Migration d'application dans le cloud et application cloud native

M346 — Concevoir et réaliser des solutions cloud

Jérôme Frossard

EPAI

27 août 2024

Plan

1. Application traditionnelle
2. Application cloud native
3. Migration d'une application traditionnelle dans le cloud

Plan

Qu’est-ce qu’une application traditionnelle ?

Avant de voir ce qu’est une application *cloud native*, voyons d’abord ce que l’on entend par *application traditionnelle*.

On entend par application traditionnelle, une application monolithique que l’on installe sur un serveur, qui utilise généralement une base de données SQL, et qui s’utilise à l’aide d’un client lourd (application de bureau) ou d’une interface web dans un navigateur.

Voyons cela de plus près en nous intéressant aux domaines suivants :

- Architecture
- Installation
- Maintenance et exploitation

Architecture

Une application est souvent conçue pour prendre en charge un ensemble de tâches ou domaines connexes (ERP, CRM, etc.).

Traditionnellement, une telle application est réalisée en intégrant un grand nombre de fonctionnalités dans une même base de code utilisée pour produire un ensemble de fichiers binaires (exécutable et bibliothèque dynamique) fortement couplés.

Un couplage fort signifie que les bibliothèques ne peuvent pas être utilisées de manière indépendante et, très souvent, qu'elles ne peuvent pas non plus être maintenues et déployées de manière indépendante.

L'ensemble forme un **monolithe** (littéralement, un bloc de pierre massif).

L'ajout d'une fonctionnalité implique généralement la reconstruction et le redéploiement de l'ensemble des binaires.

Déploiement (installation)

Pour déployer (installer) une application traditionnelle, un administrateur effectue typiquement les tâches suivantes manuellement :

- Dimensionner et provisionner un serveur capable de répondre aux besoins des utilisateurs, en tenant compte notamment de la configuration requise pour le système d'exploitation et l'application.
- Installer les dépendances système de l'application (SGBD, environnement d'exécution, bibliothèques système, etc.)
- Installer l'application à l'aide d'un gestionnaire de paquets, ou à partir d'une archive ou d'un programme d'installation.
- Configurer l'application pour l'adapter à son environnement (information de connexion au serveur SQL, mot de passe de l'administrateur, etc.)
- Créer les comptes d'utilisateur·rice·s, et gérer les permissions pour garantir un accès sécurisé et approprié aux ressources.

Maintenance et exploitation

Après le déploiement de l'application, l'administrateur doit en assurer la maintenance et l'exploitation, qui comprennent notamment les tâches suivantes :

- Surveiller les performances du serveur et ajuster les ressources ou configurations en fonction de l'évolution des besoins des utilisateurs.
- Diagnostiquer et résoudre rapidement les problèmes en cas de panne ou d'anomalie.
- Assurer l'application régulière des correctifs de sécurité, et la mise à jour du système d'exploitation pour protéger le serveur.
- Mettre à jour les dépendances de l'application (SGBD, bibliothèques système, etc.) en vérifiant leur compatibilité avec l'application.
- Sauvegarder les données et les configurations critiques pour prévenir les pertes ou permettre une reprise rapide en cas de panne.
- Gérer les comptes d'utilisateur·rice·s et les permissions pour garantir un accès sécurisé et approprié aux ressources.

Plan

Socle technologique du cloud native

Le *cloud native* repose sur un socle technologique qui comprend notamment :

- Les microservices
- Les infrastructures immuables et les conteneurs
- Le maillage de services (*service mesh*)
- Le DevOps

Ce socle technologique permet la mise en œuvre de systèmes :

- Faiblement couplés,
- Robustes,
- Pilotables,
- Observables.

Voyons cela de plus près.

Microservices

Une application cloud native est généralement constituée de composants logiciels indépendants appelés **microservices**. Chacun de ces microservices, souvent **sans état** (*stateless*), se concentre sur un problème spécifique de manière **autonome**.

Les microservices sont **faiblement couplés**. Cela signifie qu'un microservice ne dépend pas directement d'un autre. Il dépend le plus souvent d'une **API REST**, gRPC, ou GraphQL dans le cas d'une communication synchrone, et d'un format de message dans le cas d'une communication asynchrone avec une file de messages (*message queue*).

Pour modifier ou ajouter une fonctionnalité, on modifie ou on ajoute un microservice, qui peut être **maintenu et déployé individuellement**.

Associée à un maillage de service, la faible interdépendance des microservices augmente la **robustesse** de l'application en lui permettant de fonctionner en mode dégradé en cas de défaillance de l'un d'eux.

Infrastructure immuable et conteneur

L'infrastructure immuable signifie que les serveurs d'hébergement des applications natives cloud restent **inchangés après le déploiement**.

En interdisant les modifications et les mises à jour manuelles, une infrastructure immuable rend le processus de déploiement d'une application cloud native prévisible et reproductible, et permet souvent de renforcer la sécurité en réduisant la surface d'attaque.

Pour réaliser une infrastructure immuable, il est nécessaire de pouvoir créer des images des systèmes, et de pouvoir automatiser la création et l'orchestration d'une infrastructure à partir de sa description (*infrastructure as code* ou IaC).

Dans le contexte d'une application cloud native, on utilise typiquement des **images OCI** (*open container initiative*) pour créer des **conteneurs** (ou plus exactement des *pods*) dans un **cluster Kubernetes** (K8s).

Maillage de service

Le maillage de services (*service mesh*) est une couche d'infrastructure qui facilite la communication entre différents services (ou microservices).

En associant un proxy à chaque microservice, le maillage de service permet de :

- Améliorer l'**observabilité** des communications et faciliter le diagnostic en cas de problèmes.
- Sécuriser les communications avec **TLS** et des certificats à courte durée de vie.
- Faciliter la découverte des services (***service discovery*** ou SD) pour ne pas dépendre d'adresses IP (un peu comme un DNS, mais au niveau de l'application).
- Augmenter la résilience en réessayant automatiquement une requête après un échec (***automatic retry***), ou en servant de disjoncteur (***circuit breaker***) en cas de défaillance d'un service.

Le maillage de service est un élément clé de l'architecture microservice. Avec plusieurs dizaines de services, la gestion manuelle devient presque impossible.

DevOps

Le DevOps facilite la **coordination** et la **collaboration** des rôles souvent cloisonnés : développement, exploitation (*operations*), contrôle de qualité, et sécurité.

L'adoption de la **culture**, des **pratiques**, et des **outils** DevOps permet d'accélérer le cycle de vie du développement logiciel, et s'aligne parfaitement avec le cloud native.

L'automatisation et l'IaC sont des aspects importants du DevOps. Cela permet notamment :

- **L'intégration continue (CI)** consiste à intégrer fréquemment de petites modifications dans une base de code partagée pour identifier et résoudre les problèmes au plus tôt. Pour cela, il est généralement nécessaire d'automatiser le workflow du contrôle de qualité (QA), ainsi que les tests d'intégration.
- **Le déploiement continu (CD)** permet de s'assurer que les microservices sont toujours prêts à être déployés en automatisant le déploiement dans un environnement de test (*staging*) et le workflow de déploiement en production.

Les plateformes comme **GitLab** ou **GitHub** jouent un rôle essentiel dans le DevOps.

Observabilité

L'observabilité est un aspect central du cloud native et du DevOps qui facilite la détection et le diagnostic de panne ou d'anomalies.

L'observabilité comprend les pratiques suivantes :

- Collecter et centraliser les journaux (logs) pour suivre l'activité des microservices et identifier d'éventuelles erreurs.
- Mesurer et visualiser des métriques (utilisation des ressources, latence, taux d'erreur, etc.) pour surveiller la santé et les performances du système.
- Tracer les requêtes à travers les microservices (*distributed tracing*) pour suivre les dépendances et analyser le parcours d'une requête.

Ces pratiques d'observabilité basées sur des standards (Prometheus, OpenTelemetry, ELK, etc.) sont essentielles pour l'exploitation et la maintenance d'applications cloud native.

Plan

1. Application traditionnelle
2. Application cloud native
3. Migration d'une application traditionnelle dans le cloud

Pourquoi migrer une application dans le cloud ?

Lorsque vient le moment de remplacer des serveurs ou d'augmenter la capacité du centre de données d'une entreprise, il peut être intéressant d'envisager une migration totale ou partielle dans le cloud.

Comme nous l'avons vu, le cloud offre un certain nombre d'avantages. Par exemple :

- **Élasticité rapide.** Possibilité d'augmenter ou de réduire la capacité selon les besoins.
- **Accès ubiquitaire.** Accès performant aux applications pour les utilisateurs mobiles.
- **Disponibilité et redondance.** Facilite la réalisation de solutions à haute disponibilité.
- **Maintenance simplifiée.** Gestion de l'infrastructure sous-jacente par le fournisseur.
- **Évolutivité.** Facilite l'intégration de nouvelles technologies (ML, big data, IoT, etc.)
- **Meilleure gestion des coûts.** Remplacement des dépenses d'investissement (CAPEX) par des dépenses opérationnelles (OPEX) grâce à des modèles de tarification flexibles (pay-per-use, pay-as-you-go, etc.).

Stratégies de migration

On peut identifier différentes stratégies de migration dans le cloud :

- Lift and shift (*Rehosting*)
- Lift and reshape (*Replatforming*)
- Refactoring/Rearchitecting
- Retiring et retaining

Rehosting (Lift and shift)

La stratégie Lift and Shift, ou *rehosting*, consiste à déplacer une application dans le cloud sans lui apporter de modifications substantielles.

- Structure : L'application et le SGBD sont déployés dans des VM (IaaS) de la même manière qu'ils l'étaient sur site. L'architecture reste la même.
- Avantages : Cette approche est rapide et peu coûteuse en termes de temps et de ressources, car elle ne requiert pas de réécriture de code.
- Inconvénients : Elle ne permet pas de tirer pleinement parti de l'élasticité du cloud, en particulier elle ne profite pas du *scale up* et du *scale down* des solutions serverless.

Cette stratégie est donc adaptée lorsque le principal objectif est de réduire les coûts d'infrastructure rapidement sans modifier l'application.

Replatforming (Lift and reshape)

La stratégie de Lift and Reshape, ou *replatforming*, consiste à adapter certaines parties de l'application pour bénéficier de fonctionnalités spécifiques du cloud.

- Structure : La stratégie Lift and Shift est appliquée pour l'essentiel de l'application, mais certaines parties sont migrées vers des services cloud. Un exemple typique est le remplacement du SGBD par un service DBaaS.
- Avantages : Cette approche permet de réduire les coûts d'exploitation et de bénéficier de certains avantages du cloud (p. ex., la mise à l'échelle automatique du SGBD).
- Inconvénients : Elle demande davantage de ressources et de temps qu'un Lift and Shift, car elle nécessite une adaptation de certains composants aux services cloud.

Cette stratégie est idéale pour les applications qui peuvent tirer parti de certains services managés du cloud sans nécessiter de refonte complète.

Refactoring et rearchitecting

Le *refactoring* et le *rearchitecting*, consistent à revoir la conception d'une application ou de certaines parties d'une application pour l'adapter au cloud, d'abord en réduisant la dette technique, puis en intégrant des principes et fonctionnalités cloud natives.

- Structure : L'application est typiquement décomposée en microservices, et déployée automatiquement dans des conteneurs orchestrés par Kubernetes, par exemple.
- Avantages : Ce type de refonte permet de produire une application observable, capable de tirer parti du cloud pour s'adapter aux changements de charge et maintenir un haut niveau de disponibilité.
- Inconvénients : Le refactoring et le rearchitecting sont les approches les plus complexes et les plus coûteuses. Elles nécessitent une restructuration complète de l'application et une expertise dans le cloud native.

Ces stratégies sont recommandées pour des applications stratégiques à long terme, lorsque la flexibilité et la capacité d'évolution sont des priorités.

Retiring et retaining

Les deux dernières stratégies consistent à évaluer les composants de l'application pour décider de la pertinence de leur migration dans le cloud.

- Le retiring consiste à **retirer des parties de l'application de faible valeur, inutilisées ou devenues obsolètes**. Cette stratégie permet de simplifier l'architecture et réduit les coûts d'infrastructure et de maintenance.
- Le retaining consiste à **conserver certains composants sur site**, pour des raisons de sécurité, de conformité réglementaire, ou de performance (p. ex. besoin d'une faible latence). Le retaining peut être transitoire (en attendant une solution compatible) ou définitif pour des éléments qui ne peuvent être migrés.

Ces stratégies permettent de rationaliser l'architecture en ne migrant que les composants qui bénéficieraient véritablement du cloud, en limitant les coûts et les efforts de migration.