Cloud Avancé

Yves Denneulin

Objectifs de ce cours

Faire de vous des connaisseurs des architectures de cloud
  aborder des concepts fondamentaux
  regarder quelques services de base

Donner du recul sur les technologies
  (idéalement) en manipulant dessus dans des conditions réelles
  connaitre les briques technologiques de base et savoir se documenter++

Identifier en quoi le cloud est transformant

Plus axé utilisation que construction
  mais les dimensions performances et sécurité obligeront à regarder sous le capot
    ce sera votre valeur ajoutée sur le marché du travail!

Pourquoi le cloud est possible ?

- Standardisation des services et de leur invocation
  - rôle majeur de l'open source
  - généralisation des APIs dans Amazon
- Amélioration des performances réseau et ubiquité
- Micro-paiement
- Développement des techniques de virtualisation
  - machine, pile logicielle
  - réseau

Le memo de Jeff Bezos (2002)

1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols – doesn't matter.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired.
7. Thank you; have a nice day!

Plan du cours

1. Introduction au cloud
2. Conception et déploiement en cloud (Containers), server less
3. Mesure et optimisation des performances, construction d'un cloud
4. Cloud et sécurité
5. Optimisation énergétique dans les data centers
6. Après le cloud : edge/fog computing

Évaluation : présentation d'articles

Des acteurs majeurs

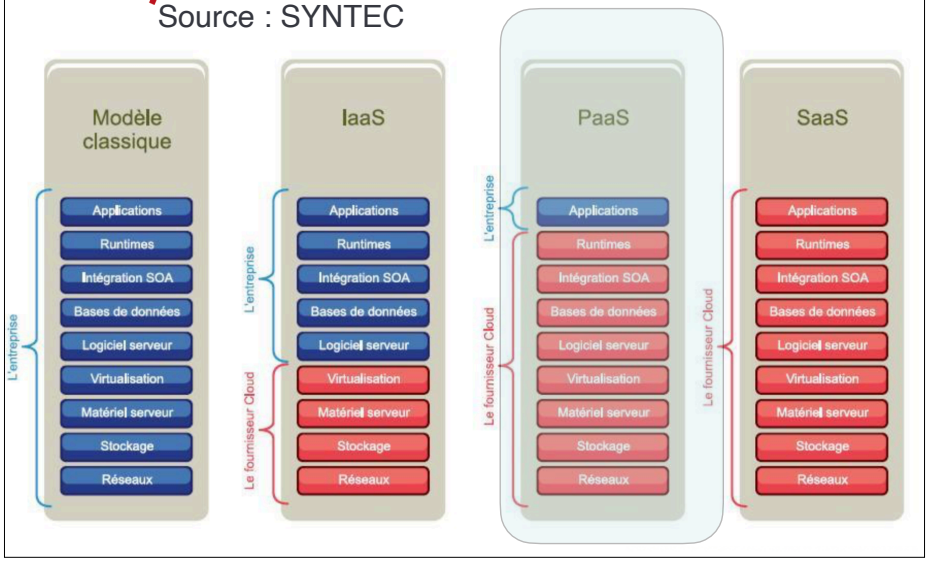
72% du marché cloud entreprises en europe pour AWS, GCP et Azure

européens de 27 à 13% en 5 ans

- AWS : le leader, générique, très évolutif
- Azure : Iaas et PaaS, support++ de Windows
- Google : du SaaS vers l'IaaS
- IBM : très axé cognitive computing, cloud hybride depuis le rachat de RedHat
- OVH : français d'envergure mondiale, historiquement IaaS
- HP : infrastructure pour cloud hybride
- Salesforce, Oracle, SAP : orienté vers leur domaine d'excellence
- ATOS, Cap,… : vente de services
- CleverCloud, Scaleway, etc. : acteurs français, offre incomplète

Vision schématique

Source : SYNTEC



PaaS ou CaaS ?

PaaS
- utilisation d'une pile logicielle déjà configurée
- PhP, Java, Node.JS, …
- développement au-dessus de cette pile
- pas d'administration à prévoir
  - mais de la mise à jour éventuelle

CaaS
- déploiement d'un environnement déjà construit
- permet plus de flexibilité
  - coexistence de versions
- peut nécessiter plus d'expertise

PaaS, quelles offres ?

AWS elastic beanstalk
Tomcat, PHP, Nginx, IIS, …

Salesforce Heroku
PHP, Ruby, NodeJS, Postgresql, …

Google App Engine
PHP, NodeJS, Go, .NET, Ruby

PaaS : avantages

- économies d'administration
  - installation, déploiement, sécurisation
- possibilité de figer une/des configurations
  - container
- flexibilité de déploiement
  - extension/contraction
  - orchestration

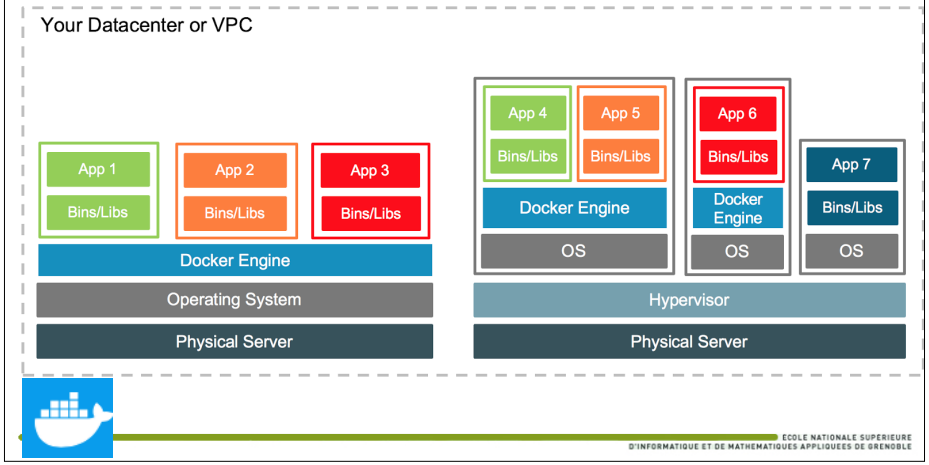
PaaS : inconvénients

- Perte d'autonomie
  - évolutivité de la pile utilisée
  - vendor lock-in (si protocoles non standards) : outils de configuration et d'administration
- robustesse de l'infrastructure matérielle sous-jacente
  - y compris réseau
- coût final (passage à l'échelle, prévisionnel)
- Technique : authentification transversale à la pile

Les containers

Idée : distribuer une application avec son environnement

Passage de l'IaaS au PaaS



Docker

Solution la plus répandue de Container

Composée de
- un constructeur de containers
  - pouvant prendre plusieurs sources
- un gestionnaire de containers
  - exécution, destruction,…
  - utilisent cgroup et namespace de Linux
  - utilisent le copy-on-write
    - permet un grain fin (un processus = un container)
- une interface REST
  - permet une automatisation de la gestion des containers
- un repository de containers

Docker (2)

Une image de container peut être stocké sous forme de
- source -> une liste d'éléments à récupérer pour assembler
  - léger, permet de figer une configuration qui fonctionne
- binaire (image) -> une archive déployable immédiatement

Une image docker est un instantané d'un système de fichiers avec quelques indications supplémentaires

Un container en cours d'exécution est
- un file system (écriture=copy-on-write)
- une pile IP (avec une adresse propre)
- un process group

qui exécute une image de container
- par défaut non interactif

Comparaison container et virtual machines

Virtual Machine
- maison : autonome (branchement électrique, réseau, etc.) et protégé
- grain de base gros : plusieurs chambres, garage, etc.
- à la base une machine complète (OS parfois customisé, …)

Container
- appartement : ensemble des réseaux partagés
  - mais porte d'entrée séparée
- flexibilité dans ce qui est fourni : nombre de chambres, garage,…
- on ajoute les composants qui sont nécessaires

Aspect dynamique
- les containers sont sans état
- pas de mise à jour -> arrêt et redémarrage
  - possible car très rapide (<1s)
- applications : des dizaines de containers qui interagissent
- support pour des microservices

Implémentation des containers

pour bien comprendre à quoi ça correspond
sous Linux (de loin le cas le plus répandu)
s'appuie sur deux éléments fournis par le noyau

namespaces : permet l'isolation
- filtre la vision du système pour chaque container
- ipc, mnt, net, pid, time, user, its,

cgroups : control groups limite l'usage des ressources
- blkio, cpu, cpuset, memory, net, pids
- v2 permet les hiérarchies de processus

Éléments de performance

Les containers partagent beaucoup de ressources (dont le noyau)
+ : faible usage de la mémoire
- : plus de contention potentielle (passage à l'échelle)

Containers plus utilisés pour faire de gros volumes d'I/O et de communication que de calcul
le recouvrement calcul/communication limite l'impact de la contention

L'utilisation d'orchestrateurs peut rallonger les circuits de communication

Partage de mémoire possible entre containers pour des ressources partagées (sécurité ?)

Éléments de performance (2)

Vision fine

Augmentation des défauts de cache (dépend de la configuration noyau)
patch refusé dans le noyau pour cause de sécurité

CPU
attention à la différence entre les environnements de développement (container seul sur la machine) et production (partage possible)
TLB peut être flushé plus souvent
plus d'IT générées par les autres containers

Mémoire
bonne utilisation : partage entre les containers peut être possible (overlayFS)

I/O
suivant la configuration, une couche supplémentaire à traverser

Stratégie de répartition des ressources

CPU
utilisation de cpusets pour restreindre les CPUs utilisables par certains containers => risque de gaspillage de ressources
autre approche : CPU shares (%age de CPU utilisable par un container)
Intel Cache Allocation Technology permet de segmenter les caches CPU entre les containers

Mémoire
mécanismes de partage similaires à ceux des processus

I/O
possibilité de fixer des limites par (groupe de) containers

Observation
attention à la vue depuis le container -> parfois partielle, parfois totale
une vraie analyse de performances doit se faire depuis l'extérieur

Virtualisation légère

On en reparle dans la partie implémentation IaaS

Open Container Interface

https://opencontainers.org/
Organisation visant à la standardisation des containers (interopérabilité!)

Définit trois spécifications
- environnement d'exécution
- format des images
- forme de distribution

Repository github avec l'ensemble des éléments de spécification et des implémentations

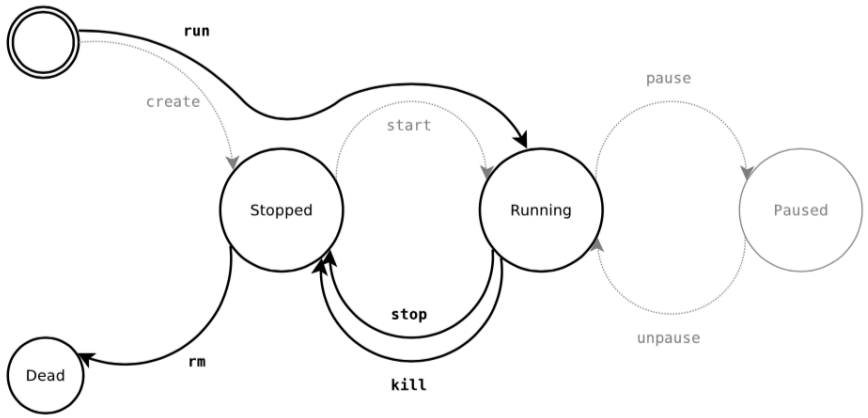
Open Container Interface : format d'une image

« donné » par Docker (https://github.com/opencontainers/image-spec/blob/main/spec.md)

Un manifeste

```
{
    "created": "2015-10-31T22:22:56.015925234Z",
    "author": "Alyssa P. Hacker <alyspdev@example.com>",
    "architecture": "amd64",
    "os": "linux",
    "config": {
        "User": "alice",
        "ExposedPorts": {
            "8080/tcp": {}
        },
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
            "FOO=oci_is_a",
            "BAR=well_written_spec"
        ],
        "Entrypoint": [
            "/bin/my-app-binary"
        ],
```


Cycle de vie



Container et cloud

Exécutable en IAAS
- il existe des instances avec Docker

Soumission directe de Containers
- Amazon Elastic Container Service
  - envois de containers à exécuter
  - facturation à la seconde + CPU et mémoire
  - idéal pour : application longue et ponctuelle, build
    - applications self contained
- Azure Container Instances
  - « serverless containers »
  - MS Flow pour construire une application en utilisant des containers comme blocs

Structure d'un fichier Docker

Dockerfile :

```
FROM debian:stretch
RUN apt-get update && \
    apt-get -y install bash fortune netcat

CMD [ « bash », «-c »,…]
EXPOSE 8080
```

dans le répertoire où se trouve ce fichier Dockerfile

```
docker build . -t mon_container
```

crée une image appelée mon_container

puis

```
docker run mon_container
```

l'exécute avec des options éventuelles (—port …)

Manipulation sur les Containers

1. Installer Docker sur votre machine ou en faire tourner une en machine virtuelle
2. créer et exécuter un container envoyant une chaine sur un port
3. créer et exécuter un container faisant un traitement sur une chaine reçue sur un port et affichant le résultat sur un autre port
4. chainer ces deux containers en utilisant —link

conseil : utiliser netcat (nc) pour écrire sur les ports, fortune (dans /usr/games) pour générer la chaine

vous pouvez aussi en profiter pour explorer le container et la machine qui l'héberge (ps, ls, top, …)

Orchestration

Avantages des containers
- entité légère (configuration et déploiement)
- permet des déploiements rapides et adaptatifs

Un système d'informations peut être constitué de centaines de containers qui s'exécutent simultanément
- impossibilité d'administrer cela manuellement
  - notamment les mises à jour
  - et la tolérance aux fautes ?

L'**orchestration** est essentielle
- différentes solutions logicielles (docker compose)
- un standard : Kubernetes

Kubernetes

Permet d'automatiser le déploiement et l'administration des applications structurées en containers
- open source par Google
- standard de fait, comme docker pour les containers
- par défaut, des containers sans état

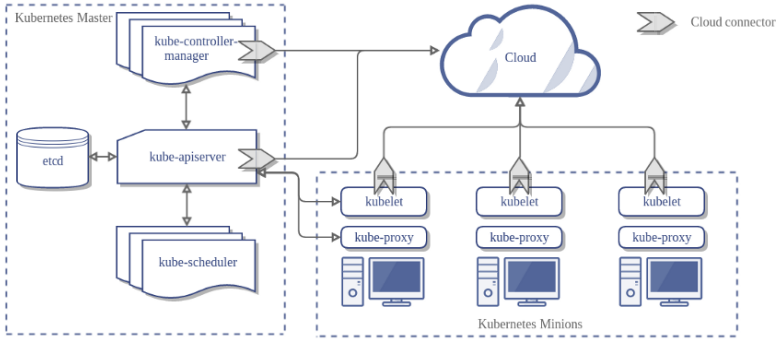
2 éléments principaux de configuration
- la composition de l'architecture (matérielle ?) sur laquelle s'exécutent les applications
- la description des applications

Composition de l'architecture matérielle
- control plane : exécute les services permettant à K8S de fonctionner
- noeuds (minions) : exécute les applications

Description des applications
- en terme de haut niveau (containers et liaisons)
- et de niveau de service voulu (réplication)

Le but de K8S est de toujours garder les applications dans l'état voulu
- nombre d'instances de containers
- ports de communication
- etc.

Architecture globale



kubernetes

L'architecture de Kubernetes

Les composants maitres
- contrôlent l'état des noeuds : actif, chargé, vide, etc.
- prennent les décisions notamment l'ordonnancement
- les principaux
  - kube-apiserver : front-end d'accès au cluster kubernetes
  - etcd : base (clé,valeur) contenant les informations du cluster
  - kube-scheduler : surveille les demandes de pods et affecte un noeud à ceux qui n'en ont pas

Les composants noeud
- kubelet : sur chaque noeud, on leur envoie des pods et s'assure qu'ils tournent
  - s'appuient sur un *container runtime* (typiquement Docker)

Tous ces éléments communiquent par REST (web service)
- ils peuvent être largement distribués

Éléments de Kubernetes

Pod : groupe de containers
- créés pour exécuter des containers ensemble
- partagent le namespace et les volumes (sont sur une même machine virtuelle)
- correspondent à un(e) (morceau d') application
  - avec un (éventuel) tag de version

Controller(s)
- node controller : surveille l'état des noeuds
- replication controller : surveille que le bon nombre de pods s'exécute
- endpoints controller : en charge des endpoints (pods+services)
- cloud-controller-manager : fait l'interface avec le service de cloud sous-jacent

Description des applications

Déploiement
- décrit comment une application doit s'exécuter
  - ensemble des containers la composant et comment les démarrer
  - nombre de copies de chaque container
- K8S s'assure que le déploiement voulu est toujours respecté
  - `kubectl run` -> crée un déploiement qui est ensuite respecté

K8S crée un objet replicaset en charge de s'assurer qu'il y en a toujours assez
- en charge d'un ensemble de pods tous identiques
- typiquement un pod pour chaque instance d'application (replication)

Voir les déploiements en cours dans un cluster K8S
```
kubectl get deployments
```

Exemple

https://kubernetes.io/fr/docs/concepts/workloads/controllers/deployment/

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```


Accès aux applications

Application (~=pod) =
- ensemble de containers en cours d'exécution
- ressources partagées entre les containers (stockage)
- communication entre les containers (ports)

Comment interagir avec l'application ?

Les services
- permet de spécifier les entrées (port) de l'application
  - K8S se charge de router les requêtes vers un déploiement (pod) capable d'y répondre
- mapping au niveau de l'application entre le port exporté et ceux des containers du pod

Exemple

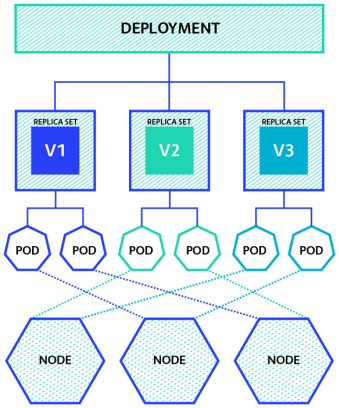
https://kubernetes.io/docs/concepts/workloads/pods/

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```


Importance du versioning

Les pods permettent d'avoir des versions différentes mais cohérentes

Durée de vie d'un pod variable
- très court : micro services
- long : application persistence



Utilisation

Cluster Kubernetes utilisable avec Docker Desktop
- facile et « rapide » pour démarrer et prototyper

Outil `kubectl` en ligne de commande pour interagir avec le master Kubernetes

Définition de configuration
- en passant des ordres et arguments par la ligne de commande
- par des fichiers YAML/JSON
  - de nombreux exemples dans la documentation

Plus sur les pods

Possibilité de leur assigner des labels et de sélectionner dessus

Définition d'affinités
- hard ou soft : imposée ou recommandé
- configuration matérielle, localisation
- peuvent être négatives

Services
- permet d'exporter un service vers un autre pod/application sur la même architecture

Ingress
- permet de fournir un service à l'extérieur de l'architecture
  - suivant le chemin permet de router vers des pods différents

Configuration commune possible (configMap)
- plus de détails dans la partie sécurité

Cloud et Kubernetes

Création et administration d'un cluster Kubernetes
- pile logicielle complexe
- environnement très évolutif

Utilisation d'un cluster clé en main (facturation à la ressource)
- Google Kubernetes Engine (GKE)
  - par les créateurs de Kubernetes, la référence
  - cluster autoscaling
- Amazon Elastic Kubernetes Service (EKS)
  - sur plusieurs zones de disponibilité
  - peut utiliser les instances spot ($$$)
- Azure Kubernetes Service
  - support de windows

Compléments

Turnkey K8S
- externalisation des master nodes
- gestion en interne des noeuds minions

L'éco-système Kubernetes
- kops, kubesray, kubeadm : installeurs, configurateurs

- Helm : K8S package manager facilite l'administration des applications

- kube-bench : benchmarking

- K8guard, copper : déboggage de clusters/applications

- powerfulseal, chaoskube : stress tests

OpenShift

Plateforme d'orchestration de containers de Redhat

Maintenant basé sur Docker et K8S

Permet de cacher la complexité de K8S et d'en accélérer le déploiement

Intégration verticale avec RedHat

Microservices

Serverless basé sur 2 concepts
- FaaS : Functions as a Service
- BaaS : Backend as a Service

Principe
- plus aucune gestion de l'infrastructure, entièrement délégué au cloud
- transmission du code et/ou des applications
- exécution entièrement prise en charge par le prestataire cloud

Adhère au concept devops
- les développeurs sont aussi les administrateurs
- permet une grande agilité
  - à condition d'avoir les infrastructures adaptées (container, orchestration)
  - et la bonne culture aussi!

Manipulations Kubernetes

Créer (fichier YAML) et lancer un déploiement pour le container serveur que vous avez créé lors du TP1
- connectez vous dessus pour vérifier qu'il fonctionne
- tuer le container lancé et observez ce qui se passe

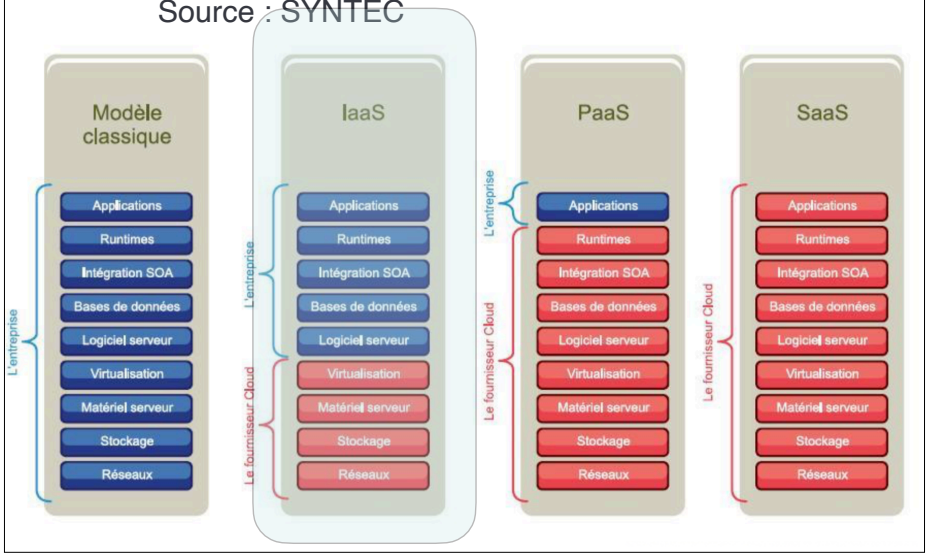
Spécifier un déploiement pour l'application que vous avez faite dans le TP Docker (YAML) avec les 2 containers
- Lancer le déploiement
- Tuer des containers et observer ce que fait K8S

(Utiliser un volume commun entre les deux containers au lieu d'un port de communication)

Essayer de lancer sur AWS/GCP (surveiller vos crédits!)
- possibilité sur GCP et AWS de voir son empreinte carbone (https://cloud.google.com/carbon-footprint)

Vision schématique

Source : SYNTEC

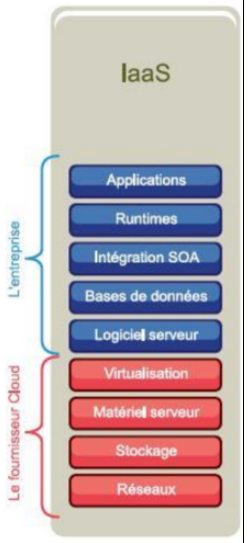


Infrastructure as a Service

- Historiquement le premier type fourni
  - d'abord pour le HPC, calcul puis stockage
  - ISP avec hébergement, milieu des 90s
- Service fourni : stockage, couche de virtualisation, matériel serveur, réseau
- permet de ne pas avoir pas à acheter et gérer de matériels
  - la notion de ressource virtuelle est centrale
    - permet le partage au niveau du datacenter

IaaS : implémentation

- Pas de standards dans le déploiement
  - vision « bas niveau » des ressources
- Notion de machines
  - virtuelles ou physiques
- Isolation (théorique?) entre les clients
  - disque
  - réseau
- Solution pour cloud privé : OpenStack, Xen, …



IaaS : services de base

- machines virtuelles (ou physiques)
- espace de stockage
  - vue fichier ou clé,valeur
  - stockage longue durée
  - (R)DBMS
- Services réseau de base : DNS, IP

IaaS : avantages

- Évolutivité
  - matérielle
  - logicielle (de base)
- Scalabilité
  - peut même être automatique

IaaS : inconvénients

- Single Point of Failure
  - Service Level Agreement
- Vendor lock-in
- Confidentialité des données et traitements
  - architecture partagée entre différents clients
  - législation
    - du site d'hébergement
    - des pays traversés
- Anticipation des coûts
  - intéressant seulement si infrastructure peu exploitée ?

IaaS : critères de choix

- localisation des ressources
- taux de disponibilité
- dimensionnement des serveurs
- qualité des liens réseau
- support système associé

- et le coût, bien évidemment

SaaS

- Exécution de services logiciels sur un site distant

- interface d'accès
  - applications dédiées
  - navigateur

- Importance de la standardisation des protocoles
  - REST, JSON, HTTP(S)
  - clé pour la standardisation et donc le déploiement de solutions
  - avec des implémentations de référence

SaaS

- utilisation quotidienne : Office 365, Gmail,…

- Implique qu'on ne maitrise techniquement (presque) plus rien en terme d'implantation et de confidentialité

- OpenSaaS : Wordpress, Wiki, …
  - code de l'application en Open Source
  - hébergement possible ou déploiement autonome
    - avec les risques de sécurité afférents

Tendance à la migration vers le SaaS

- Serverless illustre bien cela
- Dans l'intérêt des fournisseurs de cloud
  - meilleure maitrise de leur infrastructure
  - hébergement mutualisé
- modèle de coût plus opaque
  - meilleure marge

Les tendances Cloud

- IAAS : AWS, Google Cloud, Azure ont gagné (OVH en challenger poussé par l'UE)
- SaaS : l'émergence de l'IA rebat les cartes
  - Montée en gamme : IA, IoT, Analytics
- développement du multi-cloud (AWS + Azure)
- Développement du Cloud Hybrid
  - rachat de RedHat par IBM a un impact potentiel important

Cloud et contractualisation

- Importance des
  - conditions d'utilisation
  - garanties contractuelles
    - disponibilité
    - évolutions
    - conservation du secret
- Quel degré de confiance accordé à un fournisseur ?
  - sécurité = confiance

Conclusion

- Baisse des coûts
  - matériel
  - maintenance
  - compétences
- Flexibilité
  - rapidité d'évolution
  - passage à l'échelle : machines, données
- Sécurité
  - mais confiance dans le fournisseur

Problèmes potentiels

- concentration -> faible tolérance aux fautes
- risques de sécurité
- lieu d'hébergement (et législation associée)
- dépendances à un fournisseur
  - compétition pour une source de revenues réguliers
  - potentiellement prisonnier d'une solution
- complexité de la facturation, prévisionnel difficile
- coût si haute utilisation des ressources
  - exemple de Netflix