

Notes sur le module : *Cloud Natif*.

Armel Pitelet

23 février 2022

Table des matières

Remarque : *Sur la dénomination Cloud.*

Terme marketing.

Remarque : *Sur internet.*

Ensemble de réseaux qui se base sur des protocoles de communication (http, bas débit, ...). L'ensemble de ces protocoles fait internet qui est un système de communication (pas que le web). Le but premier est de transmettre de l'information via des protocoles (irc pour le chat par exemple, smtp pour les mail,...).

Remarque : *Sur la notion d'IP et l'histoire d'internet.*

Internet protocol. Le nom de version fait référence au nombre d'octet (ipv4 → 4 octets par adresse), ce qui fait un nombre important (4 milliards) mais limité d'ip. Sans adresse ip, pas de communication via internet (au sens large donc plus que juste le web). L'ipv6 est apparue pour balancer ce problème (6 octets par adresse). Début des 90 apparition du web : déblocage de l'accessibilité à l'information. Puis la partie mercantile apparaît rapidement. La notion de qualité logicielle apparaît alors (aka un service qui fonctionne) et avec les fournisseurs de serveur (accès à internet)/service aka ceux qui possèdent l'infrastructure et les points de connexion. Puis la fibre succède au câble cuivre. Les techs (et marchand associés) sont de plus en plus dépendant de cette connexion.

Note :

NAT → Network Address Translation, PAT → Port Address Translation.

Remarque : *Sur le **DNS**.*

Domain Name Service. Service distribué qui alias un ip avec un nom lisible humainement (et fait l'inverse également).

Remarque : *Sur la notion de Cloud.*

Il s'agit en fait de l'hébergement massif et centralisé de serveur destinée à la location. Contrairement à l'ancienne manière de faire le serveur n'est pas monté en physique chez un client mais est simplement loué prêt à l'emploi. Le Cloud fournit via API des ressources utilisables pour le déploiement d'une infrastructure. En générale les services Cloud sont des acronymes qui finissent en *aas* (as a Service). Le Cloud se base sur la virtualisation : une machine physique est capable d'émuler une/plusieurs machines virtuelle(s).

Remarque : *Iaas.*

Infrastructure as a Service. C'est à dire le démarrage d'une machine virtuelle par un Hyperviseur (la machine physique qui fait tourner la machine virtuelle). On alloue ensuite de la RAM, du CPU,

de la mémoire à ces machines virtuelles (dimensionnement du serveur).

Remarque : Sur la virtualisation.

Le gros intérêt est de pouvoir déplacer des VM de serveur en serveur sans couper le service de VM en question. Autre point d'intérêt : deux VM sont isolés l'une de l'autre (en terme de système, pas de réseaux).

Remarque : Faas.

Function as a service. Cette fois ci au lieu de déployer une application entière on ne déploie qu'une partie, une fonctionnalité (comme une base de donnée par exemple ou une fonction de traitement d'image). L'application se retrouve être un ensemble de fonctionnalité distribué à travers le Cloud. Cela évite d'avoir une machine (serveur) mobilisé en permanence.

Remarque : DNSaaS.

Principe du DNS mais avec la possibilité de rajouter des noms de domaines à la volé. Cela revient à mettre à disposition un serveur dns de manière dynamique.

Remarque : CaaS.

Container as a Service. Kubernetes par exemple. Docker est la technologie de containerisation et Kubernetes est le service de gestion de ces container.

Remarque : SaaS.

Software as a Service. Permet d'éviter le déploiement sur des machines d'un Software donné. Par exemple : gmail, github,...

Remarque : PaaS.

Platform as a Service. Contrairement à l'IaaS ou l'on a toute la machine virtuelle (avec installation et configuration à faire), le PaaS met à disposition une plateforme déjà configurée, sécurisée, mise à jour, mise à l'échelle... pour faire tourner du code directement.

Remarque : NaaS.

Network as a Service. Permet de créer des réseaux virtuelle (pas forcément accessible de l'extérieur mais plus des réseaux internes).

Au final le principe du Cloud est d'automatiser tous ces services (depuis une interface). Il existe aussi du DBaaS, du VPNaaS, Maas (Metric as a Service pour des stats.)

Remarque : Sur les Cloud publiques.

Ce sont des services de Cloud ouverts, qui centralisent les services Cloud au sein d'une seule entreprise (AWS de Amazon, Azure de Microsoft, OVH en France, Scaleway, Clever Cloud, Alibaba Cloud en Chine). Selon les localisations les lois sont différentes ! Par exemple le Cloud Act américain permet au gouvernement US d'accéder aux données hébergées par une entreprise Américaine quel que soit l'endroit physique où sont hébergées les données.

Remarque : Sur le Cloud Native.

Une application est dite Cloud Native si elle est conçue dans l'idée de tourner dans le Cloud.

Remarque : Private Cloud.

Le Cloud n'est pas partagé, il est centralisé (infra et logiciel) chez le client. VMware, OpenStack (solution Python open source), Proxmox (plus léger que le reste pour créer des VM à la de-

mande/volée), Kubernetes.

Remarque : Hybrid Cloud.

L'idée est de séparer les services dans différents Cloud (provider de Cloud). Au lieu de tout centraliser, tout est décentralisé chez plusieurs providers.

Remarque : Le MultiCloud.

L'application est partagée sur plusieurs Cloud différents (chez différents Cloud providers).

Remarque : Sue les **12 factors**.

Un ensemble de bonne pratique qui permettent de définir une application comme *Cloud Native Compliant*

1. Code Base : Il faut un code source partageable et versionné ! (et disponible en remote). L'idée est d'avoir une démarche qualitative. Idéalement il faut même avoir une base de code par microservice (au moins un repos par application). De plus le code ne doit pas être dépendant de où il doit être déployé (debug, release, local, ...).
2. Dependencies : Il faut garder à jour la liste des dépendances (via package manager : quelle lib, quelle version), les déclarer explicitement et les isoler. Il ne faut pas garder le code sources des dépendances au même endroit que le code source de l'application. De plus il faut considérer une dépendance comme du *readonly*. Le fait de lister les dépendance permet aussi de les auditer (pour les licences par exemple).
3. Configurations : tout ce qui peut varier entre différents déploiement ne doit pas être hardcodé dans l'application. La meilleure façon de passer de la configuration est d'utiliser les variables d'environnement. Il est aussi possible d'utiliser du yaml ou du json mais ce n'est pas l'idéal car pas facilement éditable. On peut également mettre la configuration dans un service à part dédié à la gestion des configurations (Consul, etcd). Attention à ne pas passer des *secret* dans la configuration (comme des password). **Vault** permet de centraliser et stocker les *secret*. La config se base en générale sur un système de clef valeur.
4. Backing Service : Tout ce dont la base de code a besoin peut être considéré comme un système externe (I/O, base de donnée,...) et doit donc être isolé.
5. Build, Release, Run : Build est la phase de transformation du code source en code exécutable (avec dll et ressources embarquées si besoin). Release : mise à disposition du binaire pour qu'il soit utilisable par qu'un. Run : mettre à disposition la release et la faire tourner (c'est le déploiement) dans un environnement particulier en utilisant une configurations donnée (attention sur le site en lien ils parlent d'une Release comme d'un build accompagnées d'une config). La limite entre Dev et Ops se situe entre Release et Run.
6. Process : Plusieurs instances d'une application doivent s'exécuter en différents process indépendant les uns des autres (par instance). Cette isolation peut se faire en lançant chaque instance sur une VM différente (potentiellement dans la même *Hypervisor*) ou bien via des containers (docker orchestré par kubernetes). L'idée est de sécuriser sa machine par rapport à une possible défaillance d'une des instances de l'application (ou une CVE exploitable selon les technologies utilisées dans l'application).
7. Port Binding : est le principe de mapper un port d'écoute sur un autre (Ex : docker écoute le 80 mais il est mappé par la machine sur le 5000 car les 1 à 2024 sont réservés à root). On passe par du PAT (Port Address Translation) pour rediriger le port déclaré vers un autre. L'intérêt est surtout lié à la sécurité. Cela permet également d'assurer l'unicité des ports mappés dans les dockers (car ces derniers peuvent écouter sur le même port mais de deux containers différents et la machine host doit tout de même pouvoir les distinguer. On map donc chacun des ports [identique] sur un port différent via PAT).

8. Concurrency : Il faut pouvoir déployer l'application plusieurs fois sans que les différentes instances ne bloquent l'accès aux données des autres (ex d'un fichier partagé qui ne doit pas être bloqué par une des instances [sauf au moment de l'utilisation].) Dans la même veine il est possible d'utiliser un **Redis** pour gérer l'état d'une session de connexion ou bien d'utiliser des systèmes d'authentification sans état (au lieu de devoir stocker l'état côté serveur, qui peut changer si l'état du serveur change). Voir également **jwt** pour gérer de l'authentification. Au final on doit pouvoir démarrer autant de fois que l'on souhaite l'application en parallèle sans qu'une instance ne *lock* une autre.
9. Disposability : Le code est jetable. Il ne faut pas hésiter à faire, refaire, jeter, recommencer, patcher pour ensuite nettoyer. De plus le processus correspondant doit démarrer vite, à volonté et s'arrêter correctement sans problème.

10.

Remarque : *Sur le versionning.*

Pour accompagner les bonnes pratiques sur le Code Base et les Dépendances : voir le **semantic versioning** (SEMVER). Par exemple le système version = Major.Minor.Patch. Une mise à jour de patch est juste un update de type fix qui n'affecte pas le fonctionnement de l'application. Si l'on fait une update Du Minor (en ajoutant une fonctionnalité) il faut garder la rétrocompatibilité avec les mineurs précédentes (engagement dans le SEMVER). Cette rétrocompatibilité n'est pas obligatoire lors d'un bump de la Major.

Note :

Une **CVE** est une faille de sécurité connue et référencée (exemple : log4j)