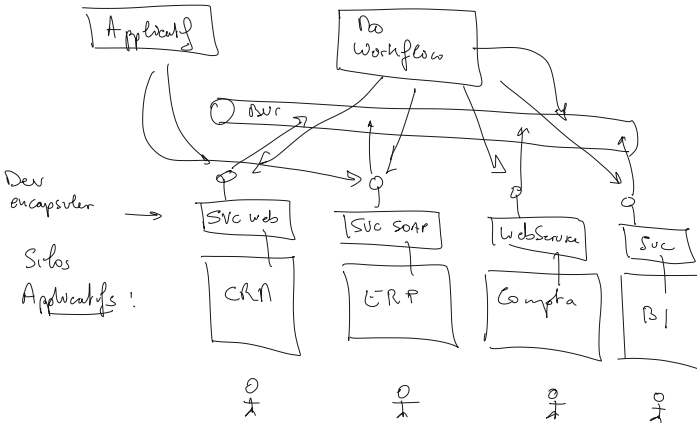


Communication		Avantages	Inconvenients
	Synchrone	Rapide Direct +Simple	indisponibilité
	Asynchrone (Message Driven Development)	+ Dispo	Petite latence (ms - secondes) + Archi, outils + Complexe

Agile

Scrum

DevOps

→ TDD → Test Driven Development

① Analyse, Specs fonctionnelles — BUG

"naturellement"

② Implémentation

③ Coder des Tests (basés sur les specs) → Résultats QA = Noyau

④ Validation + MEP.

① Coder la génération du bug dans les tests —

2' Coder les Tests basés sur les specs —

TDD → BDD Behavior

③ implémente

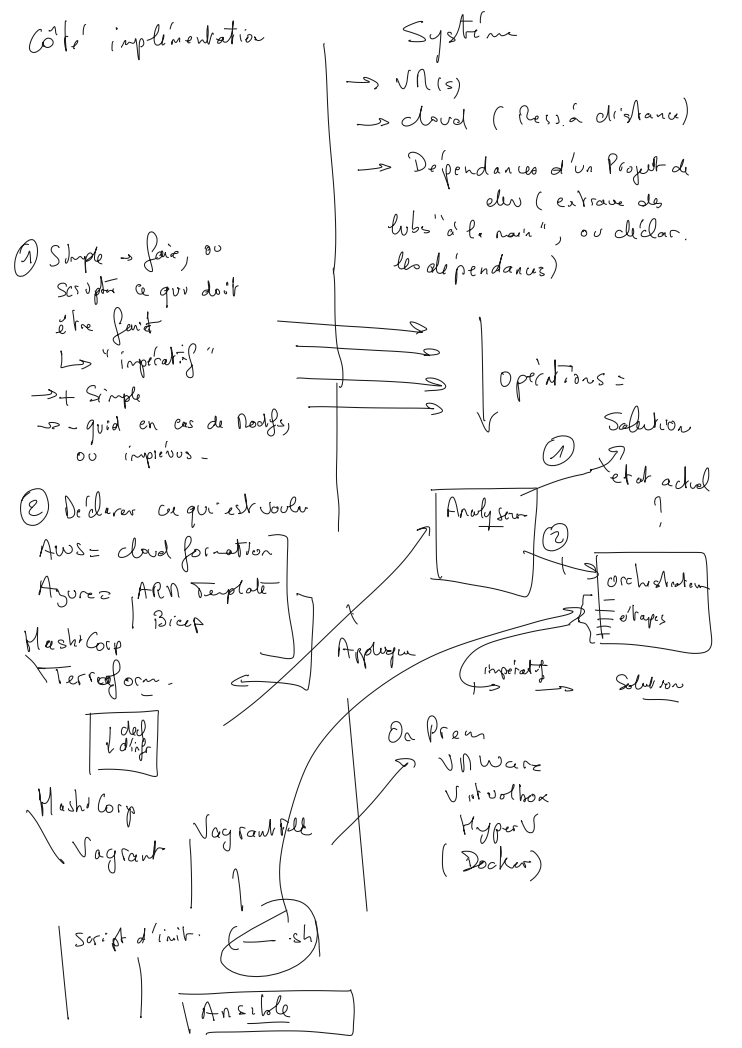
② Corrige le bug —

Dev

→ Tests → xUnit → jUnit → xUnit (BDD)

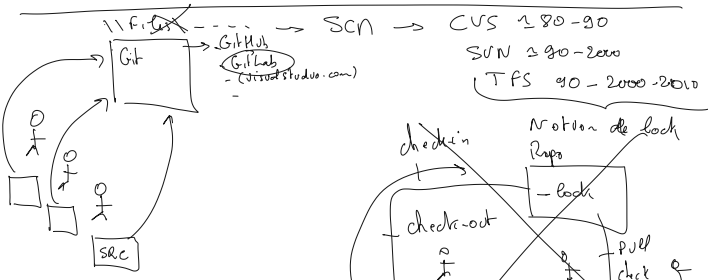
→ PhpUnit

→ nUnit.


Côté implémentation
Système
→ VM(s)
→ cloud (Ress. à distance)
→ Dépendances d'un Projet de
elles (extraire des
libs "à la main", ou déclar.
les dépendances)
(1) Simple → faire, ou
Script ce qui doit
être fait
↳ "impératif"
→ + Simple
→ - guid en cas de modifs,
ou impérieux -
Opérations =
Solution
(1)
Analyseur
état actuel
?
(2)
orchestrateur
étapes
(2) Déclarer ce qui est voulu
AWS = cloud formation
Azure = ARM Template
Bicep
HashiCorp
Terraform
Appliquer
impératif
Solution
état désiré
On Prem
VMWare
Virtualbox
HyperV
(Docker)
HashiCorp
Vagrant
Vagrantfile
Script d'init.
.sh
Ansible

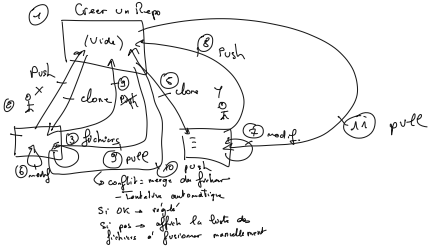
Machine de stage :
http://nbstagevm2.francecentral.cloudapp.azure.com/guacamole
admin/pass :
trainer/guacadmin
student/Pa$$w0rd

Olivier : st01
JP : St02



Git → Pas de lock

- Éducation de comm dans le groupe - 2 dév ne travaillerons pas dans le même code



Dev Java

- besoin d'un Java Development Kit → éditeur (open) oracle ibm .....
  → Version 8 11 12 15 17 18 ...
- Environnement de Dev
  → (ligne de Commande) javac ....
  → Sun NetBeans → apache
  → Oracle
  → IBM Eclipse 2000 ≈ 2008
  → idea IntelliJ
  → VS Code → beaucoup extensible non lié ....

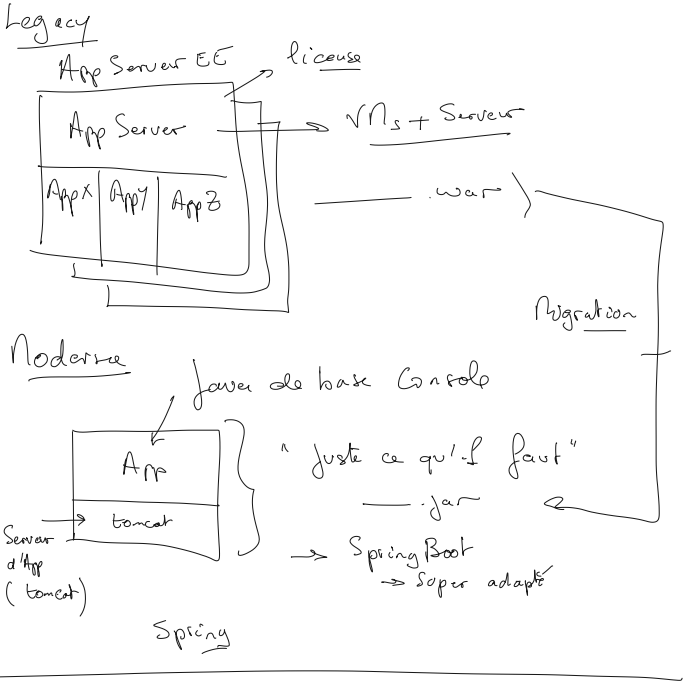
Projet

- SRC
- paramètres
- Dépendances → bibliothèques → jar (zip)

Java Console → Texte
Swing → divers Layout Graphiques

EE → .war → Tomcat → Serveur d'App (Java EE)

- Jboss
- Web Logic
- Web Sphere
- Oracle App Server



~~go~~ → Java

~ go → "Design Pattern" du GoF.

"Recettes" → Pattern → Nom
→ objectif
→ (niveau de difficulté)
→ Méthode

Patterns d'architecture logicielle

→ MVC Model Vue Controller
↳ parfait pour le front (GUI web)

↓

Struts (2) → Spring MVC

ORM Object Relationnal Mapping

↓

Hibernate → JPA

IoC Inversion of Control
↘ DI Dependancy Injection

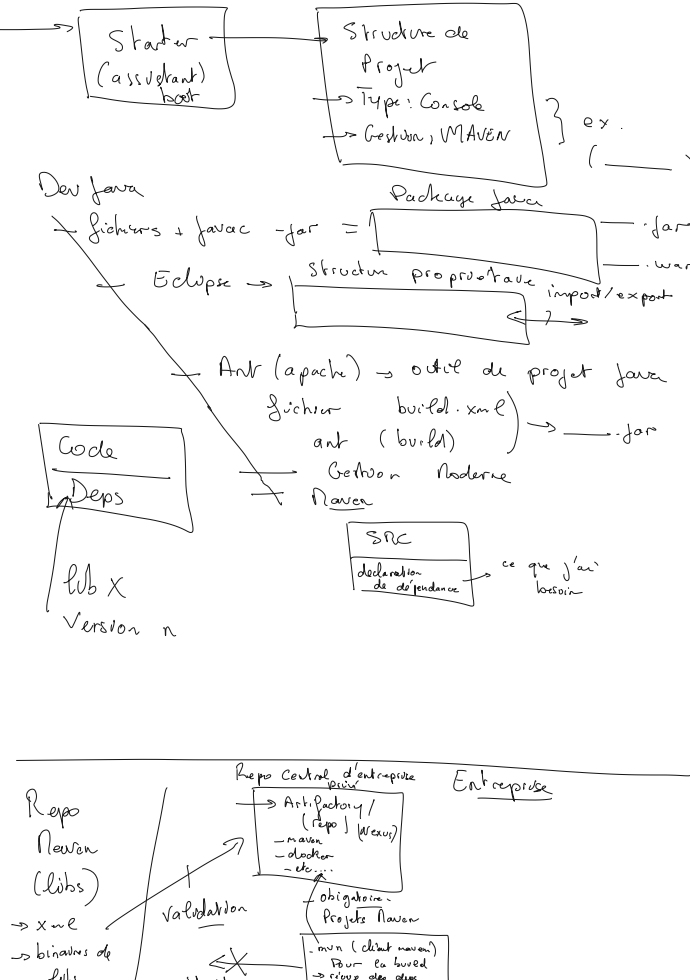
↓

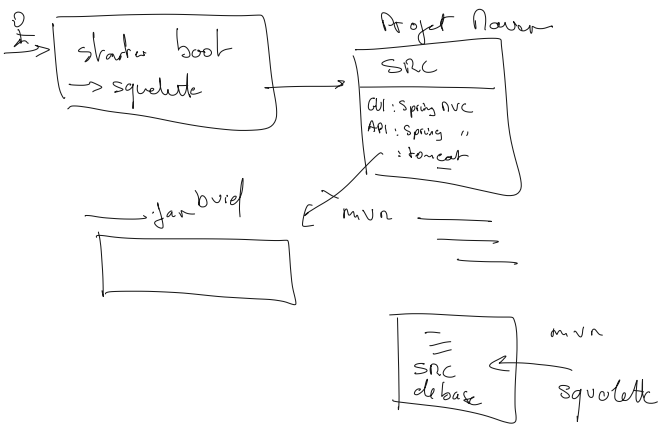
Spring (core)

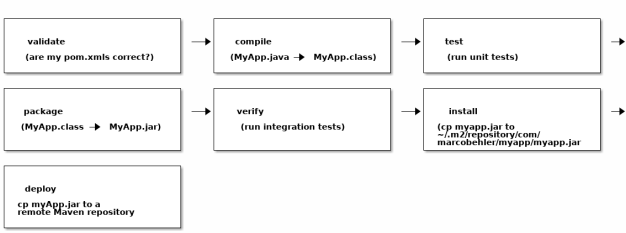
AOP → Plugins

↓

Spring AOP


Starter
(assistant)
Structure de Projet
Type : Console
Gestion : MAVEN
ex.
Dev Java
fichiers + javac -jar =
Package Java
.jar
.war
Eclipse →
Structure propriétaire
import/export
Ant (apache) → outil de projet Java
fichier build.xml
ant (build)
.jar
Code
Deps
Gestion moderne
Maven
SRC
déclaration de dépendance
ce que j'ai besoin
lib X
version n
Repo Central d'entreprise
Entreprise
Repo Maven (libs)
Artifactory / (repo) (Nexus)
- maven
- docker
- etc...
→ xml
→ binaires de libs
Validation
obligatoire
Projets Maven
mvn (client maven)
Pour la build
→ récup des deps
→ compilation → Tests → Packaging
→ .jar
interdit Réseau





mvn package (recompiler et générer le .jar)
java -jar target/xxxx.jar

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@EnableAutoConfiguration
public class MyApplication {

  @RequestMapping("/")
  String home() {
    return "Hello World!";
  }

  public static void main(String[] args) {
    SpringApplication.run(MyApplication.class, args);
  }

}
```

Créer une fonction d'API :
Créer une autre classe qui contiendra les méthodes d'API
Exemple :

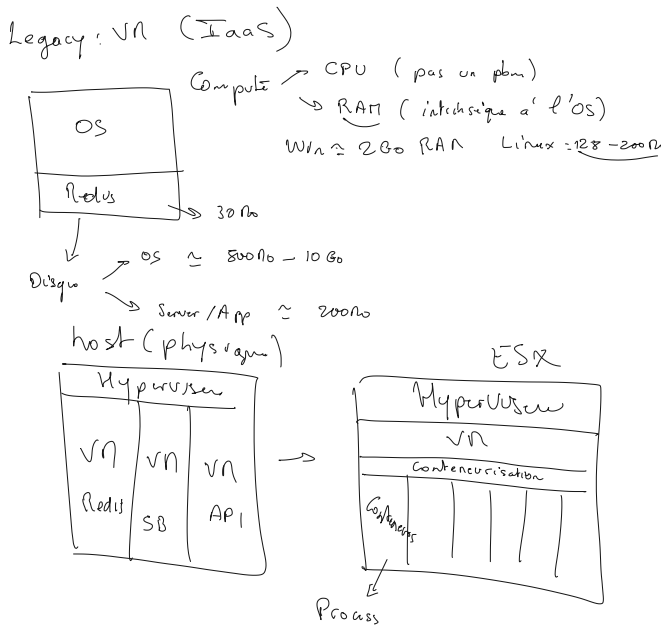
```
package com.example.demo;

import org.springframework.web.bind.annotation.*;
import java.util.*;

@RestController
class ProductsController {

  ProductsController() {
  }

  // Aggregate root
  // tag::get-aggregate-root[]
  @GetMapping("/api/products")
  List<String> all() {
    return Arrays.asList("Chai", "Chang", "AnySeed Syrup");
  }
}
```



App Springboot

→ java —jar ——— jar

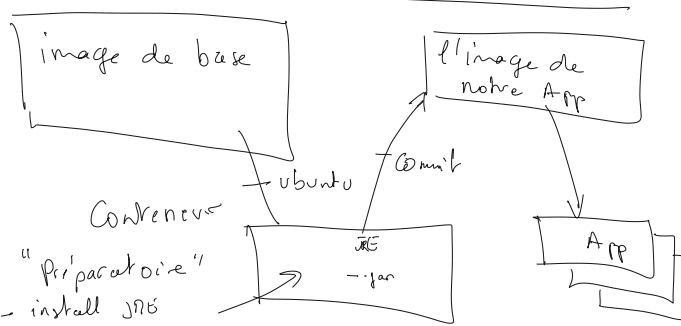
JRE

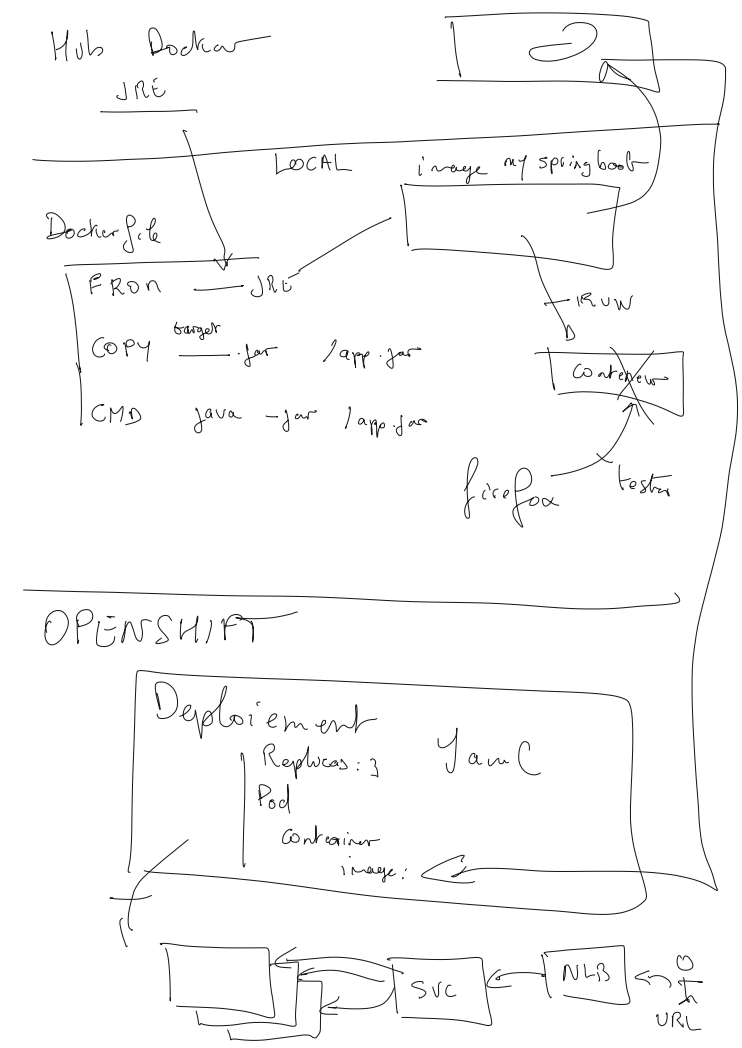
OS : - JRE dwspo
- package — jar

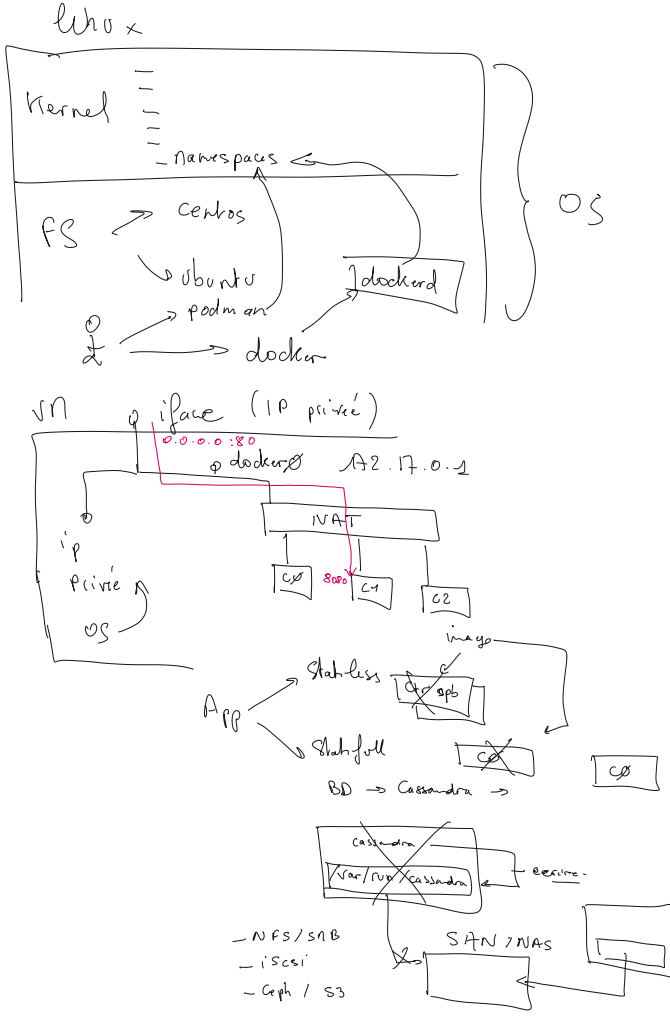
VM

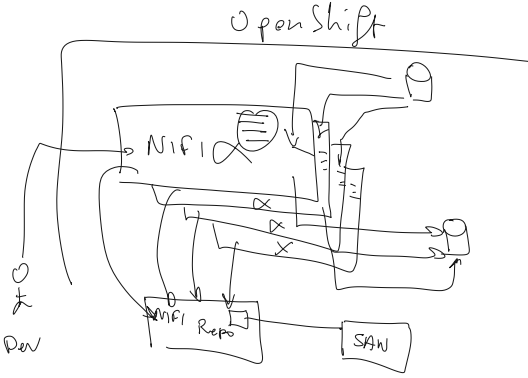
ISO — install → OS → modèle → template

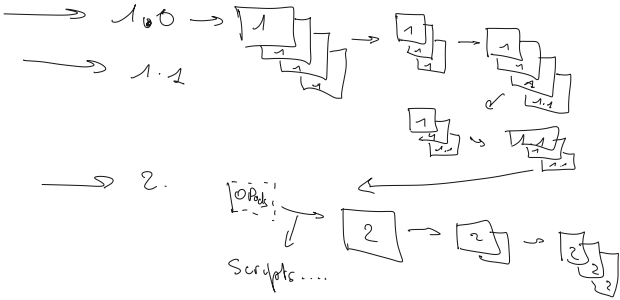
ferme de .... → OS

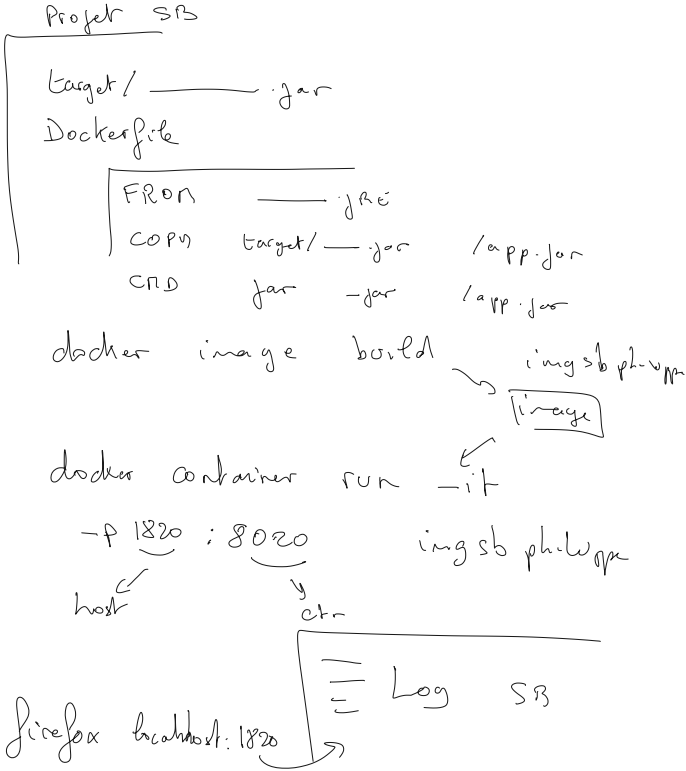



Hub Docker
JRE
LOCAL
image my springboot
Dockerfile
FROM JRE
COPY target .jar /app.jar
CMD java -jar /app.jar
RUN
D
container
firefox
tester
OPENSHIFT
Deployement
Replicas: 3
Yaml
Pod
container
image:
SVC
NLB
URL


linux
kernel
namespaces
OS
FS
centos
ubuntu
dockerd
podman
docker
VM
iface (IP privée)
0.0.0.0:80
docker0
172.17.0.1
NAT
IP privée
OS
C0
8080
C1
C2
image
App
Stateless
Stateful
BD → Cassandra →
cassandra
/var/lib/cassandra
écrire
NFS/SMB
iSCSI
Ceph / S3
SAN / NAS


OpenShift
NiFi
NiFi Repo
SAN
Dev


1.0
1.1
2.
ORM
Scripts....


Projekt SB
target/ ——— .jar
Dockerfile
FROM ——— jRE
COPY target/ ——— .jar /app.jar
CMD jar -jar /app.jar
docker image build
imgsb ph-wpr
image
docker container run -it
-p 1820 : 8020
imgsb ph-wpr
host
y
ctr
Log SB
firefox localhost: 1820

Conteneuriser l'App:

1) Fabriquer un fichier Dockerfile :
FROM mcr.microsoft.com/openjdk/jdk:17-ubuntu

RUN groupadd -r user && useradd -r -g user user
USER user

COPY target/**gitlabspringbootdemo**-0.0.1-SNAPSHOT.jar /app.jar

ENV PORT **8080**
EXPOSE $PORT

CMD ["java", "-jar", "/app.jar"]

2) recomplier l'app avec les bons paramètres
mvn package

3) construire l'image : (qui va exploiter le fichier Dockerfile qui se trouver dans le dosser "."
docker image build -t **sbdphilippe** .

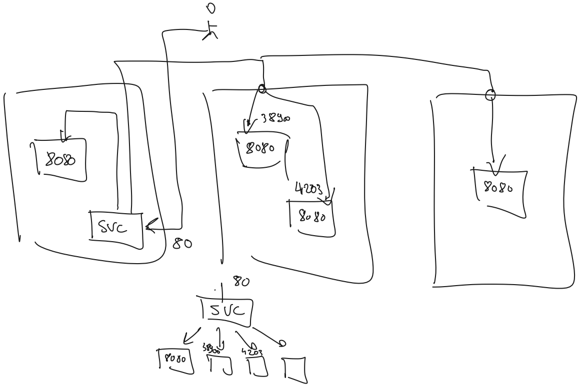
4) tester :
docker container run -it --rm -p **82**:**8080 sbdphilippe**

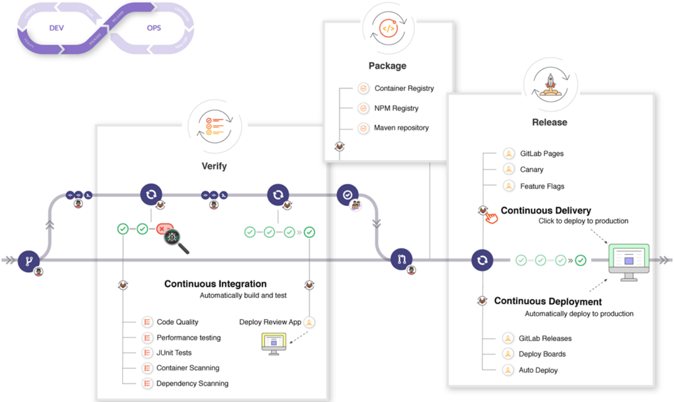
-p <port sur le host>:<port dans le conteneur>

5) s'authentifier sur le hub docker :
docker login

6) tagguer conformément l'image (selon votre compte et image)
docker image tag sbdphilippe **nboost**/sbdphilippe

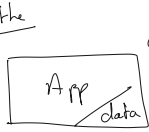
7) pousser votre image
docker image push **nboost**/sbdphilippe


8080
SVC
80
8080
4203
8080
8080
80
SVC
8080
4203

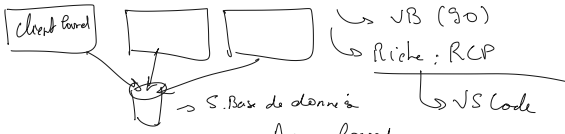

DEV
OPS
Package
Container Registry
NPM Registry
Maven repository
Release
GitLab Pages
Canary
Feature Flags
Continuous Delivery
Click to deploy to production
Verify
Continuous Integration
Automatically build and test
Code Quality
Performance testing
JUnit Tests
Container Scanning
Dependency Scanning
Deploy Review App
Continuous Deployment
Automatically deploy to production
GitLab Releases
Deploy Boards
Auto Deploy

archis des Applis →

monolithe

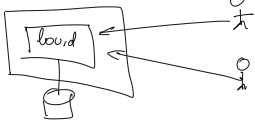


client - serveur

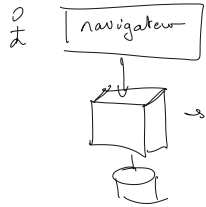


→ Serveur d'Application → App lourde
→ App légères

Serveur d'APP → Citrix TSE



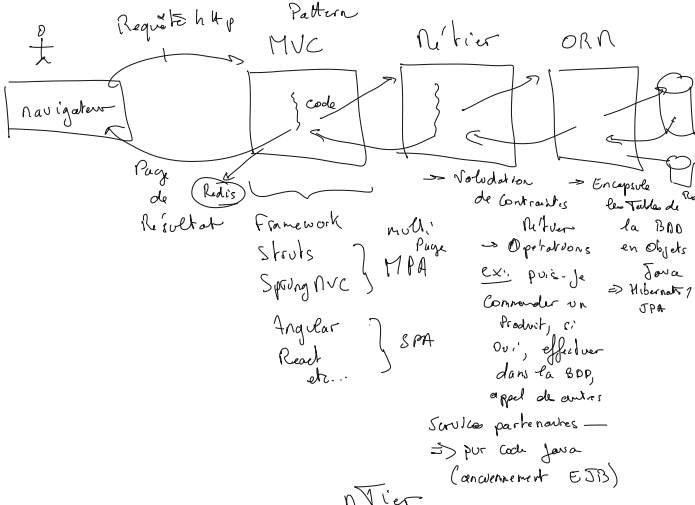
→ Serveur d'App "légère"

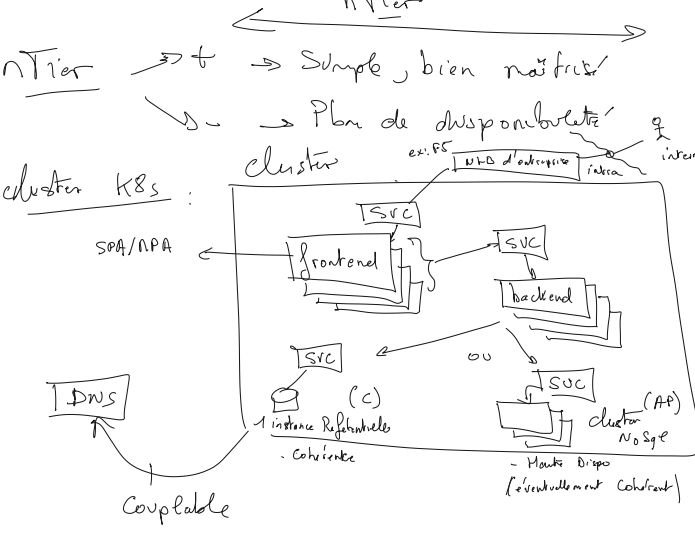


Modèle 3 Tier

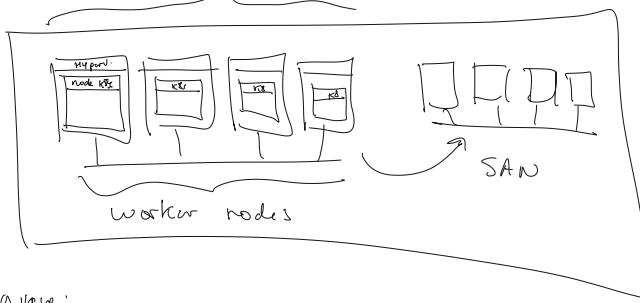
↓

front (MVC)
→ Middleware
→ Stockage


Requête http
Pattern
MVC
Mapper
ORM
navigateur
code
Page de Résultat
Redis
Framework
Struts
SpringMVC
multi Page
MPA
Angular
React
etc...
SPA
→ Validation de Contraintes
→ Encapsule les Tables de la BDD en Objets Java
Redis
Mapper
→ Opérations
ex: puis-je Commander un Produit, si oui, effectuer dans la BDD, appel de autres Services partenaires
⇒ pur code Java (Lancement EJB)
⇒ Hibernate / JPA

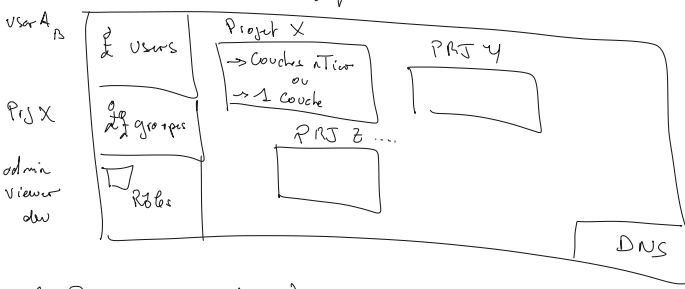

nTier
nTier
→ Simple, bien maîtrisé
→ Plan de disponibilité
cluster K8s :
cluster
ex: F5
LB d'entrée
intent
SVC
SPA/MPA
frontend
SVC
backend
SVC
ou
SVC
DNS
(C)
instance Relationnelle
- Cohérente
(AP)
cluster NoSql
- Moins Dispo
(éventuellement Cohérent)
Couplable

cluster (physique) ESX



Logique :

cluster logique



1 Projet (openShift) = 1 namespace (K8s)

→ User / groupe / roles

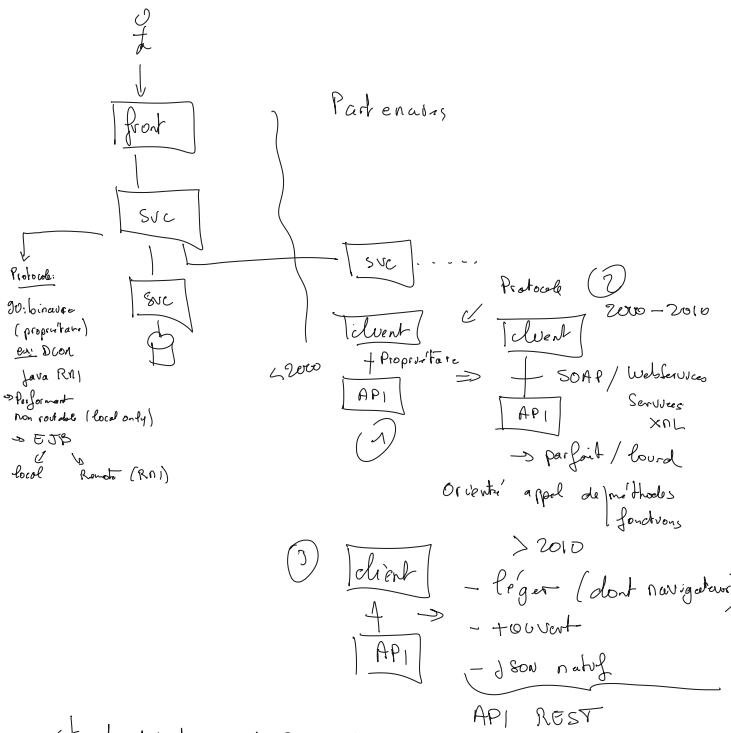
→ Ressources → Pod
→ StatefulSet (BDD)
→ Horizontal Pod Autoscaler (cloud : auto)

→ Quotas

DNS

kafka
amq

broker . cluster . local
nom de Projet



https://editor.swagger.io/

https://github.com/public-apis/public-apis

front web html → html 4 (20 ans)
↳ html 5

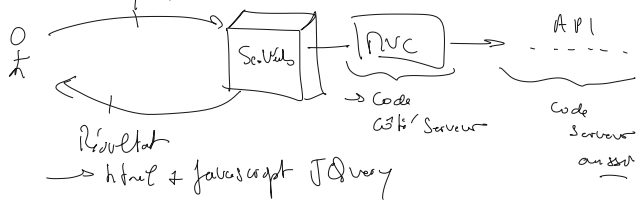
≃ 2000 (html 4)

≃ 90
Java → ~~(Applet)~~

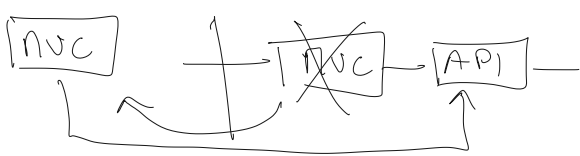
client léger → navigateur

client Riche léger → - JavaFX (Java)
- Silverlight (.net)
- Flash → Flex

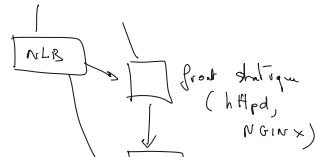
html 5



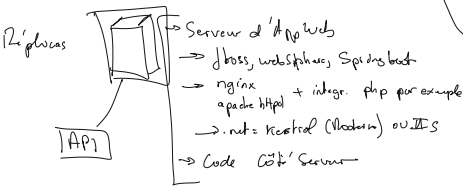
JQuery = excellente bibliothèque pour du
html "Dynamique"
≃ 2000 - 2010



Déploiement.

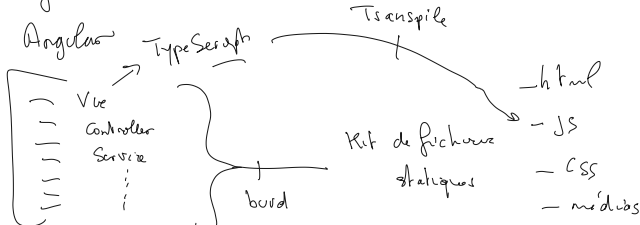


NPA



Pbm Serveur d'App → - performant
Pour le contenu statique - html - médias -
- images
- css
- js

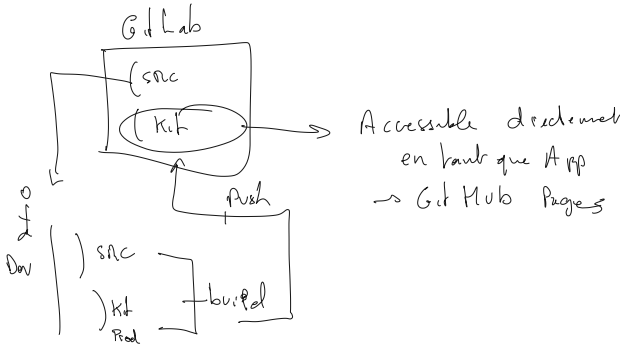
SPA



Déploiement :

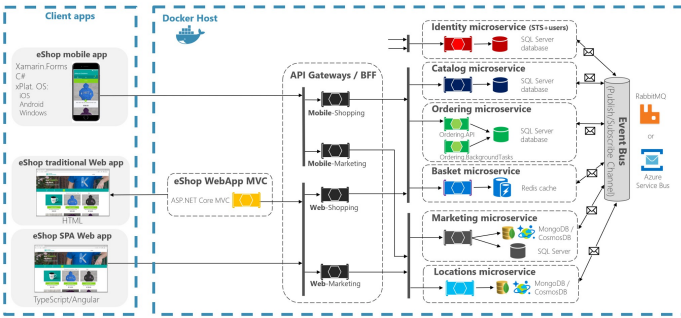
① → S. statique (Traditionnel)

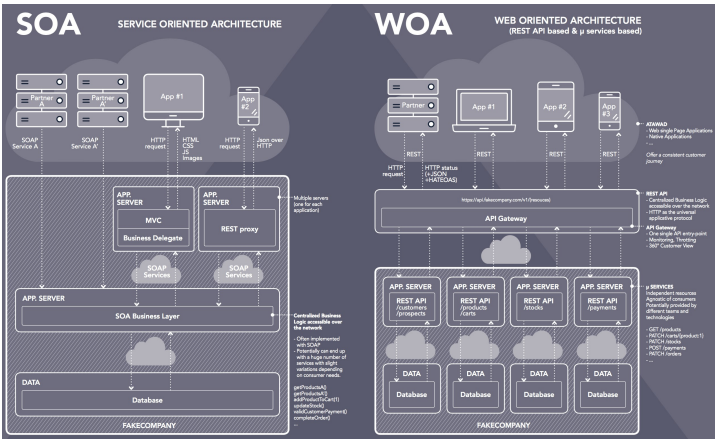
② → CDN → Répliquation intégrée + DNS.
Content Delivery Network.

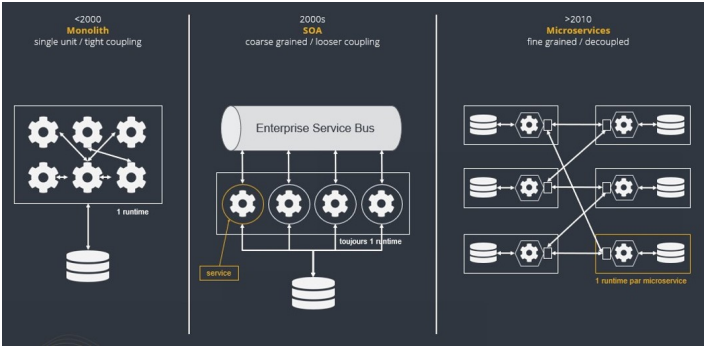


eShopOnContainers reference application

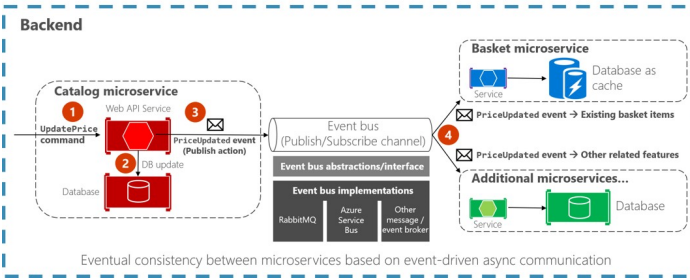
(Development environment architecture)

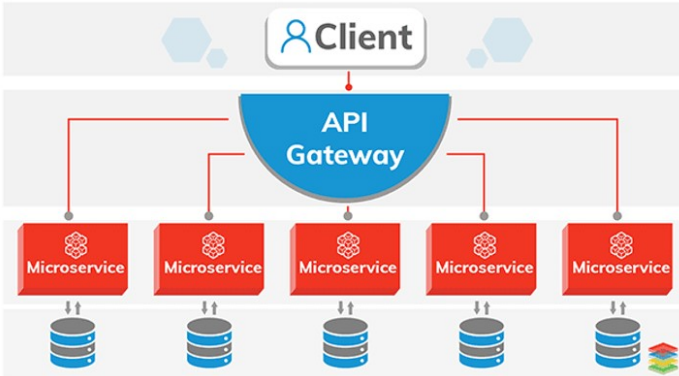


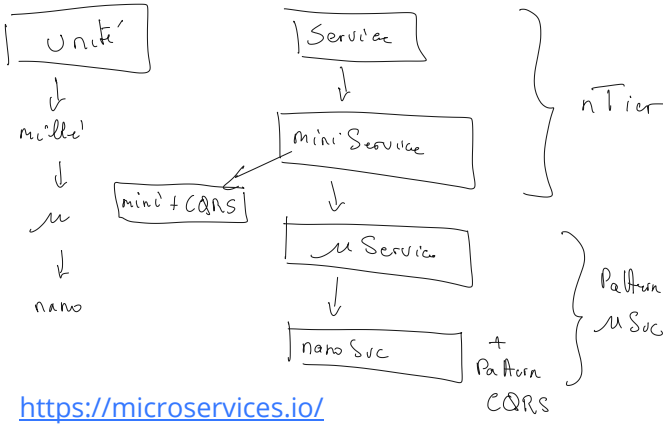

SOA
SERVICE ORIENTED ARCHITECTURE
Partner
Partner
App #1
App #2
SOAP Service A
SOAP Service A'
HTTP request
HTML CSS JS Images
HTTP request
Json over HTTP
APP. SERVER
MVC
Business Delegate
APP. SERVER
REST proxy
SOAP Services
SOAP Services
APP. SERVER
SOA Business Layer
Multiple servers (one for each application)
Centralized Business Logic accessible over the network
- Often implemented with SOAP
- Potentially can end up with a huge number of services with slight variations depending on consumer needs.
getProductsA()
getProductsA'()
addProductToCart(1)
updateStock()
validCustomerPayment()
completeOrder()
DATA
Database
FAKECOMPANY
WOA
WEB ORIENTED ARCHITECTURE
(REST API based & μ services based)
Partner
App #1
App #2
App #3
REST
REST
REST
REST
HTTP request
HTTP status (+JSON +HATEOAS)
ATAWAD
- Web single Page Applications
- Native Applications
Offer a consistent customer journey
https://api.fakecompany.com/v1/carscard
API Gateway
REST API
- Centralized Business Logic accessible over the network
- HTTP as the universal applicative protocol
API Gateway
- One single API entry-point
- Monitoring, Throttling
- 360° Customer View
APP. SERVER
REST API /customers /prospects
APP. SERVER
REST API /products /carts
APP. SERVER
REST API /stocks
APP. SERVER
REST API /payments
μ SERVICES
Independent resources
Agnostic of consumers
Free to be provided by different teams and technologies
- GET /products
- PATCH /carts/{product:1}
- PATCH /stocks
- POST /payments
- PATCH /orders
...
DATA
Database
DATA
Database
DATA
Database
DATA
Database
FAKECOMPANY


<2000
Monolith
single unit / tight coupling
1 runtime
2000s
SOA
coarse grained / looser coupling
Enterprise Service Bus
service
toujours 1 runtime
>2010
Microservices
fine grained / decoupled
1 runtime par microservice

Implementing asynchronous event-driven communication with an event bus





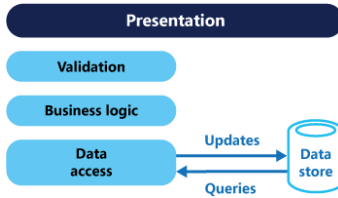


https://microservices.io/

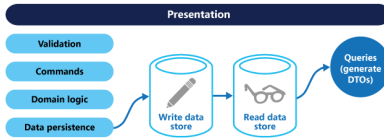
Communication via Bus : Pattern SAGA
https://microservices.io/patterns/data/saga.html

Découplage bases Lecture Ecriture :
Pattern CQRS : https://microservices.io/patterns/data/cqrs.html

Sans CQRS :



Avec CQRS :



API Synch / Asynchrone

Synch → simple
→ pas forcément disponible

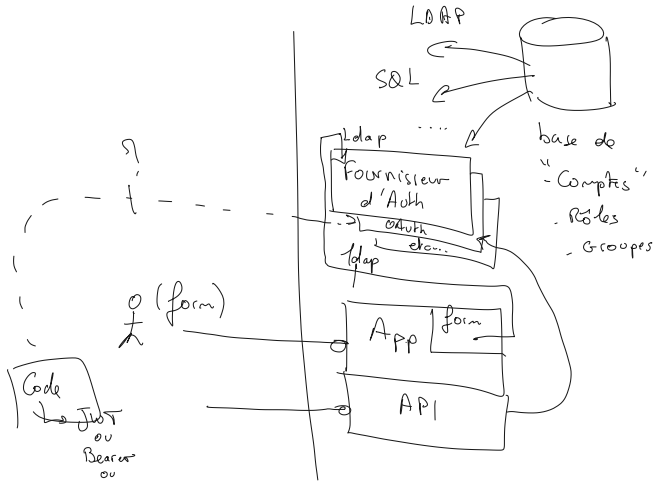
Asynch → + complexe
→ Hautement disponible

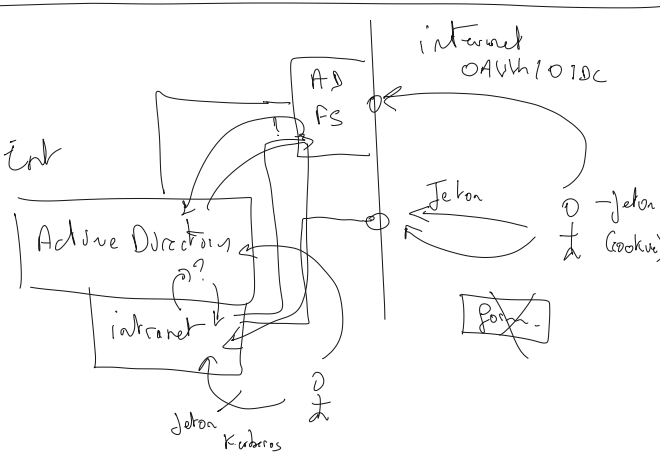
Texte, donnée → fct de hash → → Valeur de hash

	hash	
test →	123	↑ hash
toto →	132	
mdp →	hash	→ Digne

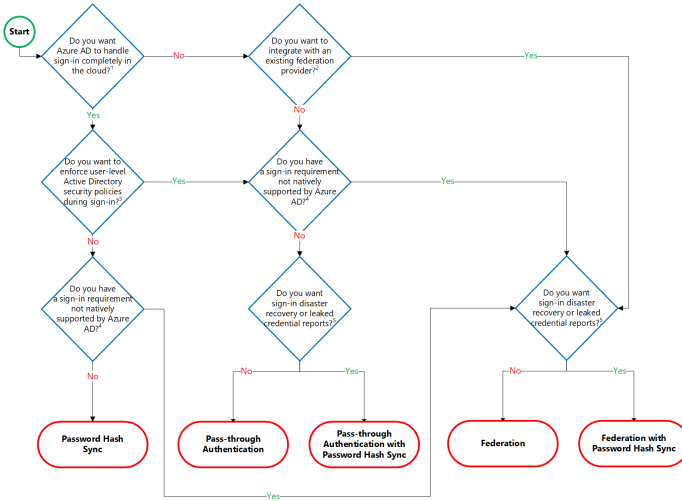
Dictio de mdp.

hash ← mdp
md5
sha1
sha2 ...


LDAP
SQL
....
ldap
Fournisseur d'Auth
OAuth
etc...
ldap
base de
- Comptes
- Rôles
- Groupes
SI
(form)
App
form
API
Code
ou
Bearer
ou
OAuth ....


intranet
OAUth / OIDC
AD FS
Int
Active Directory
intranet
Jeton
-jeton
(cookie)
Jeton
Kerberos

Exemple de choix de méthode d'auth :
Azure AD Connect pour fédérer les utilisateurs : choix à faire :



Tuto Angular / OIDC : https://learn.microsoft.com/en-us/azure/active-directory/develop/tutorial-v2-angular-auth-code

Angular qui utilise une API avec Auth :
https://learn.microsoft.com/en-us/azure/active-directory/develop/scenario-spa-overview

POC SAML avec SpringBoot et Angular :
https://github.com/isaacgarza/saml-example

Tutoriel d'initiation à Angular :
https://angular.io/tutorial/first-app

Angular Online : https://codesandbox.io/p/sandbox/angular-angular

≤ BDD

≃ 60-70 → Bandes → lecture/écriture multiple
Avant uniquement
→ Cartes → écriture unique) Séquentiel
lecture multiple

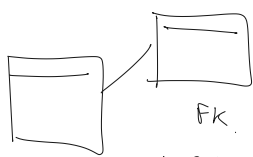
IBM → disques dur → accès aléatoires
très cher !

Besoin de système d'optimisation / exploitation.

↳ SGBDDR → Oracle v1 1979

→ exploit stockage aléatoire
Consolidé en stockage

Methode (Merise) → normalisation



→ pas de doublons
→ pas de copies
→ jointures
→ "peu performant"
↳ index etc ...

NORMAL depuis 1979

Toujours d'actualité
→ Système relationnels/Postgresql.

Atomicité
Cohérence
Isolation
Durabilité

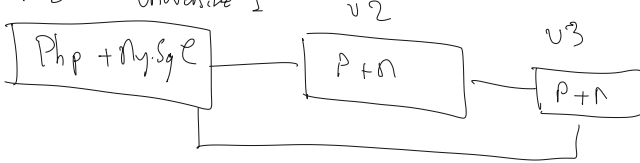
→ Silos d'entreprise

≃ 2000 → Tech internet

→ Volume ↗
→ Variabilité ↗
→ Vitesse ↗

} Si un élément n'est pas encaissable par un SGBDR
↳ passer sur du NoSQL

FB



→ Développer (Neo4J)

→ Solutions Tierces qui ont vu le jour

- Clef - Valeur (Redis) - TimeSeria
- Document (MongoDb) (InfluxDb)
- Column store (Cassandra)
- Graph (Neo4J)
- Message (Kafka)

Cohérence ?

Primordial

→ plutôt SQL

↓

Réplica et/ou Partitionnement } compliqué → Cohérence garantie

Non Primordial

→ plutôt NoSQL

→ Haute Dispo

→ Conçus nativement pour être en cluster

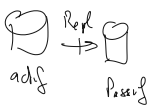
↙

" facile "

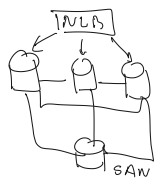
NSQL

Mysql

PG

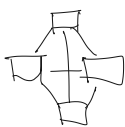


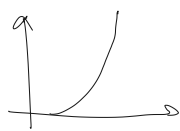
Oracle RAC

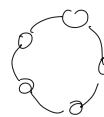


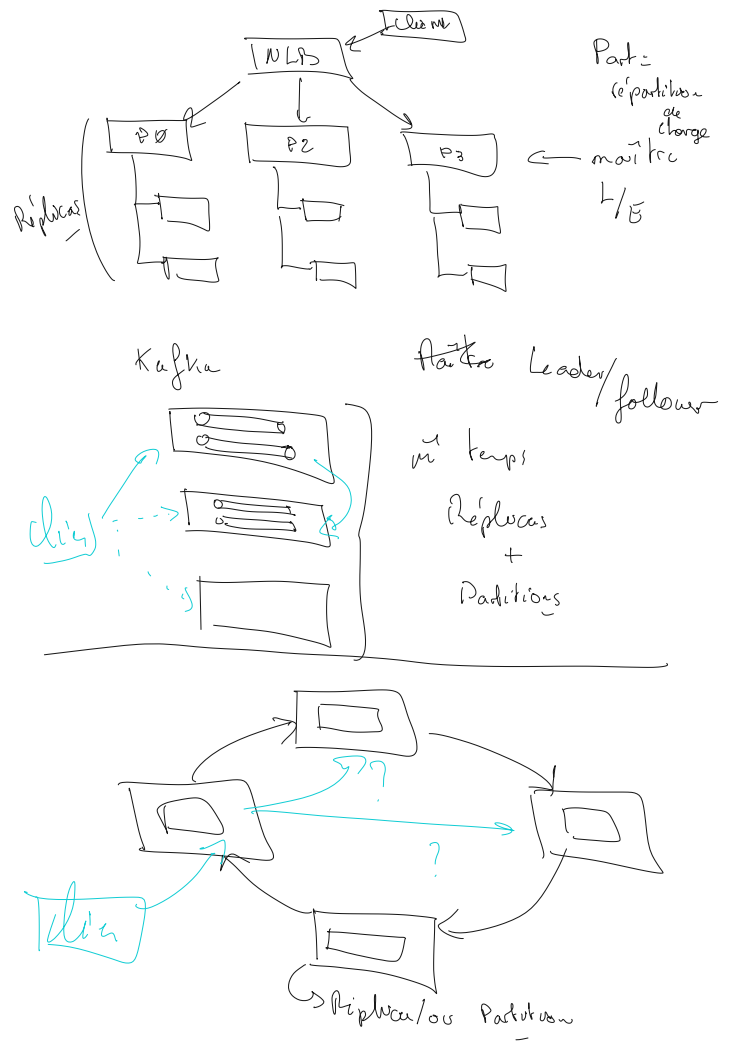
Act/Act

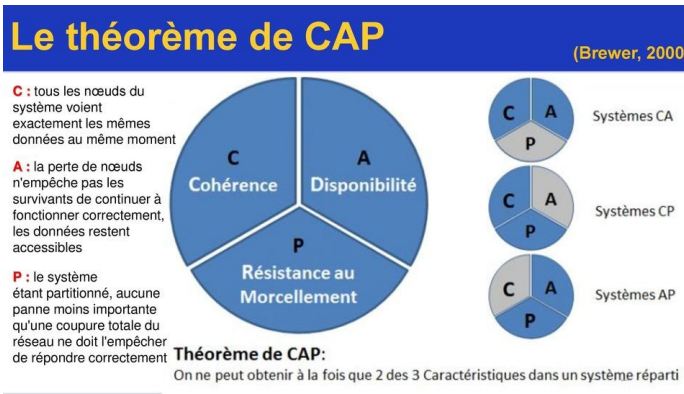
HA + Cohérence ?








Client
NLB
P0
P2
P3
Répliqués
Part.
répartition de charge
maître
L/E
Kafka
Leader/follower
Client
Répliqués + Partitions
Client
Répliqués/ou Partitions



•Les types d'authentification http :
•Basic : le mot de passe transite en clair (https recommandé !)
•Digest : le mot de passe est haché, mais sensible aux attaque rainbow table (https recommandé)
•Formulaire : les champs passent en clair
•Par Certificat : Un échange sécurisé de la clef se fait, et une correspondance à un utilisateur donné se fait côté serveur
•Authentifications modernes
•SAMLv2 (Security Assertion Markup Language) : Permet le SSO, ADFS, ...
•OpenID Connect (Oauth + Jeton json JWT): Authentification fédérée, jeton d'identité (ex Facebook uniquement pour l'authentification)
•OAuth2 : Délégation d'autorisations (exemple Facebook pour accéder à ses données)
•https://www.okta.com/fr/identity-101/whats-the-difference-between-oauth-openid-connect-and-saml/

Schéma de fonctionnement de SAML

Schéma de fonctionnement de OpenID Connect

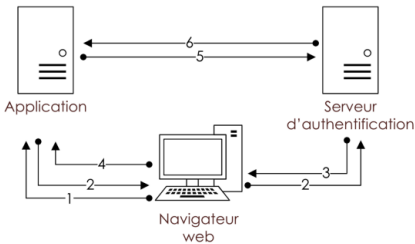


Schéma de fonctionnement de OAuth2