Cours CLOUD

1 - Introduction

- Ressource sous-utilisée doit être louée aux demandeurs
- Augmentation taux d'utilisation → rentabilité
- Réduction des coûts et des pertes
- IT Outsourcing = mettre ses infos dans le cloud (entreprise ou personnel)

- Premier service : Amazon EC2 (Elastic Computer Cloud)
  - Vente de machines virtuelles
  - Ressemble à des serveurs dédiés

- Cloud = ensemble de ressources / apps / services qui s'exécutent dans un environnement distribué et accessibles à travers des protocoles web standards
- Services généralements fournis :
  - Pay as you go
  - Illusion de ressources infinies (scalability)
  - Abstraction de l'infra hardware
  - Mutualisation entre plein d'utilisateurs

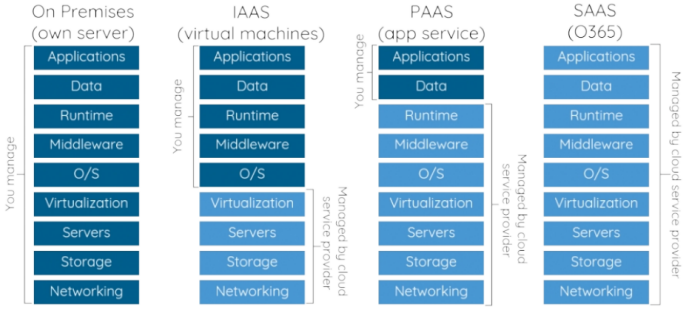
- Grid computing VS Cloud computing :
  - but de grid : diviser des tâches en sous-tâches indépendantes et assigner une tâche par machine
  - Sys distribué pour le partage collaboratif de ressources ≠ computing basé sur des ressources virtuelles
  - grid plutôt utilisé par la recherche ≠ cloud pour les entreprises & particuliers

- Rôles dans le cloud
  - Cloud providers = donne des infrastructures hardware et met des services au dessus (ex: AWS, Microsoft Azure)
  - Cloud clients = utilise les ressources des cloud platforms (ex: entreprises, particuliers)
  - Cloud resellers : construisent et vendent des services en se basant sur des plateformes cloud existantes (ils sont à la fois providers et clients) (ex: Scalr)
  - Cloud developers: produisent des outils (déploiement, réparation automatique) pour le cloud (ex: VMware, labos de recherche et entreprises)

- Bénéfice principal : pay as you go
  - Allocation / désallocation à la demande des ressources
  - pas de proc administratives
  - Accessibles partout 24/7
  - TCO (Total Cost of Ownership) réduit
  - pas d'investissements importants
  - pas de staff, pas d'infra locale, moins de licenses sofware à payer
  - Facturation à l'utilisation

- Classification par propriété
  - Community cloud = partagé par plusieurs organisations
  - Private cloud = créé par une entreprise pour son utilisation interne
  - Public cloud = créé par une entreprise pour faire du business et ouvert à tous (ex: AWS)
  - Hybrid cloud

- Classification par service
  - IaaS (Infra as a Service) = donner du stockage et des solutions de computing. Les users peuvent louer des machines ou VM (ex: AWS EC2)
  - PaaS (Platform as a Service) = donner une plateforme pour construire et exécuter des apps (ex: Google App Engine)
  - Saas (Service as a Service) = le cloud donne directement l'app dont l'utilisateur a besoin (ex: Gdoc)

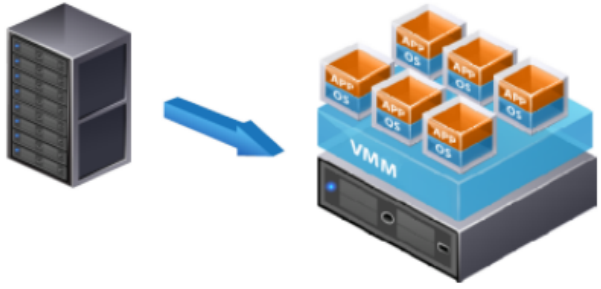


- Challenges du cloud :
  - Sécurité et confiance (où data stockée? ; vie privée ? ; lois du pays ?)
  - Service garanti
  - Impact énergétique (1.8% de l'électricité mondiale)
  - Monitoring précis pour la facturation
  - Standardisation
  - Customisation (hardware)

- Le cloud doit être élastique
  - fluctuation de la demande : les apps sont globalement sous-utilisées
  - Apps élastiques pour le client : éviter le surbooking, allouer des ressources à la demande
  - Optimiser l'infra pour le provider

- Applications élastiques
  - Augmenter sa capacité en ajoutant des machines
  - Idem en enlevant
  - L'app doit s'adapter en fonction de la charge
  - Ex: serveur web répliqué avec load balancer

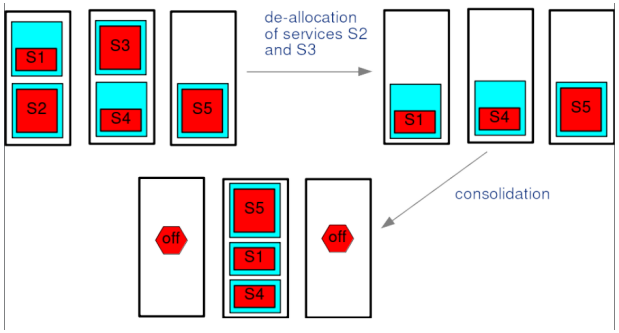
- Virtualisation : principe



- Challenge : isolation
  - Sécurité : protéger des attaques depuis d'autres VM
  - Performance : ne pas affecter les autres VM
  - Failure

- VM ⇒ Elastic cloud
  - Peuvent être bougées entre plusieurs machines physiques (migration)
  - Les machines non utilisées peuvent être coupées
  - Possibilité de surbooking

- Consolidation



2 - Virtualisation

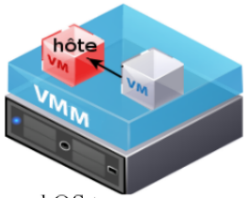
- Vente de machines en IaaS
- Définition : ensemble de techniques &/ou logiciels qui permettent de gérer plusieurs OS sur une seule machine. Ex: Xen, VMware, HyperV
- Avantages : backup & recovery ; migration

- Types de virtualisation

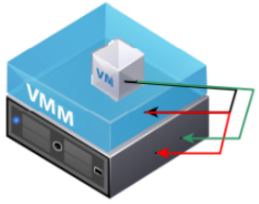
  - Full virtualisation : l'OS exécute les apps au niveau utilisateur. Ex : VMM. Chaque instruction de la VM est émulée par le VMM. L'OS exécuté sur la VM n'est pas modifié et peut être de tout type. Ex: Virtualbox. Concrètement : hardware simulation (plutôt lent)

  - OS level virtualisation : l'OS hôte inclut tous les mécanismes pour construire des containers isolés (VMs). Ces containers partagent le même OS (host). Ex: openVZ, chroot, Docker Concrètement : l'OS est modifié pour gérer plusieurs instances de lui-même ; code natif (efficace) ; isolation partielle et 1 seul type d'OS

  - Para-virtualisation (PV) : la VMM remplace l'OS hôte et se comporte comme un proxy pour accéder au hardware. L'OS est considéré comme une VM (privilégiée) et est utilisé par le VMM pour exécuter des tâches précises. Contraintes : les OS des VM doivent être modifiés. Ex: Xen, VMware
Concrètement : un mix entre simulation et code natif. Plusieurs types d'OS ; chaque VM a des ressources hardware associées et le VMM contrôle leur accès

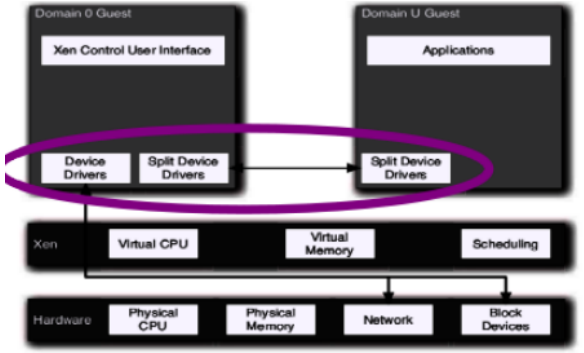


  - Hardware assisted virtualisation(HVM) : le hardware est au courant de la virtualisation. Les OS n'ont pas à être modifiés. Ex: Xen, VMware
Concrètement : natif sans modification d'OS; Le hardware peut gérer plusieurs VM ; des fois la paravirtualisation est plus rapide et flexible

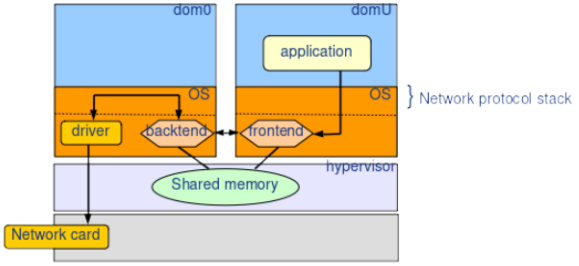


- Etude de Xen
  - Open source VM system (AWS l'utilise)
  - Superviseur (VMM), un peu comme un kernel linux
  - 3 méthodes de virtualisation : PV, HVM, combinaison des 2 (PV-HVM)

- PV avec Xen
  - Au démarrage d'une VM : le BIOS donne au kernel les infos du hardware (taille de mémoire par ex) ; le kernel est chargé en mémoire ; il initialise les structures de données ; le programme "init" est lancé
  - Challenges : comment donner aux différentes VM des versions différentes du BIOS ? comment donner aux VM une mémoire fixe alors qu'elle peut être fragmentée ? Comment faire en sorte qu'une VM ne puisse pas accéder à la mémoire des autres ?
  - On assigne des vCPUs à chaque VM ; les vCPUs sont programmés sur le CPU physique
  - Le VMM donne des pages virtuelles aux VM (illusion d'une mémoire continue) ; P2M (Physical to Machine) mapping
  - Split driver model : front-end dans la VM implémente un faux driver → back et front agissent comme un programme client-serveur

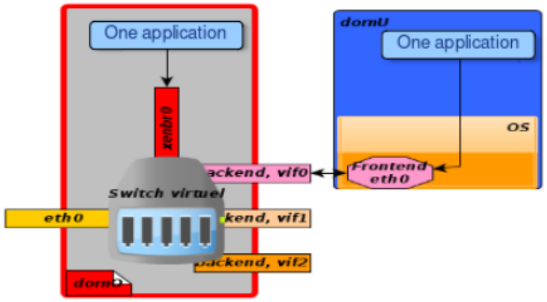


- Networking avec Xen
  - Le vrai driver est installé dans dom0
  - Le back-end reçoit et envoie des paquets à / depuis le driver
  - Interopération par 2 mécanismes : mémoire partagée & signaux



- 3 configurations network :
  - Route mode : IP routing entre les VM
  - NAT mode : traductions d'adresses en IP locales
  - Bridge mode : gestion au niveau MAC

- Bridge mode networking avec Xen
  - Chaque interface MAC a un représentant dans dom0
  - Redirection vers le back-end de destination
  - Les VM sont accessibles depuis dehors

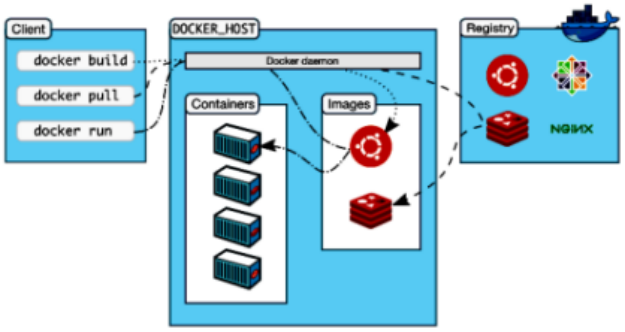


- Démarrer une VM Xen
  - Installer Xen sur la machine physique
  - Construire l'image de la VM
  - Editer la conf de la VM (# of vCPUs, quantité de RAM, network interface)
  - Utiliser la commande Xen pour démarrer la VM

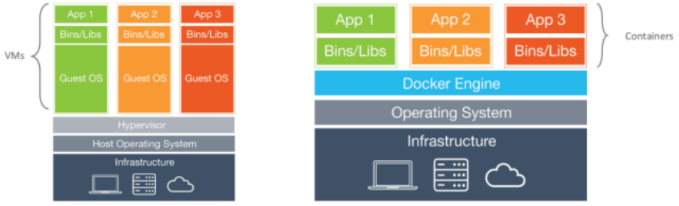
3 - Docker

- Généralités
  - Construction de VM très légères (containers)
  - Virtualisation au niveau OS
  - Outils user-friendly pour gérer les containers
  - Utilisé pour l'intégration continue
  - Pas de migration en temps réel
  - Open source
  - Un container contient tout ce qu'il faut pour exécuter du une app
  - Containers isolés les uns des autres
  - Communication possible entre plusieurs containers grâce à des canaux prédéfinis

- Architecture client-serveur
  - Registre = réservoir d'images. Peut être local ou distant
  - Client = shell
  - Host = coeur du système : créer des VM



- Image docker
  - Union File System : image = succession de couches ; chaque couche décrit une modification du file system
  - Peu de données pour créer un file system à partir d'une image ; efficace
  - Les images standard peuvent être réutilisées
  - Une modif ne génère pas un nouveau file system en entier, juste une couche



- Un container contient l'application et toutes ses dépendances mais partage le kernel avec les autres
- Tourne comme un processus isolé
- Lié à aucune infra : peut tourner partout dans le cloud

- Commandes utiles
  - Installation : `wget -qO- https://get.docker.com/ | sh`
  - Lancer un container :

```
docker run -it ubuntu bash
```

start flags image application

- Voir les images locales :

```
docker images
```

```
hagimont@hagimont-pc:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              latest              cd6d8154f1e1        12 days ago
84.1MB
hagimont@hagimont-pc:~$
```

- Chercher une image dans le hub :

```
docker search hagimont
```

```
hagimont@hagimont-pc:~$ docker search hagimont
NAME                      DESCRIPTION                 STARS
        AUTOMATED
hagimont/docker-whale                                 0

hagimont/hagi             my repo                     0

lwapet/projet_docker      ENSP - hagimont Daniel      0
```

- Voir les containers : docker ps -a
- Créer une image depuis un container : docker commit <hash> <repo>:<name>
- Créer une image avec un Dockerfile

```
# This is a comment
FROM ubuntu
RUN apt-get update && apt-get install -y apache2
```

Then docker build -t <name>:<version>

- Communication entre les containers : créer un bridge
  - docker network create <name>
  - Démarrer une VM avec accès à ce bridge : docker run -d -net <networkname> <imagename>
  - Redirection de port : exemple

```
docker run -d -p 80:5000 hagimont/apache
```

Any connection on port 80 of the host is forwaded to port 5000 of the container

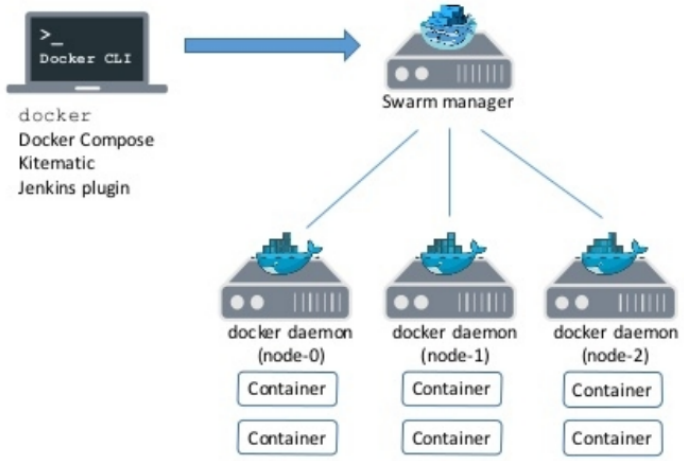
- Ecosystème:
  - Docker machine : installer des hôtes docker sur un réseau
  - Docker compose : définir et lancer des apps sur plusieurs containers
  - Kitematic : interface graphique pour administrer des hôtes docker
  - Docker swarn : gestion de cluster d'hôtes docker (réplication, load balancer, recovery, …)

- Docker compose
  - Fichier YAML pour configurer les apps et services sur chaque container
  - Une seule commande ⇒ Tout créer et démarrer
  - Permet de rendre les environnements répétables, isolés et rapides

- Usage principal : intégration continue
  - Techniques qui permettent d'accélérer la livraison d'un software en réduisant son temps d'intégration
  - Vérification et compilation de code
  - Exécution de tests unitaires
  - Livraison de versions de test
  - Génération de rapports automatiques
  - Outils : Anthill, Atlassian Bamboo, Build Forge, …
  - Le container capture les dépendances

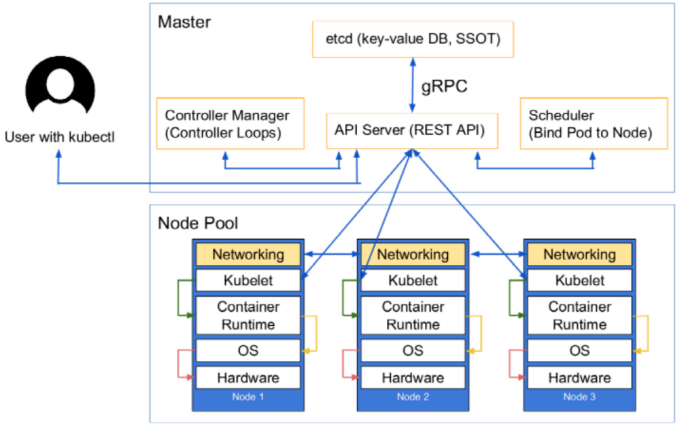
4 - Kubernetes et Openstack

- Docker Swarn : solution native de docker pour faire du clustering
  - Le cluster devient un unique virtual host
  - Permet de gérer et programmer des containers sur un cluster



- Spécifités
  - Il faut ouvrir une connexion TCP sur chaque noeud pour communiquer avec le Swarn Manager ; Installer docker sur chaque noeud

- Kubernetes = container d'orchestration de système ; abstraction de l'infra physique grâce au concept de noeud
  - Abstraction de milliers de noeuds dans un cluster
  - Donne des méthodes industrielles pour gérer les apps
  - L'admin définit le "desired state" et Kubernetes convertit le "current state" en "desired state"



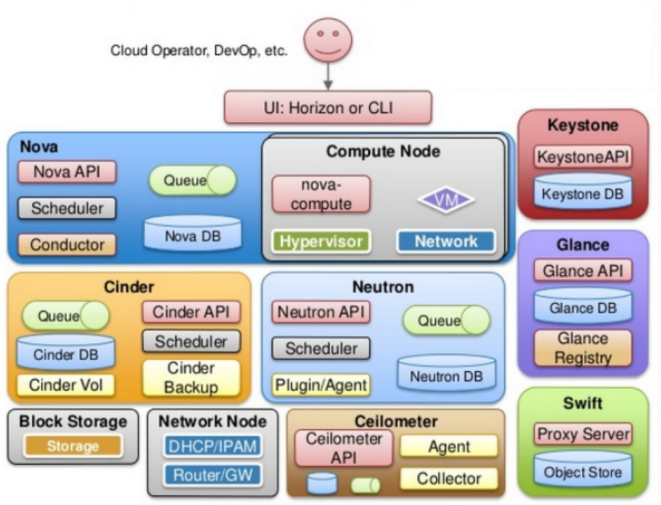
  - etcd = stocke toutes les informations et les autres services lisent et stockent dedans
  - scheduler = choisit le container et y place le bon noeud
  - controller manager = vérifie les statuts des noeuds
  - agent Kubelet = écoute les requêtes du maître

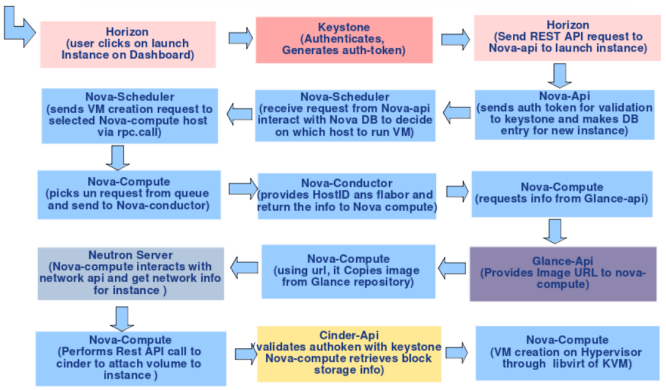
- Concepts de Kubernetes
  - Pods = groupe d'un ou plusieurs containers qui partagent du stockage, un réseau et une spec sur comment faire tourner les containers. Représente une **app Kubernetes**
  - Déploiement = donner des mises à jour déclaratives pour les pods. Décrit un "desired state"
  - Services = une manière abstraite d'exposer une app qui tourne sur des pods
  - Namespace = clusters virtuels qui tournent sur le même cluster physique

- Fonctionnalités de Kubernetes
  - Self-heating = Kubernetes redémarre les containers qui plantent, les remplacent, les tuent, bref ils ne les exposent pas au client si ils ne sont pas prêts à servir
  - Automatic bin packing = gestion des containers pour donner du CPU ou de la RAM adapté aux besoins des utilisateurs
  - Automated rollouts and rollbacks = automatisation de la mise en place d'un état désiré. Par ex, automatisation de création de containers pour des déploiements
  - Service discovery and load balancing → rendre le déploiement stable en distribuant le trafic
  - Storage orchestration = monter un stockage système automatiquement

- Kubernetes VS Swarn
  - Supporte de grosses demandes et plus complexes ≠ Swarn = solution plus simple et rapide pour démarrer

- OpenStack = cloud OS





5 - Services pour le cloud

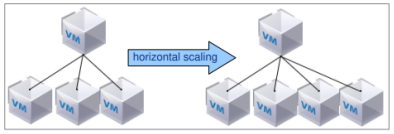
- Donnés pour faciliter la gestion d'apps cloud : développement, création de VMs, déploiement, stockage, admin
- De nombreuses formes : Modèles de prog ou d'intégration (PaaS) ; plugins pour les IDE et test dans le cloud ; API REST qui permettent d'effectuer des actions dans le cloud

- Services de création de VM
  - Installation dans une VM standard et sauvegarde de l'image
  - Customisation d'un OS minimaliste
  - Déploiement unifié dans un cloud hétérogène

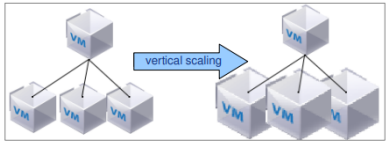
- Services de déploiement
  - Installation d'apps : installation dynamique sur des VM standard ; construction des images de VM
  - Config et lancement d'apps : centralisé ou distribué

- Stockage
  - Données utilisées par l'appli (gérées par le cloud ou par l'utilisateur) ; images ou snapshots de VM
  - Propriétés attendues : durable, sécurisées, accessibles 24/7 et de partout

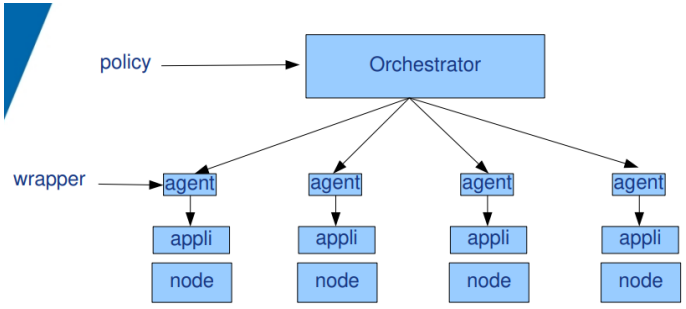
- Admin
  - Monitoring = observation des runtime conditions, détection d'événements particuliers (fail d'une app ou d'une VM, ressource trop utilisée, dégradation de la QoS, intrusion dans une appli, …). Nécessite une communication dans toute l'infra pour rassembler les données. Utilisé pour lancer des reconfigurations
  - reconfiguration
  - tolérance aux fautes : cibler la disponibilité des apps
    2 stratégies principales :
    - anticipation : réplication
    - réparation : détection, restart, backups réguliers
  - scalability ou elasticity = détection de situations d'overload → allocation de ressources supplémentaires
    2 stratégies :
    - allocation de ressources aux VM qui accueillent l'app (vertical scaling)



  - création de nouvelles VM (horizontal scaling)



  - sécurité (ex: Cloudify, Roboconf)



- Exemples de services :
  - Public cloud
    - AWS (IaaS, PaaS, SaaS)
      - EC2 (IaaS)
      - Beanstalk (PaaS) → déploiement rapide d'appli web
    - Windows Azure (IaaS, PaaS, SaaS)
      - Microsoft Dynamic CRM (SaaS) (ex: Xbox, Office)
    - GoogleApp Engine (GAE) (PaaS)
  - Private clouds : openNebula, openStack

- Consolidation
  - Motivation : en moy <10% du CPU utilisé
  - Conslidator = calcule un plan de consolidation qui minimise le nombre de VM utilisées ; exécute le plan ; suspend les VM vides
  - Migration de VM en temps réel
  - Pour savoir quand consolider : prédiction ; planification ; on the fly

- Optimisation : ballooning