réel, au fil de l'eau

Recommandé → Push / Pull → BUS, Broker

Batch = Par acoups, par lot

↳ Techniquement facile

Modèle de Message - Synchrone → "comme si je vous passe un coup de fil"

Asynchrone

↳ "je vous envoie un message"

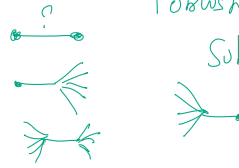
Synchrone
- informations de suite
- Transactionnel
- obligation de disponibilité
- besoin d'Accusé Réception (Transaction)

Transaction = Toute une opération doit avoir lieu intégralement ou pas du tout

Asynchrone → envoi de message
- Pas besoin de disponibilité
- Garantie de remise
- Transactionnel
- plus complexe à mettre en œuvre

Point à Point

Publisher / Subscriber



Message Queue

NoSQL Services
MongoDB

Apache MQ RabbitMQ

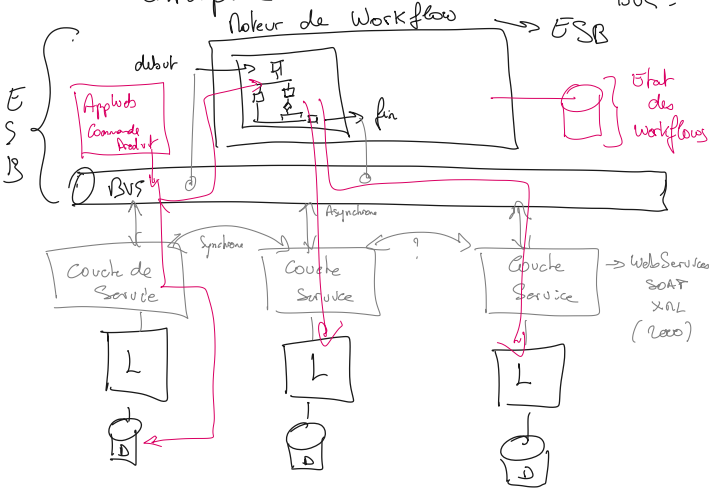
Cluster: Kafka

Compensation / Corrélation
=
Correspondance d'id dans des systèmes hétérogènes et récupération sur erreurs.

Architecture SOA, ESB

Service Oriented Architecture

Enterprise Service BUS => Juste le Bus.



Couche Service (SOA). Au niveau de l'entreprise on développe une surcouche commune pour parler le même langage (XML)

Evénement de bus = réception d'un message

Moteurs ESB:

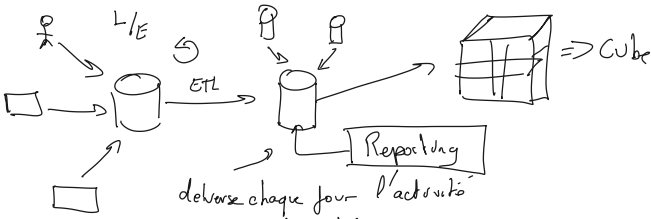
Language BPEL

→ implémentation dans ex WebSphere

→ Microsoft BizTalk

BDD Relationnelles

Scale-up → Monter en Puissance la machine



→ Cumul d'activité
→ Entrepôt
→ Volume plus important

BI → Analyse de valeur par dimensions

Cube = Aggrégats précalculés pour des requêtes instantanées

Data Mining → BI

Data Engineering → Big Data

Modèle Traditionnel Relationnel Cohérent BI

Cohérence ?

Modèle Big Data

3V - Volume
- Vitesse
- Variabilité (de schéma)

Métier du Data*

Cluster MA à l'échelle d'internet

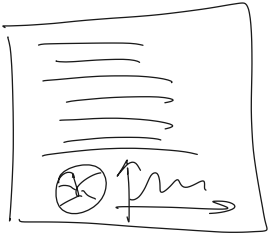
BI Traditionnelle : outils de Reporting

→ Crystal Reports
BO
Bi it
SSRS
Report Builder

→ BI Moderne
→ Power BI, Qlik, Tableau, Microstrategy

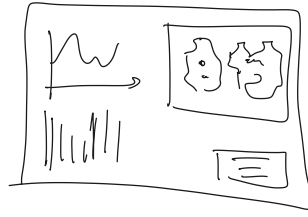
Reporting

→ Sur le détail (beaucoup de données)
→ Peu de graphiques (Résumés)

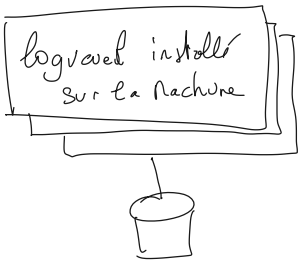


Analyse de Données

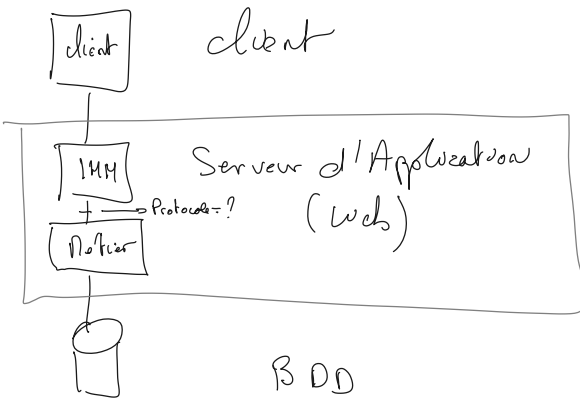
→ Sur les graphiques et les résumés
→ éventuellement du détail
→ Compatible avec le Big Data (3V)

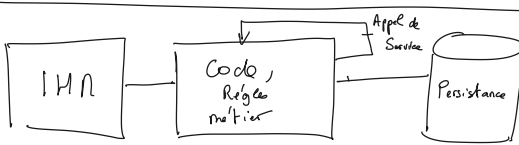


clients lourds



Application web

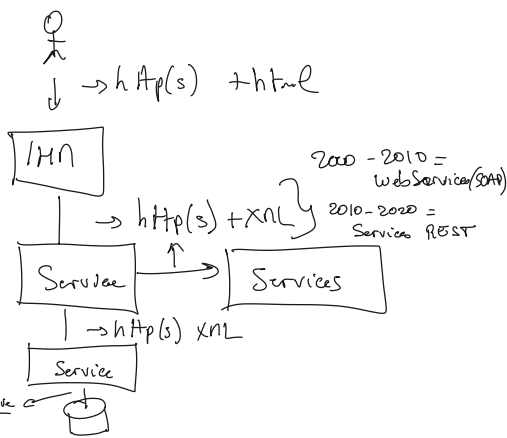




Communication entre les couches

- "Propriétaire" → avant
  Entre un client et un serveur dont il sera difficile d'utiliser d'autres Technologies -
  IBM CORBA
- "Open", basée sur des standards -
  exemple : - Texte XML
  - Protocole de comm: http(s)

Utilisateur → http + html (affichage)
machine(process) → http + XML (Contenu)



XML → Lourd

↳ JSON → natif, aussi puissant que le XML

début des Services REST → aussi dans le Temps

→ Harmonisation

Depuis > 2010 → Uniquement les Services REST

Representational State Transfer

http + JSON etc ...

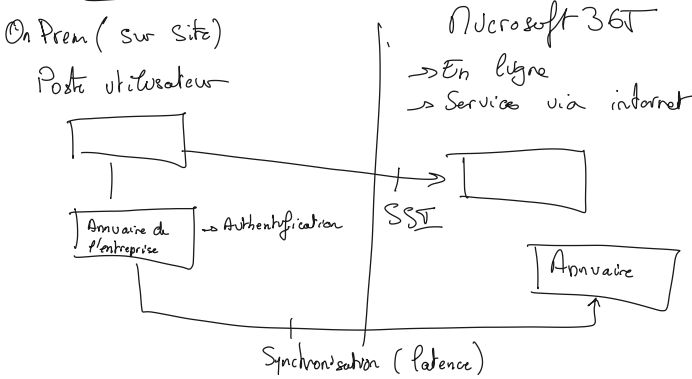
1/3 → Tier (Pas tiers)

Monolithique → Apps Web 3-tiers → Microservices

1980 - 2000	2000 - 2010+	2015+ (pour les très gros projets)

Microsoft 365

Déploiements Hybride



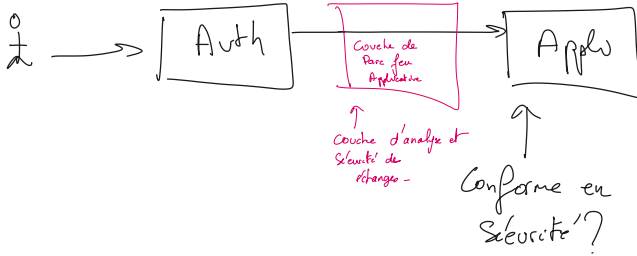
SSI = Same Sign In (même authentification)

SSO = Single Sign On (auth. unique (1 seule fois))

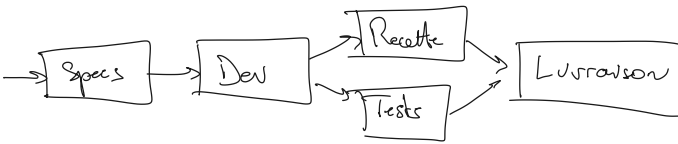
Actuellement = Authentification avec Protocole Standard Sur Internet

SAML — Open Auth

(WebSSO)



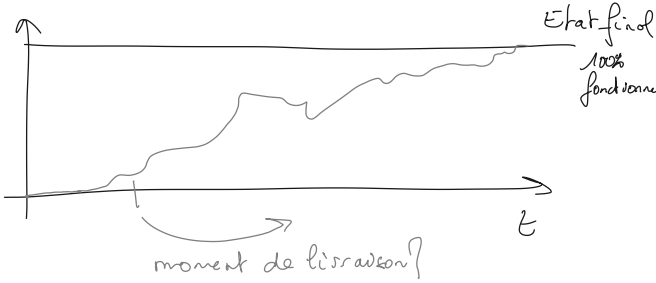
Test Driven Development



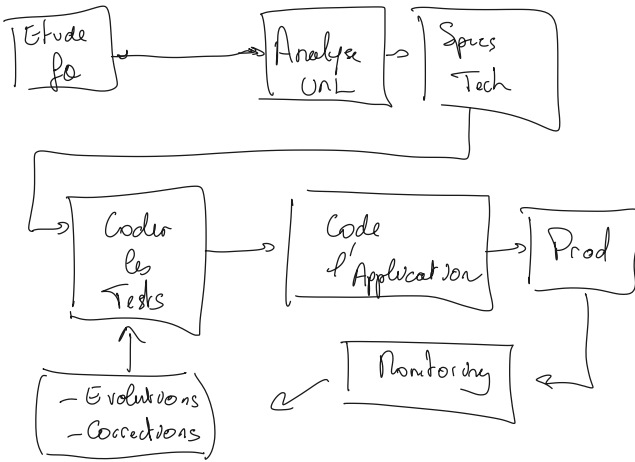
Tests
- IHM
Unitaire
- charge

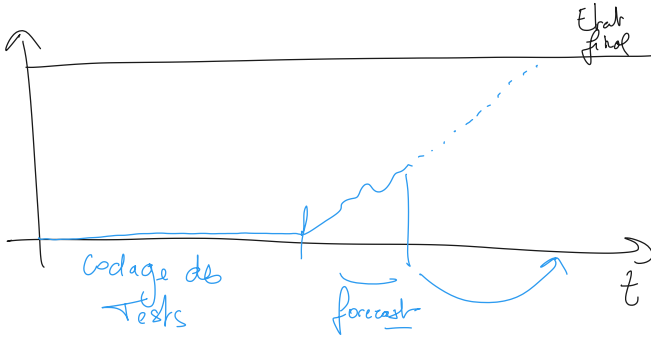
Intégration
fonctionnels -
Non régression

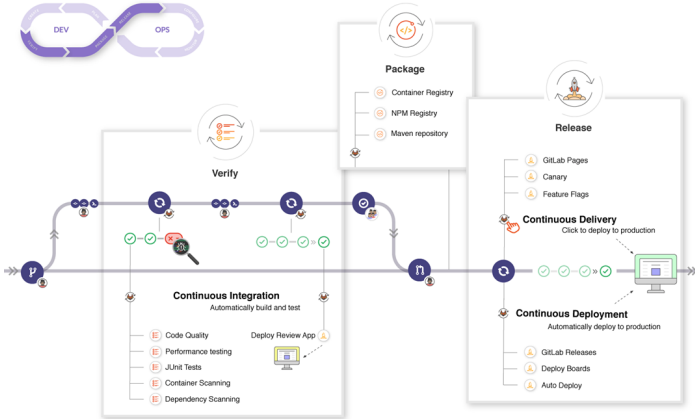
Tests Non Exhaustifs



Approche TDD :







Code Quality: - Règle d'entreprise
- Performance
La Référence = - Conformité
Sonar Qube - Risques de sécurité

Scan de dépendances: Recherche de dépendances avec des failles, erreurs ou risques

Registres: (Registry)

- Déf: Stockage d'éléments binaires -

Code Source → compilation → élément binaire (de la build)

Machine Virtuelle → Images/Templates → éléments binaires

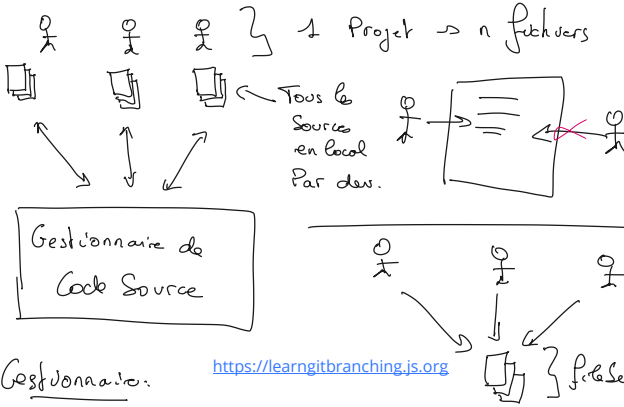
Images de Conteneurs → éléments binaires
Configurations (fichiers de), Clefs (certificats) → éléments binaires

- Registres d'images de VM
- " d'images de conteneurs
- " de builds (jar, paquages.rar, etc...)
- " de configs, et secrets

Gestions de Versions -

Svn (Subversion), TFS, Mercurial

Git



Gestionnaire:

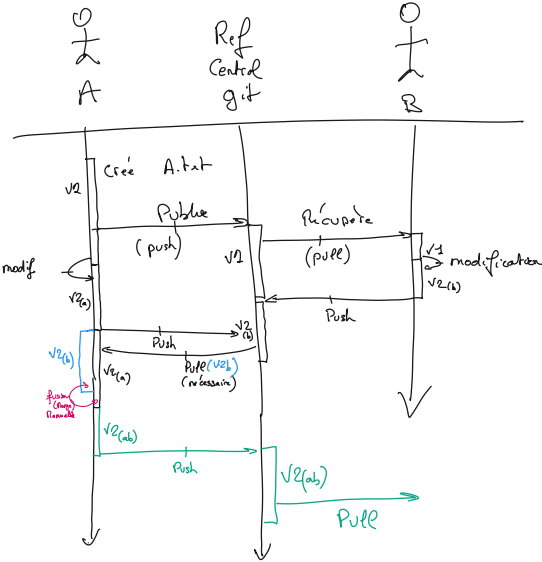
- Stockent les versions
- s'occupent du verrouillage et synchro/cohérence des code partagé

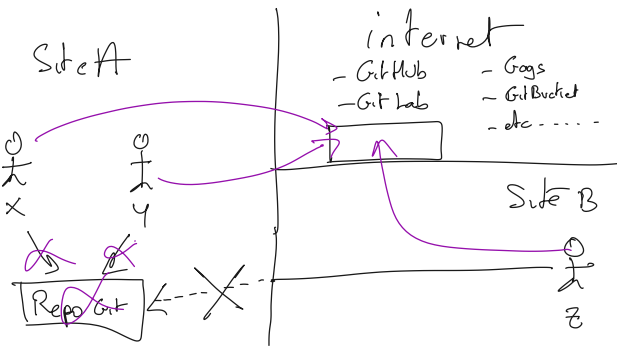
Avant:

Svn, Mercurial, TFS ⇒ Verrou Par fichier

Maintenant: Git (dev. par Linus Torvald)

- Simple dépôt central
- Synchro côté client


A
Ref
Central
git
B
Crée A.txt
V2
Publie
(push)
V2
Récupère
(pull)
V1
modification
V2(1)
modif
V2(a)
Push
V2(b)
V2(1)
Push
Pull(V2b)
(nécessaire)
V2(a)
Push
Pull
(Merge
Nécessaire)
V2(ab)
Push
V2(ab)
Pull


Site A
Internet
- GitHub
- Git Lab
- Gogs
- GitBucket
- etc.....
Site B
X
Y
Z
Repo Git

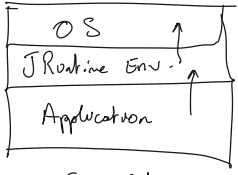
Langages / Plateformes

90 → C++ (C) → Dos, Windows, Linux
   → Cobol → Mini

90 → Delphi, C++, VB (A) 6 → faute "bricolage"
   Applis Commerciales

≥ 95 → Php (open Source)
     → Java (SUN)

Java Objectifs : inspiré du C++
- indépendant de la plateforme
- Simplification des concepts objet Mémoire.
- Plein de bibliothèques.



← Application "gérée"

Compilation

Dev → pseudo code → livre → Compilation à la volée au runtime

Dev — Prod

⇒ BCP Moins performant que le C++

90

Desktop → Delphi, C++, VB6 (⇒ depuis 98)

Web : Php

Java → client lourd ⇒ CATASTROPHE !
Web

2000 ↓ Sans

2002 → Microsoft .net v. 1.0
→ plateforme, framework.

Objectif Concurrencer Java

Java 1, 1.1 ...

Java 2 → 1.4

Java 5 → 1.5

6
7
8
11
19
} année de sortie

Actuellement
- JEE, .net 5 } 70% des dev
- Php } 20%

Reste Django, Go, Node (API) ....

Types de dev

- Apps et APIs web → 80%
- Clients lourd
  - Windows → .net
  - linux → .net Xamarin
    Qt (c++)
  - Android, iOS → - Solutions prop
    - Qt
    - Xamarin (c#)
- "charge de calcul" (Analyse de données, IA, ...)
  → environnement Spark :
    - Scala (Jupyter notebook)
    - Python
- Début du No/Low Code / Serverless
  → juste un navigateur pour dev.
  → On ne cible plus l'architecture, mais des URLs.

- Mini Systèmes : RPG, GAP, Cobol, --
- Plateforme Big Compute
  Hadoop
  - Pig, Hive, Java, MapReduce

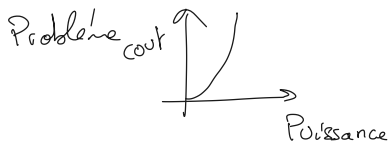
Big Data → Trop de données

Big Compute → Trop de calcul

Calcul → 1 Machine

HPC → High Performance Computing
→ "Super Calculateurs"

Grid Computing → Des ordinateurs en grille



~ 2000 Google → Recherche → résultats publiés.

"Big Table" "Big Compute"

"Map Reduce" → Paradigme pour distribuer un calcul sur plusieurs machines.

2004 à 2006, Doug Cutting développe Hadoop.

Hadoop → Java

Meta langage qui produisent du Java

Python → Java

100 lignes de code → 1 calcul

Hadoop

- Java : 100 lignes
- Python : 15 lignes.
- Scala : 3 lignes

Hadoop ← calcul distribué

Calcul
- → ok sur 1 machine
- → besoin d'un cluster

Spark (Apache)

- Tourne en autonome
- Tourne sur un cluster Hadoop

Compétences de Développeurs

Générations de Langages :

1ère Génération (≈ années 80) : Programmation naturelle : basic, script .... : Spaghetti

2nde Génération (80) Prog structurée :

C, pascal Prog modulaire → globale (var fonctions)

3ème Génération Prog Orientée objet

Semi objet : C++ (structuré et/ou objet)
Delphi, Php

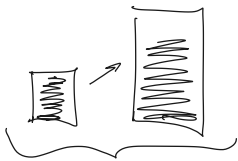
Pur objet : Java, .net

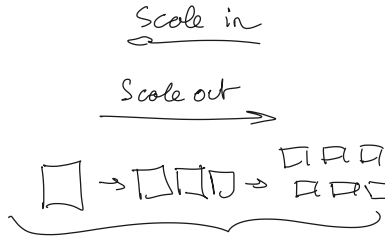
4ème Génération : Programmation fonctionnelle

→ Passer des fonctions à des fonctions -

Langages fonctionnels

F# (.net)
Scala
} Simplifie l'écriture





Analyse de données → Vous

DataBricks

Fournisseur cloud de cluster Hadoop

Community → gratuit → découverte
→ Commander un abonnement
→ Code dans le web (Serverless)

R

Langage Par et pour les scientifiques –

→ Simple, au niveau scripting –

Jargon est spécifique à la science :

Table → (objet ?)
Ligne → observation
Colonne → variable

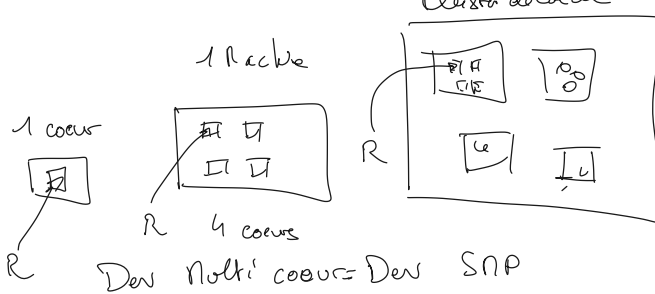
} à vérifier

jargon de R

→ "On a de tout et de n'importe quoi"

→ Limites techniques → Mono thread à la base

1 thread = 1 coeur de processeur



Des multi coeur = Des SMP

R n'est donc pas conçu pour le Big Data / Big Compute mais la Richesse de algorithmes -

---

Microsoft a développé un moteur R maison Compatible SMP haute performance

Deployable sur Azure

Avec bibliothèque Microsoft hautement optimisées -

RShiny -> environnement de développement

RStudio

Data / Big Data

Régle de 3V : Vitesse
Volume
Variabilité

Variabilité → Modifications de schéma, type, structure des données

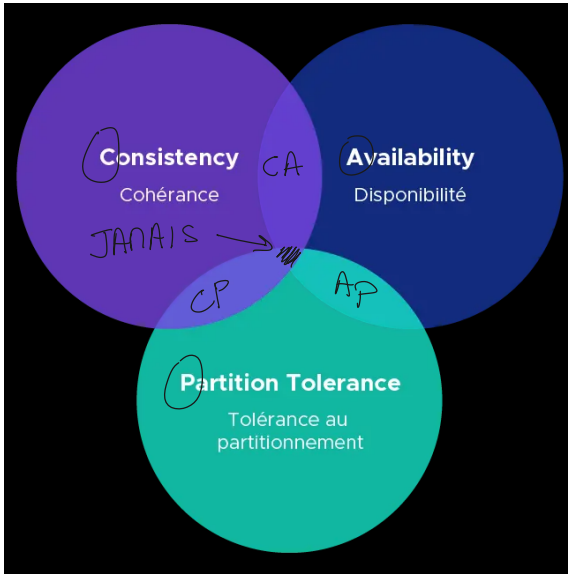
Cohérence

Relationnel

Big Data

Cohérence - 1 donnée validée existera toujours en tant que tel en tout endroit

THÉORÈME DE CAP



JAMAIS CA et P en même temps

Cohérence

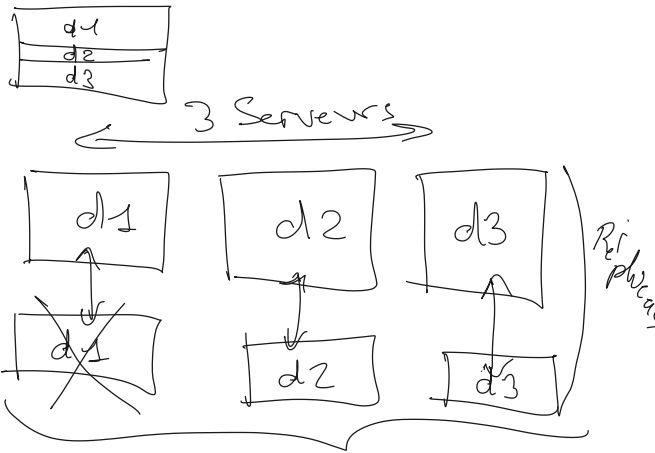
Disponibilité → Système qui est TOUJOURS fonctionnel (pas forcément performant)

Partitionnement → Système éclaté pour répartir la charge de travail

→ Performance au détriment de la Disponibilité

Availability
et
Partitionning ensemble :

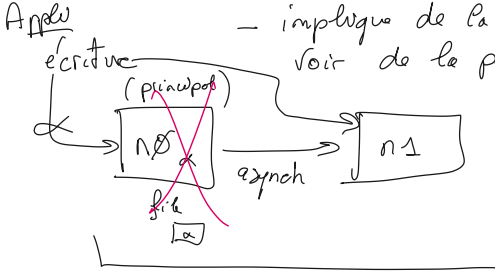
Soit 3 lignes d1, d2, d3



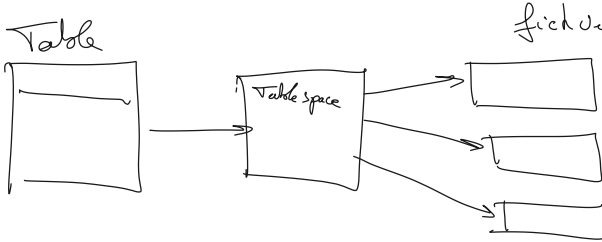
A + P

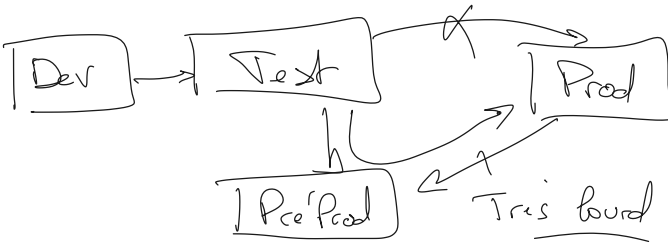
La Réplication

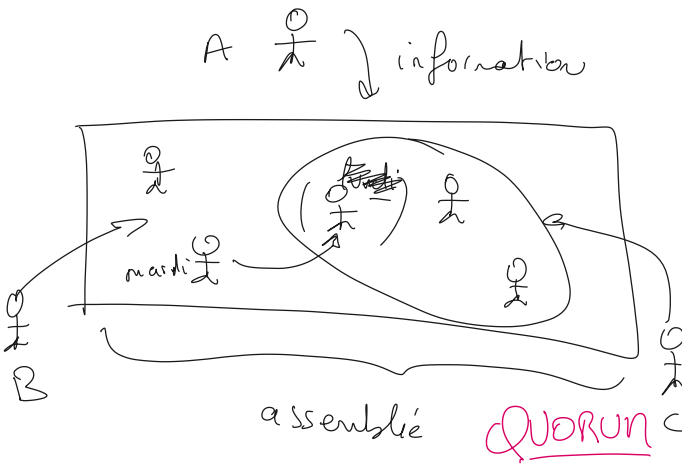
- Synchrone → - garantie d'état exact de la donnée partout
  - implique que toutes les machines soient disponibles -
- Asynchrone → des files d'attentes pour les mises à jour des autres serveurs.
  - implique de la latence, voir de la perte de données.



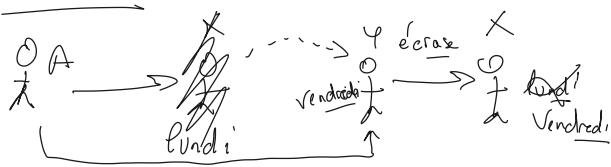
ORACLE Table Space

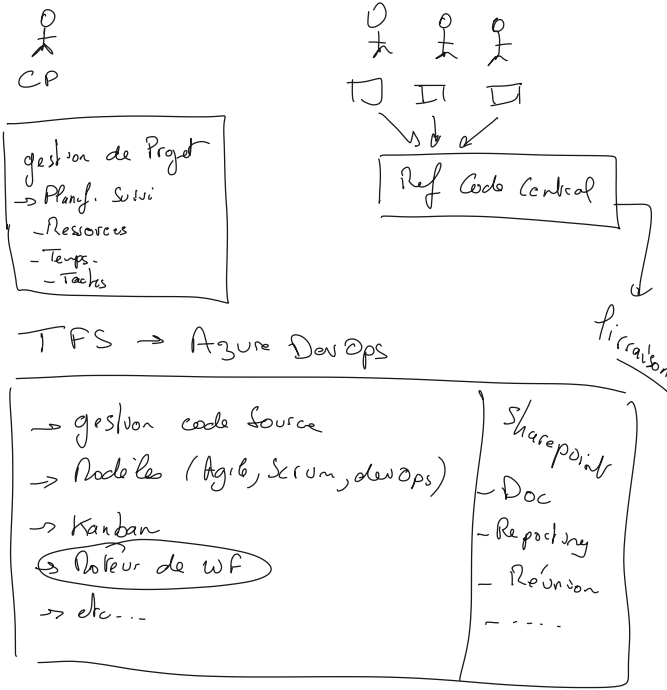


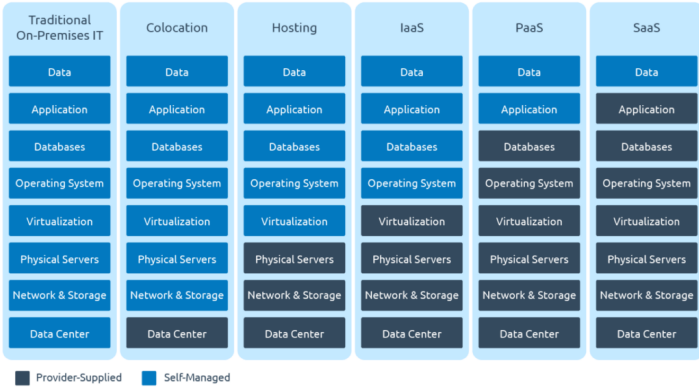




Protocole GOSSIP




CP
gestion de Projet
→ Planif. Suivi
- Ressources
- Temps.
- Tâches
Ref Code Central
TFS → Azure DevOps
→ gestion code source
→ Modèles (Agile, Scrum, devOps)
→ Kanban
→ Moteur de WF
→ etc...
Sharepoint
- Doc
- Repository
- Réunion
- ....
Jenkins → le Moteur de WF opensource le + réputé



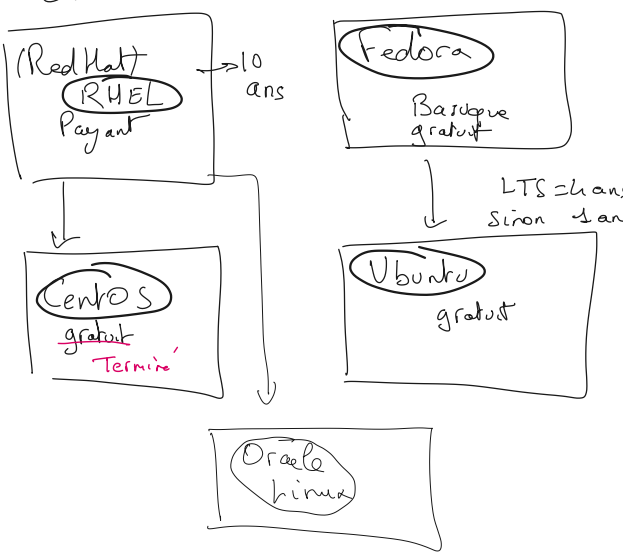
Red Hat Rancher → « plateforme » → empilement de logiciels.

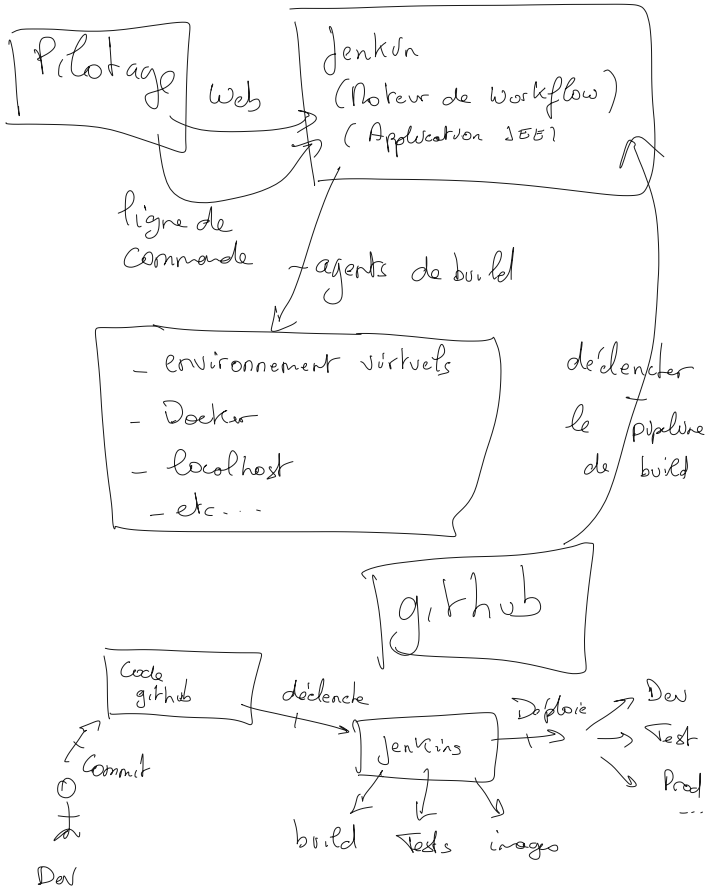
= OS + logiciels configurés –

VM + Softs + confs –

AWS + Azure + GCP ne fournissent pas nativement mais il y a de la demande → le déploiement y est possible

Directement chez Red Hat


OS
RedHat
RHEL
Payant
10 ans
Fedora
Basique gratuit
LTS = 4 ans
Sinon 1 an
CentOS
gratuit
Terminé
Ubuntu
gratuit
Oracle Linux


Pilotage
Web
Jenkins
(Moteur de workflow)
(Application JEE)
ligne de commande
agents de build
- environnement virtuels
- Docker
- localhost
- etc...
déclencher le pipeline de build
github
Code github
déclenche
Jenkins
Déploie
Dev
Test
Prod
Commit
Dev
build
Tests
image

Mercredi :

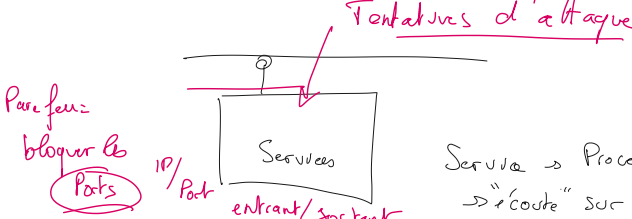
Big Data , NoSql

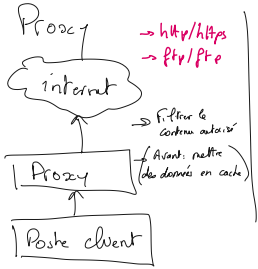
Architecture Microservices + Tuto (inclus Conteneurs)

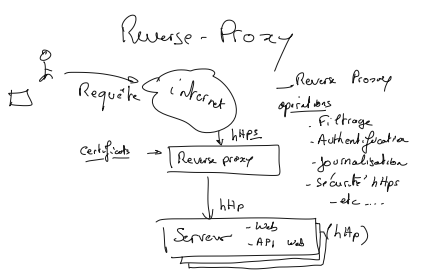
PKI & Crypto

① Firewall + WAF + Proxy + DMZ + API Managers

Firewall pare feu





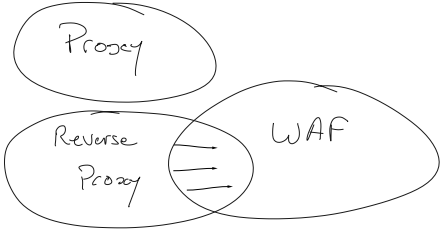


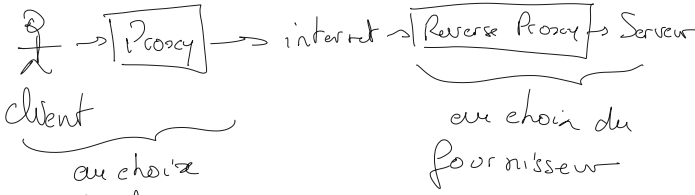
Avant : Tout ce réseau était Matériel

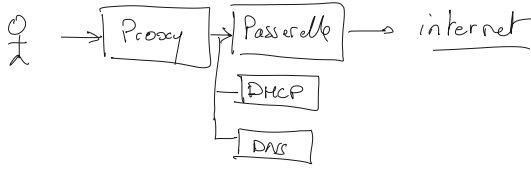
→ routeur - proxy
→ firewall

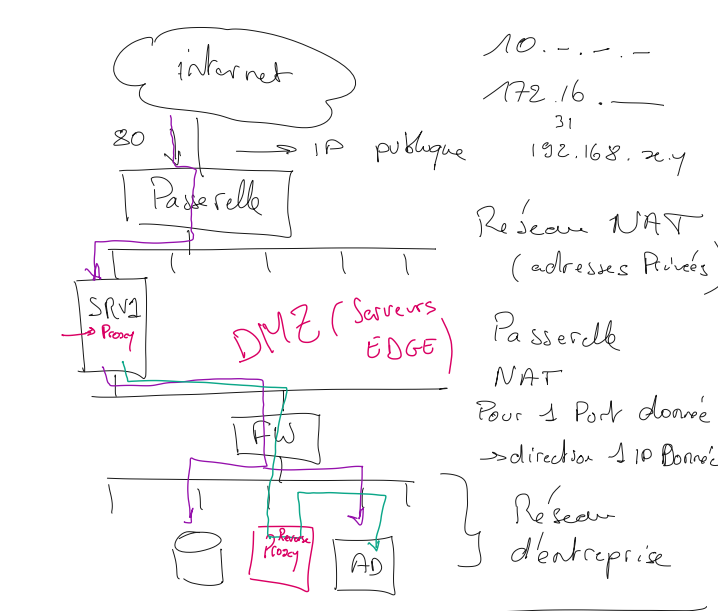
Maintenant : Software Defined Networking

→ Tous les composants sont logiciels

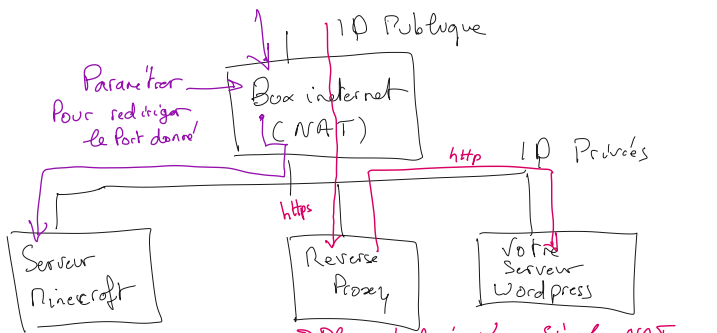


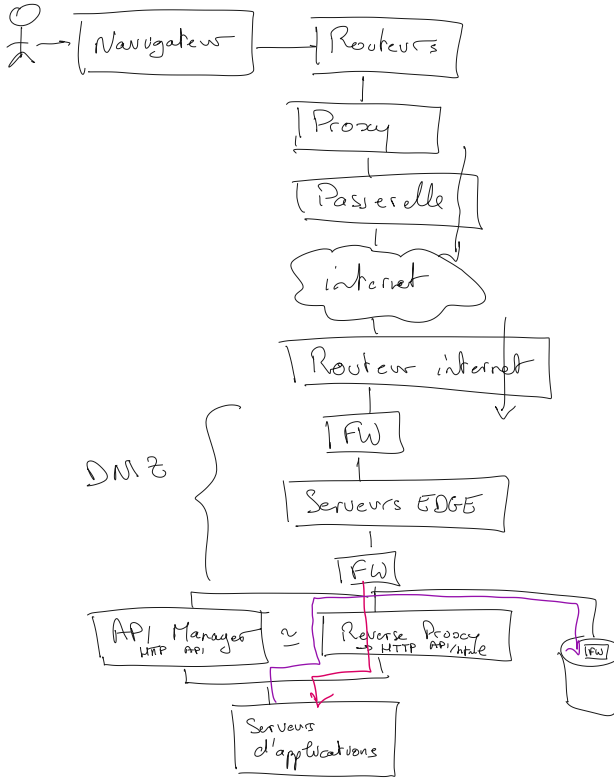






NAT / Reverse Proxy




Navigateur
Routeurs
Proxy
Passerelle
internet
Routeur internet
FW
DMZ
Serveurs EDGE
FW
API Manager
HTTP API
≃
Reverse Proxy
HTTP API/stale
FW
Serveurs d'applications

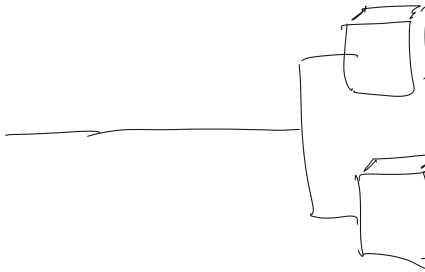
Firewall → analyse bas niveau souvent du hardware

Reverse Proxy → analyse à plus haut niveau le contenu des échanges pour consolider

objectif = pas la sécurité "assez simple" Peu de logique

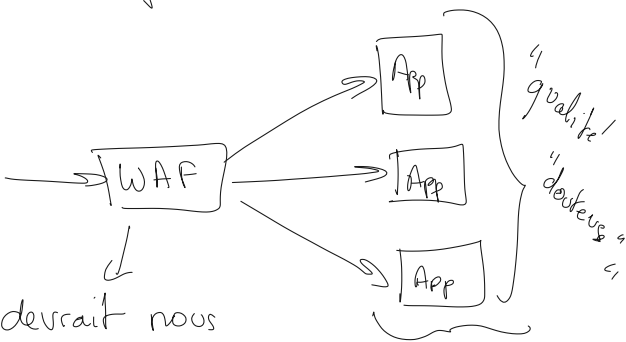
Web Application Firewall

→ Solution logicielle spécifique
→ hautement sécurisée
→ Avec Intelligence artificielle
→ Peu détecter bien des tentatives d'attaques,
→ intégre l'équivalent d'un Reverse Proxy



Physique (mag d'OS)

→ Applicatifs
- → bugs
- → failles de dev.



devrait nous
Protéger des attaques

devraient
être sécurisées
→ jamais à 100%

- mot de passe → "phrase" → caractères employés ?
  → Alphanumérique
  a-z A-Z 0-9
  &*#, >< ....
  ≃ 100 caractères sur 256 bit %1

- Secrets = équivalent aux mots de passe

- Mot de passe sous forme binaire
  exploitant n bits 64 bit, 128, 256, 2048 bits ....
  en octets
  ↳ besoin de le stocker → Dans un fichier → Certificats } généré par une bibliothèque (souvent: openssl)

Algorithmes

Hashage

Transformation irréversible

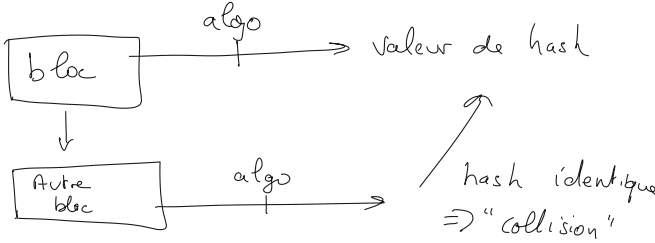
valeur → hash

utile pour vérifier un contenu

Algos → MD5, SHA 1, 2

Cryptage / chiffrement

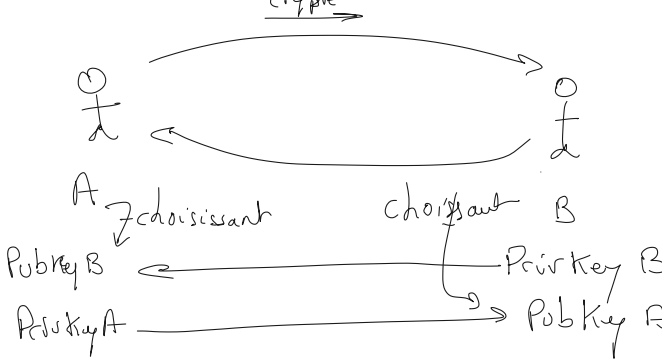
- Symétrique = même secret pour crypter / décrypter
  ex: 3DES (algo)
- Asymétrique
  → 1 mdp pour crypter
  → 1 mdp pour décrypter
  ex: RSA (algo)

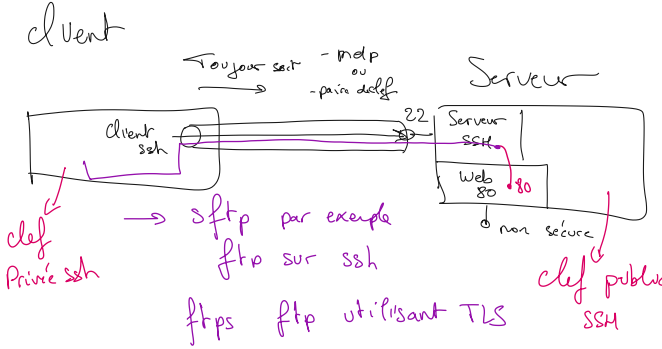


SHA1 → obsolète

$SHA2_{|3}$ ⇒ à employer

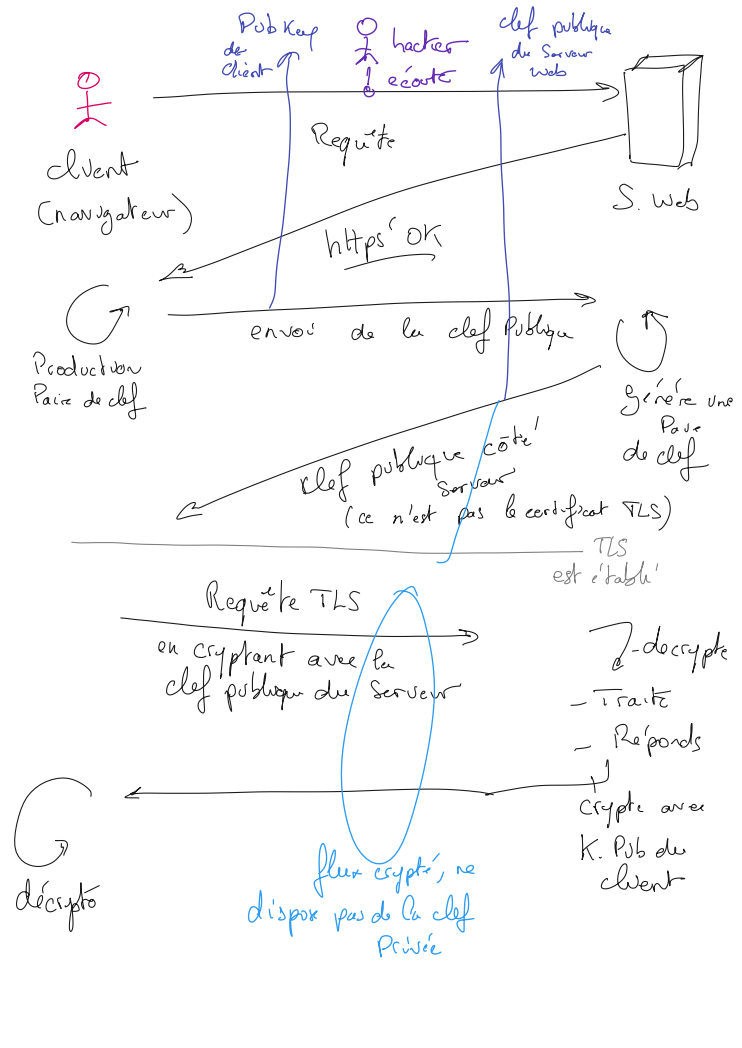
Mail chiffrés





Une liste des ports standards : https://fr.wikipedia.org/wiki/Liste_de_ports_logiciels

RFC = la bible d'internet : https://www.ietf.org/rfc/


Pub Key de client
hacker écoute
clef publique du Serveur Web
client (navigateur)
Requête
S. Web
https OK
envoi de la clef Publique
Production Paire de clef
Génère une Paire de clef
clef publique côté Serveur (ce n'est pas le certificat TLS)
TLS est établi
Requête TLS
en cryptant avec la clef publique du Serveur
- decrypte
- Traite
- Réponds
Crypte avec K. Pub du client
décrypte
flux crypté, ne dispose pas de la clef Privée

Big Data

→ Systèmes Non relationnels

→ Dénormalisation est la bonne pratiques
les pratiques relationnelles, si possible, sont néfaste (performance)

→ Base "No Sql" car ne sont pas capable de travailler en SQL: non implémenté
table
cas particulier: Cassandra (langage CQL)
Serveur de BDD NoSql le + proche des bases Sql

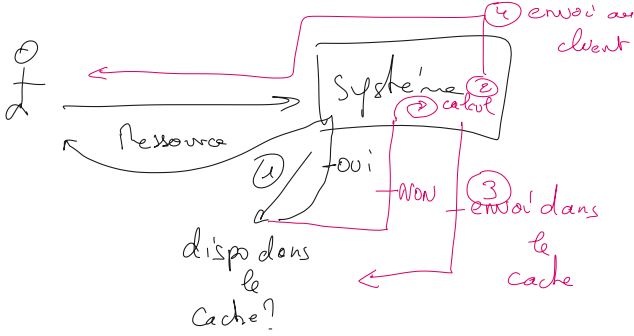
→ Orienté Haute Disponibilité
au détriment de la cohérence

→ Solutions très souvent Open Source

→ Organisés par Catégories.
- certains serveur peuvent avoir plusieurs de ces catégories.

(1) Type clef-valeur → le + Simple
SGBD orienté haute perf pour stocker/
Récupérer une valeur (de tout type)
→ Utilisé souvent pour la mise en cache
→ Ex: REDIS

Mise en cache
a une clef on affecte une valeur et date de péremption



---

(2) Type Column family

ex: Cassandra

Stockage de Type Table avec colonnes pouvant apparaître dynamiquement (schéma souple)

ligne 1 : id, nom de produit, prix

ligne 2 : id, nom prod, prix, commentaires

---

(3) Types Document

identique au clef-valeur mais le Serveur est capable d'exploiter le Contenu du Doc

Ex: MongoDb Doc : format JSON

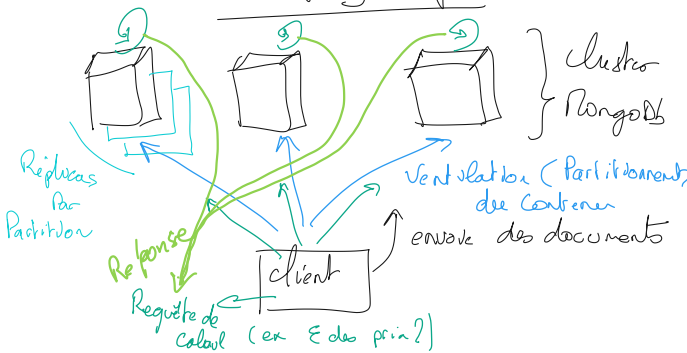
Ex: doc 1 = Nom de Produit, Prix

doc 2 = Nom, Prix, commentaire

Requête = Σ des Prix de Produits

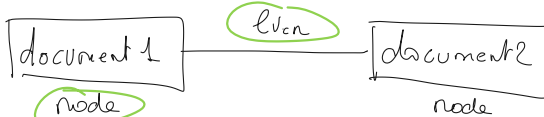
MongoDB : Big Data

Big Compute



---

(4) Type Graph Ex: Neo 4J



données + métadonnées
ex Contenu d'un post FB + date, public ?, etc...

Base de Type Graph sont conçues pour analyser les relations -

→ Plein de scénario
- Qui a fait quoi
- gamme de Temps
- Tendances

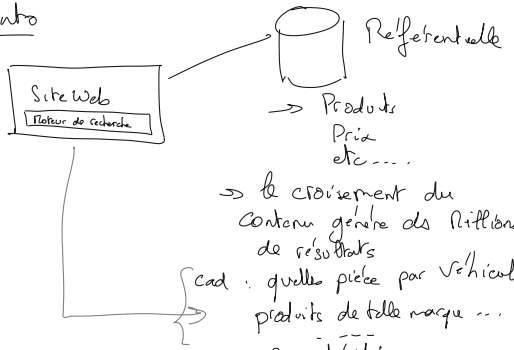
- Cousins/ enfants

(5) Types Sémantiques

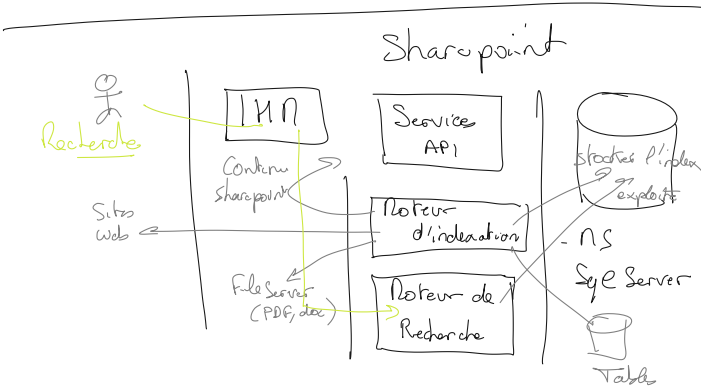
Objectif : analyser le contenu Texte pour des recherches puissantes et performantes.

Ex : Elastic Search

ex : Noranto



basé sur le moteur Lucene



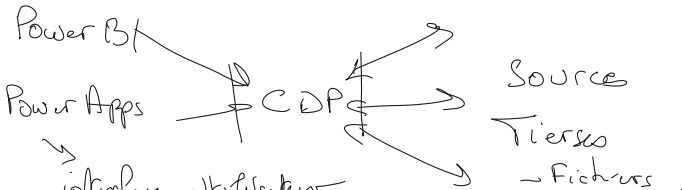
Data Verse

≈ 2005 MS rachète Dynamics (CRM)
→ CRM avantage = scène souple derrière BDD type SQL Server
c'est-à-dire → créer des entités, champs dynamiquement (customization)

évolue

2011, 2013 .... Dynamics 365
→ intégré à Microsoft 365
↙ OnPrem ↘ cloud

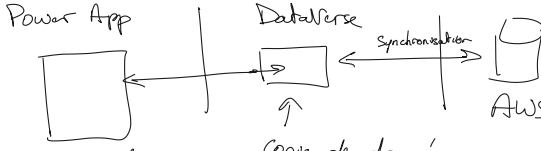
Moteur Souple de Stockage Relationnel est Devenu CDP (Common Data Platform Services)

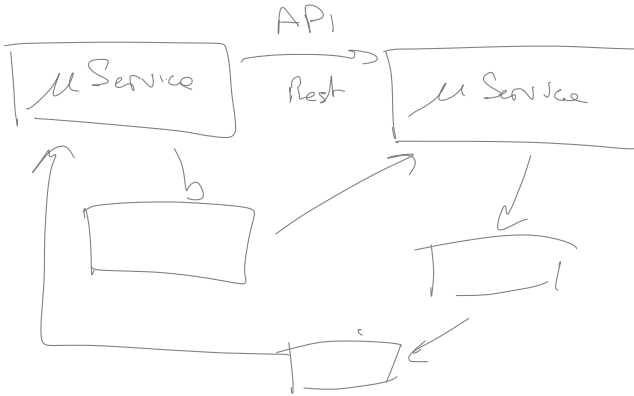


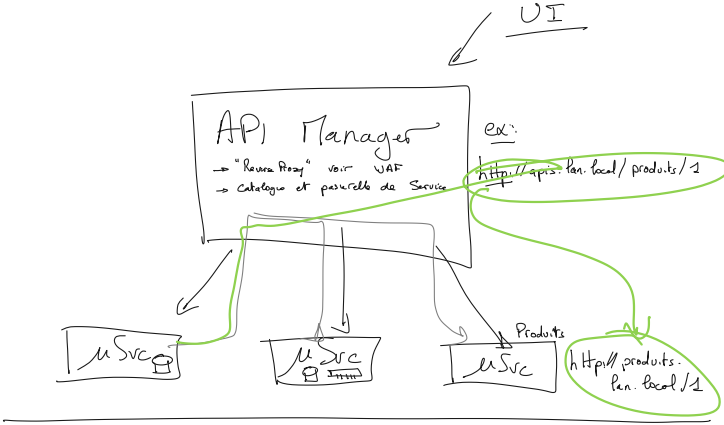
en 2021 CDP Renommé en Data Verse

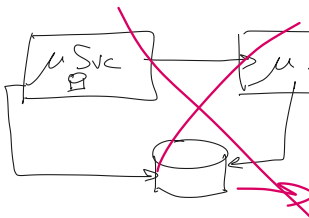
Ex: Vous avez une BDD AWS DynamoDB

- On créé un Référentiel "DataVerse"







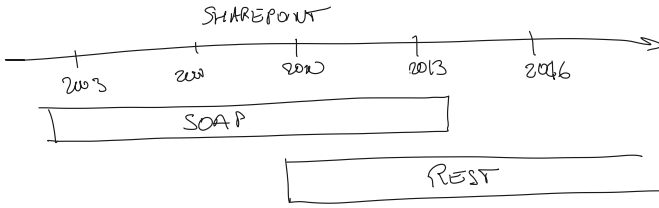


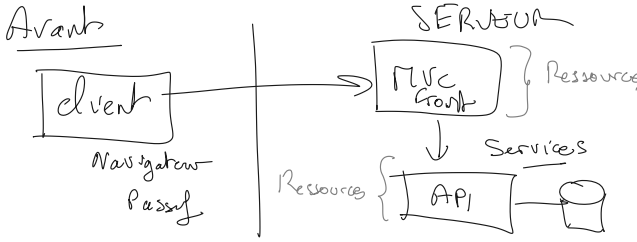
Point Central = Maillon faible = non conforme

Objectifs.

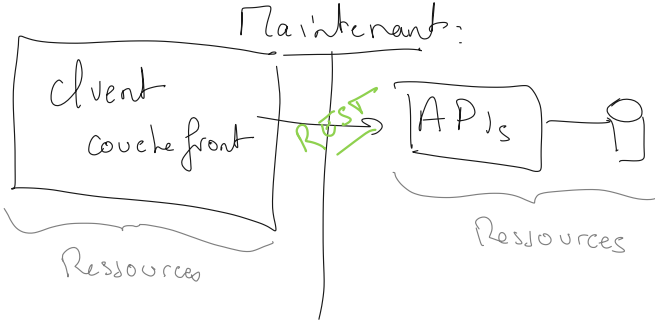
① → HA

② → Réactivité (DevOps)

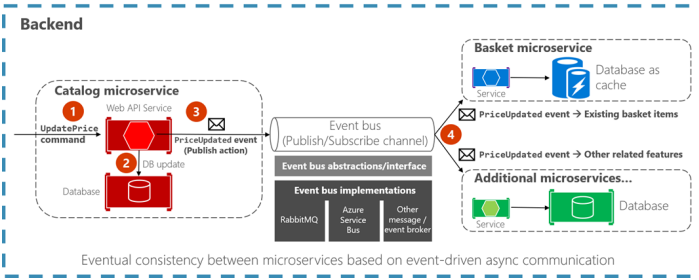




AngularJS → MVC côté navigateur

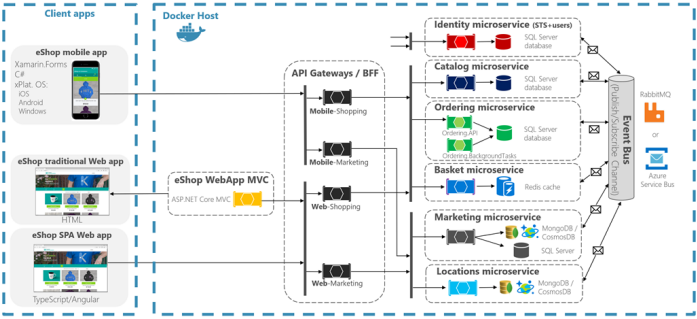


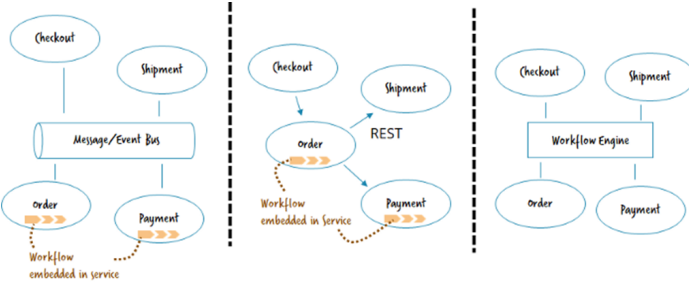
Implementing asynchronous event-driven communication with an event bus



eShopOnContainers reference application

(Development environment architecture)



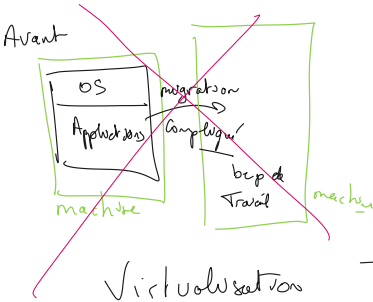


Virtualisation / Conteneurisation

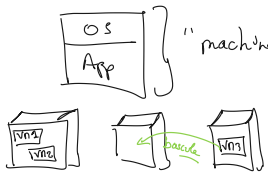
Docker

Kubernetes

Avant



Virtualisation → VMWare ESX ou Hyper V KVM



→ 1 VM tourne sur 1 SRV

→ 1 SRV = 0..n VM

Inconvénients :

OS → Consomme beaucoup de RAM

Beaucoup d'IOps Disque (lecture/Ecritures)

ex : 1 Serveur d'App consomme 1Go RAM + ½ 500Go pour l'OS

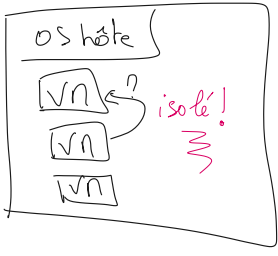
⇒ Pas Efficace

Solution :

N'isoler que les Process / Filesystème / Réseau

dans un seul OS

Virtualisation

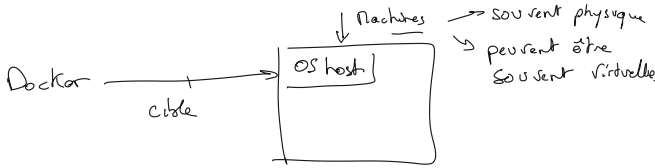


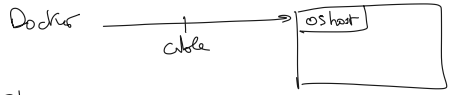
Solutions de Conteneurisation

→ implémenter au niveau OS (noyau) → "namespaces" sous Linux

→ inexploitable entant que tel

⇒ DOCKER est la solution logicielle la + répandue pour l'exploiter





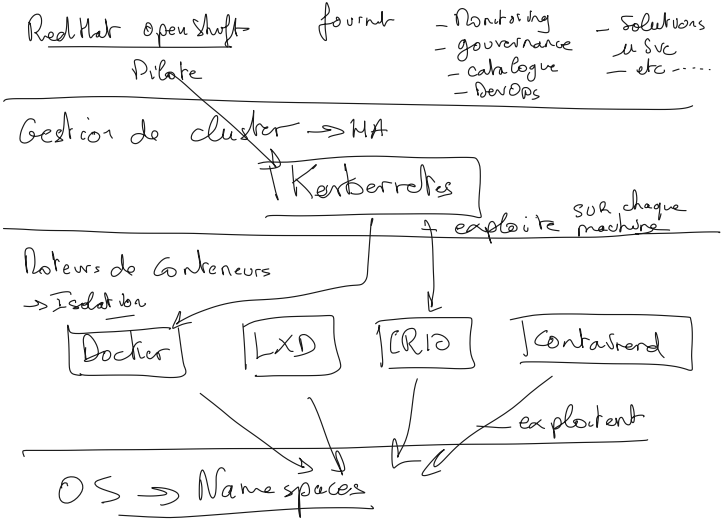
Pbm host qui tombe
besoin de bascule vers un autre host
Docker → trop limité

DOCKER

Community → gratuit
↙
docker CE

Enterprise → payant
↘
Docker EE
↳ Cluster de serveurs
→ HA

Google → leur orchestrateur de conteneurs → Kubernetes → gratuit → HA


RedHat openShift
fournit
- Monitoring
- gouvernance
- cataloguer
- DevOps
- Solutions
u SVC
- etc .....
Pilote
Gestion de cluster -> HA
Kerberetes
exploite sur chaque machine
Moteurs de Conteneurs
-> Isolation
Docker
LXD
CRIO
Containerd
exploitent
OS -> Name spaces