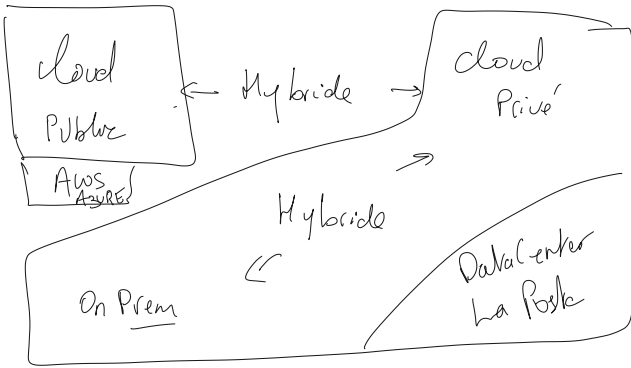
Bonjour tout le monde

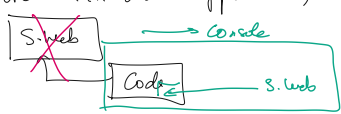


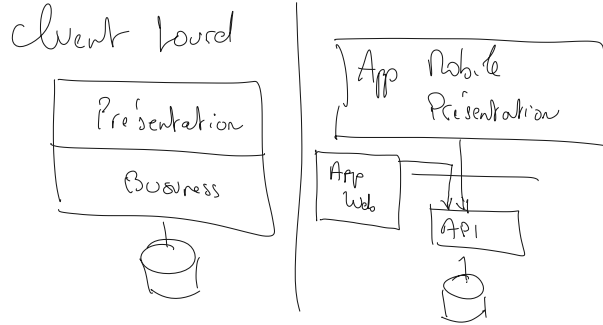
→ Gouvernance de données
→ Montée en charge

La Poste

Paas Conteneurs
RedHat
Open Sh. ft
(cloud Plateforme)

cloud Privé (XaaS)
→ VM
→ Serveur de BDD
→ " de fichiers
etc... (en développement)
(Pas OpenStack)





Avant (n-Tier)

front → Présentation - HTML

middle → Business → Contraintes
↘ Règles de gestion

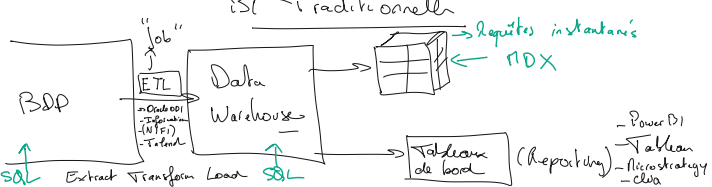
back → BDD

Maintenant { "back" = API - DAO
BDD : Storage

"Haute Disponibilité" = - Tolérance de Panne
- Répartition de charge

↳ objectif pour les Applications critiques -

SLA (Service Level Agreement)



Règle des 3V

- Vitesse
- Volume
- Variabilité (du schéma)

→ Si 1 des règles est "mise à mal"
Les alors une BDD relationnelle n'est plus adaptée (Oracle, SqlServer, MySql, etc...)

MapReduce → Plateforme analytique
Via Hadoop (Kit d'outils)

Non Propriétaire

Cloudera
Horton works
→ Distribution hadoop pour éviter une installation à la main

→ Simplification de la plateforme en SaaS
→ DataBricks (.com → SaaS)

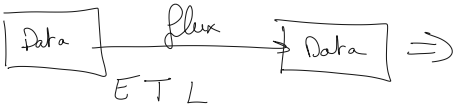
Hadoop sur Azure = HDInsight
DataBricks sur Azure Databricks Community Home

Editeur / Exécuteur de code via le Web = Jupyter Notebooks

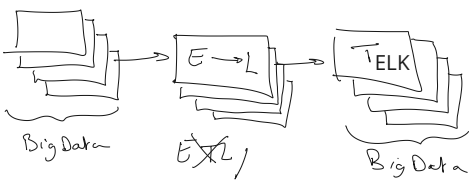
https://community.cloud.databricks.com/login.html

ETL → Job, batch - 1 execution = 1 lot de data

⇒ Devoir travailler en flux Continu, "Presque temps réel"



⇒ Big Data



→ ~~Redis~~ ~~Cassandra~~

→ Mongo Db

Elastic Search ✓

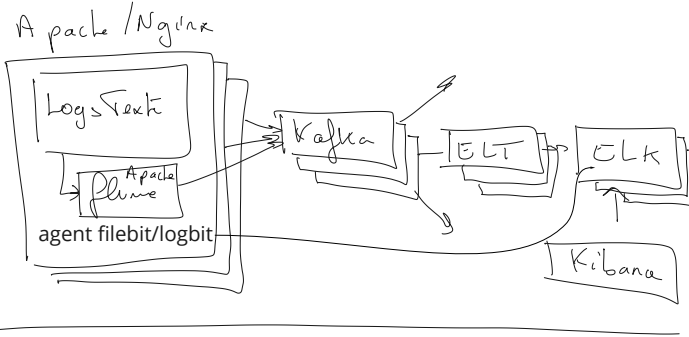
↳ Code

ELT

Extract Load Transform

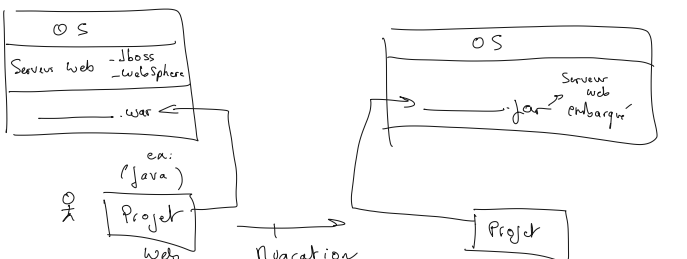
⇒ Apache NiFi

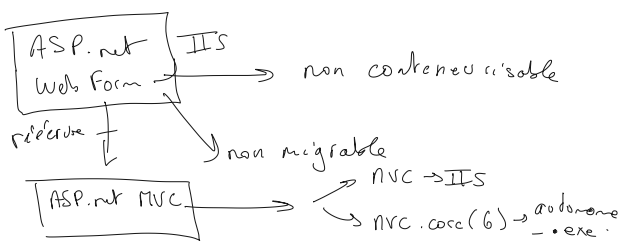
ELK Extract



Conteneurisation d'App Web

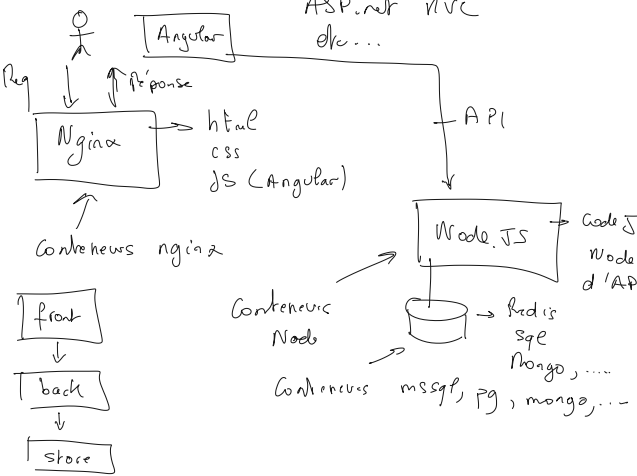
Virtualisation : ⟶ Conteneurisation

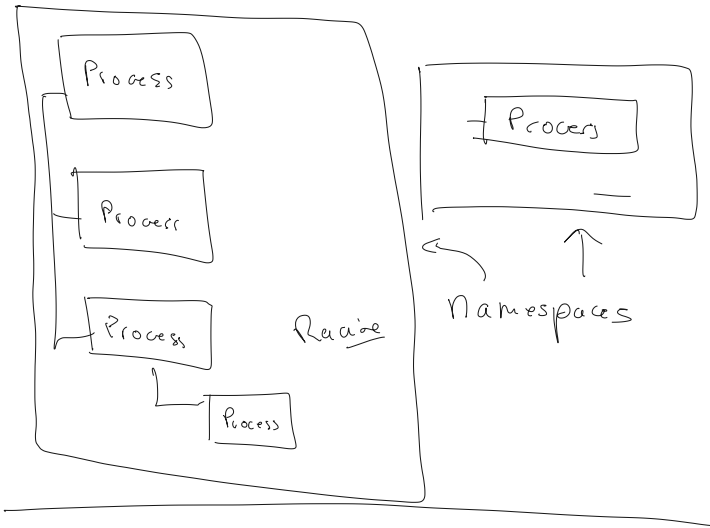


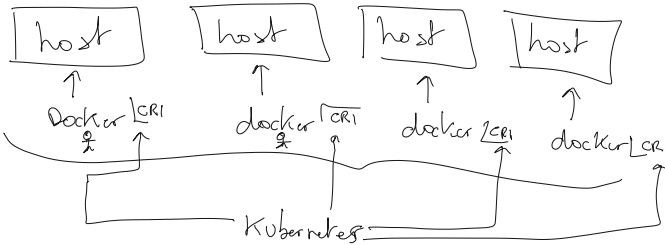


front → MVC → client (Angular)
↓ ou
→ Serveur (Spring boot)

back → API → JS (Node.js)
Python (python)
Spring boot
ASP.net MVC
etc...



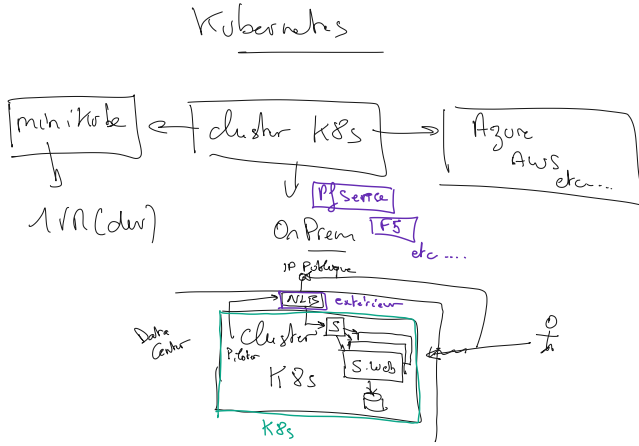




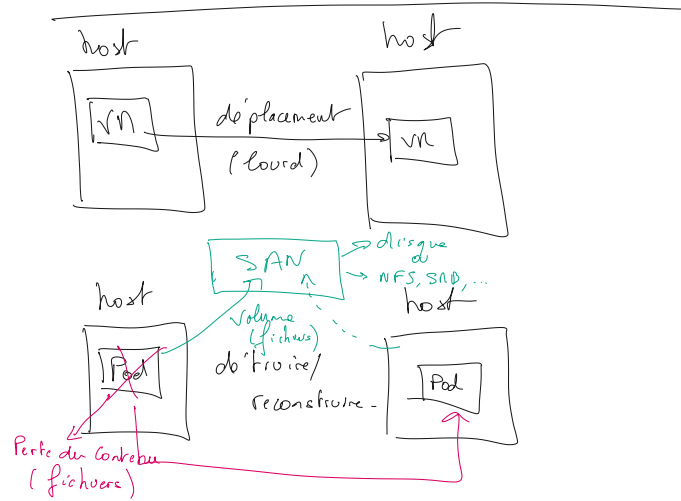
CRI → containerd (docker)

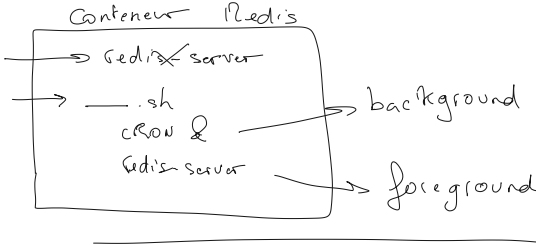
CRIO
RunC
Podman

Docker → image (oci) Standard
↓
Kubernetes → runc



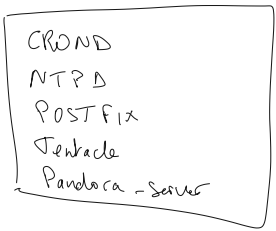
Mercredi matin : démo cluster Kubernetes sous Azure avec test de charge, et augmentation des réplicas.



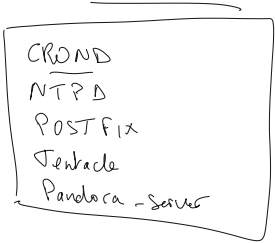


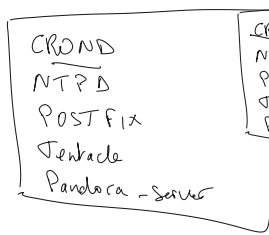
Pandora FMS

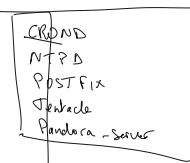
Estado actual. Réplica = 1

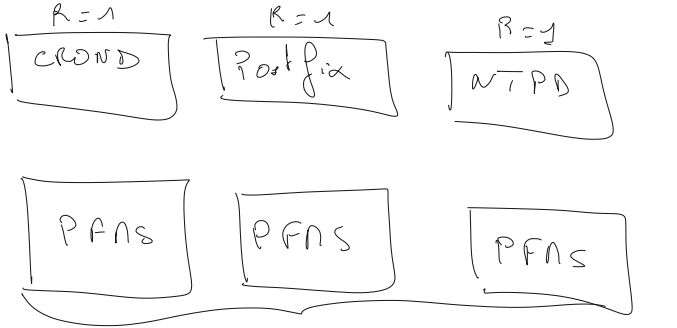


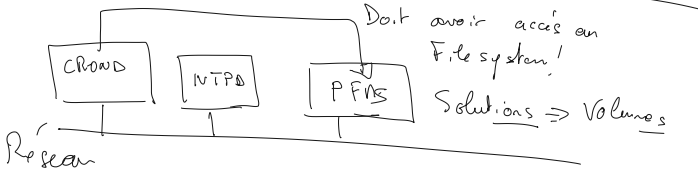
Réplica = 3:









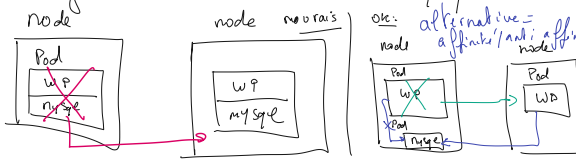


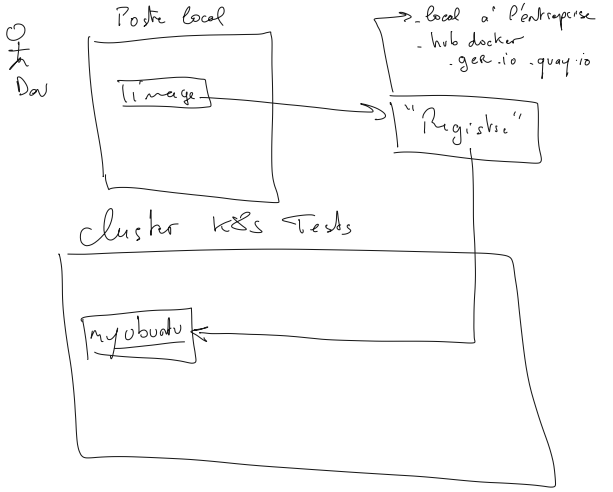
Docker → Conteneur

Kubernetes → Pod (Un Pod encapsule 1..n Conteneurs)
n Conteneurs par Pod est une mauvaise pratique

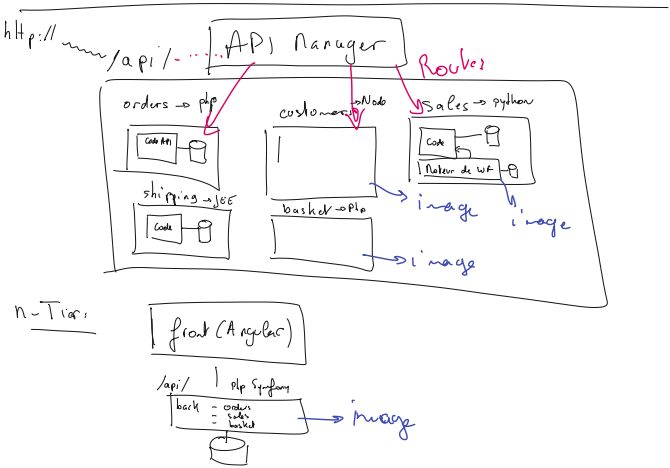
1 Pod est sur un Node

ex : front WP + base MySQL





Slide des architectures microServices

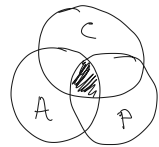


Objectifs : Système scalable
Haut rendement Disponible

→ Si la BDD est un goulet → c'est le maillon faible

Solutions : Monter un cluster de BDD

Rappel de concept :



- Soit cohérent, pas HA
- Soit HA, éventuellement cohérent

choix à prendre -

→ NVCC

BDD Relationnelles → pas adaptées en AP

→ partir sur du NoSql

① choisir le type d'approche -

- clé - Valeur (Redis)
- Document (MongoDb) — 1 clé → 1 Document (json par ex.) → peut exploiter le contenu du document -
- Graph
- Column Store (Cassandra)

...

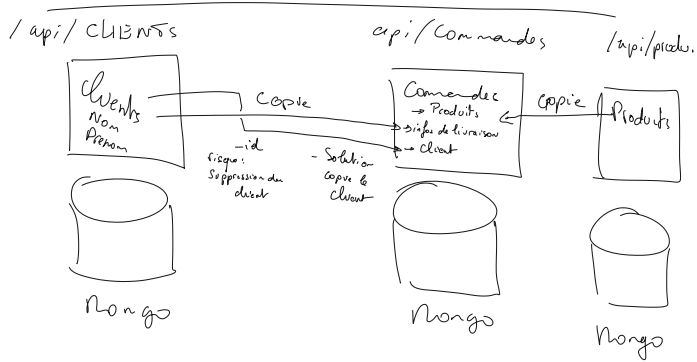
Il n'existe aucun fonction de contrainte d'intégrité référentielle dans les bases NoSql -
Il n'y a pas de jointures, mais des recherches (Lookup)

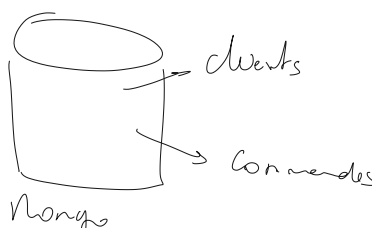
Ex: Cassandra est un "Equivalent" de Tables et SQL
- Pas de jointures, relations.
- Append Only — Insert, Update — Pareil

Ex2: MongoDb est une collection de docs. JSON

ex: Clients (collection 1) → schémas peuvent évoluer

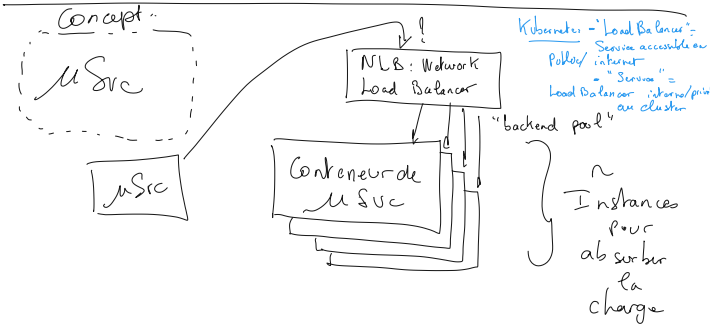
Commande (collection 2) →

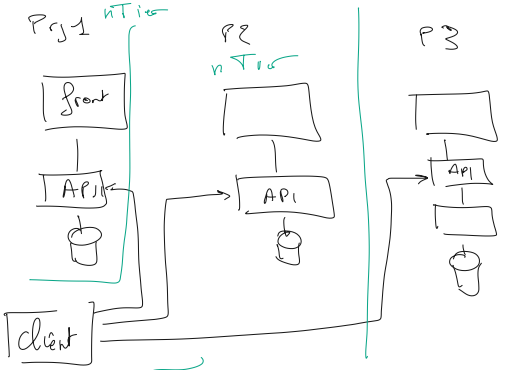




Fo de l'API. (Gateway)

- point d'entrée unique
- départ de la sécurité (Auth.)
  - consommateurs, "facturation"
- ~~WAF~~
- Routage selon des règle
- Gestion de Quotas
- Service de documentation / recherche d'API (Swagger)
- Gouvernance
- gestion centralisée des certificats -
- Reporting / Dashboard opérationnel -
- Point central des APIs -



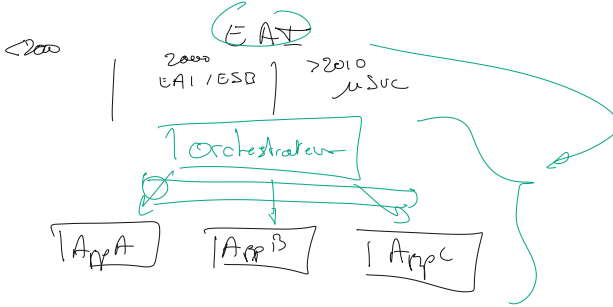


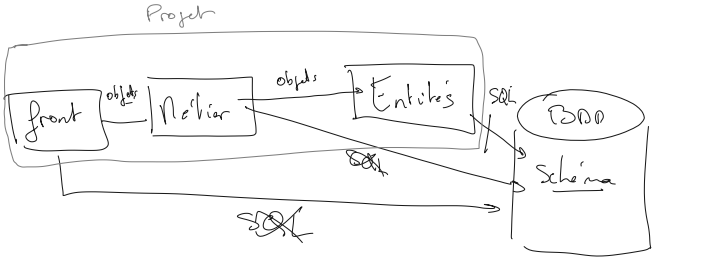
→ Pas 1 ensemble de Services découpés, découplés, qui Travaillent ensemble

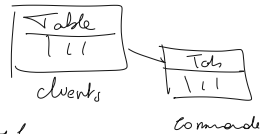
→ Pas de Pattern Pub Sub

→ n Silos au lieu d'une solution Complète

≠ Non, pas une archi en µSvc



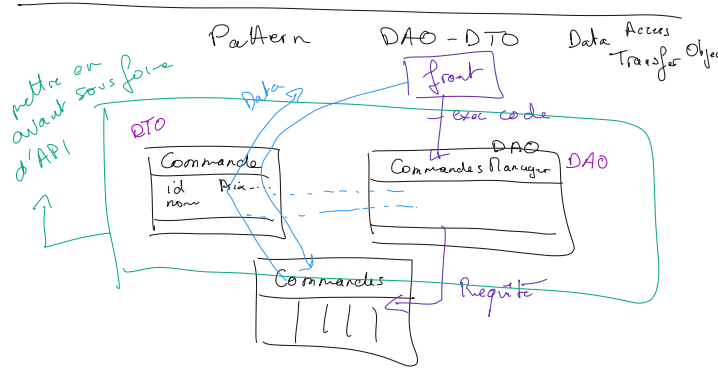




Entité = Terme de qualification de classe/instances ayant pour rôle de correspondre/"mapper" à/de Table

ex : Table Commandes → classe Commande

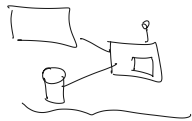
```
class Commande {
    int id;
    string adresse;
```



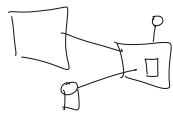
ex.
Kubectl → créés manuellement (impératif)
ag → du contenu

① —

② générer le déclaratif

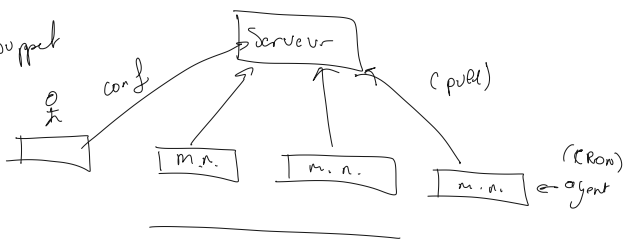


③ appliquer

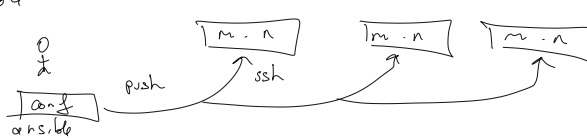


---

chef / puppet



Ansible



Ag vm create
vagrant create
création de VM ... (1) →

(coque extérieure)

VM

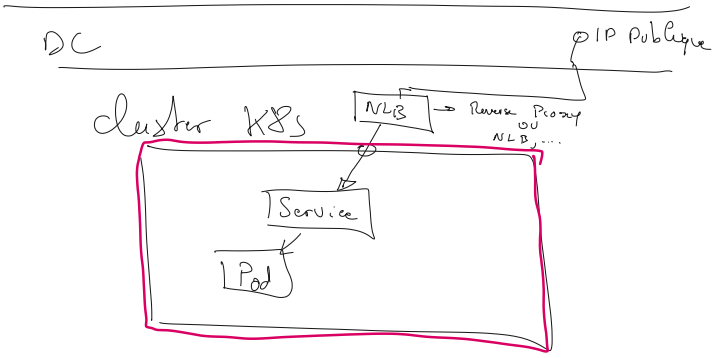
Paquets
virag
fichiers users
etc...

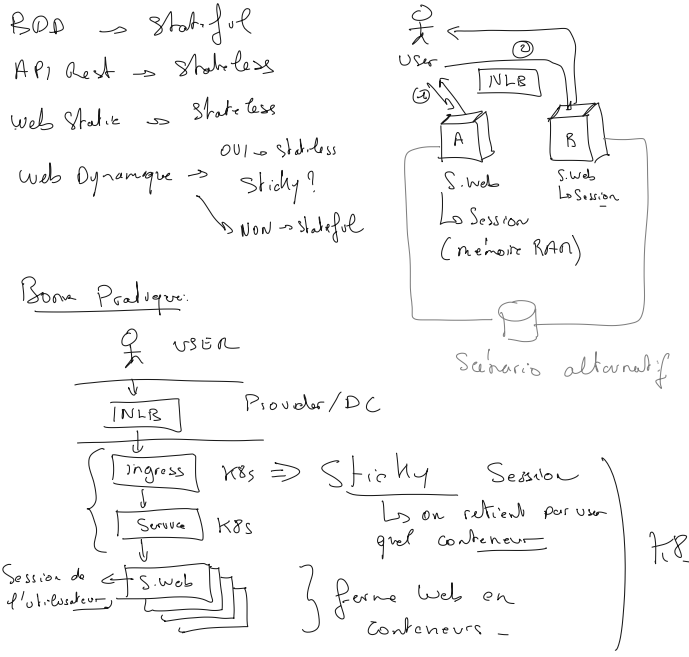
I IaaS (API)
(python)

↓ enchaîner les

(2) ——— yaml (Ansible) → configurer le contenu

Ansible + modules = job
↳ de base (conf d'OS)
↳ extensions
↳ utilisateurs (que vous faites)





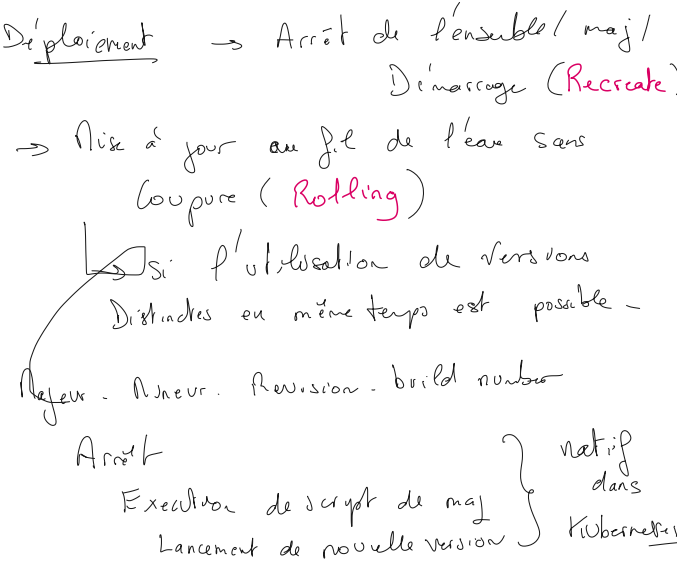
CI/CD
Continuous Integration
Continuous Deployment / Delivery

Français :
IC / DC ou IC / LC
Livraison Continue Déploiement Continu

Types de tests :

• Accessibility testing
• Acceptance testing
• Black box testing : tester un système inconnu.
• End to end testing : tester un workflow fonctionel
• Functional testing : s'assurer qu'il fait exactement ce qui était prévu.
• Interactive testing : eq tests manuels
• Interface tests
• Integration testing : test en environnement équivalent à la prod dans son ensemble.
• Load testing
• Non functional testing : catégorie de tests d'interface (conformité), accessibilité, performance, ...
• Performance testing : recherche des meilleurs métriques, benchmark.
• Regression testing
• Sanity testing : contrôle que les bugs ciblés sont bien corrigés
• Security testing : contrôle face aux menaces
• Single user performance testing
• Smoke testing : valider les fonctions critiques d'un système, envers la stabilité.
• Stress testing : test selon des conditions exceptionnelles de charge, au dela des specs.
• Unit testing
• White-box testing : teste les entrées sorties d'un logiciel bien connu, concernant le design, l'utilisabilité, la sécurité, la stabilité.

3 Ways to Test
There are 3 ways you can do testing.
• Manual testing is the most hands-on type of testing and is employed by every team at some point. Of course, in today's fast-paced software development lifecycle, manual testing is tough to scale.
• Automated testing uses test scripts and specialized tools to automate the process of software testing.
• Continuous testing goes even further, applying the principles of automated testing in a scaled, continuous manner to achieve the most reliable test coverage for an enterprise. Keep reading to learn more about the differences between automated testing vs. manual testing and how continuous testing fits in.
• BDD
 • Frameworks
  ○ Cucumber (most popular)
  ○ Concordion.Net
   § Small open-source C#
   § Extends nunit
  ○ Concordion
   § Java based
  ○ BDDfy
   § Simple to use, compatible with test runners
   § . Net
  ○ Specflow
   § Part cucumber familly for. Net, Gherkin parser, most used in. Net
  ○ Gherkin : est le langage (DSL) : given, when, then
  ○ Quantum
   § Java based
  ○ Jbehave
   § Java based
  ○ Codeception
   § Php based
  ○ Zephyr scale
   § Free
   § Ruby, Java, Javascript, and C# (SpecFlow).
  ○ JDave : on top of JUnit
  ○ TestLeft
   § .NET, C#, Java, Jenkins, and more



https://nvd.nist.gov/

Manières de livrer et valider des nouvelles versions :
https://blog.christianposta.com/deploy/blue-green-deployments-a-b-testing-and-canary-releases/