LeMagIT
L'informatique pour et par les pros
TechTarget
E-handbook
Contenu PRO+

# Serverless : bien comprendre cette architecture applicative

## Dans ce guide



## Introduction.

Le serverless computing correspond à une des nouvelles tendances émergentes de l'IT moderne placées sur le radar 2019 de Gartner. Il faut dire que la promesse de cette architecture logicielle a de quoi attirer l'attention : se dédouaner purement des problématiques d'infrastructure, de provisioning de ressources mais aussi - et c'est là un point capital - d'épargner les équipes d'exploitation des douloureuses tâches de déploiement et de maintenance des applications en place. Du DevOps, qui crée une passerelle entre les développeurs et les ops, on passe au NoOps avec le serverless. Le code s'exécute, les ressources s'ajustent automatiquement, la facture est calculée selon les usages.

L'on comprend immédiatement que le terme même de serverless est survendu. Car des serveurs, il en reste bien. Ils sont associés à des containers et des microservices qui apportent cette ultime flexibilité. Le « sans serveur » est justement là pour illustrer cette absence d'opération d'exploitation.

## Dans ce guide



Comme toutes tendances émergentes, peu d'entreprises s'y sont risquées. Car le serverless, l'approche applicative à-la-fonction, rompt avec une habitude de développement. Apprivoiser le concept est un premier pas, mais maîtriser les outils et connaître le marché est le plus difficile.

Ce guide essentiel a justement pour objectif de vous aider à comprendre les concepts qui entourent le serverless, à s'interroger sur le bien-fondé de cette architecture, pour en fin de compte savoir si cette architecture est bien adaptée pour vous et votre entreprise.

Dans ce guide



# Serverless : les pour et les contre

**Chris Moyer,** ACI Information Group

**Le serverless apporte certes de nombreux avantages pour les développeurs, mais cela ne va pas sans compromis. Une compréhension globale est nécessaire, qui pour la plupart, fait encore défaut.**

Le terme "serverless" est aujourd'hui au cœur de nombreux débats. Si les questions sont multiples certaines reviennent souvent, et illustrent une confusion entre containers et serverless ainsi que leurs avantages respectifs. Les deux architectures constituent certes des approches modernes de la gestion des applications, mais chacune d'entre elles présente des avantages spécifiques.

La meilleure façon de comprendre la différence entre les containeurs et l'architecture serverless est d'observer les communautés de développeurs respectives. La plupart de la documentation portant sur les containers Docker traite des questions relatives à la gestion de l'infrastructure et de ses outils. Ces derniers permettent de gérer plus facilement le matériel ou les machines virtuelles sous-jacentes et de répartir les containers sur plusieurs serveurs ou

**Dans ce guide**



instances AWS. La documentation pour le serverless tend à se concentrer avant tout  sur la création d'applications serverless.

Fondamentalement, le serverless permet aux développeurs de se concentrer sur l'écriture de code. Avec ce terme, les serveurs semblent se retirer de l'équation. Mais en fait, il signifie que le développeur n'a pas à se soucier de la gestion des ressources d'infrastructure. Alors que des services comme Amazon Elastic Compute Cloud (EC2) exigent que vous fournissiez des ressources pour le système d'exploitation et l'application, une architecture serverless demande simplement de la quantité de  ressources que nécessite une seule demande de la fonction. Par exemple, une suite de tests Web peut nécessiter 128 Mo de RAM pour un seul site Web. Même si vous déployez 10 millions de copies de cette fonction, chaque fonction individuelle n'a besoin que de 128 Mo. Elles peuvent même fonctionner simultanément. Le serverless se concentre sur ce dont chaque requête individuelle a besoin, puis s'adapte automatiquement.

## Approches de développement serverless

Les Il existe plusieurs approches en matière de développement serverless. Les développeurs qui utilisent généralement un framework traditionnel, comme Flask, Rails ou Express, ont la possibilité de choisir un framework adapté au serverless, comme Chalice pour Python ou Serverless pour Node.js.

Ils sont en effet identiques dans leur approche ; ce qui facilite la transition pour ces développeurs.

Malheureusement, cette approche de framework unique impose des limites tant en termes de taille d'application que de complexité de l'approche. Les développeurs risquent de se heurter rapidement à des problèmes lorsqu'ils tenteront de migrer cette application vers une application serverless.

Par exemple, une fonction lambda AWS se limite à environ 50 Mo. Cela comprend toutes les dépendances, car celles-ci doivent être incluses au moment du déploiement.

## Quelles approches en matière de développement ?

Autre point clé : AWS CloudFormation impose une limite à la complexité des API. Le développeur devra donc les séparer s'il y a trop de points d'extrémité ou d'opérations. En outre, les pièges du monolithisme s'appliquent également : il devient plus difficile de mettre à niveau les services, de les entretenir. De plus, on dispose d'un seul point de défaillance pour l'environnement. Cependant, avec une seule fonction, les démarrages à froid sont plus faciles à gérer.

Les microservices nécessitent une approche différente dans le serverless. Le recours à un framework est certes possible, mais il faut diviser l'API en

## Dans ce guide



plusieurs microservices. Cela  permet de partager le code entre services qui communiquent via les invocations AWS Lambda. Prenons l'exemple d'une entreprise dont le système d'envoi email marketing repose sur plusieurs microservices. Son application est hébergée à partir d'Amazon CloudFront : il utilise un service pour permettre aux utilisateurs de prendre un template, un autre service pour choisir les destinataires et un troisième service pour l'envoi de courriels. Chacun de ces services est également divisé en microservices distincts. Le service d'emailing bâtit d'abord une liste de destinataires dans une fonction. Ensuite, il transmet le modèle de courriel et la liste de destinataires à une autre fonction, qui divise cette liste de destinataires et transmet chaque destinataire, plus le courriel, à une troisième fonction pour faire l'envoi par courriel.

Les fonctions serverless sont souvent chaînées - ce qui permet  d'atténuer la limite des cinq minutes pour le runtime, ainsi que la limite de taille de 50 Mo. Dans l'exemple du système d'emailing, la première fonction, qui gère la construction de la liste des destinataires, a accès à Amazon DynamoDB (le service NoSQL d'AWS) pour extraire les destinataires. Mais il n'a pas besoin d'avoir le code installé pour traiter le modèle de courriel ou envoyer les messages.

La dernière fonction, en charge de l'emailing, n'a pas besoin d'accéder à DynamoDB, mais elle a besoin de savoir comment construire le modèle à partir des données d'entrée. Surtout, aucune de ces fonctions n'a besoin d'être

exposée via Amazon API Gateway. Au lieu de cela, il s'agit d'un service séparé qui prend une requête de l'utilisateur, l'authentifie et la transmet directement au service courriel par le biais d'un appel AWS Lambda.

Pour les services complexes, comme l'exemple du système d'emailing ci-dessus, les développeurs peuvent choisir d'utiliser les fonctions AWS Step Functions au lieu de connecter les fonctions Lambda à la main. Cela apporte un support plus poussé de la gestion des erreurs et peut gérer automatiquement le transfert d'état et de données entre les fonctions. Chaque fonction est encore complètement isolée, mais l'état et les transitions sont tous gérés par AWS.

## Outils de débugage d'architectures serverless

Traditionnellement, les développeurs pouvaient simplement se connecter au système, exécuter l'application, et consulter les logs et les entrées pour déboguer. Dans une architecture serverless, il n'y a pas de serveur sur lequel se connecter, et l'exécution locale peut être beaucoup plus compliquée. Certains plug-ins AWS, tels que Serverless Offline et SAM Local, permettent d'exécuter la majorité des applications hors ligne. Cependant, ceux-ci ne fonctionnent pas bien si par exemple une procédure d'autorisation est effectuée à partir d'un autre référentiel ou s'il y a plusieurs fonctions qui doivent être chaînées.

Dans de nombreux cas, les développeurs doivent exécuter leur propre système pour le développement et le test, puis apporter des modifications sur un compte AWS.

Il existe plusieurs outils pour identifier les problèmes au niveau de l'application et pour assurer un suivi des performances. AWS X-Ray peut tracer automatiquement les problèmes grâce à d'autres appels vers AWS. La plupart des applications nécessite quelques lignes de code pour activer cette radiographie. Il peut ensuite afficher la cartographie du réseau et signaler les problèmes (le débit provisionné sur les tables DynamoDB ou les limites de concurrence dans Lambda). Les logs  provenant d'erreurs standards sont dirigés vers Amazon CloudWatch et peuvent être ingérés dans une instance Amazon Elastic Search.

## Le compromis serverless

Dans l'ensemble, le serverless permet aux équipes de développement de se concentrer davantage sur le produit et ses résultats, mais il faut prévoir davantage d'outillage pour gérer la planification, les tests et le monitoring. D'où la nécessité de cartographier le projet : cela permet de décider si ou non une architecture de microservices est possible ou souhaitable.

Si elle est correctement exécutée, une architecture serverless peut faire gagner du temps dans le développement de nouvelles fonctionnalités et peut évoluer à

## Dans ce guide



l'infini. Si en revanche, les développeurs sautent l'étape de la planification, il faut s'attendre à ce que cela soit plus difficile à gérer.

Dans ce guide



# Le Serverless ne convient pas à toutes les applications

**Zachary Flower,** développeur web freelance et journaliste

**Même si les plateformes dites Serverless ont la capacité de réduire à la fois la complexité de l'infrastructure et les coûts, elles ne constituent pas la meilleure option pour certaines applications, comme celles exploitant les mécanismes du multi-cloud.**

Pour une entreprise, les critères qu'il convient d'évaluer avant de se plonger dans le Serverless sont nombreux. Chaque API a des exigences distinctes ; il est donc difficile d'identifier celles qui sont ou ne sont pas éligibles aux infrastructures dites Serverless. En général, on parle de Serverless (littéralement sans serveur) pour illustrer qu'une application se repose sur des services Cloud, invoqués de façon éphémère et sur lesquels s'appuient certaines fonctions logiques. Le développeur exécute son code au sein de ces Faas (Function-as-a-service) à la demande.

Il existe toutefois plusieurs méthodes pour identifier si un back-end Serverless est la bonne solution pour une application, quand cela est pertinent et pourquoi cela devient pertinent. Réduire la complexité et rationaliser l'infrastructure sont généralement deux motivations clé.

# Réduire la complexité

Les API sont complexes par nature. Leur utilisabilité mise à part, elles restent des composants à géométrie variable. Du stockage de données à la logique métier, les éléments à prendre en compte sont nombreux. Si l'on écarte les grandes API de logiciels monolithiques, les plateformes Serverless constituent un bon moyen pour réduire la complexité d'une application et au final de l'ensemble de l'entreprise.

A l'inverse des grandes applications, celles s'adossant à une architecture composée de microservices sont logiquement les premières éligibles à ces infrastructures Serverless. La migration d'une telle application vers un backend Serverless permet aux développeurs de minimiser le nombre de ces microservices et de rationaliser encore plus les endpoints. Résultat, le coût de l'infrastructure se retrouve lui-aussi abaissé, car chaque fonction Serverless fonctionne à la demande – et est donc facturée en fonction de l'usage.

Comme les applications en microservices, les API qui définissent une fonction unique sont idéales pour le Serverless. Leur empreinte réduite permet de les encapsuler dans des fonctions elles-mêmes plus ciblées et donc plus petites.

## Dans ce guide



# Rationnaliser l'infrastructure

Par définition, avec  les plateformes Serverless, l'entreprise n'a plus à se préoccuper des serveurs de l'infrastructure – comme tout ce qui est placé dans le Cloud finalement. Cela procure donc un effet de liberté chez les développeurs qui peuvent rationaliser aussi bien la taille de leurs équipes que le nombre d'API – le tout avec des coûts plus bas.

Le Serverless permet également de simplifier le dimensionnement. Les API, dont l'usage nécessite un dimensionnement aléatoire, conviennent à ces plateformes qui sont capables de gérer ces demandes imprévisibles. En éliminant ainsi cette contrainte, celle d'avoir à gérer les ressources de l'infrastructure, le Serverless permet de consacrer plus de temps à créer des API plus robustes et optimisées.

Toutefois, le mode du Serverless est généralement associé au verrou-vendeur. En gros, les infrastructures Serverless conviennent mieux à des applications et des API qui se reposent déjà sur la même plateforme Cloud. Les API que l'on héberge soi-même et qui s'adossent à des socles génériques de virtualisation n'auront pas accès à l'une des promesses du Serverless : une intégration simplifiée entre les autres services Cloud de la même plateforme du même fournisseur.

Dans ce guide



# Quand il ne faut pas choisir d'infrastructure Serverless

Si les plateformes Serverless peuvent réduire la complexité et les coûts, elles ne conviennent pas à toutes les applications et toutes les workloads. Si elles conviennent aux applications bâties sur des microservices, elles sont moins adaptées aux applications monolithiques – à moins d'y appliquer une refonte en profondeur.

Il faut également savoir que les économies de coûts ne pourront pas être réalisées si l'on applique une mécanique Serverless à des processus sur le long terme. Ces workloads nécessitent plutôt une infrastructure plus classique.

Les applications ou les API qui nécessitent d'avoir à jongler entre plusieurs plateformes Cloud ne sont également pas adaptées à ces infrastructures. La cause : la verrou-vendeur. Chaque plateforme Serverless dispose de ces propres caractéristiques. Toute migration d'une plateforme vers une autre est difficile et chronophage.

Enfin, pour certaines applications nécessitant des performances élevées, le Serverless impliquera quelques ajustements. Pour exécuter sa première requête à une fonction, une application Serverless a besoin de temps. Ces requêtes peuvent être lentes puisque le fournisseur doit allouer les bonnes ressources pour supporter cette fonction.

## Dans ce guide



Les applications qui, en plus, nécessitent un niveau élevé de fiabilité, doivent s’accommoder de cette contrainte à la main en conservant la fonction dans un mode actif.

Dans ce guide



# Serverless : ce qu'il faut considérer avant de se lancer

**George Lawton,** rédacteur searchMicroservices

**Le Serverless a certes attiré l'attention de nombreux développeurs. Mais il faut rester particulièrement attentif aux problèmes de portabilité.**

L'informatique Serverless a été au centre de nombreux discours l'année dernière. Amazon a bien sûr pris la tête du peloton avec ses fonctions Lambda. Et les entreprises, elles, semblent suivre le mouvement.  Cependant, le chemin pourrait bien être chaotique pour celles qui veulent utiliser le Serverless en environnement multi-Cloud, pensent certains experts.

« Les fournisseurs de cloud ne voient probablement pas leur intérêt à créer des passerelles pour faciliter l'accès au Serverless », constate Rich Sharples, directeur principal des produits chez Red Hat. « Mais à terme, cela sera bénéfique aux utilisateurs qui ont adopté le multi-cloud ou ceux qui souhaitent être agnostiques en termes de cloud. Les architectes d'entreprise doivent évaluer les risques entre être verrouillé sur une plateforme cloud, et bénéficier de services, comme un monitoring optimisé, ajoute le responsable.

## Les difficultés du Serverless

Bien Actuellement, les fournisseurs de Cloud proposent de nouvelles plateformes Serverless, tout en promettant de délivrer les développeurs de la gestion de l'infrastructure. Cependant, cela peut conduire au fameux verrou-vendeur. Les développeurs utilisent en effet les services spécifiques à chaque plateforme, comme l'orchestration des fonctions par exemple.

Pour résoudre ce problème, la Cloud Native Computing Foundation (CNCF) travaille à améliorer la portabilité avec des spécifications, comme Kubeless et OpenWhisk. Les développeurs peuvent utiliser les bibliothèques d'API et des frameworks Serverless pour créer des applications portables entre Clouds, a déclaré Michael Coté, directeur du marketing technique chez Pivotal. Il s'attend à l'émergence de nombreux frameworks pour .Net, Java, Ruby, Python et PHP. Il s'attend également à une bataille entre développeurs pour connaître quel sera le meilleur des langages pour le Serverless.

## A architecture Serverless, services réduits

La croissance des offres, et les gains en matière de réactivité, devrait également permettre aux entreprises de développer, de tester et de déployer des services plus réduits, s'accordent à dire certains experts.

Les entreprises auront ainsi la possibilité d'assembler et de désassembler les applications d'une manière qui n'était pas possible jusqu'alors.

Ces modèles Serverless fonctionnent pour les applications simples et qui ne nécessitent pas d'état, commente Ravi Mayuram, CTO chez Couchbase (base NoSQL). Ces services stateless fonctionneront de pair avec des bases de données JSON et les applications. Si une interface JSON est présente dans la base, les entreprises auront d'importantes possibilités d'évolution.

## Identifier le bon cas d'usage

Malgré le battage médiatique, le Serverless en est encore à ses débuts. Amazon a certes pris les devants avec son offre Lambda, et ce, malgré des SLA limités. Les entreprises devront donc évaluer la performance de ces nouvelles applications, et identifier les bons cas d'usage avant de s'engager, affirment les experts.

Pour adopter les frameworks Serverless, il faut une bonne dose d'optimisme, de foi et d'indifférence, pense Owen Garrett, responsable des produits chez Nginx. Optimisme, car les développeurs doivent prier qu'une fonction se déclenche dans le bon timing ; foi car, si cette fonction ne se déclenche pas une fois, il se peut qu'elle s'exécute une autre fois ; et l'indifférence quant à savoir si elle fonctionne réellement.

Il sera également plus difficile de débuguer et de récupérer des informations sur les pannes car les applications Serverless ne stockent pas d'état, disent les experts. Les développeurs auront donc besoin de nouveaux outils. Les architectes d'entreprise, de leur côté, auront une responsabilité : créer des schémas directeurs et de la documentation pour les tests et les déploiements, au risque d'être confrontés à la fragmentation de l'architecture.

## Dans ce guide

Dans ce guide



# FaaS : bien comprendre les modèles de déploiements

**Tom Nolle,** CIMI Corporation

**Si le modèle Function-as-a-service attire les développeurs, le besoin de bonnes pratiques se fait encore sentir. Le modèle FaaS demande de bien comprendre les modèles de déploiements. Cet article fait le point.**

Le FaaS, ou Function as a service, a tout pour séduire les amateurs d'applications natives pour le cloud. Ce concept architectural permet de créer des applications sans état qui réagissent à des événements. Pour cela, il s'adosse aux microservices, qui s'exécutent sur une infrastructure serverless, avec une allocation dynamique des ressources.

Toutefois peu d'équipes de développement savent finalement quand et comment passer au FaaS. Il est donc utile de suivre une feuille de route type. Celle-ci doit tenir compte de deux facteurs principaux : la façon dont les données sont associées aux processus dans l'application et les capacités technique de la plateforme.

# Les modèles d'implémentation du FaaS

Il existe deux modèles d'implémentation possibles pour le FaaS. Le premier est le modèle workflow, qui est similaire aux modèles d'application multicomposants traditionnels. Dans ce modèle, on visualise la tâche soit comme une transaction, soit comme une entrée qui doit être traitée pour créer une sortie. Le deuxième type est le modèle flux d'événements, dans lequel chaque tâche représente un événement, un changement de condition ou d'état.

Lorsqu'il s'agit de l'exécution d'une fonction de base dans le cloud ou même dans le data center, le dispositif prévoit qu'un événement déclenche l'exécution d'une fonction spécifique. Ces applications FaaS qui réagissent à des flux d'événements s'appuient sur chaque fonction de la chaîne pour générer un événement secondaire ; cet événement secondaire suit des règles de traitement qui invoqueront ensuite une fonction différente. Cela crée donc une cascade d'événements qui se termine soit par une réponse de contrôle, soit par une transaction en back-end traditionnelle où le flux d'événements sert de front-end à d'autres processus métier.

Les applications qui reposent sur des flux d'événements ne s'appuient pas sur des outils spécifiques car le lien entre un processus et les données est fondamental pour la mise en œuvre de ces fonctions. Les applications sont donc, dans un sens, auto-organisées et auto-dimensionnées. Dans une implémentation pratique, il suffit de s'assurer que la chaîne d'événements est

correctement en place et que la latence reste faible pour identifier, charger et exécuter chaque fonction dans la chaîne.

## Évaluer les offres du marché

En grande partie, les entreprises n'utiliseront pas les flux d'événements purs parce que leurs applications en place ne se reposent que sur des workflows. La structure d'une application est plus figée lorsqu'elle utilise le modèle FaaS pour les applications de workflow. Parce que l'information suit une séquence de traitement, il est nécessaire de mettre en place un mécanisme qui dirige les traitements vers les processus les uns après les autres, plutôt que de façon automatisée.

Si votre application FaaS est sans état, vous pouvez utiliser Step Functions d'AWS et Logic Apps de Microsoft pour définir et contrôler la structure.

## Lambda et Step Functions

La vision d'Amazon en matière d'informatique fonctionnelle est peut-être la plus radicale de toutes. AWS Lambda, son service FaaS, suppose que presque tout chez AWS peut à la fois générer des événements et être déclenché par des événements. On pourrait même dire qu'AWS Lambda s'intéresse davantage au workflow et à l'orchestration des applications métier qu'au traitement des flux

d'événements générés en externe. Cependant, Lambda permet aussi d'intégrer des applications aux services AWS via des événements. Il s'agit donc vraiment d'une application de gestion de flux d'événements en raison du fait qu'elle est automatisée.

Step Functions est une bonne option car elle définit un workflow qui est suivi par des fonctions et des composantes applicatives. Le service fournit un deuxième niveau d'orchestration en plus de Lambda.

## Azure Functions et Logic Apps

Microsoft a l'avantage de la base installée Windows et d'une importante population d'utilisateurs Office en environnement professionnel. Toutes ces activités peuvent à la fois générer des événements et être déclenchées par des événements. Ce qui donne à Azure Functions une capacité inégalée pour créer des fonctions qui s'intègrent aux outils existants du legacy. En raison de cet avantage, beaucoup considèrent Microsoft comme le leader sur le marché du cloud d'entreprise.

Azure Logic Apps, qui vous permet de créer des liens d'intégration bâtis sur des événements à l'aide de formulaires, étend d'autant plus cet avantage. Logic Apps optimise l'orchestration des applications par les événements et les fonctions, même auprès des utilisateurs les plus avertis. Logic Apps est

particulièrement doué pour intégrer les outils Microsoft et les applications Azure aux médias sociaux - ce qui est un atout pour les entreprises.

## Google Cloud Pub/Sub, Functions et Firebase

Google a son propre modèle d'orchestration de fonctions, avec Google Cloud Pub/Sub, Google Cloud Functions et Google Firebase. Firebase et sa capacité à activer des fonctions liées à la base de données, en réaction à des événements, est celle qui intéresse le plus les entreprises.

Cependant, Google n'a pas exposé beaucoup de fonctions à et n'a rien à intégrer avec les outils sur site, à moins que les utilisateurs n'utilisent Google Docs.

## Un dernier mot sur la fonction de service

Pour le moment, l'approche prise par les fournisseurs de cloud est beaucoup trop différente pour imaginer pourvoir déplacer les fonctions d'un cloud à l'autre, sans qu'il n'y ait beaucoup de travail à réaliser. Il convient d'abord d'identifier la plateforme qui réponde le mieux à vos besoins à court terme, puis de la tester sur cette plate-forme. Le problème de transfert de données arrivera par la suite.

## Dans ce guide



Pourquoi cette approche ? Parce que les applications fonctionnelles et leurs outils évoluent très rapidement. Il est probable qu'à terme, tous les fournisseurs de cloud computing fournissent les mêmes fonctions. Mais pour l'instant, difficile de prédire. La bonne pratique consiste donc à y aller par étape.

Dans ce guide



# Le tandem serverless – microservices : quels sont les bénéfices ?

**Tom Nolle,** CIMI Corporation

**Cet article montre que coupler le serverless aux microservices donne accès à certains avantages, notamment dans la gestion des processus métier et des applications.**

Lorsqu'un nouveau modèle cloud émerge, il attire inévitablement l'attention.

Au moment où les fournisseurs de cloud ont commencé à offrir des services dits serverless, cela a suscité l'intérêt des clients. Aujourd'hui, pourtant, la confusion règne sur ce qu'est vraiment le serverless et comment il se rapporte aux microservices, aux fonctions, etc. Ce serverless est-il vraiment un modèle de service révolutionnaire, ou s'agit-il simplement d'une tendance très buzzy aux cas d'usage limités ? La réponse dépend de vos besoins.

## Dans ce guide



# Comment les services sans serveur et les microservices sont-ils liés les uns aux autres ?

Michel Le serverless est un modèle d'hébergement cloud où une application, ou un composant, se charge et s'exécute à la demande. Parce que les composants serveless peuvent se charger à la demande, n'importe où et se dimensionner en fonction des besoins, rien n'est stocké entre chaque utilisation. Cela implique une structure d'application différente. Vous disposez d'un composant sans état, ou un composant qui ne se souvient de rien. Les composants sans état sont souvent désignés comme une fonction, un « lambda » ou un microservice.

Les microservices sans état marquent une évolution des modèles de développement : un changement vers une plus grande élasticité dans le déploiement des applications et un plus grand partage des composants. Cette tendance particulière porte sur l'hébergement de microservices sur des containers. Ces derniers proposent en effet des coûts inférieurs à ceux des machines virtuelles (VM) et les outils d'orchestration, tels que Kubernetes, se prêtent au déploiement de microservices.

Dans ce guide



# Quand est-il judicieux d'utiliser à la fois les services sans serveur et les microservices ?

Articuler une application autour d'une kyrielle de composants est une mauvaise pratique car cela augmente la latence et la complexité opérationnelle. Pour les applications traditionnelles, on peut s'attendre à une progression des architectures de microservices. On peut également s'attendre à ce que les aspects de sécurité, de mise à l'échelle et de partage des microservices y trouvent refuge. En ce qui concerne le transactionnel, cette limite de composant devrait en freiner la croissance.

Si un changement doit intervenir dans la façon dont les applications utilisent les microservices, alors ce changement doit porter sur la façon dont ils sont invoqués. Sémantiquement parlant, si le modèle traditionnel appelle les microservices, alors le modèle alternatif doit quant à lui déclencher les microservices.

C'est là qu'il existe une convergence entre les microservices et les services serverless. Quand AWS et Microsoft ont présenté leurs services Serverless, ceux-ci ciblaient principalement les applications déclenchées par des événements - comme celles liées à l'IoT. Ces applications s'exécutent de façon intermittente et ne justifient donc pas l'utilisation de VM ou de serveurs cloud persistants.

Dans ce guide



Cependant, l'IoT n'est pas le cas d'usage premier pour ces fournisseurs de cloud. La portée du serverless s'est élargie.

## L'orchestration des applications et des processus métier permet de combiner microservices et serverless

Le premier cas d'usage du serverless et des microservices dans les entreprises porte sur l'orchestration d'applications et de processus métier pilotés par les événements. Presque tout ce qu'une entreprise fait avec l'informatique s'apparente à un ensemble chorégraphié d'étapes successives. Etapes qui se retrouvent au sein de plusieurs applications et souvent, utilisées par différentes équipes. Avec cette nouvelle approche, le serverless ou les microservices ne traitent pas les événements IoT, mais orchestrent les flux de travail des applications. Cette mission pourrait bien transformer tout le modèle du serverless.

Cette dimension comporte trois éléments. Premièrement, les applications doivent avoir la capacité de générer des événements et leurs déclencheurs. Deuxièmement, un mécanisme doit permettre d'activer les processus en fonction de ces déclencheurs. Enfin, les flux de travail ou les séquences de tâches - à la fois normales et anormales – doivent être définies pour structurer

## Dans ce guide



les workflows et les  événements. Les fournisseurs de cloud disposent tous de ces trois éléments.

Les outils d'orchestration et d'intégration tels que Step Functions chez AWS ou Logic Apps de Microsoft permettent de séquencer les composants applicatifs en fonction des événements générés par d'autres applications. Microsoft propose par exemple des connecteurs qui permettent de déclencher des changements de documents dans Office. Ces connecteurs peuvent ensuite lancer d'autres applications et mettre à jour des bases de données. Ils peuvent également déclencher des fonctions et des microservices qui s'exécutent à la demande.

AWS et Microsoft proposent tous deux des cas d’usage pour illustrer l'utilisation de Step Functions et Logic Apps. Ces exemples montrent que les composants serverless et les composants microservices peuvent jouer les mêmes rôles dans l'orchestration d'applications métier et des processus que ceux traditionnellement joués par les ESB.

Au lieu de simplement contrôler ce qui est activé sur la base de l’analyse des données, on peut contrôler les processus sur la base de signaux provenant d'autres processus.

## Dans ce guide



# Accéder à plus de contenu exclusif PRO+

Vous avez accès à cet e-Handbook en tant que membre via notre offre PRO+ : une collection de publications gratuites et offres spéciales rassemblées pour vous par nos partenaires et sur tout notre réseau de sites internet.

L'offre PRO+ est gratuite et réservée aux membres du réseau de sites internet TechTarget.

## Profitez de tous les avantages liés à votre abonnement sur: http://www.lemagit.fr/eproducts

Images; Fotolia

## Dans ce guide






Cyrille Chausson | *Rédacteur en Chef*
TechTarget
22 rue Léon Jouhaux, 75010 Paris
www.techtarget.com