AYI NEDJIMI Consultants

Serverless Security : Sécuriser Lambda et Functions Cloud

Catégorie : Cloud Security Lecture : 9 min Publié le : 12/03/2026 Auteur : Ayi NEDJIMI

Guide sécurité serverless complet : protection Lambda Azure Functions GCP, gestion IAM moindre privilège, validation entrées et monitoring runtime.

L'adoption du serverless a franchi un cap décisif en 2026, avec une majorité d'organisations utilisant au moins un service de fonctions cloud en production. AWS Lambda, Azure Functions et Google Cloud Functions permettent d'exécuter du code sans gérer d'infrastructure serveur, offrant une scalabilité automatique, un modèle de facturation à l'utilisation et une réduction significative de la charge opérationnelle. Cependant, cette abstraction de l'infrastructure crée un faux sentiment de sécurité qui conduit à des erreurs graves de configuration et de développement. Le modèle de responsabilité partagée se déplace vers le haut de la pile applicative : si le cloud provider gère le système d'exploitation, le runtime et le réseau, le client reste entièrement responsable du code, des permissions, des dépendances et de la configuration des événements déclencheurs. Ce guide détaille les menaces spécifiques au serverless, les bonnes pratiques de sécurité par cloud provider et les outils de protection adaptés aux architectures event-driven, en s'appuyant sur les retours d'expérience de nos audits de sécurité d'applications serverless dans des secteurs réglementés.

Résumé exécutif

Guide de sécurité serverless complet : protection des fonctions AWS Lambda, Azure Functions et Google Cloud Functions. Gestion IAM, validation des entrées, sécurisation des dépendances, monitoring et bonnes pratiques de configuration. La migration vers le cloud transforme radicalement les paradigmes de sécurité : responsabilité partagée, identités éphémères, surfaces d'attaque distribuées et configurations complexes multiplient les vecteurs de compromission. Les équipes sécurité doivent adapter leurs compétences et leurs outils à ces nouveaux environnements tout en maintenant une visibilité complète sur les ressources déployées. Ce guide technique détaille les approches éprouvées en production, les pièges courants à éviter et les stratégies de durcissement prioritaires pour sécuriser efficacement vos workloads cloud en 2026. Chaque recommandation est issue de retours d'expérience concrets en environnement entreprise.

Retour d'expérience : lors de l'audit d'une plateforme e-commerce bâtie sur AWS Lambda et API Gateway, nous avons identifié que 34 fonctions Lambda partageaient un seul rôle IAM avec la politique `AdministratorAccess`. L'exploitation d'une injection dans une fonction de traitement de commandes a permis d'accéder à toutes les tables DynamoDB, les buckets S3 et les secrets Secrets Manager du compte. La séparation des rôles avec moindre privilège par fonction a éliminé 95 % des chemins d'escalade tout en ne nécessitant que deux jours de travail.

Surface d'attaque des architectures serverless

La surface d'attaque d'une architecture serverless diffère fondamentalement de celle des applications traditionnelles. Les fonctions serverless sont déclenchées par des **événements** provenant de sources multiples : API Gateway, files SQS, topics SNS, buckets S3, streams DynamoDB, événements CloudWatch, et de nombreuses autres intégrations. Chaque source d'événement constitue un point d'entrée potentiel pour l'injection de données malveillantes. Contrairement aux serveurs classiques où le trafic passe par un unique point d'entrée réseau, les fonctions serverless exposent une surface multi-vectorielle que les *WAF traditionnels* ne couvrent pas complètement.

Le modèle d'exécution éphémère des fonctions serverless crée des défis spécifiques pour la sécurité. Les fonctions sont instanciées à la demande, exécutent leur logique et sont potentiellement recyclées. Cette éphéméralité complique le monitoring traditionnel basé sur les agents résidents mais offre un avantage de sécurité : un attaquant qui compromet une fonction perd son accès au recyclage de l'instance. Cependant, les **warm starts** réutilisent des instances existantes, créant un risque de persistance de données sensibles en mémoire entre les invocations. Le partage du runtime entre invocations peut permettre la fuite de données si les variables globales ne sont pas gérées correctement. Consultez Google Cloud Security pour les recommandations officielles d'AWS sur la sécurité Lambda. Notre article sur Container Security Docker Runtime Protection détaille les aspects complémentaires de la sécurité cloud AWS.

Gestion IAM serverless et moindre privilège

La gestion des permissions IAM est le pilier le plus critique de la sécurité serverless. Chaque fonction doit disposer de son propre **rôle d'exécution IAM** avec des permissions strictement limitées aux ressources et actions nécessaires. L'anti-pattern le plus fréquent est le partage d'un rôle unique entre toutes les fonctions d'un projet, créant un rayon d'explosion maximal en cas de compromission. Les permissions doivent être spécifiées au niveau de la ressource individuelle : plutôt que d'autoriser `s3:GetObject` sur tous les buckets, restreignez à l'ARN du bucket spécifique et si possible au préfixe utilisé par la fonction.

Les frameworks d'Infrastructure as Code comme **AWS SAM**, **Serverless Framework** et **Terraform** facilitent la définition de permissions granulaires via des templates. Les *permissions boundaries* définissent les limites maximales de permissions qu'un rôle Lambda peut obtenir, empêchant l'escalade même si un développeur configure un rôle trop permissif. **IAM Access Analyzer** analyse les invocations réelles des fonctions pour recommander des politiques resserrées basées sur l'usage effectif. L'utilisation de **tags de ressource** dans les conditions IAM permet de créer des politiques dynamiques qui s'adaptent à l'environnement (dev, staging, prod) sans duplication. Notre guide sur Oauth Oidc Abus Consent Securite approfondit les stratégies de gestion des identités et des accès cloud. Les benchmarks du Azure Defender for Cloud fournissent des référentiels détaillés pour l'évaluation des configurations IAM.

Validation des entrées et protection contre l'injection

Les fonctions serverless reçoivent des événements dont le format et le contenu varient selon la source. L'**injection d'événements** est l'attaque la plus répandue : un attaquant forge des données malveillantes dans l'événement déclencheur pour exploiter une vulnérabilité dans le code de la fonction. Les vecteurs d'injection incluent les payloads API Gateway (SQLi, NoSQLi, XSS, XXE), les noms d'objets S3 (path traversal, command injection), les messages SQS/SNS (désérialisation dangereuse) et les données de formulaire (SSRF). La validation des entrées doit être appliquée à **chaque source d'événement** avec des schémas de validation stricts.

Les **API Gateway validators** filtrent les requêtes malformées avant qu'elles n'atteignent la fonction, réduisant la surface d'attaque et les coûts d'exécution. Les *schémas JSON* définissent les types, formats et contraintes attendus pour chaque endpoint. Au niveau du code, les bibliothèques de validation comme `joi` (Node.js), `pydantic` (Python) et `bean-validation` (Java) structurent la validation avec des messages d'erreur informatifs. L'**encodage de sortie** contextuel prévient les attaques XSS lorsque les fonctions génèrent du contenu HTML ou JSON. Le logging structuré de chaque invocation avec les métadonnées de l'événement (sans les données sensibles) facilite la détection et l'investigation des tentatives d'injection. Consultez AWS Security pour les bonnes pratiques de sécurité applicative sur Google Cloud Functions. Notre article sur Cloud Disaster Recovery Pra Resilience explore les aspects réseau complémentaires de la protection cloud.

Source d'événement	Vecteur d'attaque	Protection recommandée	Priorité
API Gateway	SQLi, XSS, SSRF	Validation schéma, WAF, rate limiting	Critique
S3 Events	Path traversal, injection nom fichier	Validation nom objet, sandbox traitement	Haute
SQS/SNS	Désérialisation, injection payload	Validation schéma message, DLQ	Haute
DynamoDB Streams	Manipulation données, DoS	Validation structure, rate limiting	Moyenne
CloudWatch Events	Manipulation règle, timing	Restriction IAM création règles	Moyenne
Cognito Triggers	Injection attributs utilisateur	Validation stricte claims, sanitization	Haute

Sécurité des dépendances et supply chain serverless

Les fonctions serverless reposent sur des écosystèmes de packages riches (npm, pip, Maven) qui introduisent des risques de **supply chain** significatifs. Les attaques de typosquatting, les compromissions de mainteneurs et l'injection de code malveillant dans les packages populaires constituent des menaces croissantes. Le scan des dépendances avec **Snyk**, **Dependabot** ou **npm audit** dans le pipeline CI/CD identifie les vulnérabilités connues. Le **lockfile** (`package-lock.json`, `requirements.txt` avec hashes, `go.sum`) garantit la reproductibilité des builds et protège contre la substitution de packages.

Les **Lambda Layers** (AWS) et les équivalents sur Azure et GCP permettent de centraliser les dépendances partagées, facilitant leur mise à jour et leur scan. La *génération de SBOM* documente l'inventaire complet des composants tiers, facilitant la réponse aux nouvelles vulnérabilités par l'identification rapide des fonctions affectées. L'utilisation de **runtimes personnalisés** compilés (Go, Rust) réduit la surface d'attaque par rapport aux runtimes interprétés qui incluent l'écosystème complet du langage. Le *vendoring* des dépendances (inclusion dans le repository plutôt que téléchargement au build) élimine la dépendance aux registres de packages pendant le build mais complexifie la maintenance. Notre guide sur Top 10 Outils Securite Kubernetes 2025 détaille les stratégies de sécurité de la supply chain logicielle. Les recommandations de Google Cloud Security couvrent les aspects spécifiques de la sécurité Lambda.

Mon avis : le serverless offre un avantage de sécurité fondamental souvent sous-estimé : l'absence de serveurs à patcher et l'éphéméralité des instances réduisent considérablement la fenêtre d'exploitation post-compromission. Cependant, cette force est annulée si les permissions IAM sont trop larges, car une seule fonction compromise avec AdministratorAccess donne un accès total au compte. Le moindre privilège par fonction est le investissement le plus rentable en sécurité serverless.

Comment sécuriser des fonctions AWS Lambda en production ?

La sécurisation de fonctions Lambda en production suit une approche structurée couvrant les permissions, le code, la configuration et le monitoring. **Permissions** : créez un rôle IAM dédié par fonction avec les permissions minimales nécessaires, utilisez les conditions de ressource pour restreindre l'accès aux ARN spécifiques et activez IAM Access Analyzer pour identifier les permissions inutilisées. **Code** : validez toutes les entrées avec des schémas stricts, utilisez des paramètres SSM ou Secrets Manager plutôt que des variables d'environnement pour les secrets critiques, implémentez un logging structuré sans données sensibles et gérez les erreurs sans exposer les détails internes. **Configuration** : limitez le timeout au minimum nécessaire pour réduire le risque de déni de service économique, configurez la mémoire au juste nécessaire, activez le VPC access uniquement si la fonction doit accéder à des ressources privées et configurez les reserved concurrency pour prévenir les invocations excessives. **Monitoring** : activez AWS X-Ray pour le tracing distribué, configurez des alarmes CloudWatch sur les métriques d'erreur et de durée, et intégrez les logs avec votre SIEM pour la corrélation avec les événements de sécurité. Notre article sur Secrets Sprawl Collecte Guide fournit un guide complémentaire sur la sécurité des architectures cloud-native.

Pourquoi le serverless ne signifie-t-il pas sécurité automatique ?

Le terme serverless crée une perception trompeuse selon laquelle l'absence de serveurs visibles signifie l'absence de risques de sécurité. La réalité est que le modèle de responsabilité partagée se déplace vers le haut de la pile sans diminuer la responsabilité totale du client. Le cloud provider gère le système d'exploitation, le runtime, les correctifs de sécurité de l'infrastructure et la disponibilité. Le client reste entièrement responsable du **code applicatif** et de ses vulnérabilités, des **permissions IAM** qui déterminent le rayon d'action de chaque fonction, de la

configuration des déclencheurs qui définissent les points d'entrée, des **dépendances tierces** et de leurs vulnérabilités, et de la **gestion des données** incluant le chiffrement et la conformité. L'absence de serveurs à patcher est un avantage réel mais insuffisant si le code est vulnérable aux injections, les permissions sont trop larges et les dépendances ne sont pas mises à jour. La surface d'attaque a changé de nature, pas diminué en importance.

Quelles sont les menaces spécifiques au serverless en 2026 ?

Le paysage des menaces serverless en 2026 comprend des attaques spécifiques à ce paradigme. L'**injection d'événements** exploite la multiplicité des sources de déclenchement pour injecter des données malveillantes via des canaux non protégés par les WAF traditionnels. Les **rôles IAM surpermissifs** amplифient l'impact de toute compromission en donnant accès à des ressources bien au-delà du besoin de la fonction compromise. Le **déni de service économique** déclenche des millions d'invocations pour générer des factures astronomiques, exploitant le modèle pay-per-use. Le **détournement pour cryptominage** exploite la scalabilité automatique pour exécuter du code de minage sur des milliers d'instances simultanées. Les *attaques par réutilisation de contexte* exploitent les warm starts pour accéder aux données en mémoire de l'invocation précédente. L'**exfiltration via les variables d'environnement** cible les secrets stockés en clair dans la configuration de la fonction. Les consultants de Azure Defender for Cloud ont documenté plusieurs de ces scénarios d'attaque dans leurs benchmarks de sécurité cloud. La surveillance proactive avec les outils de détection adaptés au serverless est essentielle pour contrer ces menaces.

À retenir : la sécurité serverless repose sur quatre piliers : le moindre privilège IAM par fonction, la validation exhaustive des entrées provenant de toutes les sources d'événements, la sécurisation des dépendances dans le pipeline CI/CD, et le monitoring adapté au modèle éphémère et event-driven. Le serverless n'élimine pas les responsabilités de sécurité, il les recentre sur le code, les permissions et les données.

Vos fonctions Lambda disposent-elles chacune de leur propre rôle IAM avec moindre privilège, ou partagent-elles un rôle unique trop permissif ?

Sources et références : CISA · Cloud Security Alliance

Perspectives et prochaines étapes

Le serverless continue d'évoluer vers des modèles toujours plus abstraits avec les step functions, les workflows visuels et les intégrations directes entre services sans code intermédiaire. Cette évolution réduit la surface d'attaque liée au code mais augmente la complexité des configurations et des permissions à gérer. Les outils de sécurité serverless gagnent en maturité avec des solutions spécialisées qui comprennent nativement le modèle event-driven et les spécificités de chaque cloud provider. L'intégration de la sécurité dans les frameworks de développement serverless permet un shift-left efficace où les politiques de sécurité sont définies en même temps que le code applicatif.

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr