Optimisation des coûts sur le Cloud avec AWS Lambda et Boto3

1 Pourquoi optimiser les coûts sur le Cloud ?

Lorsque nous utilisons des services cloud comme AWS, certaines ressources peuvent devenir **obsolètes** (stale) au fil du temps, ce qui peut :

☑ **Engendrer des coûts supplémentaires** (ex: stockage inutile de snapshots EBS).
☑ **Diminuer la gestion efficace des ressources**.
☑ **Créer des risques de sécurité** si les anciennes ressources ne sont pas bien gérées.

💡 **Problème :** Les instantanés EBS (snapshots) permettent de sauvegarder l'état des volumes EC2, mais **s'ils ne sont plus utilisés, ils continuent d'occuper de l'espace de stockage et génèrent des coûts**.

---

2 Solution : Automatiser la gestion des ressources avec AWS Lambda et Boto3

Pour éviter ces coûts inutiles, nous allons utiliser **AWS Lambda** pour automatiser la suppression des snapshots EBS non utilisés.

☑ Pourquoi AWS Lambda ?

- Service **serverless** : pas besoin de gérer des serveurs.
- **Exécution automatique** en réponse à des événements.
- **Optimisation des coûts** : on ne paie que pour le temps d'exécution.

☑ Pourquoi Boto3 ?

- C'est le **SDK officiel d'AWS pour Python**.
- Permet d'interagir avec les services AWS via **API**.
- Facile à utiliser et bien documenté.

---

3 Fonctionnement de la solution

1. **Créer une fonction Lambda en Python**.
2. **Accorder les permissions nécessaires** à Lambda via IAM.
3. **Lambda utilise Boto3 pour** :
    - Lister tous les snapshots EBS du compte AWS.
    - Vérifier s'ils sont attachés à un volume EC2 actif.
    - Supprimer ceux qui ne sont plus utilisés.
4. **Déclencher la fonction** automatiquement via **EventBridge** (ex: une exécution quotidienne).

4️⃣ Pourquoi utiliser Python pour cette solution ?

💡 **Python est le langage recommandé pour AWS Lambda avec Boto3** car :
☑ Il est **nativement supporté par AWS Lambda**.
☑ Il est **simple et efficace** pour traiter des tâches automatisées.
☑ Il possède **Boto3**, un SDK puissant et optimisé pour AWS.
☑ Il **démarre plus vite** que d'autres langages comme Java ou C#.

📌 **En résumé :** Python + AWS Lambda est **le choix idéal** pour automatiser des tâches cloud comme l'optimisation des coûts sur AWS. 🚀

Différence entre "Add a Trigger" et "Add a Destination" dans AWS Lambda

Lorsque tu configures une fonction **AWS Lambda**, tu as deux options :

- **"Add a Trigger"** → Déclenche l'exécution de la fonction.
- **"Add a Destination"** → Définit où envoyer les résultats après l'exécution de la fonction.

1️⃣ "Add a Trigger" (Ajouter un déclencheur)

📌 **Définition** : Un **trigger** est un événement qui déclenche l'exécution de la fonction Lambda.
📌 **Exemples courants de déclencheurs** :

- **API Gateway** → Pour exposer Lambda via une API REST.
- **S3** → Pour exécuter Lambda lorsqu'un fichier est ajouté à un bucket.
- **DynamoDB Streams** → Pour traiter les changements dans une base de données.
- **CloudWatch Events** → Pour exécuter Lambda sur un planning régulier (ex: toutes les 5 minutes).
- **SNS / SQS** → Pour traiter des messages en file d'attente.

📌 **Quand l'utiliser ?**
☑ Quand tu veux que Lambda **réagisse automatiquement** à un événement spécifique.
☑ Quand tu veux **déclencher l'exécution de la fonction** à partir d'un service AWS.

◇ **Exemple d'utilisation :**
Si tu ajoutes **un trigger S3**, la fonction Lambda sera appelée **chaque fois qu'un fichier est téléchargé dans un bucket S3**.

2️⃣ "Add a Destination" (Ajouter une destination)

📌 **Définition** : Une **destination** est un service où AWS Lambda **envoie le résultat** de l'exécution après qu'elle soit terminée.
📌 **Deux types de destinations :**

- **Succès (on success)** → Si la fonction s'exécute **sans erreur**.
- **Échec (on failure)** → Si la fonction rencontre **une erreur**.

📌 **Destinations possibles :**

- **SQS** → Envoyer le résultat dans une file d'attente.
- **SNS** → Notifier un autre système.
- **Lambda** → Appeler une autre fonction Lambda après l'exécution.
- **EventBridge** → Enregistrer l'événement pour analyse.

📌 **Quand l'utiliser ?**
☑️ Quand tu veux **envoyer les résultats** de ta fonction vers un autre service AWS.
☑️ Quand tu veux **gérer les erreurs automatiquement** (ex: envoyer les erreurs vers une file SQS).

🔹 **Exemple d'utilisation :**
Si Lambda **traite des données** et réussit, il peut envoyer le résultat à une **autre fonction Lambda** via **une destination**. S'il échoue, il peut envoyer un message d'erreur à **SNS ou SQS** pour que l'équipe IT soit informée.

📌 Différence principale entre Trigger et Destination

Fonctionnalité	Trigger (Déclencheur)	Destination
📌 **Rôle**	Démarre l'exécution de Lambda	Envoie le résultat après l'exécution
🔥 **Quand ça s'applique ?**	Avant l'exécution	Après l'exécution
🎯 **Objectif**	Lancer Lambda en réaction à un événement	Rediriger le résultat vers un autre service
🛠️ **Exemples**	API Gateway, S3, DynamoDB, CloudWatch, SNS, SQS	SQS, SNS, Lambda, EventBridge
🔘 **Gère les erreurs ?**	❌ Non	☑️ Oui (on failure)

🔹 Exemple pratique

- **Trigger** : Une fonction Lambda est déclenchée par **un fichier S3 ajouté**.

- **Destination** : Si Lambda réussit, il envoie les résultats à **une autre Lambda** ; s'il échoue, il envoie un message d'erreur à **SQS**.

💡 **Conclusion**
👉 **"Add a Trigger"** = Déclenche l'exécution de Lambda.
👉 **"Add a Destination"** = Envoie le résultat après exécution (succès ou échec).

🚀 **Meilleure pratique** : Toujours ajouter une **destination d'échec** (`on failure`) pour suivre les erreurs et éviter la perte de données.

SNS et SQS : Définition et Différences

AWS propose deux services de messagerie couramment utilisés pour la communication entre applications :

- **SNS (Simple Notification Service)** → Service de **messagerie Pub/Sub (publish-subscribe)**
- **SQS (Simple Queue Service)** → Service de **file d'attente de messages (message queueing)**

---

1 SNS (Simple Notification Service) - Notification Pub/Sub 📢

📌 **Définition** :
SNS est un **service de messagerie basé sur la publication/souscription** (**Pub/Sub**). Il permet d'envoyer **un message à plusieurs abonnés** (applications, emails, SMS, Lambda, SQS, etc.).

📌 **Fonctionnement** :

1. Une **application ou un service publie un message** dans un **topic SNS**.
2. Tous les **abonnés au topic** reçoivent le message en temps réel.

📌 **Cas d'utilisation** :
☑ Envoi d'**alertes et notifications** (ex: **CloudWatch Alarm** envoie une alerte à SNS pour notifier un administrateur).
☑ Communication **événementielle** entre microservices.
☑ Diffusion de **messages à plusieurs destinations** (email, SMS, Lambda, SQS, HTTP, etc.).

📌 **Exemple** :

- Une **application e-commerce** envoie une **notification de confirmation de commande** via SNS.
- SNS transmet ce message à plusieurs abonnés :

- **Un email** au client ☑
- **Un SMS** au livreur ☑
- **Une autre Lambda** pour mise à jour en base de données ☑

---

2 SQS (Simple Queue Service) - File d’attente de messages 📥

📌 **Définition** :
SQS est un **service de file d’attente de messages** où les messages sont stockés **jusqu'à ce qu'ils soient traités** par un consommateur (serveur, Lambda, EC2, etc.).

📌 **Fonctionnement** :

1. Une application envoie un **message dans une file SQS**.
2. Un **consommateur** récupère et traite les messages **un par un**.
3. Une fois traité, le message est **supprimé de la file**.

📌 **Cas d'utilisation** :
☑ Gestion des **tâches en arrière-plan** (ex: une **file SQS stocke des tâches pour qu'une Lambda les exécute** en différé).
☑ **Désynchronisation des services** (ex: un service envoie un message dans SQS, et un autre le récupère plus tard).
☑ **Gestion des pics de charge** (ex: une application Web envoie des requêtes dans SQS pour être traitées progressivement).

📌 **Exemple** :

- Une **application de traitement d'images** reçoit 1000 images à convertir en PDF.
- Elle place ces tâches dans **une file SQS**.
- Une **Lambda ou une instance EC2** récupère les tâches et les exécute **une par une**, évitant ainsi une surcharge du serveur.

📌 Différence entre SNS et SQS

Critère	SNS (Pub/Sub - Notifications)	SQS (File d'attente - Message Queue)
Type de communication	**Diffusion (fan-out)**	**File d’attente (FIFO ou Standard)**
Transmission des messages	**Instantanée** (envoi à plusieurs abonnés en même temps)	**Asynchrone** (stockage jusqu'à consommation)

Critère	SNS (Pub/Sub - Notifications)	SQS (File d'attente - Message Queue)
Destinataires possibles	Plusieurs abonnés (Email, SMS, Lambda, SQS, HTTP)	Un ou plusieurs consommateurs (EC2, Lambda, etc.)
Cas d'usage principal	Notifications, alertes, communication événementielle	Traitement différé, gestion des tâches en file d'attente
Conservation des messages	Non	Oui (jusqu'à 14 jours)
Répétition des messages	Un message est envoyé **une seule fois**	Un message peut être lu **plusieurs fois** si pas supprimé
Ordre des messages	Non garanti	**FIFO disponible** pour garantir l'ordre

---

◇ Quand utiliser SNS ou SQS ?

☑ **Utiliser SNS si :**

- Tu veux **envoyer un message à plusieurs destinataires** en même temps.
- Tu veux **notifier** un système ou des utilisateurs en **temps réel** (ex: confirmation de paiement, alerte système).

☑ **Utiliser SQS si :**

- Tu veux **stocker et traiter des messages de manière asynchrone**.
- Tu as un **processus qui doit récupérer les messages progressivement** sans surcharge.
- Tu veux assurer **la persistance des messages** pour les traiter plus tard.

◇ Peut-on utiliser SNS et SQS ensemble ?

Oui ! **Meilleure pratique AWS → SNS + SQS** pour une architecture robuste 🚀

- **SNS envoie un message à plusieurs files SQS**
- Chaque service peut récupérer et traiter les messages à son rythme

◇ **Exemple** :

- Une application SNS **publie un événement de nouvelle commande**.
- Deux services abonnés reçoivent le message via **deux files SQS distinctes** :
  - Une **file SQS** pour la **facturation** ☑
  - Une **file SQS** pour la **gestion des stocks** ☑

💡 **Conclusion** :

👉 **SNS = Notification instantanée vers plusieurs abonnés**

👉 **SQS = File d’attente pour traiter des messages de manière asynchrone**