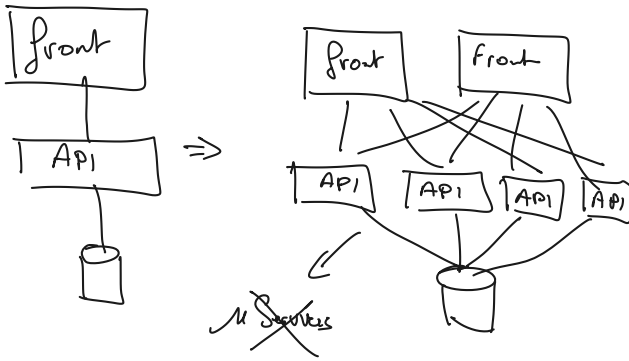
µ Services



Services

monolithique

⇓

mini Services → mini Suc (CQRS)

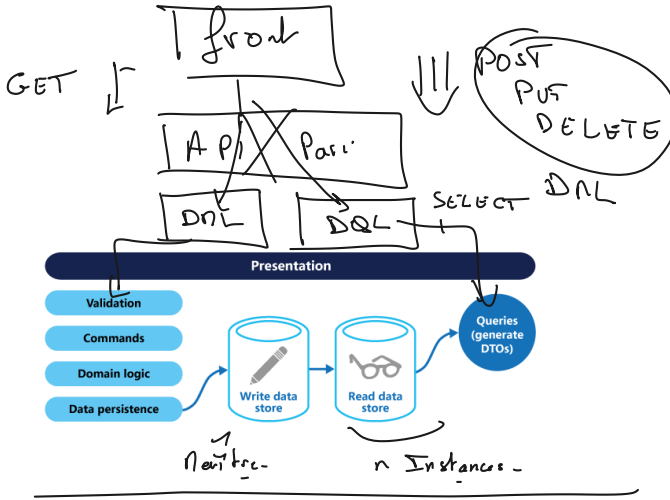
⇓

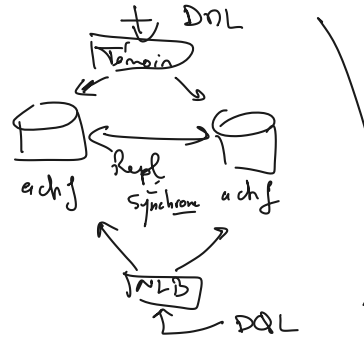
µ Services

archi. µ Suc

⇓ pattern CQRS

nano Services


front
GET
POST
PUT
DELETE
API
Pass
DML
DQL
SELECT
DML
Presentation
Validation
Commands
Domain logic
Data persistence
Write data store
Read data store
Queries (generate DTOs)
Write loc-
n Instances -


DML
Domain
Repl
Synchron
achf
achf
VNLB
DQL
Vous

Traditionnellement Des web → fonction de la Technologie

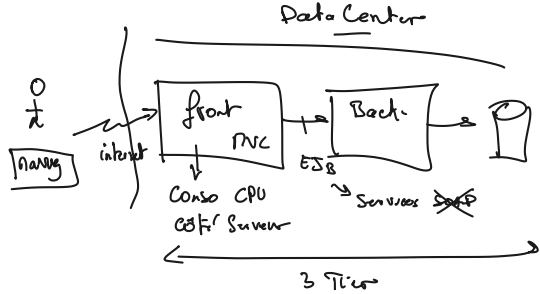
Php → Page php → calcul + affichage

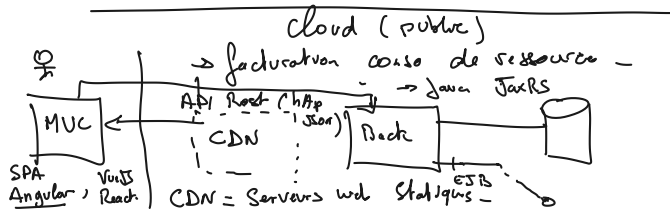
.net → ASP .net web form

Java J2EE → Servlets

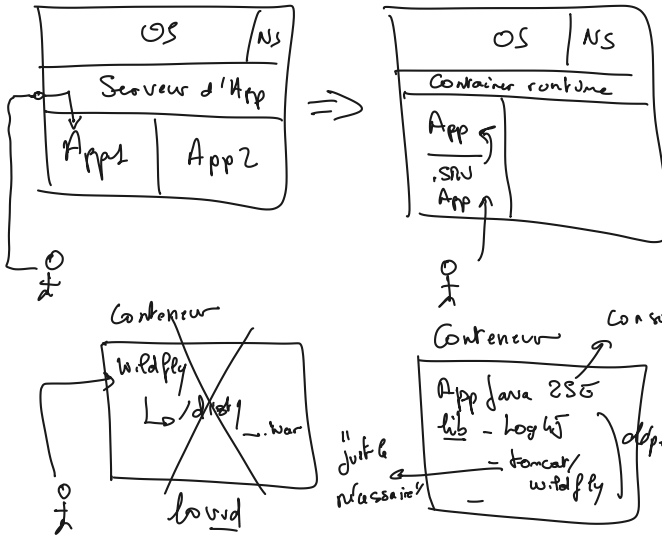
Pattern Model Vue Controller

Java J2EE | Struts (obs) (pbm sécurité)
Spring MVC → assez utilisé





actuellement : VM



Modèle Cattle vs Pet

cluster Kubernetes

→ cluster de conteneurs —

Grains → Conso de ressource globalement + faible

→ Déploiement très rapides (/ ordre de la seconde / minute en VM)

Modèle "Cattle vs Pet"

Modèle Pet (de Compagnie)

↳ Serveurs - "éduqués" à la main
- nourris " " "
Hérité - "entretenus" " " " "

Modèle Cattle (Bétail)

→ Tout est automatisé

→ Conception
→ build
→ livraison
→ Redémarrage (en cas de soucis)

CI/CD
DevOps

Prod (élastique)
en fonction de la
Demande

Métaphore concernant le conteneur

⇒ Vendra des frites

→ Conteneur

SLA n-gth

Service Level Agreement

5.9 = 99.999 %

99us secondes / jour

99.9999999 % / an

• Couplages
  ○ **Fort**
    § Simple, facile à comprendre
    § Problématique pour l'évolution et les mises à jour
  ○ **Faible**
    § Le client peut décider de quelle implémentation d'une interface obtenir.
    § Les évols doivent respecter l'interface.
    § La mise à jour de l'interface implique la mise à jour des clients.
    § Nécessite l'implémentation de pattern.
  ○ **Lâche**
    § Le client demande des implémentations d'un type d'interface. Il en trouvera, ou non.
    § Une implémentation pourrait ne pas être trouvée, il faudra faire avec.
  ○ **Découplé**
    § Chaque composant est développé sans tenir compte des autres, sauf rester conforme aux specs lors des évolutions (respect du versioning sémantique)

Ils doivent publier des interfaces, et une documentation, qui restera toujours ce qu'elle est au fil du temps, au risque de casser le système.

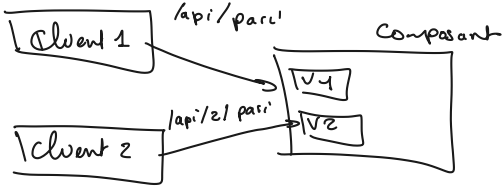
Versions des Services :

/api/client

/api/pari → v1.x

↑

toute révisions conformes.



BDD Relationnelle

Souvent → Act.f / Passif

HA → Toujours Up (SLA < 99.999%)

- Jamais saturé (CPU < 20%)
- pas de locks

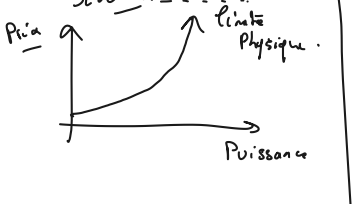
Scale Up → Augmenter la puissance

→ Perf CPU brut, Ccache (RAM)

→ " disque, Réseau

SPOF

Scale down

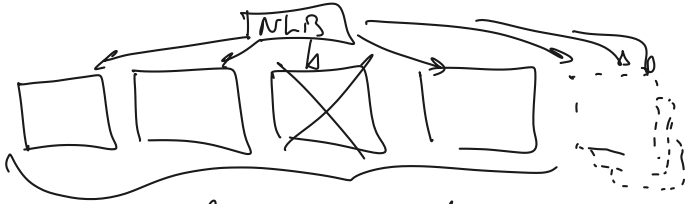


Déploiement Modernes :

Scale Out = Ajouter des Déploiement/Machines

Scale In

Si Gestion automatique = Système Élastique



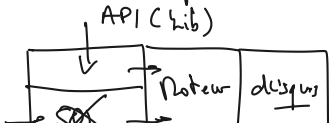
"cluster de Déploiement"

"Coût" du matériel bonus avec la puissance au mieux.

⇒ Prédéploiement prédictif

Scale Out : BDD Relationnelles ne sont pas adaptées

Solution préconisée → BDD NoSql



→ Déterminés pour un type de charge -

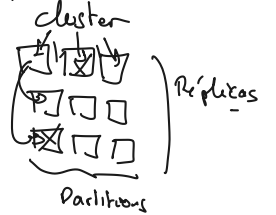
⇒ Il faut connaître les requêtes pour déterminer le shéma

Types de charge sont

- Stockage en colonne
- clef-valeur (incapacité de traiter le contenu)
  - très rapide (ex: Redis pour du cache)
- Document clef-valeur + capacités de calcul

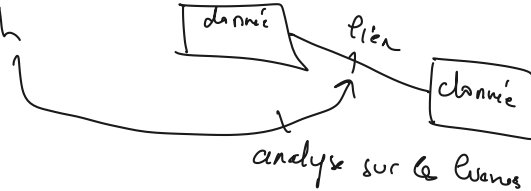
ex: MongoDb (HA)

→ Requêtes Big Compute (calcul distribué)



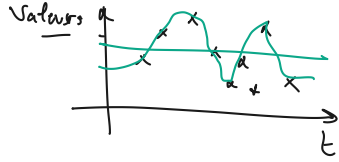
⇒ Oracle RAC

- Graph



- Time Series

ex Influx Db

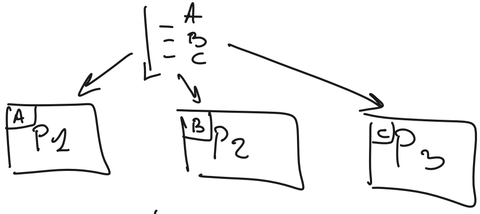


- Message

ex: Kafka → KSqeDb

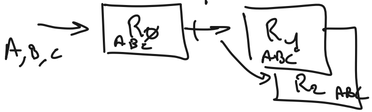
Mosquitto (IoT)

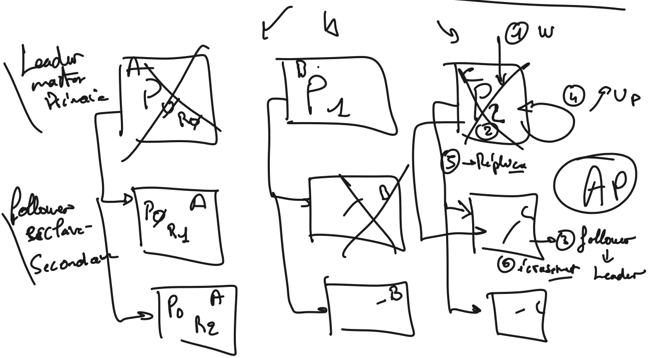
Bases NoSql → Prévues pour la HA-

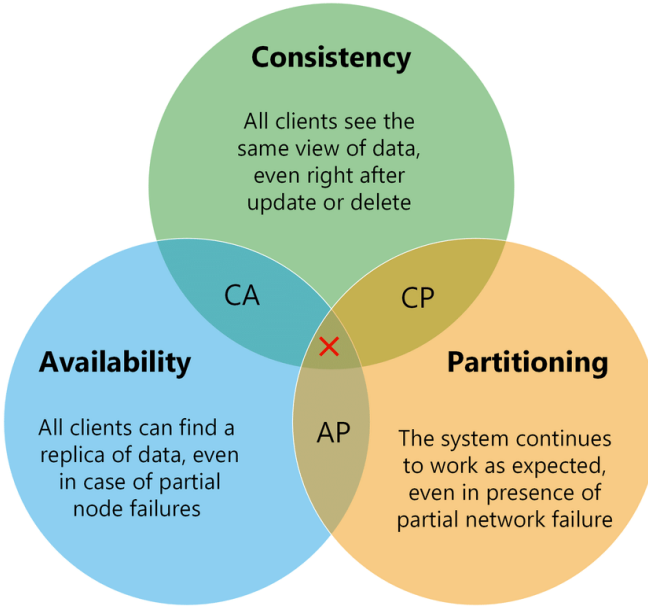


x3 risque de panne

Synchrone → - performant







⑥ Se prémunir des écrasements :

Concept MVCC

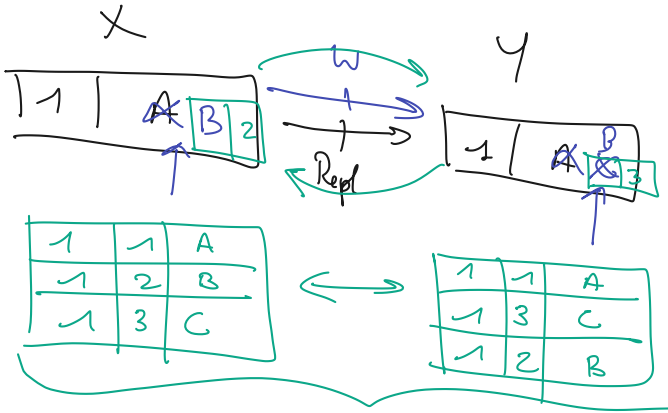
Multi Version

Concurrency check.

Collection (Table)

PK (int)	"Version" → incrément → datetime	Valeurs
1		A
2		B
3		C

défini automatiquement sur l'écriture

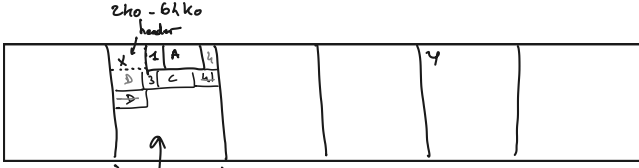


Select - .... (order by version DESC TOP 1)

1	3	C

1	3	C

RowStore (Stockage Relationnel de base)



écho - block
header

fichier de données

- Page = bloc

int (PK)	varchar(50)

W → Page = 1 unité d'IO

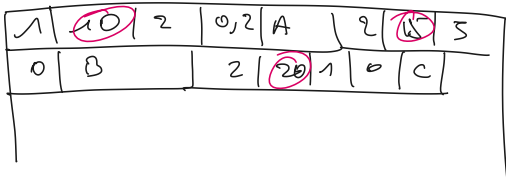
RAM → cache global des pages

log

synchrone

BI → Tables de faits, dimensions
schémas en flocon, étoile

int (PK)	float (prix)	int (entier)	float (remise)	Nom varchar()



Select SUM(Qté) From produits

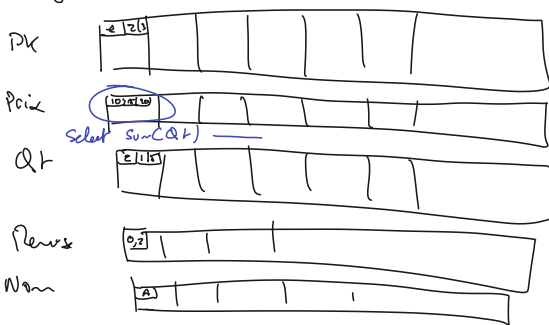
RAM

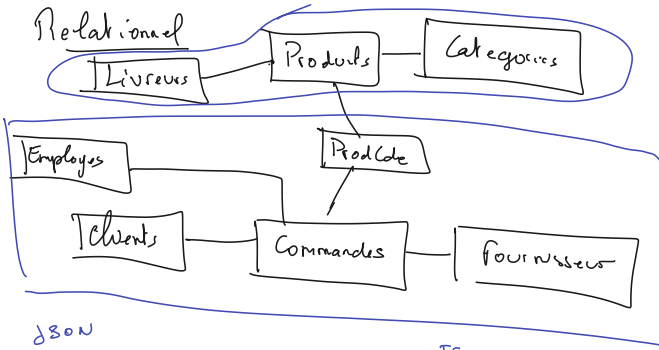
- Tout le détail ?
- juste la col Qté ?

"Column Store" →

Insert 1, 10, 2, 0,2, 'A' ⇒ Ventilation = Très mauvais en perf.

1 fichier par colonne





JSON

Commandes

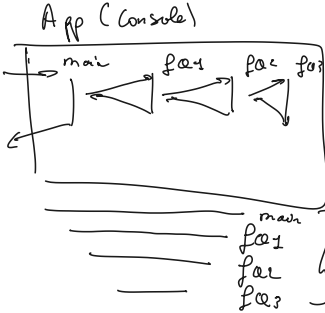
```
{ "client1",
  "commandes", [
    { id: 1,
      date: --
      Produits: [
        {id: "pomes": 3
         qte: 2,
         Pain: 10 ....
```

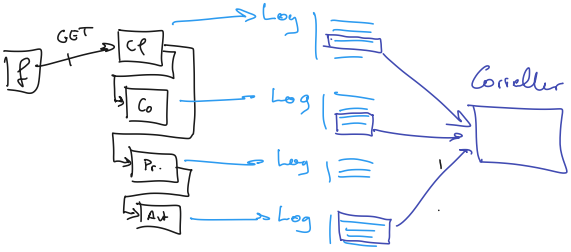
JSON

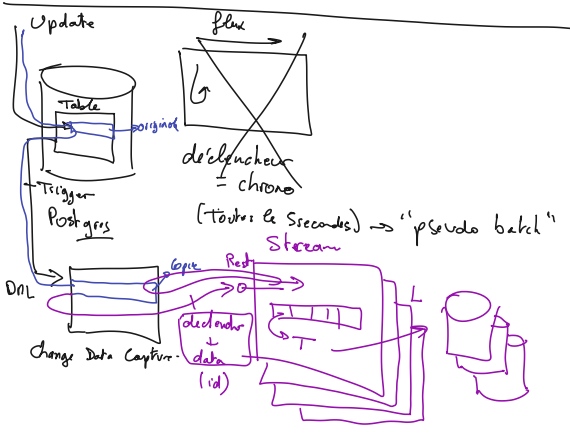
Produits

API :

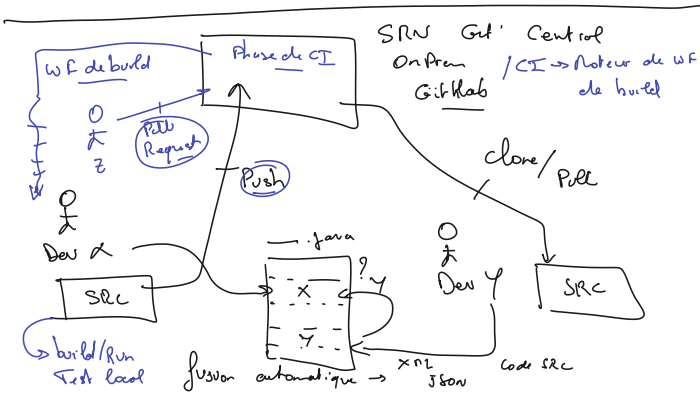
- client
- commande
- Produit ← Copie de données structurées, Pas de jointures !


App (Console)
main
foo1
foo2
foo3
main
foo1
foo2
foo3
Pile d'appel

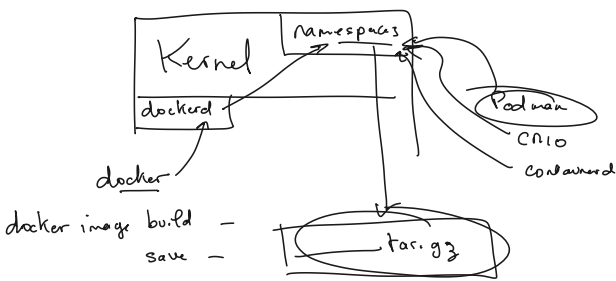

GGT
f
Cl
Co
Pr.
Ant
Log
Log
Log
Log
Corrélateur

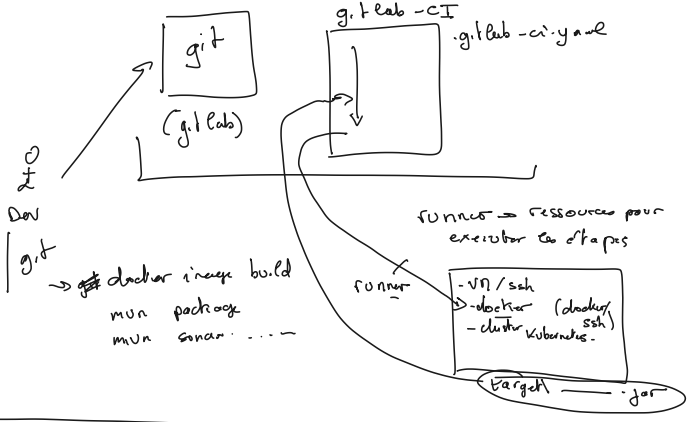

Update
Table
original
Trigger
Postgres
flux
déclencheur
- chrono
(toutes les 5 secondes) → "pseudo batch"
Stream
DML
Change Data Capture
copie
Redo
déclencheur
data
(id)
L
T

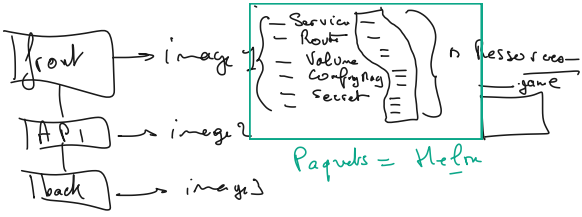
Installation de Openshift local :
https://www.redhat.com/sysadmin/install-openshift-local



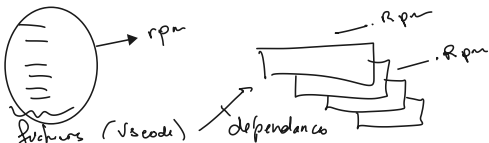
Exercice :
Découvrir les failles de certaines bibliothèques ou systèmes
(exemple : centos, wildfly, log4j, ...)
https://nvd.nist.gov/vuln/search







installer un package sous un OS

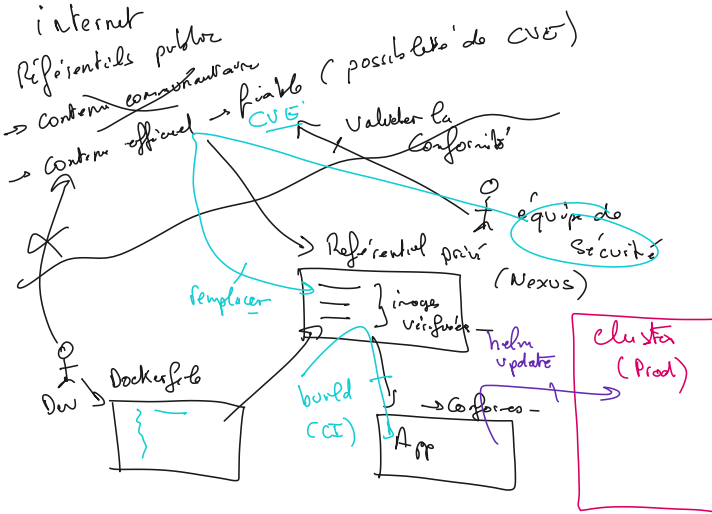


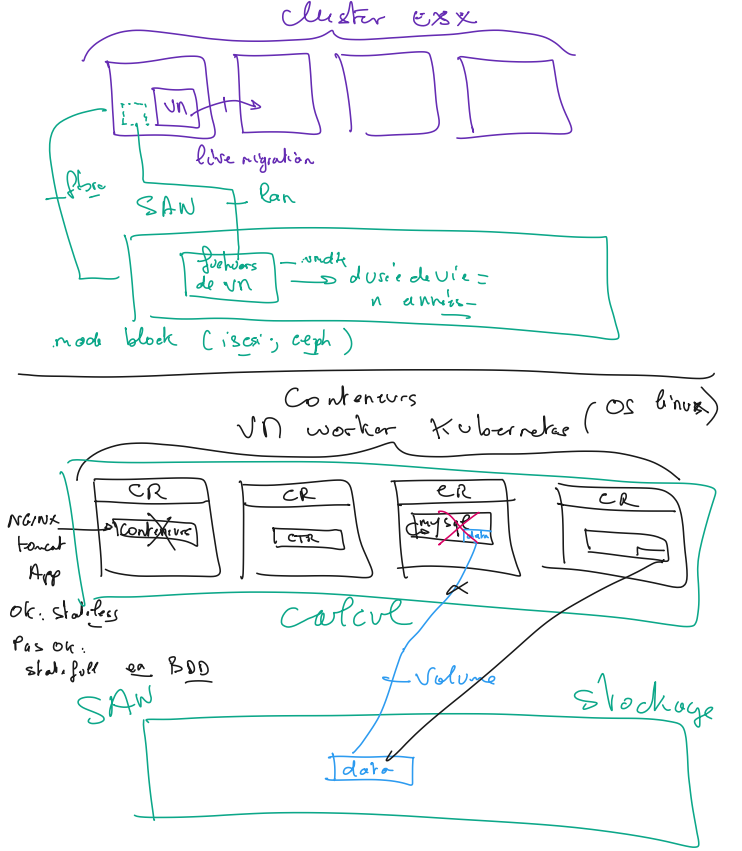
Ubuntu → apt

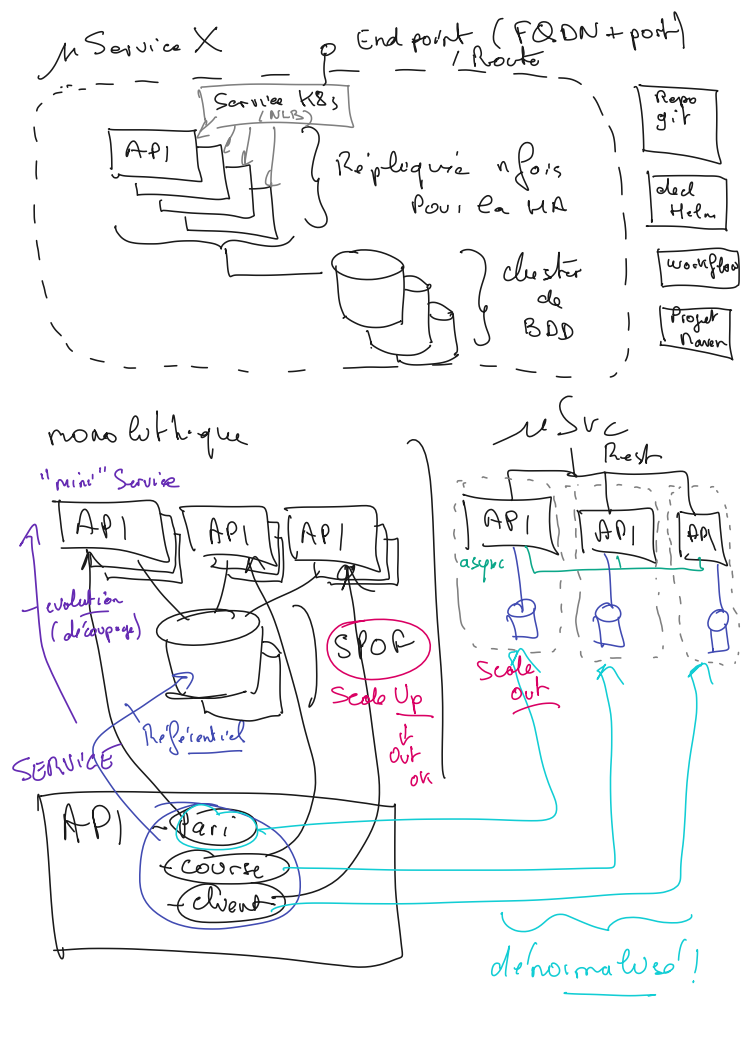
Centos → yum

Java → maven

.net → Nuget


internet
Référentiels publics
→ contenu communautaire
→ contenu officiel
→ fiable (possibilité de CVE)
CVE
Valider la conformité
équipe de sécurité
Référentiel privé
(Nexus)
remplacer
images vérifiées
Dockerfile
Dev
build (CI)
App
→ Conforme
helm update
cluster (Prod)


cluster esx
VM
live migration
fibre
SAN
lan
fichiers de VM
vmdk
disque de vie = n années
mode block (iscsi ; ceph)
Conteneurs
VM worker Kubernetes (OS linux)
CR
CR
CR
CR
NGINX
tomcat
App
Conteneur
CTR
mysql
data
ok : stateless
Pas ok : stat. full en BDD
calcul
volume
SAN
stockage
data


µ Service X
Endpoint (FQDN + port)
/ Route
Service K8s
(NLB)
API
Réplication n fois
pour la HA
cluster
de
BDD
Repo
git
deck
Helm
workflow
Projet
Maven
monolithique
"mini" Service
API
API
API
évolution
(développement)
SPOF
Scale Up
⇓
out
on
Référentiel
SERVICE
API
Tari
course
Event
µSvc
Rest
API
API
API
async
Scale
out
déformation!

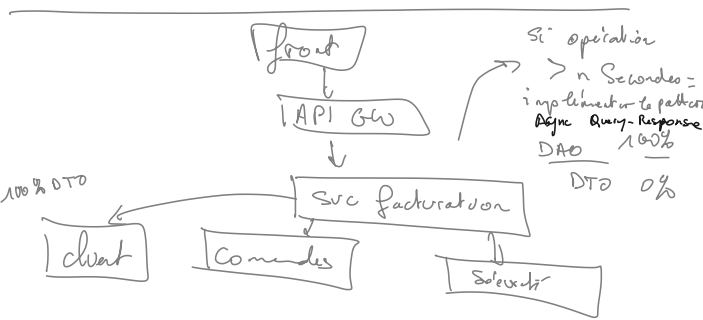
Tutoriel OpenShift gratuit en ligne :
https://developers.redhat.com/courses/openshift/getting-started

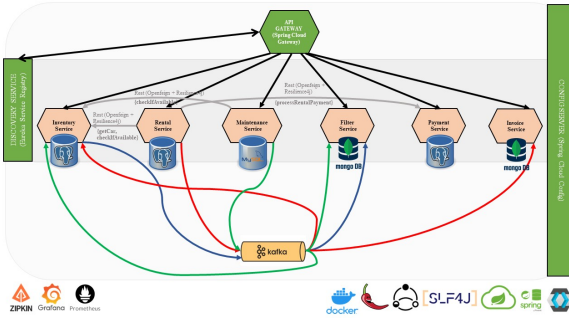
Procédure pour déployer si vous utilisez le cluster Azure :

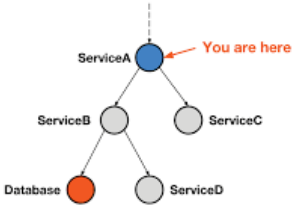
1. Allez dans la perspective Developper
2. Cliquez sur Project / new Project, donnez un nom
3. Cliquez sur Topology
4. Clic droit dans la surface, Add->Container Image
5. Choisissez une image disponible sous Docker, par exemple : nboost/springbootdemo
6. Spécifiez le port auquel l'image écoute les requêtes (ex 8080 pour springbootdemo)
7. Cliquez sur Create
8. Une fois le déploiement Up, ouvrez la route (Détail du déploiement, ou petite icône en haut à droite du cercle sur Topology)

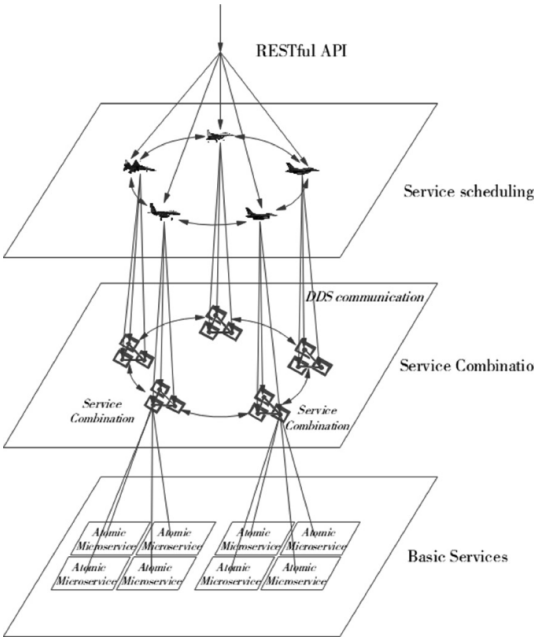
! Openshift interdit l'execution de conteneurs root. Des images officielles (par exemple nginx) ne tourneront pas. Par exemple, utilisez
quay.io/openshifttest/nginx-alpine
nginxinc/nginx-unprivileged
Ce ne sera donc pas le port 80 (exploitable que par root)

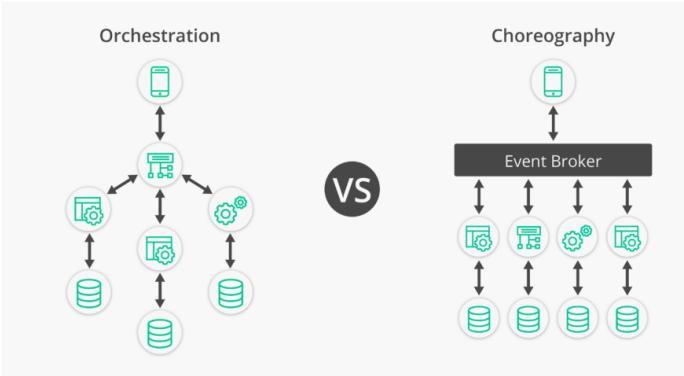
Autres tuto généraux : https://developers.redhat.com/learn/openshift

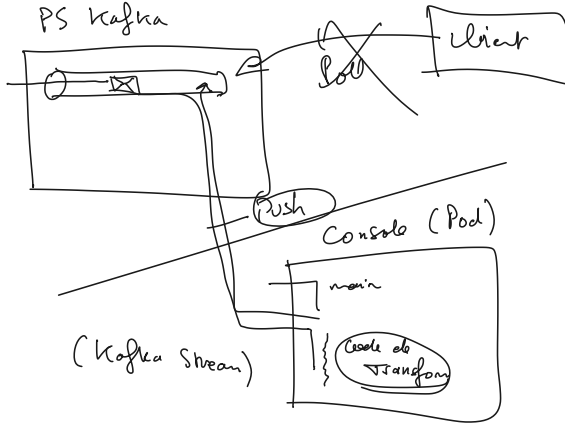



API GATEWAY (Spring Cloud Gateway)
Eureka Service Discovery
CLOUD SERVICES (Spring Cloud Config)
Inventory Service
Rental Service
Maintenance Service
Fibre Service
Payment Service
Invoice Service
kafka
mongo DB
mongo DB
ZIPKIN
Grafana
Prometheus
docker
SLF4J
spring


ServiceA
You are here
ServiceB
ServiceC
Database
ServiceD


RESTful API
Service scheduling
DDS communication
Service Combination
Service Combination
Service Combination
Atomic Microservice
Atomic Microservice
Atomic Microservice
Atomic Microservice
Atomic Microservice
Atomic Microservice
Atomic Microservice
Atomic Microservice
Basic Services


Orchestration
VS
Choreography
Event Broker


PS kafka
Pod
Client
Push
Console (Pod)
main
(Kafka Stream)
Create obj Transform