Optimiser l'économie de l'entreprise avec les architectures sans serveur

Septembre 2017

Mentions légales

Ce document est fourni à titre informatif uniquement. Il présente les offres de produits et pratiques actuelles d'AWS à la date de publication de ce document. Ces informations sont susceptibles d'être modifiés sans avis préalable. Il incombe aux clients de procéder à leur propre évaluation indépendante des informations contenues dans ce document et chaque client est responsable de son utilisation des produits ou services AWS, chacun étant fourni « en l'état », sans garantie d'aucune sorte, qu'elle soit explicite ou implicite. Ce document n'offre pas de garantie, représentation, engagement contractuel, condition ou assurance de la part d'AWS, de ses sociétés apparentées, fournisseurs ou concédants de licence. Les responsabilités et obligations d'AWS vis-à-vis de ses clients sont régies par les contrats AWS. Le présent document ne fait partie d'aucun contrat et ne modifie aucun contrat entre AWS et ses clients.

Table des matières

Résumé

Ce livre blanc a pour but d'aider les dirigeants principaux de l'information (CIO), les dirigeants principaux de la technologie (CTO) et les architectes senior à obtenir davantage d'informations sur les architectures sans serveur et leur impact sur la mise sur le marché, l'agilité des équipes et l'économie informatique. En éliminant les serveurs non ou sous-utilisés au niveau de la conception et en simplifiant sensiblement les conceptions des logiciels cloud, l'approche sans serveur va rapidement transformer le paysage informatique.

Ce livre blanc couvre les bases de l'approche sans serveur ainsi que le portfolio sans serveur d'AWS. Il inclut un certain nombre d'études de cas qui illustrent le gain radical d'agilité et d'avantages économiques des sociétés actuelles qui adoptent l'approche sans serveur. Ce document montre comment les organisations de toutes tailles peuvent exploiter les architectures sans serveur pour concevoir des systèmes réactifs et basés sur les événements, ainsi que leur capacité à fournir des microservices cloud à des coûts infimes par rapport à la normale.

Introduction

De nombreuses entreprises tirent déjà avantage de l'exécution d'applications dans le cloud public, dont des économies sur les coûts grâce à une facturation à l'utilisation et une agilité accrue par l'utilisation de ressources informatiques à la demande. Plusieurs études sur divers types d'applications et secteurs ont montré que la migration d'architecture d'applications existantes vers le cloud réduisait le coût total de possession (TCO) et améliorait le délai de mise sur le marché.[1]

En lien avec les solutions sur site et cloud privé, le cloud public simplifie grandement la création, le déploiement et la gestion des flottes de serveurs, ainsi que les applications qui y sont hébergées. Cela dit, les entreprises d'aujourd'hui ont des options supplémentaires allant au-delà des architectures de serveur classiques ou sur machines virtuelles pour profiter d'avantages du cloud public. Bien que le cloud élimine le besoin pour les entreprises d'acheter et d'entretenir leur propre matériel, l'architecture sur serveur, *quelle qu'elle soit*, les oblige à chaque fois à repenser leur évolutivité et fiabilité. En outre, les entreprises doivent relever le défi de la correction et du déploiement de ces flottes de serveurs à mesure de l'évolution de leurs applications. Sans oublier qu'elles doivent dimensionner leurs flottes de serveurs en fonction de pics de charge, puis tenter de les réduire pour rentabiliser les coûts aux moments opportuns, le tout en préservant l'expérience des utilisateurs finaux et l'intégrité des systèmes internes. Les serveurs non ou sous-utilisés s'avèrent aussi couteux qu'inefficaces. Les analystes estiment qu'en pratique, 85 % des serveurs ont une capacité sous-utilisée.[2]

Les services de calcul sans serveur comme AWS Lambda sont conçus pour traiter ces problèmes en proposant aux sociétés une manière différente d'approcher la conception des applications, une approche présentant des coûts et des délais de mise sur le marché fondamentalement réduits. *AWS Lambda élimine la complexité du traitement des serveurs à tous les niveaux de l'échelle technologique. Le service apporte un modèle de facturation à la demande qui évite les coûts liés aux capacités de calcul non exploitées.* En outre, les fonctions Lambda permettent aux organisations d'adopter en toute simplicité les architectures de microservices. L'élimination d'infrastructure et le passage à un modèle Lambda offre des avantages économiques doubles :

- Les problèmes comme ceux des serveurs non utilisés disparaissent purement et simplement, tout comme leurs conséquences économiques. Un service de calcul sans serveur comme AWS Lambda ne se retrouve jamais « à froid », car des frais ne sont engagés qu'en cas de fonctionnement utile avec une granularité de facturation à l'échelle de la milliseconde.
- La gestion de flotte, y compris l'application de correctifs de sécurité, les déploiements et la surveillance des serveurs, n'est plus nécessaire. Cela signifie qu'il n'est pas nécessaire d'entretenir les outils, processus et rotations sur demande associés nécessaires au fonctionnement 24 heures sur 24 et 7 jours sur 7 de la flotte de serveurs. L'utilisation de Lambda pour la création de microservices permet aux organisations de se montrer plus agiles. Débarrassées du souci que représente la gestion des serveurs, les entreprises peuvent rediriger leurs ressources informatiques rares vers ce qui compte : leur activité.

Grâce à des coûts d'infrastructure réduits, des équipes plus agiles et plus centrées, et un délai de mise sur le marché réduit, les entreprises ayant déjà adopté l'approche sans serveur prennent un avantage non négligeable sur la concurrence.

Compréhension des applications sans serveur

Les avantages de l'approche sans serveur citée ci-dessus sont attractifs, mais que faut-il prendre en considération pour une mise en place concrète ? Qu'est-ce qui différencie une application sans serveur de son homologue conventionnel basé sur un serveur ?

Les applications sans serveur sont conçues de telle sorte que les développeurs puissent se concentrer sur leur compétence de base : rédiger la logique commerciale de base. Nombre de composants essentiels de l'application, comme les serveurs Web, et tous les gros œuvres indifférenciés, comme le logiciel permettant de gérer la fiabilité et l'évolutivité, sont complètement retirés au développeur. Il ne lui reste qu'une approche claire et fonctionnelle où la logique commerciale n'est déclenchée qu'en cas de besoin : l'envoi d'un message par un utilisateur mobile, le chargement d'une image vers le cloud, un flux entrant d'archives, etc. Une approche asynchrone basée sur les événements de la conception des applications, même si elle n'est pas requise, est très courante dans les applications sans serveur, car elle s'imbrique parfaitement dans le concept de code s'exécutant (et engendrant des coûts) uniquement lorsqu'il y a des tâches à accomplir.

Une application sans serveur s'exécute dans le cloud public, sur un service tel qu'AWS Lambda, qui s'occupe de la réception d'événements ou d'invocations de clients, avant d'exemplifier et d'exécuter le code. Ce modèle propose un certain nombre d'avantages par rapport à la conception conventionnelle d'applications basées sur serveur :

- Plus besoin de provisionner, déployer, mettre à jour, surveiller ou gérer de quelque autre manière les serveurs. Tout le matériel et les logiciels de serveur actuels sont gérés par le fournisseur cloud.
- L'application se dimensionne automatiquement en fonction de son utilisation actuelle. Cette fonctionnalité est fondamentalement différente par rapport aux applications conventionnelles qui nécessitent une flotte réceptrice et une gestion de capacité explicite pour le dimensionnement à l'échelle du pic de charge.
- En plus du dimensionnement, la disponibilité et la tolérance aux pannes sont intégrées. Aucune configuration, gestion et aucun codage ne sont nécessaires pour tirer parti de ces capacités.
- Il n'y a aucuns frais pour la capacité non exploitée. Plus besoin de préprovisionner ou surprovisionner de la capacité (ce n'est pas possible, en fait). Au lieu de cela, la facturation se fait à la demande et se base sur la durée d'exécution du code.

Cas d'utilisation d'une application sans serveur

Le modèle d'application sans serveur est générique et s'applique à pratiquement tous les types d'application, de l'application Web de start-up à la plate-forme d'analyse boursière d'entreprise Fortune 100. Voici quelques exemples :

- **Applications et sites Web :** l'élimination des serveurs rend possible la création d'applications Web qui ne coûtent pratiquement rien lorsqu'il n'y a pas de trafic tout en se dimensionnant pour gérer les pics de charge, même ceux qui ne sont pas prévus.
- **Backends mobiles :** les backends mobiles sans serveur offrent aux développeurs un moyen de se concentrer sur le développement client pour créer en toute simplicité des backends parfaitement dimensionnés, sécurisés et hautement disponibles sans devenir expert en conception de systèmes distribués.

- **Traitement de médias et de journaux :** l'approche sans serveur propose un parallèle naturel qui simplifie le traitement de charges de travail exigeantes en ressources sans la complexité de la création des systèmes multithread ou le dimensionnement manuel de flottes de calcul.
- **Automatisation informatique :** des fonctions sans serveur peuvent être jointes à des alertes et des moniteurs pour apporter une personnalisation en cas de besoin. Les tâches cron et d'autres exigences d'infrastructure informatique sont nettement plus faciles à mettre en place en supprimant la nécessité de posséder et entretenir des serveurs pour leur usage, en particulier lorsque ces tâches et exigences sont occasionnelles ou de nature variable.
- **Backends d'IoT :** la possibilité de fournir du code, y compris des bibliothèques natives, simplifie le processus de création de systèmes cloud capables de mettre en place des algorithmes propres aux appareils.
- **Chatbots (assistants vocaux compris) et autres systèmes webhook :** l'approche sans serveur convient parfaitement à tout système webhook, comme les chatbots. Leur capacité à effectuer des actions (exécution de code, par exemple) uniquement en cas de besoin, comme lorsqu'un utilisateur demande des informations d'un chatbot, en fait une approche directe et typiquement à moindre coût pour ces infrastructures. Par exemple, la majorité des Skills d'Alexa pour Amazon Echo sont mises en place grâce à AWS Lambda.
- **Clickstream et autres processus de données de diffusion en temps quasi réel :** les solutions sans serveur apportent la flexibilité pour un dimensionnement en fonction du flux de données, ce qui leur permet de faire correspondre des exigences de débit sans la complexité de la création d'un système de calcul évolutif pour chaque application. En cas d'appairage avec une technologie comme Amazon Kinesis, AWS Lambda pour fournir un traitement rapide des archives pour l'analyse clickstream, les déclenchements de données NoSQL, les informations boursières et plus encore.

En plus de cas d'utilisation largement adoptés que nous avons vus plus tôt, les entreprises appliquent également l'approche sans serveur aux domaines suivants :

- Big Data, comme les problèmes de réduction de mappage, le transcodage de vidéo ultra rapide, analyse boursière et simulations Monte Carlo exigeantes en ressources pour les applications de prêt. Les développeurs ont découvert qu'il était plus facile d'effectuer un parallèle avec une approche sans serveur,[3] en particulier en cas de déclenchement par des événements, ce qui les a conduits à appliquer davantage de techniques sans serveur à un large éventail de problèmes de Big Data sans avoir besoin de gestion d'infrastructure.
- Faible latence, traitement personnalisé pour les applications Web et actifs fournis via des réseaux de déploiement de contenu. En faisant passer la gestion d'événements sans serveur à la limite d'Internet, les développeurs peuvent tirer parti d'une latence plus faible et de la capacité de personnaliser facilement les récupérations et les recherches de contenu. Cela ouvre un nouveau spectre de cas d'utilisation à latence optimisée en fonction de l'emplacement du client.
- Les appareils connectés qui apportent des fonctionnalités sans serveur comme les fonctions AWS Lambda pour l'exécution dans des appareils d'Internet des Objets (IoT) commerciaux, résidentiels et portables. Les solutions sans serveur comme les fonctions Lambda offrent une abstraction naturelle du matériel physique (et même virtuel) sous-jacent ce qui leur permet de passer plus facilement de centres de données à la limite, et d'une architecture matérielle à l'autre, sans perturber le modèle de programmation.
- Logique personnalisée et traitement des données dans des appliances sur site comme AWS Snowball Edge. Comme elles découplent la logique commerciale à partir des détails de l'environnement d'exécution, les applications sans serveur peuvent aisément fonctionner dans une grande diversité d'environnements, y compris sur une appliance.

Typiquement, les applications sans serveur sont conçues à l'aide d'une *architecture de microservices* dans laquelle une application est séparée en composants indépendants réalisant des tâches discrètes. Ces composants, constitués de fonctions Lambda individuelles et d'API, de files d'attente de messages, de bases de données et d'autres composants, peuvent être déployés, testés et dimensionnés indépendamment. En fait, les applications sans serveur conviennent parfaitement aux microservices du fait de leur modèle à fonction. En évitant les conceptions et les architectures monolithiques, les organisations peuvent devenir plus agiles, car les développeurs peuvent déployer de manière incrémentielle et remplacer ou mettre à niveau des composants individuels, comme le niveau de base de données, au besoin.

Dans de nombreux cas, la simple isolation de la logique commerciale d'une application est la seule chose nécessaire pour la convertir en application sans serveur. Les services comme AWS Lambda prennent en charge les langages de programmation populaires et permettent l'utilisation de bibliothèques personnalisées. Les tâches au long cours sont exprimées comme des flux de travail composés de fonctions individuelles opérant dans des périodes raisonnables, ce qui permet au système de redémarrer ou paralléliser des unités individuelles de calcul, selon les besoins.

Le sans serveur est-il toujours approprié ?

Presque toutes les applications modernes peuvent être modifiées pour s'exécuter correctement et dans la plupart des cas d'une manière plus économique et dimensionnable, sur une plate-forme sans serveur. Cependant, il arrive parfois que le sans serveur ne soit pas le meilleur choix :

- lorsque l'objectif consiste explicitement à éviter d'apporter des modifications à une application ;
- lorsqu'un contrôle fin sur l'environnement est nécessaire (spécification particulière de correctifs de système d'exploitation ou accès aux opérations de mise en réseau à bas niveau pour une exécution correcte du code) ;
- lorsqu'une application sur site n'a pas été migrée vers le cloud.

Evaluation de la plate-forme sans serveur d'un fournisseur cloud

Lors de l'architecturage d'une application sans serveur, les entreprises doivent prendre en considération autre chose que les fonctionnalités de calcul sans serveur exécutant le code de l'application. Des applications sans serveur complètes nécessitent une large gamme de services, d'outils et de capacités s'étendant au stockage, à la messagerie, au diagnostic et plus encore. Le portfolio sans serveur incomplet ou fragmenté d'un fournisseur cloud peut être problématique pour les développeurs sans serveur qui peuvent avoir à revenir à une architecture sur serveur s'ils ne parviennent pas à coder à un niveau constant d'abstraction.

Une plate-forme sans serveur se compose de l'ensemble de services comprenant l'application sans serveur (composants de stockage et de calcul) ainsi que des outils requis pour écrire, créer, déployer et diagnostiquer des applications sans serveur. L'exécution d'une application sans serveur dans la production nécessite une plate-forme fiable, flexible et de confiance capable de gérer les demandes de petites start-ups aux corporations multinationales. La plate-forme doit se dimensionner en fonction de *tous* les éléments d'application et assurer une certaine fiabilité de bout en bout. Comme avec les applications conventionnelles, l'aide apportée aux développeurs pour qu'ils parviennent à créer et fournir des solutions sans serveur est un défi à plusieurs facettes. Pour répondre aux besoins d'entreprises à grande échelle sur divers secteurs, une plate-forme sans serveur doit fournir les capacités suivante :

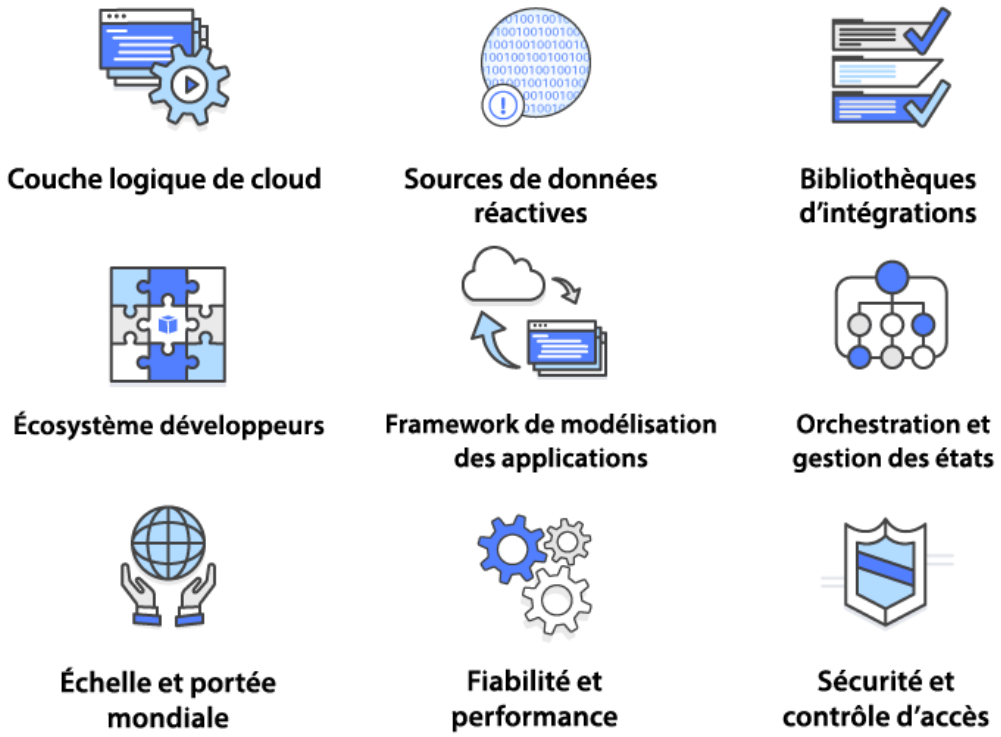


Figure 1 : Capacités d'une plate-forme sans serveur

- *Couche logique cloud* hautes performances, dimensionnable et fiable.
- *Sources d'événement et de données* premières réactives et simple *connectivité à des systèmes tiers*.

- *Bibliothèques d'intégration* permettant aux développeurs de démarrer en toute simplicité et d'ajouter de nouveaux modèles rapidement et en toute sécurité à des solutions existantes.
- Un *écosystème de développeurs* bouillonnant aidant les développeurs à découvrir et appliquer des solutions dans divers domaines et pour un large ensemble de systèmes tiers et de cas d'utilisation.
- Collection de *frameworks modélisant des applications* d'adaptation aux objectifs.
- *Orchestration* pour une gestion de flux de travail et d'état.
- *Echelle globale* et large portée incluant la certification de programme d'assurance.
- *Fiabilité intégrée et performances à l'échelle* sans avoir besoin de provisionner une capacité à tout niveau d'échelle.
- *Sécurité intégrée* avec *contrôle d'accès flexible* pour les ressources et services premiers et tiers.

Au cœur de toute plate-forme sans serveur se trouve la *couche logique cloud* responsable de l'exécution des fonctions représentant la logique commerciale. Comme ces fonctions sont souvent exécutées en réponse à des événements, *l'intégration aux sources d'événement premières et tierces* est essentielle pour simplifier l'expression des solutions et leur permettre de se dimensionner automatiquement en réponse aux charges de travail variables. Par exemple, les fonctions sans serveur peuvent devoir s'exécuter à chaque fois qu'un objet est créé dans un stockage d'objets ou pour chaque mise à jour effectuée sur une base de données NoSQL sans serveur. Les architectures sans serveur éliminent tout le code de dimensionnement et de gestion généralement requis pour intégrer chaque système, faisant ainsi passer cette charge opérationnelle au fournisseur cloud.

La réussite d'un développement sur une plate-forme sans serveur nécessite qu'une entreprise soit capable de démarrer facilement et de trouver des modèles prêts à l'emploi pour des cas courants d'utilisation, qu'ils impliquent des services premiers ou tiers. Ces *bibliothèques d'intégration* sont essentielles pour véhiculer des modèles efficaces, comme le traitement de flux d'archives ou la mise en place de webhooks, en particulier pendant la période où les développeurs passent d'une architecture sur serveur à une architecture sans serveur. Un besoin étroitement lié est un *écosystème large et varié* autour de la plate-forme de base. Un écosystème

large et actif aidant les développeurs à se préparer à découvrir et utiliser des solutions de la communauté et simplifiant la contribution à de nouvelles idées et approches. Au vu de la variété des chaînes d'outils utilisées pour la gestion du cycle de vie d'applications, un écosystème sain est également nécessaire pour s'assurer que chaque technologie créée par un langage, un IDE et une entreprise dispose des délais d'exécution, des plug-ins et des solutions open source nécessaires à l'intégration de la création et du déploiement d'applications sans serveur dans les approches existantes. Il est également essentiel pour les applications sans serveur d'exploiter des investissements existants, y compris la connaissance des frameworks par les développeurs comme Express, Flask et des langages de programmation populaires. Un large écosystème apporte un boost important dans divers domaines et permet aux développeurs de réorienter le code existant de façon mieux préparée dans une architecture sans serveur.

Les *frameworks de modélisation d'application*, comme la spécification ouverte AWS Serverless Application Model (AWS SAM), permettent aux développeurs d'exprimer les composants qui constituent une application sans serveur et activent les outils et flux de travail nécessaires pour créer, déployer et surveiller ces applications. Un autre framework essentiel à la réussite d'une plate-forme sans serveur est *l'orchestration et la gestion d'état*. La vaste nature sans état du calcul sans serveur nécessite un mécanisme complémentaire pour l'activation des flux de travail au long cours. Les solutions d'orchestration permettent aux développeurs de coordonner les multiples composants d'application liés courants dans une application sans serveur tout en activant ces applications pour qu'elles soient composées de petites fonctions à courte durée de vie. Les services d'orchestration simplifient également la gestion des erreurs et offrent une intégration à des systèmes et flux de travail hérités incluant ceux qui s'exécutent plus longtemps que le permettent généralement les fonctions sans serveur elles-mêmes.

Pour assister les clients du monde entier, y compris les corporations multinationales à portée mondiale, une plate-forme sans serveur doit fournir *une activité à l'échelle mondiale*, y compris des centres de données et des emplacements périphériques dans le monde entier. Les emplacements périphériques sont essentiels à l'apport d'un calcul sans serveur et à faible latence aux utilisateurs finaux. Comme la plate-forme, plutôt que le développeur d'applications, est responsable de l'apport de l'évolutivité et de la forte disponibilité des applications sans serveur, sa *fiabilité* intrinsèque est essentielle. Les fonctionnalités comme les nouveaux essais intégrés et les files d'attente de lettres mortes pour les événements non traités aident les développeurs à construire des systèmes robustes avec une fiabilité bout en bout

exploitant l'approche sans serveur. Les performances sont tout autant essentielles, en particulier la faible latence (frais généraux), à condition que le les délais d'exécution du langage et que le code client soient exemplifiés à la demande dans une application sans serveur.

Enfin, la plate-forme doit disposer d'un large éventail de *contrôle d'accès et de sécurité*, y compris la prise en charge de réseaux virtuels privés, les autorisations en fonction des rôles et des accès, l'intégration robuste avec l'authentification par API et les mécanismes de contrôle d'accès (dont des systèmes hérités et tiers), ainsi que la prise en charge des éléments d'application de chiffrement comme les paramètres variables d'environnement. Les systèmes sans serveur, par leur conception, proposent d'emblée un niveau supérieur de sécurité et de contrôle pour les raisons suivantes :

- **Gestion de flotte de premier ordre avec application de correctif de sécurité :** dans un système comme AWS Lambda, les serveurs exécutant des requêtes sont constamment surveillés, mis en cycle et analysés à des fins de sécurité. Ils peuvent être corrigés dans les quelques heures qui suivent la mise à disposition des mises à jour de sécurité, contrairement aux flottes de calcul de nombreuses entreprises ayant des SLA plus longs pour l'application de correctifs et de mises à jour.
- **Durée de vie des serveurs limitée :** chaque machine exécutant le code du client dans AWS Lambda est mise en cycle plusieurs fois par jour, ce qui limite son exposition aux attaques et garantit la mise à jour constante du système d'exploitation et l'application de correctifs.
- **Authentification par demande, contrôle d'accès et audit :** chaque demande de calcul exécutée sur AWS Lambda, quelle qu'en soit la course, est individuellement authentifiée, l'accès aux ressources spécifiques lui est accordé et elle est entièrement auditée. Les demandes arrivant de l'extérieur des centres de données AWS par Amazon API Gateway offrent des systèmes supplémentaires de défense face à Internet, y compris les défenses contre les attaques DoS. Les entreprises passant aux architectures sans serveur peuvent utiliser AWS CloudTrail pour profiter d'informations détaillées sur les utilisateurs qui accèdent aux systèmes et avec quels privilèges, et ils peuvent utiliser AWS Lambda pour traiter les archives d'audit par programme.

La plateforme sans serveur AWS

Depuis la sortie de Lambda en 2014, AWS a créé une plate-forme sans serveur complète. Le service dispose d'une collection de services entièrement gérés permettant aux organisations de créer des applications sans serveur pouvant s'intégrer en toute transparence aux autres services AWS et aux services tiers. La Figure illustre un sous-ensemble de composants dans la plate-forme sans serveur AWS ainsi que leurs relations.

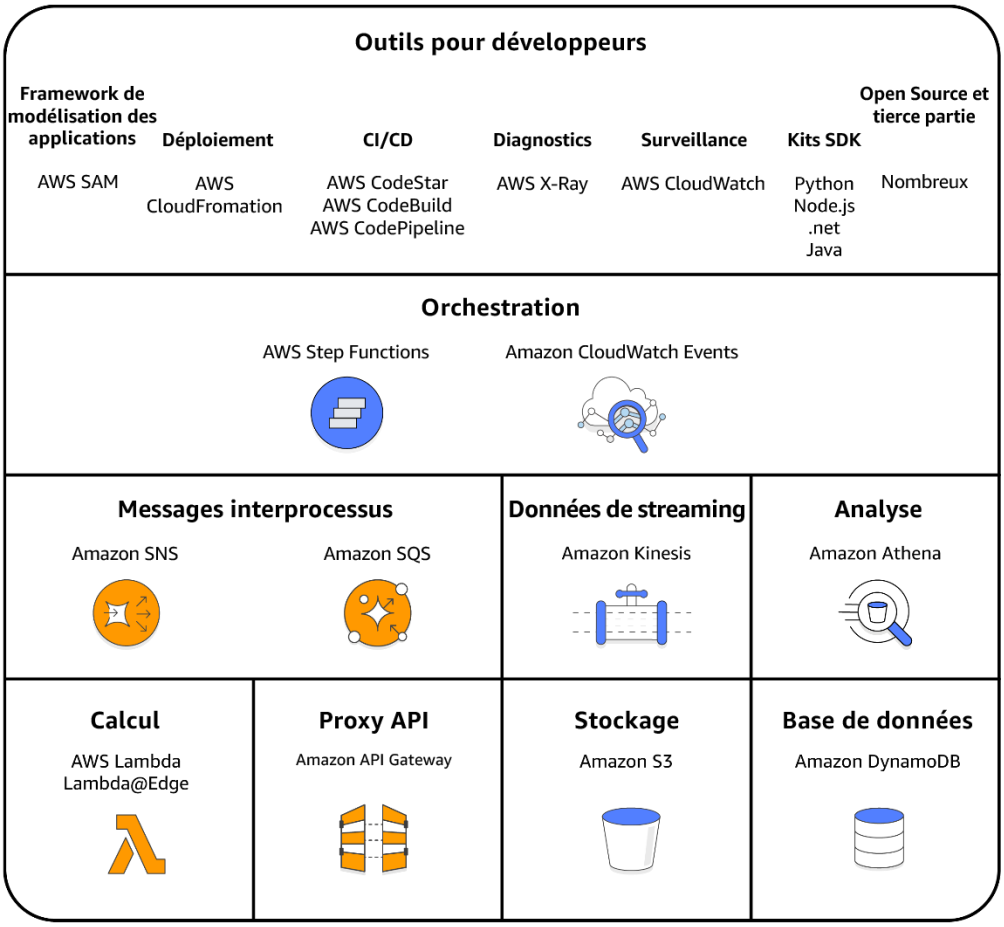
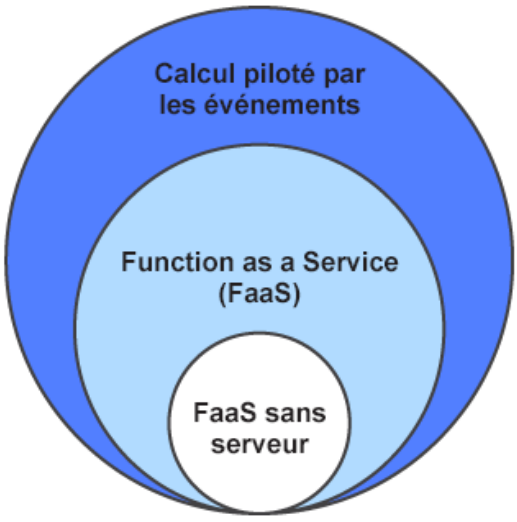


Figure 2 : Composants de plate-forme sans serveur AWS

Capacités de la plate-forme sans serveur AWS

AWS affiche toutes les capacités de base identifiées dans la section précédente comme conditions pour une plate-forme sans serveur complète. La couche logique cloud est fournie par AWS Lambda, une offre de calcul sans serveur ni provision à grande échelle basée sur des fonctions. AWS Lambda est complété par AWS Lambda@Edge, qui fournit une assistance similaire pour l'exécution de fonctions Lambda à extrêmement faible latence grâce à un routage optimisé pour edge, ainsi que par AWS Greengrass, qui permet aux fonctions Lambda de s'exécuter sur des appareils connectés, y compris des appliances comme AWS Snowball.

Les fonctions Lambda peuvent être aisément déclenchées par divers événements premiers ou tiers, ce qui permet aux développeurs de créer des systèmes réactifs orientés événements (cf. Figure 3) sans l'embarras conventionnel de la configuration et de la gestion d'infrastructure. Lorsqu'il y a plusieurs événements simultanés, Lambda exécute simplement davantage de copies de la fonction en parallèle, chacune répondant à chaque déclenchement individuel. Les fonctions Lambda s'adaptent précisément à la taille de la charge de travail et en fonction de la demande individuelle. Par conséquent, il n'y a pas moyen d'avoir un serveur ou un conteneur non utilisé. Le problème de dépenses d'infrastructure gaspillées est éliminé *d'emblée* avec les architectures utilisant les fonctions Lambda.



FaaS, ou *Function as a Service* (fonction en tant que service), est approche de création de systèmes de calcul orientés événements reposant sur des fonctions étant une unité de déploiement et d'exécution. Le *FaaS sans serveur* est un type de FaaS dans lequel aucune machine ou conteneur virtuels n'est présent dans le modèle de programmation et dans lequel le fournisseur offre une évolutivité sans provision et une fiabilité intégrée.

Figure 3 : Relation entre le calcul orienté événements, le FaaS et le sans serveur.

Les capacités de calcul sans serveur d'AWS fournies par Lambda sont un élément clé des services gérés suivants proposés par AWS, chacun s'intégrant en toute transparence aux autres :

- Amazon API Gateway : points de terminaison HTTP pour les fonctions Lambda dont une gamme complète de capacité des proxy d'API et de gestion d'API.
- Amazon S3 : les fonctions Lambda peuvent être utilisées comme déclencheurs automatiques d'événements lorsqu'un objet est créé, copié et supprimé.
- Amazon DynamoDB : les fonctions Lambda peuvent être utilisées pour traiter tout ou partie des modifications apportées à un tableau de base de données.
- Amazon SNS : les messages peuvent être routés vers des fonctions Lambda en vue de traitement, ce qui permet d'ajouter la possibilité de répondre de manière dynamique au contenu publié.
- Amazon SQS : les messages dans les files d'attente peuvent être traités aisément par les fonctions Lambda.
- Amazon Kinesis Streams : le traitement d'archive en ordre de données de flux est fourni par les fonctions Lambda, ce qui simplifie la création de moteurs d'analyse en temps quasi réel.
- Amazon Kinesis Firehose : les fonctions Lambda peuvent être appliquées automatiquement aux archives ingérées par un Firehose, ce qui simplifie l'ajout de la transformation, le filtrage et les capacités d'analyse dans un flux de données.
- Amazon Athena : les fonctions Lambda peuvent être automatiquement déclenchées pour chaque objet d'un ensemble de résultats d'une demande.
- AWS Step Functions : plusieurs fonctions Lambda peuvent être orchestrées pour créer des flux de travail au long cours pour les traitements orientés humains et automatisés.
- Amazon CloudWatch Events : les fonctions Lambda peuvent être utilisées pour répondre automatiquement aux événements dont les événements tiers.
- Amazon Aurora : les déclenchements de base de données peuvent être écrits comme des fonctions Lambda.

Lambda fournit une bibliothèque d'intégration contenant des plans pour un large choix de services tiers, dont Slack, Algorithmia, Twilio, Loggly, Splunk, SumoLogic, Box et d'autres, ce qui permet aux développeurs de créer des applications réactives qui incluent l'analyse, des algorithmes avancés, des communications et bien plus en seulement quelques lignes de code. Un large choix de frameworks d'applications Web, dont Express (pour les applications NodeJS) et Flask (pour les applications Python), a été amélioré pour bien fonctionner avec les fonctions Lambda. Les projets d'applications Web open source incluent le framework Serverless, Sparta, Chalice et bien d'autres.

Les applications sans serveur se composent généralement de plusieurs éléments : une fonction, voire plus, une base de données sans serveur (comme Amazon DynamoDB) et soit une API que les clients doivent appeler, soit une source d'événements qui déclenche l'application. Pour maintenir ces éléments en ordre, AWS utilise SAM, le Serverless Application Model de spécification open. Avec SAM, les développeurs peuvent aisément décrire les fonctions, les API, les sources d'événements, les tableaux de base de données et d'autres éléments d'une application sans serveur. L'utilisation de SAM aide également les développeurs à gérer toutes les étapes du cycle de vie de développement logiciel, et AWS propose un large choix d'outils et de services pour cela. Cela inclut la prise en charge native du test et du débogage locaux de leur IDE de prédilection (ou par ligne de commande) via SAM Local, par déploiement d'applications SAM à l'aide d'AWS CloudFormation et la prise en charge de CI/CD GitHub pour les applications SAM conçues dans AWS CodePipeline. En plus de la prise en charge première, un nombre de frameworks open source, de fournisseurs CI/CD et de fournisseurs de gestion de performances propose la prise en charge de SAM et des fonctions Lambda, y compris le framework Serveless, Claudia, CloudBees, Datadog et bien d'autres. Pour davantage d'exemples, consultez Serverless Application Developer Tooling.[4]

Après la création d'une fonction Lambda, ou dans le cas d'une application SAM éventuellement plusieurs fonctions Lambda opérant ensemble, les développeurs peuvent aisément la surveiller en utilisant des mesures et des journaux créés automatiquement disponibles dans Amazon CloudWatch et CloudWatch Logs. AWS propose également AWS X-Ray, une solution d'analyse de performances et de suivi de demande inter-services qui permet aux développeurs de suivre l'opération et le comportement de fonctions individuelles, ainsi que les événements qu'elles traitent.

Les offres de la plate-forme sans serveur AWS ont une portée globale et prennent en charge AWS Lambda et Amazon API Gateway dans virtuellement toutes les régions AWS du monde. Lambda@Edge est disponible dans tous les emplacements périphériques.[5] Lambda propose une gamme de fonctionnalités visant à aider les clients à améliorer la fiabilité de leurs applications, y compris les nouveaux essais automatiques pour les événements asynchrones et commandés, ainsi que les files d'attente de lettres mortes pour capture les événements non traités par l'application. L'intégration en profondeur avec Amazon Virtual Private Cloud (Amazon VPC) et la portée flexible des capacités de contrôle d'authentification et d'accès fournies par AWS Lambda permettent aux organisations de créer des applications sécurisées en accord avec les bonnes pratiques, comme le principe de moindre privilège. La gestion et la sécurité des utilisateurs finaux est également simplifiée : Amazon Cognito propose une autorisation et une authentification pouvant être combinée avec Amazon API Gateway et AWS Lambda, ce qui permet l'enregistrement d'utilisateurs sans serveur et des capacités de connexion, y compris l'intégration à des fournisseurs de réseaux sociaux comme Facebook et des référentiels d'entreprise.

Études de cas

Les entreprises ont appliqué des architectures sans serveur à des cas d'utilisation à partir de validation boursière pour la construction de sites Web d'e-commerce au traitement de langage naturel. AWS Lambda et le reste du portfolio sans serveur d'AWS proposent la flexibilité de créer un large choix d'applications incluant celles nécessaires aux programmes d'assurance comme la conformité PCI ou HIPAA. Les sections suivantes illustrent certains des cas d'utilisation les plus courants, mais ne sont en rien une liste exhaustive. Pour avoir une liste exhaustive de références client et de la documentation sur les cas d'utilisation, consultez Serverless Computing.[6]

Sites Web sans serveur, applications Web et backends mobiles

Les approches sans serveur sont idéales pour les applications où la charge peut varier dynamiquement. L'utilisation d'une approche sans serveur signifie qu'aucun coût de calcul n'est encouru lorsqu'il n'y a pas de trafic d'utilisateur final tout en offrant quand même un dimensionnement permettant de répondre à une forte demande, comme une vente flash sur un site d'e-commerce ou une mention de réseau social générant une soudaine vague de trafic. Par rapport aux approches

d'infrastructure traditionnelles, il est également souvent extrêmement moins coûteux de développer, déployer et opérer un backend Web ou mobile lorsqu'il a été architecturé sans serveur.

AWS fournit aux développeurs les services dont ils ont besoin pour créer rapidement ces applications :

- Amazon S3 propose une solution d'hébergement simple pour le contenu statique.
- AWS Lambda, en conjonction avec Amazon API Gateway, offre une prise en charge pour les demandes d'API dynamiques à l'aide de fonctions.
- Amazon DynamoDB propose une solution de stockage simple pour la session et l'état par utilisateur.
- Amazon Cognito fournit un moyen simple de gérer l'enregistrement d'utilisateurs finaux, l'authentification et le contrôle de l'accès aux ressources.
- AWS SAM peut être utilisé par les développeurs pour décrire les différents éléments d'une application.
- AWS CodeStar peut configurer une chaîne d'outils CI/CD en seulement quelques clics.

Pour en savoir plus, consultez le livre blanc AWS Serverless Multi-Tier Architectures qui fournit des informations détaillées sur l'examen des modèles pour la création d'applications Web sans serveur.[7] Pour les architectures de référence complètes, consultez Serverless Reference Architecture for creating a Web Application[8] et Serverless Reference Architecture for creating a Mobile Backend[9] sur GitHub.

Exemple client : Bustle.com

Bustle.com est un site Web d'actualités, de divertissement, de style de vie et de mode s'adressant aux femmes. Il a constaté une économie d'environ 84 % en passant à une architecture sans serveur basée sur AWS Lambda et Amazon API Gateway. Les ingénieurs de Bustle ont gagné en agilité, ce qui leur a permis de se concentrer sur la création de nouvelles fonctionnalités produit au lieu de s'occuper de la gestion et du dimensionnement de l'infrastructure. L'équipe de Bustle est désormais plus efficace avec la moitié de l'effectif normalement requis pour créer et opérer des sites à l'échelle de Bustle. Le backend sans serveur de Bustle prend également en charge les applications iOS pour deux de ses propriétés Web (Bustle et Romper). Pour en savoir plus, consultez l'étude de cas de Bustle.[10]

Backends d'IoT

Les avantages d'une architecture sans serveur apportés aux applications Web et mobiles simplifient également la création de backends d'IoT et de systèmes de traitement d'analyse sur appareil qui se dimensionnent en toute transparence selon le nombre d'appareils. Pour avoir un exemple d'architectures de référence, consultez Serverless Reference Architecture for creating an IoT Backend sur GitHub.[11]

Exemple client : iRobot

iRobot, qui crée des robots comme le robot de ménage Roomba, utilise AWS Lambda en conjonction avec le service AWS IoT pour créer un backend sans serveur pour sa plate-forme d'IoT. En utilisant une architecture sans serveur, l'équipe d'ingénierie d'iRobot n'a pas à se soucier de la gestion de l'infrastructure ou à écrire manuellement du code pour gérer la disponibilité et le dimensionnement. Cela leur permet d'innover plus rapidement et de rester concentrés sur les clients. Pour en savoir plus, consultez les diapos de leur présentation AWS re:Invent 2016 ou Serverless IoT Back Ends (IOT401)[12] ou regardez la vidéo.[13]

Traitement des données

Les plus grandes applications sans serveur traitent des volumes massifs de données, dont la plupart en temps réel. Les architectures typiques traitant des données sans serveur utilisent une combinaison d'Amazon Kinesis et AWS Lambda pour traiter des données de diffusion. Sinon elles combinent Amazon S3 et AWS Lambda pour déclencher du calcul en réponse à la création d'objets ou la mise à jour d'événements. Lorsque les charges de travail nécessitent une orchestration plus complexe qu'un simple déclenchement, les développeurs peuvent utiliser AWS Step Functions pour créer des flux de travail d'état et au long cours qui invoquent une ou plusieurs fonctions Lambda alors qu'elles progressent. Pour en savoir plus sur les architectures de traitement de données sans serveur, consultez les articles suivants sur GitHub :

- Serverless Reference Architecture for Real-time Stream Processing[14]
- Serverless Reference Architecture for Real-time File Processing[15]
- Image Recognition and Processing Backend reference architecture[16]

Exemple client : FINRA

La Financial Industry Regulatory Authority (FINRA, autorité de réglementation du secteur financier) a utilisé AWS Lambda pour créer une solution de traitement de données sans serveur lui permettant d'effectuer 500 milliards de validations de données sur 37 milliards d'événements de marché boursier chaque jour. Lors de sa présentation à l'AWS re:Invent 2016 intitulée The State of Serverless Computing (SVR311),[17] Tim Griesbach, directeur senior chez FINRA, a dit avoir trouvé que Lambda allait être pour l'entreprise la meilleure option pour cette solution cloud sans serveur. Avec Lambda, le système s'était montré plus rapide, moins coûteux et plus évolutif. Ainsi, à la fin de la journée, les coûts avaient été réduits de plus de 50 % ... et il était possible de suivre cette progression chaque jour, même chaque heure.

Exemple client : Thomson Reuters

Thomson Reuters, entreprise multimédia et d'information, a créé une solution d'analyse commerciale sans serveur qui permet aux équipes produit d'analyser aisément les données d'utilisation produit. La solution combine AWS Lambda, Amazon Kinesis Streams et Amazon Kinesis Firehose pour la collecte et le traitement de données d'événements de diffusion pour analyse. Le résultat, Product Insight, a été lancé avec deux mois d'avance et a dépassé les attentes techniques.

Anders Fritz, responsable senior de l'innovation produit chez Thomson Reuters, a dit que l'objectif initial était d'adapter 2 000 événements par seconde. Les tests montrent que Product Insight sur AWS peut traiter jusqu'à 4 000 événements par seconde et en un an, il était possible de s'attendre à une augmentation à plus de 10 000 événements par seconde. Cette figure représente plus de 25 milliards d'événements par mois. Même avec ce débit élevé, le système n'a perdu aucune donnée depuis son introduction. Du fait de l'architecture de basculement robuste et des capacités techniques d'AWS, aucun événement n'a été perdu depuis le début de la collecte de données, d'après Fritz. Pour en savoir plus, consultez l'article Cas d'étude de Thomson Reuters[18] ou regardez la présentation AWS re:Invent 2016 Real-time Data Processing Using AWS Lambda (SVR301).[19]

Big Data

AWS Lambda correspond parfaitement à de nombreuses charges de travail de traitement parallèle à haut volume. Pour avoir un exemple d'architectures de référence utilisant MapReduce, consultez l'article Reference architecture for running serverless MapReduce jobs.[20]

Exemple client : Fannie Mae

Fannie Mae, source leader de financement pour les prêteurs hypothécaires, exploite AWS Lambda pour exécuter une charge de travail « malheureusement parallèle » pour sa modélisation financière. Fannie Mae utilise les processus de simulation Monte Carlo pour projeter les futurs flux d'argent de prêts qui l'aident à gérer les risques de prêts. La société a découvert que ses grilles HPC existantes ne correspondaient plus à ses besoins commerciaux grandissants. Elle a créé sa nouvelle plate-forme sur Lambda et le système s'est dimensionné à hauteur de 15 000 exécutions de fonctions en cours pendant le test. Le nouveau système a exécuté une simulation sur 20 millions de prêts terminée en 2 heures, soit 3 fois plus rapidement que l'ancien système. Grâce à une architecture sans serveur, Fannie Mae peut désormais exécuter de manière rentable des simulations Monte Carlo à grande échelle du fait qu'elle ne paie pas les ressources de calcul non utilisées. La société peut également accélérer ses calculs en exécutant plusieurs fonctions Lambda simultanément. Fannie Mae a également constaté un délai de mise sur le marché nettement inférieur à la normale, car elle a été en mesure de dispenser une gestion et une surveillance de serveurs, ainsi que la capacité d'éliminer la majeure partie du code complexe auparavant requis pour gérer le dimensionnement et la fiabilité de l'application. Pour plus d'informations, consultez la présentation de Fannie Mae à l'AWS Summit 2017 SMC303: Real-time Data Processing Using AWS Lambda.[21]

Automatisation informatique

L'approche sans serveur élimine les frais généraux de gestion des serveurs, ce qui simplifie la création et la gestion de la plupart des tâches d'infrastructure, comme le provisionnement, la configuration, la gestion, les alertes/surveillances et les tâches cron chronométrées.

Exemple client : Autodesk

Autodesk, qui crée un logiciel d'ingénierie et de modélisation 3D, utilise AWS Lambda pour automatiser les processus de gestion et de création de son compte AWS dans son organisation d'ingénierie. Autodesk estime avoir réalisé des économies à hauteur de 98 % (mise en facteur en économies estimées en heures de travail passées à provisionner des comptes). Elle peut désormais provisionner des comptes en seulement 10 minutes au lieu des 10 heures nécessaires avec l'ancien processus d'infrastructure. La solution sans serveur permet à Autodesk de provisionner automatiquement des comptes, de configurer et mettre en application des normes et exécuter des audits grâce à une automatisation accrue et moins points de contact manuels. Pour plus d'informations, consultez la présentation d'Autodesk à l'AWS Summit 2017 SMC301: The State of Serverless Computing.[22] Rendez-vous sur GitHub pour découvrir le service Autodesk Tailor.

Cas d'utilisation supplémentaires

Les cas d'utilisation décrits dans la section précédente ne sont que la partie émergée de l'iceberg des possibilités des offres sans serveur de Lambda et AWS. D'autres cas d'utilisation incluent une puissante compréhension du langage humain par des chatbots créés à l'aide d'Amazon Lex et AWS Lambda, un Edge Computing global à faible latence utilisant Lambda@Edge avec Amazon CloudFront et un puissant traitement de fichiers sur site avec les fonctions Lambda dans AWS Snowball. Voici quelques capacités passionnantes de cette approche polyvalente. Pour en savoir plus, consultez AWS Lambda.[23]

Conclusion

Les approches sans serveur sont conçues pour résoudre deux problèmes majeurs de la gestion informatique : les serveurs non utilisés qui handicapent le budget des entreprises sans apporter de valeur, ainsi que les coûts de création et d'exploitation de flottes de serveurs, et de logiciels de serveurs, qui détournent les équipes de la création de valeur client bien différentiée. AWS Lambda et les autres offres sans serveur d'AWS résolvent ces problèmes de tout temps en éliminant les serveurs, les conteneurs, les disques et d'autres ressources au niveau de l'infrastructure du modèle de programmation et de facturation. Par conséquent, les développeurs peuvent fonctionner avec un modèle d'application clair qui les aide à accélérer les déploiements. De plus, les entreprises ne paient que pour le travail utile. Le moyen

le plus simple et le plus rapide d'architecturer des systèmes réactifs sur événements et de déployer des microservices cloud consiste à utiliser des architectures sans serveur. Pour en savoir plus et consulter les livres blancs concernant les rubriques liées, consultez Serverless Computing and Applications.[24]

Participants

Les personnes et organisations suivantes ont participé à l'élaboration de ce document :

- Tim Wagner, responsable général d'AWS Serverless Applications chez Amazon Web Services

Suggestions de lecture

Pour plus d'informations, consultez les éléments suivants :

- Serverless Reference Architectures with AWS Lambda par Werner Vogels, CTO chez Amazon.com
- AWS re:Invent 2016: The State of Serverless Computing (présentation) par Tim Wagner, responsable général d'AWS Serverless Applications
- The economics of serverless cloud computing par Owen Rogers, directeur de recherche chez 451 Research

Architectures de référence

- Applications web
- Backends mobiles
- Backends d'IoT
- Traitement des fichiers
- Traitement des flux
- Traitement de la reconnaissance des images
- MapReduce

Révisions du document

Date	Description
Septembre 2017	Première publication

Notes

[1] https://www.forbes.com/sites/moorinsights/2016/04/11/tco-analysis-demonstrates-how-moving-to-the-cloud-can-save-your-company-money/#537e2bd07c4e
http://www.cloudstrategymag.com/articles/86033-understanding-tco-cloud-economics

[2] En 2012, Gartner a estimé que l'utilisation de centres de données oscillait entre 7 et 12 % (http://www.nytimes.com/2012/09/23/technology/data-centers-waste-vast-amounts-of-energy-belying-industry-image.html). Une étude menée en 2008 par McKinsey l'a estimée à 6 % (https://www.sallan.org/pdf-docs/McKinsey_Data_Center_Efficiency.pdf). Un rapport d'Accenture sur l'analyse d'un ensemble d'applications basées sur EC2 a montré une utilisation d'environ 7 % (http://ieeexplore.ieee.org/document/6118751/). En 2014, une étude de NRDC et Anthesis a montré qu'en 2013, plus de 30 % des serveurs étaient entièrement « comateux » (connectés mais sans réelle valeur ajoutée) (http://anthesisgroup.com/wp-content/uploads/2015/06/Case-Study_DataSupports30PercentComatoseEstimate-FINAL_06032015.pdf).

[3] Occupez le cloud : Eric Jonas et al., *Distributed Computing for the 99%*, https://arxiv.org/abs/1702.04024.

[4] https://aws.amazon.com/serverless/developer-tools

[5] https://aws.amazon.com/lambda/edge/

[6] https://aws.amazon.com/serverless

[7] https://d0.awsstatic.com/whitepapers/AWS_Serverless_Multi-Tier_Architectures.pdf

[8] https://github.com/awslabs/lambda-refarch-webapp

[9] https://github.com/awslabs/lambda-refarch-mobilebackend

[10] https://aws.amazon.com/solutions/case-studies/bustle/

[11] https://github.com/awslabs/lambda-refarch-iotbackend

[12] https://www.slideshare.net/AmazonWebServices/aws-reinvent-2016-serverless-iot-back-ends-iot401

[13] https://www.youtube.com/watch?v=gKMaf5E-z7Q

[14] https://github.com/awslabs/lambda-refarch-streamprocessing

[15] https://github.com/awslabs/lambda-refarch-fileprocessing

[16] https://github.com/awslabs/lambda-refarch-imagerecognition

[17] https://www.youtube.com/watch?v=AcGv3qUrRC4&feature=youtu.be&t=1153

[18] https://aws.amazon.com/solutions/case-studies/thomson-reuters/

[19] https://www.youtube.com/watch?v=VFLKOy4GKXQ&feature=youtu.be&t=1449

[20] https://github.com/awslabs/lambda-refarch-mapreduce

[21] https://www.slideshare.net/AmazonWebServices/smc303-realtime-data-processing-using-aws-lambda/28

[22] https://www.slideshare.net/AmazonWebServices/smc301-the-state-of-serverless-computing-75290821/22

[23] https://aws.amazon.com/lambda

[24] https://aws.amazon.com/serverless