Engineer

Serverless

Exécuter du code sans gérer les serveurs qui l'hébergent

Ce que c'est

Le serverless est un mode d'exécution dans lequel l'équipe écrit et déploie l'application tandis qu'un fournisseur exploite les machines sous-jacentes, en adapte la capacité et les entretient. Le livre blanc du CNCF Serverless Working Group, qui emploie l'expression plus longue « informatique sans serveur », le présente comme la possibilité de construire et d'exécuter des applications sans qu'un membre de l'équipe ait à gérer de serveur.

Le nom exagère la réalité, et le livre blanc le dit clairement : « Des serveurs restent nécessaires pour faire fonctionner une plateforme serverless. » Les serveurs existent toujours, tout comme le travail d'exploitation. Ce qui change, c'est la personne qui s'en charge. L'équipe ne choisit plus la taille des machines, n'installe plus le système d'exploitation, ne lui applique plus de correctifs et ne décide plus du nombre d'instances à exécuter. Elle remet du code ou une image de conteneur, avec une description des conditions dans lesquelles ce code doit s'exécuter.

Le livre blanc distingue deux volets. Les fonctions en tant que service, ou FaaS, relèvent de l'informatique événementielle : de petits morceaux de logique s'exécutent en réaction à un événement. Ce qui démarre une fonction s'appelle un déclencheur ; il résulte généralement d'un événement comme une requête HTTP entrante ou une modification dans une base de données. Le backend en tant que service, ou BaaS, constitue l'autre volet : des services tiers accessibles par API remplacent des parties entières des fonctionnalités de l'application. Les fonctions sont censées être sans état et de courte durée ; c'est donc du côté BaaS que l'état est conservé.

Deux produits montrent l'étendue du modèle. AWS Lambda se décrit comme un service de calcul serverless qui exécute du code sans provisionner ni gérer de serveurs. Sa forme d'origine est la fonction événementielle : un gestionnaire et un déclencheur. Azure Container Apps se présente comme un service qui exécute des conteneurs applicatifs pendant que le fournisseur exploite l'infrastructure. Le premier reçoit une fonction, le second un conteneur entier, et les deux confient les machines au fournisseur.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que le serverless ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des serveurs dimensionnés selon vos usages réels, migrés vers le cloud ou hors du cloud, avec une estimation préalable de la facture mensuelle.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 20 applications Docker sur Google Cloud et AWS — 20 applications conteneurisées déployées et surveillées

20 applications Docker sur Google Cloud et AWS Contactez-nous →

Pourquoi c'est important

Le modèle de facturation est la première raison pour laquelle les équipes s'y intéressent. Le livre blanc décrit une tarification à l'usage, facturant seulement le temps d'exécution du code, sans frais lorsque rien ne tourne. Lambda facture une fonction événementielle par requête et selon le temps d'exécution consommé, mesuré en gigaoctets-secondes. Un service actif une heure par jour paie cette heure, plutôt qu'un serveur qui attend pendant les vingt-trois autres.

La deuxième raison est l'ajustement de capacité dans les deux sens. Lambda crée et détruit des environnements d'exécution lorsque le trafic augmente ou diminue. Le livre blanc décrit des fonctions dont le nombre d'instances tombe à zéro lorsqu'il n'y a plus de travail. Microsoft précise que Container Apps adapte sa capacité au trafic HTTP, aux événements ou à la charge du processeur et de la mémoire, et que la plupart des applications peuvent descendre à zéro instance.

La troisième raison est le travail dont l'équipe n'a plus à se charger. AWS compte l'entretien des serveurs, le provisionnement de capacité, l'ajustement de capacité et l'application de correctifs parmi les tâches assurées par Lambda. Pour une petite équipe, ces corvées peuvent remplir une semaine sans rien livrer. Avec une plateforme serverless, elles deviennent la responsabilité du fournisseur.

Les cas d'usage du livre blanc montrent où ce modèle convient le mieux : réagir aux modifications d'une base de données, traiter les données de capteurs, gérer des flux de données et exécuter de courtes tâches planifiées qui seraient autrement des tâches cron. La liste de Microsoft pour Container Apps ajoute les points d'accès d'API, les traitements en arrière-plan et les microservices.

Le même livre blanc avertit aussi que le serverless peut sembler être l'option la plus économique au premier abord, et demande de considérer l'ensemble des coûts avant de s'engager. La facturation à l'usage avantage les charges qui arrivent par pics et restent inactives la plupart du temps. Le calcul diffère pour une charge qui occupe une machine en permanence. Le débogage représente lui aussi un coût : le livre blanc indique qu'il peut être plus difficile qu'avec des offres IaaS ou PaaS.

À quoi cela ressemble en pratique

La documentation AWS des gestionnaires Python contient un exemple qui reçoit une commande, produit un reçu textuel et le stocke dans un bucket Amazon S3. En voici une version abrégée :

```
import os

import boto3

# Runs in the Init phase, once per execution environment.
s3_client = boto3.client("s3")


def lambda_handler(event, context):
    order_id = event["Order_id"]
    bucket_name = os.environ.get("RECEIPT_BUCKET")
    receipt_content = (
        f"OrderID: {order_id}\n"
        f"Amount: ${event['Amount']}\n"
        f"Item: {event['Item']}"
    )
    key = f"receipts/{order_id}.txt"
    s3_client.put_object(Bucket=bucket_name, Key=key, Body=receipt_content)
    return {"statusCode": 200, "message": "Receipt processed successfully"}
```

Le gestionnaire reçoit exactement deux arguments. L'événement porte les données à traiter, et un événement JSON arrive sous forme de dictionnaire Python. Ce gestionnaire attend donc une entrée de cette forme :

```
{"Order_id": "12345", "Amount": 199.99, "Item": "Wireless Headphones"}
```

Le contexte porte des informations sur l'invocation elle-même. Si le fichier s'appelle `lambda_function.py`, le gestionnaire est nommé `lambda_function.lambda_handler`, valeur proposée par défaut par la console Lambda pour une nouvelle fonction. Le nom du bucket vient d'une variable d'environnement plutôt que du code, conformément à la recommandation d'AWS pour transmettre les paramètres d'exploitation. Le code ne contient pas d'informations d'authentification : Lambda les fournit au SDK par ses propres variables d'environnement.

L'endroit où le client est créé dans le code compte. Pour les fonctions événementielles à la demande sans SnapStart ni Managed Instances, Lambda fait passer un environnement d'exécution par trois phases : Init, Invoke et Shutdown. Le code extérieur au gestionnaire s'exécute une fois, pendant Init, et AWS limite la tentative initiale d'Init à 10 secondes. Le gestionnaire s'exécute ensuite pour chaque événement, dans le délai maximal de la fonction. Lorsqu'un environnement est réutilisé pour l'événement suivant, ce qu'AWS appelle un démarrage à chaud, le client créé pendant Init est toujours présent et n'est pas reconstruit.

Le serverless ne se limite pas aux fonctions. Sur ce site, la réalisation consacrée au provisionnement d'un environnement Azure sous forme de code décrit deux services, `api` et `www`, déployés sur Azure Container Apps, avec leurs images de conteneurs conservées dans l'Azure Container Registry du projet. Microsoft présente Container Apps comme une plateforme serverless pour conteneurs : ces deux services constituent donc la partie serverless de cette infrastructure. Chaque environnement est décrit par un même ensemble de modèles Bicep paramétrés, ce qui permet à la réalisation de présenter un nouvel environnement comme une pull request.

Le choix inverse peut aussi être le bon. La réalisation consacrée à l'examen d'une infrastructure et au déplacement des charges hors d'AWS part d'une configuration AWS largement surdimensionnée par rapport à l'usage de l'entreprise, qui comportait encore des services inutiles. Les services inutilisés ont été supprimés, la capacité restante ajustée et les charges déplacées hors d'AWS. La facturation serverless est un moyen de ne plus payer de capacité inactive ; examiner et supprimer l'infrastructure inutilisée en est un autre. Il

faudrait comparer les charges et coûts réels pour déterminer ce qui coûte le moins : la réalisation consigne des changements, pas un ratio de coûts à périmètre identique.

Où les problèmes surviennent

La plupart des défaillances du serverless viennent d'une erreur : traiter une fonction comme un petit serveur toujours allumé, ce qu'elle n'est pas.

Les démarrages à froid retardent certaines requêtes. Lorsqu'aucun environnement n'est prêt, Lambda doit télécharger le code, préparer un environnement et exécuter le code Init avant que le gestionnaire voie l'événement. AWS indique que les démarrages à froid touchent généralement moins de 1 % des invocations, durent de moins de 100 millisecondes à plus d'une seconde, sont plus fréquents dans les fonctions de développement et de test, et que le temps d'Init est facturé. La solution consiste à limiter le travail de la phase Init, en important seulement ce dont la fonction a besoin et en chargeant à la demande les objets rarement utilisés. Lorsque le délai compte, la simultanéité provisionnée maintient un nombre choisi d'environnements initialisés à l'avance.

Les données conservées dans l'environnement sont exposées ou disparaissent. Chaque invocation s'exécute indépendamment, et un environnement peut être réutilisé ou non. AWS remplace aussi les environnements à intervalles de quelques heures pour mettre le runtime à jour, même à trafic constant. Après un plantage ou un dépassement de délai, une réinitialisation ne vide pas le répertoire `/tmp`. Les données d'un événement peuvent donc réapparaître pendant un autre, ou disparaître sans avertissement. La solution consiste à ne conserver ni données utilisateur ni événements dans l'environnement, et à placer ce qui doit persister dans une base de données ou un service de stockage.

Le même événement est traité deux fois. Une fonction peut recevoir des événements en double. Les bonnes pratiques d'AWS demandent du code idempotent, qui valide chaque événement et traite un doublon comme l'original. Dans l'exemple ci-dessus, la clé du reçu dérive de l'identifiant de commande : une seconde exécution écrase donc le même objet au lieu de créer un second reçu. Tout effet de bord qui ne peut pas être répété sans risque, comme envoyer un email ou débiter une carte, nécessite sa propre trace de ce qui a déjà été fait.

Les limites strictes arrêtent le travail en cours. Pour les fonctions événementielles hors Lambda Managed Instances, une invocation dure au maximum 900 secondes, soit 15 minutes. Une requête synchrone et une réponse synchrone mise en tampon sont chacune limitées à 6 Mo, une requête asynchrone à 1 Mo et l'ensemble des variables d'environnement à 4 Ko. Dépasser une limite produit une erreur, pas un simple ralentissement. Il faut tenir compte de ces limites dès la conception : découper les traitements longs en étapes, conserver les données volumineuses dans un système de stockage et en transmettre une référence, et déplacer ce qui ne convient pas vers une plateforme de conteneurs ou une machine virtuelle.

Le trafic admis dépasse la capacité d'exécution simultanée. Le quota par défaut de Lambda est de 1 000 exécutions simultanées par région, alors qu'API Gateway autorise par défaut 10 000 requêtes par seconde. La porte d'entrée peut donc accepter davantage de requêtes que les fonctions derrière elle ne peuvent en exécuter. Les nouveaux comptes commencent avec des quotas de simultanéité et de mémoire réduits, qu'AWS relève automatiquement selon l'utilisation. La solution est de demander une augmentation avant le lancement et d'effectuer un test de charge au pic attendu.

Une fonction qui se déclenche elle-même fait monter la facture. Une fonction qui s'invoque, ou démarre un processus qui l'invoque à nouveau, peut provoquer un flot d'invocations, toutes facturées. AWS recommande d'éviter les invocations récursives. Si un volume inattendu apparaît, il conseille de fixer immédiatement la simultanéité réservée de la fonction à `0`, ce qui empêche de nouvelles exécutions jusqu'à la correction du code.

Le modèle de programmation vous lie à une plateforme. Le livre blanc cite le modèle de programmation de la plateforme, son interface d'événements et de messages et ses offres BaaS comme sources de dépendance au fournisseur. La solution est de garder la logique métier dans des fonctions ordinaires que le gestionnaire se contente d'appeler, et de privilégier des plateformes comme Container Apps, capables d'exécuter une image de conteneur standard depuis n'importe quel registre, lorsqu'un déplacement ultérieur est réellement envisageable.

Pour aller plus loin

- Livre blanc CNCF Serverless, version 1.0 — Les définitions du serverless, du FaaS et du BaaS par le groupe de travail, ainsi que leurs inconvénients.

- Qu’est-ce qu’AWS Lambda ? — Ce que Lambda gère à votre place, ses types de fonctions et son modèle de facturation.
- Définir un gestionnaire de fonction événementielle Lambda en Python — L’exemple du reçu au complet, le nommage du gestionnaire et les bonnes pratiques de code.
- Comprendre le cycle de vie d’un environnement d’exécution Lambda — Les phases Init, Invoke et Shutdown, les démarrages à froid et la réutilisation des environnements.
- Quotas Lambda — Les limites d’exécution simultanée, de mémoire, de durée et de charge utile, et la façon de les augmenter.
- Présentation d’Azure Container Apps — La plateforme serverless de Microsoft pour les conteneurs et la réduction à zéro instance.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu’est-ce que le serverless ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/serverless/