Déploiement d'une application web sur EC2 avec Auto Scaling et Load Balancer

Guide Complet de Déploiement - 9 Jobs Détaillés

La Plateforme - École du numérique pour tous

Table des Matières

1 - Introduction au Projet

Ce projet a pour objectif de créer une infrastructure cloud complète en utilisant les services AWS. Vous apprendrez à déployer des applications, gérer des bases de données, analyser des données et automatiser des tâches.
Le projet est divisé en 9 jobs progressifs, du déploiement basique d'une application web jusqu'à l'automatisation complexe des workflows.

Prérequis

- Compte AWS avec accès au tier gratuit
- Docker installé localement
- Connaissances de base en Linux/Unix
- Compréhension basique des concepts cloud

Ressources Gratuites

- Tier gratuit AWS : https://aws.amazon.com/free
- LocalStack pour tester localement : https://localstack.cloud/
- 14 jours gratuits avec LocalStack

2 - Guide Complet de Connexion SSH

Qu'est-ce que SSH ?

SSH (Secure Shell) est un protocole de réseau cryptographique pour accéder à distance à vos serveurs EC2. Il vous permet de contrôler complètement votre instance via une interface ligne de commande.

Étape 1 : Créer une paire de clés

Lors de la création d'une instance EC2 :

1. Naviguez jusqu'à 'Paires de clés' dans le wizard de lancement
2. Cliquez sur 'Créer une nouvelle paire de clés'
3. Donnez un nom : ma-clé-ec2
4. Choisissez le format : RSA
5. Cliquez sur 'Créer une paire de clés'
6. Un fichier .pem sera téléchargé automatiquement - CONSERVEZ-LE EN LIEU SÛR !

Étape 2 : Configurer les permissions du fichier de clé

Sur Linux/Mac, ouvrez un terminal :

```
chmod 400 ma-clé-ec2.pem
```

Étape 3 : Obtenir l'adresse IP publique

7. Allez à EC2 > Instances
8. Sélectionnez votre instance
9. Notez l'adresse 'IP publique' (ex: 54.123.45.67)

Étape 4 : Se connecter en SSH

Sur Linux/Mac :

```
ssh -i /chemin/vers/ma-clé-ec2.pem ubuntu@54.123.45.67
```

Remplacez :

- /chemin/vers/ma-clé-ec2.pem par le chemin réel du fichier
- 54.123.45.67 par l'adresse IP de votre instance
- ubuntu par ec2-user si vous utilisez Amazon Linux

Sur Windows (avec PuTTY) :

10. Téléchargez PuTTY et PuTTYgen
11. Convertissez le .pem en format .ppk avec PuTTYgen
12. Ouvrez PuTTY et entrez l'IP de votre instance
13. Allez à Connection > SSH > Auth
14. Sélectionnez votre fichier .ppk et connectez-vous

Windows avec PowerShell :

```
ssh -i "C:\cheminkey\nomkey.pem"
admin@ec2-35-181-50-140.eu-west-3.compute.amazonaws.com

icacls "C:\aws-keys\aws_arch.pem" /inheritance:r

icacls "C:\aws-keys\aws_arch.pem" /grant:r "$($env:USERNAME):(F)"
```

Étape 5 : Accepter la clé du serveur

À la première connexion, SSH vous demandera d'accepter la clé d'hôte. Tapez 'yes' et appuyez sur Entrée.

Étape 6 : Transférer des fichiers avec SCP

Pour uploader un fichier vers votre instance :

```
scp -i ma-clé-ec2.pem mon-fichier.txt ubuntu@54.123.45.67:/home/ubuntu/
```

Pour télécharger un fichier de votre instance :

```
scp -i ma-clé-ec2.pem ubuntu@54.123.45.67:/home/ubuntu/fichier.txt .
```

Commandes Utiles une fois connecté

- Mettre à jour les paquets : sudo apt update && sudo apt upgrade -y
- Voir l'espace disque : df -h
- Voir l'utilisation CPU/mémoire : free -h
- Voir les processus en cours : top
- Installer un paquet : sudo apt install nom-du-paquet
- Voir les logs : sudo tail -f /var/log/syslog

Dépannage

Erreur : 'Permission denied'

Solution : Vérifiez que les permissions du fichier .pem sont correctes avec : chmod 400 ma-clé-ec2.pem

Erreur : 'Connection timed out'

Solution : Vérifiez que votre groupe de sécurité autorise le port 22 (SSH) et que l'instance est en cours d'exécution

Erreur : 'Host key verification failed'

Solution : Acceptez la clé d'hôte en tapant 'yes' à la première connexion

3 - Job 1 : Auto Scaling + Load Balancer

Guide complet - Application web scalable avec HTTPS

Objectif

Déployer une application web sur un serveur EC2 Debian avec une mise à l'échelle automatique pour gérer la charge. Mettre en place un Load Balancer pour distribuer le trafic entre les instances et ajouter HTTPS pour sécuriser les connexions.

Ressources AWS Utilisées

15. EC2 (Elastic Compute Cloud)
16. Launch Template
17. Auto Scaling Group
18. Application Load Balancer (ALB)
19. Security Groups
20. SNS (Simple Notification Service) - optionnel
21. AWS Certificate Manager - optionnel

Étapes Détaillées

Étape 1 : Créer une instance EC2 Debian

- Accédez à la console AWS et naviguez vers EC2
- Cliquez sur 'Lancer une instance'
- Sélectionnez une AMI : Debian 12 (version stable actuelle)
- Choisissez le type d'instance : t2.micro (gratuit)
- Configurez les détails réseau (VPC, subnet)
- Ajoutez du stockage : 20 GB d'EBS suffisent
- Configurez les groupes de sécurité : autorisez ports 80 (HTTP), 443 (HTTPS) et 22 (SSH)
- Créez une nouvelle paire de clés (aws_arch) ou utilisez une existante

Étape 2 : Se connecter via SSH (Windows PowerShell)

Utilisateur Debian : admin (pas ubuntu)
Ouvrez PowerShell en ADMIN et exécutez :

```
icacls "C:\aws-keys\aws_arch.pem" /inheritance:r
icacls "C:\aws-keys\aws_arch.pem" /grant:r "$($env:USERNAME):(F)"
ssh -i "C:\aws-keys\aws_arch.pem" admin@<IP_DE_VOTRE_INSTANCE>
```

Étape 3 : Installer Nginx, PHP et créer la page d'accueil

⭐ **IMPORTANT : Cette étape installe l'application sur la première instance**
Une fois connecté à l'instance EC2 Debian via SSH, exécutez :

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y nginx php-fpm php-common
sudo systemctl start nginx && sudo systemctl enable nginx
```

Créez la page d'accueil avec les métadonnées AWS (voir script complet sur GitHub).
Vérifiez : curl localhost (doit retourner la page HTML)

Étape 4 : Créer un Launch Template

Le modèle de lancement définit comment les instances seront créées automatiquement :

- Accédez à EC2 > Launch Templates
- Créez un nouveau modèle
- Name : 'debian-nginx-template'
- AMI : Debian 12
- Instance type : t2.micro
- Key pair : aws_arch
- Security groups : celle créée à l'étape 1
- Advanced > User data : COLLEZ LE SCRIPT COMPLET (voir GitHub)

⚠️ **IMPORTANT : Vérifiez que la case 'User data has already been base64 encoded' est DÉCOCHÉE** ☐

Étape 5 : Configurer l'Auto Scaling Group

L'Auto Scaling Group gère le nombre d'instances automatiquement :

- Allez à EC2 > Auto Scaling Groups
- Créez un nouveau groupe Auto Scaling
- Name : 'debian-asg'
- Sélectionnez le Launch Template : 'debian-nginx-template'
- Availability Zones : Cochez eu-west-3a, eu-west-3b, eu-west-3c
- Min capacity : 1 | Desired capacity : 2 | Max capacity : 4

Étape 6 : Créer et configurer un Application Load Balancer

Le Load Balancer distribue le trafic entre les instances :

- Allez à EC2 > Load Balancers
- Créez un Application Load Balancer
- Name : 'debian-alb'
- Scheme : 'Internet-facing'
- Zones : eu-west-3a, eu-west-3b, eu-west-3c
- Listeners : HTTP (port 80)

Créer Target Group :

22. Type : 'Instances'
23. Name : 'debian-targets'
24. Protocol : HTTP, Port : 80
25. Health check path : '/index.php'
26. Healthy threshold : 2

Étape 7 : Ajouter HTTPS et certificat auto-signé (optionnel)

⭐ Étape 7A : Générer un certificat auto-signé

Connectez-vous en SSH à la première instance et exécutez :

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \-keyout
/etc/ssl/private/nginx-selfsigned.key \-out /etc/ssl/certs/nginx-selfsigned.crt
```

Étape 8 : Ajouter les notifications SNS (optionnel)

- AWS Console > SNS > Topics > 'Create topic'
- Name : 'asg-notifications'
- Créez la subscription (Email) et confirmez via email
- EC2 > Auto Scaling Groups > debian-asg > Notifications
- Events : Cochez 'Launch' et 'Terminate'

Étape 9 : Tester l'infrastructure

- Attendez 2-3 minutes le temps que le Load Balancer soit actif
- EC2 > Load Balancers > 'debian-alb'
- Copiez le 'DNS Name' du load balancer
- Test HTTP : http://<DNS_NAME> → Redirection vers HTTPS
- Rafraîchissez F5 plusieurs fois
- L'Instance ID change = le Load Balancer distribue bien le trafic ✅

Sécurité

27. Configurez les groupes de sécurité pour autoriser uniquement le trafic nécessaire
28. Limitez l'accès SSH (port 22) à vos adresses IP (pas 0.0.0.0/0)
29. Utilisez les certificats SSL pour HTTPS

Coûts

30. Le tier gratuit d'EC2 vous offre 750 heures par mois sur t2.micro
31. Debian : gratuit (pas de frais de licence comme RHEL)
32. 2 instances t2.micro = 1500 heures/mois (dépasse le tier gratuit, environ 15€/mois)
33. Load Balancer : environ 16€/mois
34. SNS : gratuit (1 000 notifications/mois)

4 - Job 2 : Héberger un site web statique avec CloudFront

Objectif

Héberger un site web statique (HTML, CSS, JavaScript) en utilisant S3 et distribuer le contenu globalement avec CloudFront pour une meilleure performance et sécurité.

Ressources AWS Utilisées

35. S3 (Simple Storage Service) - Stockage fichiers
36. CloudFront (Content Delivery Network) - Distribution globale
37. IAM (gestion des permissions)

Pourquoi ?

- S3 = Stockage des fichiers de site (HTML, CSS, JS)
- CloudFront = Vitesse (200+ serveurs mondiaux, latence réduite)
- Ensemble = Site rapide + sécurisé + scalable

Étapes Détaillées

Étape 1 : Créer un Bucket S3

- Allez à S3 > Buckets
- Cliquez sur 'Create bucket'
- Bucket name : mon-site-2024 (doit être unique)
- Region : eu-west-3
- Object Ownership : ACLs disabled (Bucket owner enforced)
- Block Public Access : DÉCOCHEZ ☐ (pour permettre accès public)
- Cliquez 'Create bucket' ✓

Étape 2 : Uploader vos fichiers

Fichiers à préparer :

38. index.html (OBLIGATOIRE - page d'accueil)
39. error.html (optionnel - page erreur)
40. styles.css (optionnel - feuille de style)
41. app.js (optionnel - JavaScript)

Étape 3 : Configurer les Permissions

⚠️ Vous avez 'bucket owner enforced' → Utiliser Bucket Policy

Bucket > Permissions > Bucket policy > Edit

Collez cette policy (remplacez 'mon-site-2024' par votre nom) :

```
{  "Version": "2012-10-17",  "Statement": [{    "Effect": "Allow",    "Principal": "*",    "Action": "s3:GetObject",    "Resource": "arn:aws:s3:::mon-site-2024/*"  }]}
```

Étape 4 : Activer Hébergement Statique

- Bucket > Properties > Static website hosting > Edit
- Enable ☑ Hébergement de site web statique
- Index document : index.html
- Error document : error.html

- Cliquez 'Save changes' ✓

Étape 5 : Créer une Distribution CloudFront

⚠️ CRITIQUEMENT IMPORTANT

Origin domain DOIT être : mon-site-2024.s3.eu-west-3.amazonaws.com (PAS le s3-website-...)

- CloudFront > Create distribution
- Origin domain : sélectionnez votre bucket S3
- Default root object : index.html
- Viewer protocol policy : Redirect HTTP to HTTPS
- Cliquez 'Create distribution' ✓

Status : 'InProgress' - attendez 5-15 minutes (normal !)

Étape 6 : Tester et Vider le Cache

Quand CloudFront = 'Enabled' (vert) :

42. CloudFront > Invalidations > Create invalidation
43. Path : /* (vider le cache)
44. Attendez 'Completed' (2-5 min)

Étape 7 : Accès Final

- CloudFront > Distributions > Votre distribution
- Copiez le 'Domain name' (ex: d123abc.cloudfront.net)
- Ouvrez : https://d123abc.cloudfront.net ✅ Vous voyez votre site !

Coûts

45. Stockage S3 : 5 GB gratuit (tier gratuit)
46. CloudFront : 1000 minutes gratuites par mois (tier gratuit)
47. Au-delà : S3 = 0.023€/GB/mois, CloudFront = 0.085€/GB
48. CONCLUSION : Entièrement gratuit pour ce projet ✅

Troubleshooting

❌ 403 Forbidden sur S3

49. Vérifiez Bucket Policy (étape 3) est correcte
50. Vérifiez nom du bucket dans Resource
51. Vérifiez Block Public Access = OFF

❌ AccessDenied CloudFront

52. Attendez 5-15 min (distribution en cours de création)
53. Créez Invalidation (étape 6) pour forcer rafraîchissement
54. Videz cache navigateur en mode Incognito

Points Importants à Retenir

1. Bucket Policy

Avec 'ACLs disabled', il faut IMPÉRATIVEMENT utiliser Bucket Policy. C'est le seul moyen de donner accès public.

2. CloudFront Origin Domain

DOIT être s3.region.amazonaws.com (REST API), PAS s3-website-region.amazonaws.com.

3. Index.html

Doit être défini comme 'Default root object' dans CloudFront.

4. Invalidations

Quand vous modifiez un fichier, créez une invalidation (/*) pour forcer CloudFront à rafraîchir le cache.

5 - Job 3 : Déployer une base de données relationnelle (RDS)

Objectif

Déployer une base de données relationnelle gérée (MySQL ou PostgreSQL) avec des sauvegardes automatiques et des snapshots pour un stockage sécurisé des données.

Ressources AWS Utilisées

55. Amazon RDS (Relational Database Service)
56. MySQL ou PostgreSQL
57. IAM (gestion des permissions)
58. Security Groups (firewall)

Étapes Détaillées

Étape 1 : Créer une instance RDS

- Allez à RDS > Bases de données
- Cliquez sur 'Créer une base de données'
- Choisissez le moteur : MySQL 8.0 ou PostgreSQL 13+
- Sélectionnez 'Tier gratuit' comme modèle
- Choisissez l'instance : db.t2.micro (gratuit)
- Configurez le stockage : 20 GB suffit
- Configurez Connectivity : VPC = default, Public accessibility = Yes, Create new security group = rds-security-group

Étape 2 : Configurer les paramètres de la base de données

- Identifiant principal : admin
- Mot de passe principal : créez un mot de passe fort (ex: MyPassword123!@#)
- Nom de la base de données initiale : testdb
- Instance class : db.t2.micro (gratuit)
- Allocated storage : 20 GB
- Storage type : gp3

Étape 2.5 : Configurer le Security Group (IMPORTANT !)

⚠️ **CETTE ÉTAPE EST CRITIQUE - SANS ELLE, VOUS NE POUVEZ PAS ACCÉDER À LA BD**

- Une fois la BD créée, allez à EC2 > Security Groups
- Sélectionnez 'rds-security-group'
- Cliquez sur 'Edit inbound rules'
- Ajoutez une règle : Type = MySQL/Aurora, Protocol = TCP, Port = 3306, Source = 0.0.0.0/0
- Cliquez 'Save rules'

Étape 3 : Configurer les sauvegardes automatiques

Note : Les backups automatiques sont GRATUITS (inclus dans le tier gratuit jusqu'à 35 GB)

- Période de rétention des sauvegardes : 7 jours
- Fenêtre de sauvegarde : 03:00 UTC
- Copy tags to automated backups : ☑ Coché
- Auto minor version upgrade : ☑ Coché
- Enable deletion protection : ☑ Coché

Étape 4 : Se connecter à la base de données

Récupérez l'endpoint RDS : RDS > Databases > my-database
Endpoint exemple : my-database.cza6s2m48ld0.eu-west-3.rds.amazonaws.com
OPTION A : Utiliser AWS Query Editor (RECOMMANDÉ)
RDS > Query Editor, Sélectionnez votre BD, Écrivez des requêtes SQL directement
OPTION B : Utiliser MySQL CLI

```
mysql -h ENDPOINT -u admin -p
```

Coûts

59. 750 heures d'utilisation gratuites par mois (db.t2.micro)
60. 20 GB de stockage gratuit par mois
61. 35 GB de backups automatiques gratuit
62. Au-delà : 0.023€ per GB/mois
63. CONCLUSION : Entièrement gratuit pour ce projet ✅

Points Importants à Retenir

1. Security Group

C'est le firewall qui autorise les connexions. ESSENTIEL pour accéder à la BD. Port 3306 (MySQL) ou 5432 (PostgreSQL).

2. Backups Automatiques

Quotidiens, rétention 7 jours (configurable jusqu'à 35 jours), inclus dans le tier gratuit.

3. Deletion Protection

Activé pour prévenir suppression accidentelle. Important en production.

6 - Job 4 : Créer une API RESTful sans serveur (Serverless)

Résumé du projet

Dans ce job, vous apprendrez à créer une API REST complète et fonctionnelle.

64. AWS Lambda: fonction serverless pour la logique métier
65. API Gateway: point d'entrée HTTP
66. CloudWatch Logs: suivi et debugging

Architecture

Flux de requête: CLIENT → API Gateway → Lambda → CloudWatch → Response

Guide détaillé

Étape 1 : Créer la fonction Lambda

- Aller à Lambda Console
- Créer une fonction: Nom=hello-api, Runtime=Python 3.11
- Créer avec permissions de base

Étape 2 : Copier le code

Remplacer le code par le script Python lambda_function.py
Cliquer Deploy (voir sur GitHub dans scripts + code + site > Lambda)

Étape 3 : Créer API Gateway

- Aller à API Gateway Console
- Créer API REST: Nom=hello-api
- Type=API publique, TLS 1.2

Étape 4 : Créer ressource /hello

Clic droit sur /, sélectionner Create Resource, Resource name = hello

Étape 5 : Créer méthode GET

- Cliquer sur /hello, Create method
- Sélectionner GET
- Integration: Lambda function
- Cocher Lambda proxy integration
- Lambda function = hello-api

Étape 6 : Déployer l'API

Cliquer Deploy API, Stage = prod

Étape 7 : Tester l'API

PowerShell :

```
Invoke-WebRequest -Uri 'https://xxx.execute-api.eu-west-3.amazonaws.com/prod/hello'
-Method GET
```

Résultat: StatusCode 200

Étape 8 : Voir les logs

CloudWatch > Log groups > /aws/lambda/hello-api, Voir vos requêtes loggées

Commandes CLI

67. `aws lambda list-functions --region eu-west-3`
68. `aws logs tail /aws/lambda/hello-api --follow`
69. `curl https://xxx.execute-api.eu-west-3.amazonaws.com/prod/hello`

Troubleshooting

70. 502 Bad Gateway: Vérifier CloudWatch Logs
71. 403 Forbidden: Vérifier Lambda proxy integration
72. Logs vides: Tester l'API d'abord
73. Timeout: Augmenter timeout Lambda à 30 secondes

Conclusion

74. Vous avez une fonction Lambda fonctionnelle
75. Vous avez une API REST publique
76. Vous pouvez tracer les logs
77. Le tout gratuit dans le free tier AWS

7 - Job 5 : Surveillance et alertes avec CloudWatch

Objectif

Mettre en place une surveillance et des alertes pour une application AWS. Monitorer les ressources (CPU, mémoire) et recevoir des notifications lorsque des seuils sont dépassés.
À la fin de ce guide, vous aurez :

78. Un Topic SNS configuré pour les alertes
79. Des alarmes CloudWatch sur vos ressources
80. Des notifications par email en temps réel
81. Un dashboard pour visualiser les métriques
82. 0 euros de frais (free tier AWS)

Ressources AWS Utilisées

Ce job utilise deux services AWS :

83. CloudWatch - Service de monitoring centralisé pour collecter logs, métriques et créer alarmes
84. SNS (Simple Notification Service) - Service de notifications pour envoyer des messages (email, SMS, Lambda)

Étapes Détaillées

Étape 1 : Accéder à CloudWatch

Approche 1: Via la Console AWS (Dashboard)

- Allez à https://console.aws.amazon.com
- Cherchez 'CloudWatch' dans la barre de recherche (en haut)
- Cliquez sur 'CloudWatch' dans les résultats
- Dans le menu à gauche, cliquez sur 'Tableaux de bord'
- Cliquez sur le bouton 'Créer un nouveau tableau de bord'

Étape 2 : Ajouter des Métriques

Via la Console AWS

- Sur le tableau de bord, cliquez sur 'Ajouter un widget'
- Sélectionnez le type de widget (ex: Nombre, Graphique en ligne)
- Cliquez sur 'Configurer les métriques'
- Sélectionnez le namespace (ex: AWS/EC2)
- Sélectionnez la métrique (ex: CPUUtilization)
- Cliquez sur 'Créer un widget'

Étape 3 : Créer des Alarmes

Via la Console AWS

- Allez à CloudWatch > Alarmes
- Cliquez sur 'Créer une alarme'
- Cliquez sur 'Parcourir les métriques'
- Sélectionnez une métrique (ex: AWS/EC2 > CPUUtilization)
- Sélectionnez l'instance EC2
- Définissez le seuil (ex: 70 pour 70%)
- Statistic: Average, Period: 5 minutes
- Comparison operator: >= (greater than or equal)

Via la CLI AWS

```
aws cloudwatch put-metric-alarm --alarm-name ec2-cpu-high \--alarm-description 'CPU > 70%'
--metric-name CPUUtilization \--namespace AWS/EC2 --statistic Average --period 300
--threshold 70 \--comparison-operator GreaterThanOrEqualToThreshold \--evaluation-periods 2
--alarm-actions arn:aws:sns:eu-west-3:ACCOUNT_ID:ServerAlerts
```

Étape 4 : Configurer les Notifications SNS

Via la Console AWS

- Allez à SNS > Topics
- Cliquez sur 'Créer un topic'
- Nom du topic: ServerAlerts, Type: Standard
- Cliquez 'Créer un topic'
- Cliquez sur 'Créer une subscription'
- Protocol: Email, Endpoint: votre-email@example.com
- Vérifiez votre email et cliquez sur le lien de confirmation

Coûts

CloudWatch et SNS sont inclus dans le free tier AWS :

85. CloudWatch Alarms: 10 alarmes gratuites par mois
86. SNS Notifications: 1000 emails gratuits par mois
87. CloudWatch Logs: 5GB de logs gratuit par mois
88. CloudWatch Metrics: Gratuites (standard)

IMPORTANT: Tant que vous ne dépassez pas ces limites, vous ne payerez rien.

Checklist Finale

89. Topic SNS 'ServerAlerts' créé
90. Email confirmé dans la subscription SNS
91. Au moins 1 alarme CloudWatch créée
92. L'alarme est associée au Topic SNS
93. Dashboard créé (optionnel)
94. Tester l'alarme: aws cloudwatch set-alarm-state --alarm-name ec2-cpu-high --state-value ALARM

8 - Job 6 : Automatiser les pipelines de données (AWS Glue)

Vue d'ensemble

Job 6 automatise complètement l'ingestion et la transformation de données avec AWS Glue. Le pipeline extrait des données brutes d'un bucket S3, les nettoie, et génère des fichiers Parquet optimisés pour les requêtes analytiques.

Objectif

Créer un pipeline ETL entièrement automatisé qui :

95. Détecte automatiquement le schéma des données
96. Nettoie et transforme les données (suppression des valeurs NULL)
97. Génère des fichiers Parquet optimisés
98. Automatise l'exécution via des triggers

Architecture

Flux de données

S3 (CSV) → Crawler → Glue Catalog → Job → S3 (Parquet)

Composants

1. Amazon S3 (Stockage)

- Input: s3://monsitetomrib/glue-data/input/
- Output: s3://monsitetomrib/glue-data/output/
- Scripts: s3://monsitetomrib/glue-scripts/

2. AWS Glue Crawler

Nom: data-crawler, Fonction: Détecte le schéma des données CSV

3. AWS Glue Job

Nom: customers-transform, Langage: PySpark, Output: Format Parquet

Résultats et Tests

État du Pipeline

- S3 Input (CSV) ✅ OK
- IAM Role ✅ OK
- Crawler (data-crawler) ✅ READY
- Catalog Table (input) ✅ Created
- Job (customers-transform) ✅ Created
- S3 Output (Parquet) ⏳ Generated

Statistiques de transformation

Lignes INPUT: 7, Lignes OUTPUT: 5, Lignes filtrées: 2, Taux de nettoyage: 28.6%

Analyse des coûts

- Glue Crawler: 1M runs gratuit
- Glue Job: ~$0.50/mois
- S3 Storage: ~$0.15/mois
- TOTAL: ~$0.65/mois

Conclusion

- ✅ Pipeline entièrement automatisé et sans serveur
- ✅ Détection automatique du schéma via Crawler
- ✅ Transformation et nettoyage des données
- ✅ Output optimisé en format Parquet
- ✅ Coûts minimes (~$0.65/mois)

9 - Job 7 : Analyser les données avec Athena et QuickSight

Objectif

Effectuer des requêtes SQL sur des données S3 avec Athena et visualiser les résultats avec QuickSight.
Architecture: Données S3 → Athena (SQL) → Résultats → QuickSight (Dashboards)
Durée: 2-3 heures

Ressources

99. Amazon S3 - Stockage des données
100. Amazon Athena - Requêtes SQL sans serveur
101. Amazon QuickSight - Dashboards et visualisations

Étape 1 : Préparer S3 (15 min)

Utilisez bucket existant: s3://monsitetomrib/data/
Fichiers CSV à créer:

- customers.csv: customer_id, name, email, country, signup_date
- orders.csv: order_id, customer_id, amount, order_date, product

Étape 2 : Créer les Tables Athena (20 min)

2.1 Configurer Athena
AWS Console → Athena → Query Editor, Settings → Query result location → s3://monsitetomrib/athena-results/
2.2 Créer la DATABASE

```
CREATE DATABASE IF NOT EXISTS job7_analytics;
```

Étape 3 : Tester les Requêtes SQL (20 min)

Requête 1: Voir les données

```
SELECT * FROM customers LIMIT 5;
```

Requête 2: Clients par pays

```
SELECT country, COUNT(*) as countFROM customersGROUP BY countryORDER BY count DESC;
```

Requête 3: JOIN (Clients + Commandes)

```
SELECT c.name, SUM(o.amount) as total_spentFROM customers cLEFT JOIN orders o ON
c.customer_id = o.customer_idGROUP BY c.customer_id, c.nameORDER BY total_spent
DESC;
```

Étape 4 : Setup QuickSight (15 min)

102. AWS Console → QuickSight → Sign up → Standard Edition ($12/mois) → Free trial 1 month
103. Permissions S3: QuickSight → Manage account → Add S3 bucket (monsitetomrib)
104. Data source Athena: QuickSight → Data Sources → New → Athena → Name: Athena-Job7

Étape 5 : Créer les Datasets (10 min)

- Dataset CUSTOMERS: QuickSight → Datasets → New → Athena-Job7 → Table: customers
- Dataset ORDERS: QuickSight → Datasets → New → Athena-Job7 → Table: orders

Étape 6 : Créer les Dashboards (60 min)

Créez des analyses avec visualisations: Total clients (KPI), Clients par pays (Bar chart), Commandes par date (Line chart), Produits vendus (Bar chart)
Publiez les dashboards: Menu (•••) → Publish as dashboard → Nommer → Publish

Coûts

- S3: ~$0 (5 GB free année 1)
- Athena: $0 (1M queries free/mois)
- QuickSight: $0 (1 mois free essai)
- TOTAL CE MOIS: $0 (free tier)

10 - Job 8 : Déployer des applications conteneurisées (ECS/Fargate)

Objectif

Déployer une application conteneurisée sans gérer l'infrastructure sous-jacente en utilisant ECS avec Fargate.

Ressources AWS Utilisées

105. ECS (Elastic Container Service)
106. Fargate
107. ECR (Elastic Container Registry)

Ressources Créées

- Repository ECR: app-ecr
- Cluster ECS: cluster-ecs-app
- Task Definition: app-ecr-task
- Service: app-ecr-task-service
- Security Group: ecs-app-sg

Étape 1 : Créer l'application et l'image Docker

Fichiers du projet

app.js

```
const express = require('express');const app = express();const PORT = process.env.PORT || 3000;app.get('/', (req, res) => {  res.json({ message: 'Bienvenue sur mon app ECS Fargate!' });});app.get('/health', (req, res) => res.json({ status: 'healthy' }));app.listen(PORT, '0.0.0.0', () => console.log(`Server on port ${PORT}`));
```

Dockerfile

```
FROM node:18-alpineWORKDIR /appCOPY package*.json ./RUN npm install --productionCOPY . .EXPOSE 3000CMD ["node", "app.js"]
```

Build et test local

```
docker build -t mon-app-ecs .docker run -d -p 3000:3000 --name test-ecs mon-app-ecs
```

Vérifier : http://localhost:3000

Étape 2 : Créer un repository ECR

Dashboard:
ECR → Create repository → Repository name: app-ecr → Tag immutability: Mutable → Create
CLI:

```
aws ecr create-repository --repository-name app-ecr --region eu-west-3
```

Étape 3 : Pousser l'image vers ECR

```
# Login ECRaws ecr get-login-password --region eu-west-3 | docker login \--username AWS --password-stdin 703216717306.dkr.ecr.eu-west-3.amazonaws.com# Tag l'imagedocker tag mon-app-ecs:latest \703216717306.dkr.ecr.eu-west-3.amazonaws.com/app-ecr:latest# Pushdocker push 703216717306.dkr.ecr.eu-west-3.amazonaws.com/app-ecr:latest
```

Étape 4 : Créer un cluster ECS

Dashboard:
ECS → Clusters → Create cluster → Cluster name: cluster-ecs-app → Infrastructure: AWS Fargate (serverless) → Create
CLI:

```
aws ecs create-cluster --cluster-name cluster-ecs-app --region eu-west-3
```

Étape 5 : Créer une Task Definition

108. ECS → Task definitions → Create new task definition
109. Task definition family: app-ecr-task
110. Launch type: AWS Fargate
111. OS: Linux/X86_64
112. CPU: .25 vCPU | Memory: .5 GB
113. Container: Name: app-ecs, Image URI: 703216717306.dkr.ecr.eu-west-3.amazonaws.com/app-ecr:latest, Port: 3000
114. Clic Create

Étape 6 : Créer un Service ECS

115. ECS → Clusters → cluster-ecs-app → Services → Create
116. Launch type: FARGATE
117. Family: app-ecr-task
118. Service name: app-ecr-task-service
119. Desired tasks: 1
120. Networking: VPC défaut, Security group: Create new → ecs-app-sg
121. Inbound: Custom TCP, Port 3000, Source 0.0.0.0/0
122. Public IP: Turned ON ✓
123. Clic Create

Étape 7 : Tester l'application

124. ECS → Clusters → cluster-ecs-app → Tasks
125. Clic sur la task → récupérer Public IP
126. Ouvrir: http://<IP_PUBLIQUE>:3000

Résultat attendu: {"message":"Bienvenue sur mon app ECS Fargate!","version":"1.0.0",...}

Coûts

- Fargate: 750 heures/mois gratuites
- ECR: 500 MB stockage gratuit
- Cluster ECS: Gratuit
- Task Definition: Gratuit

Nettoyage

```
# Supprimer le serviceaws ecs update-service --cluster cluster-ecs-app \--service
app-ecr-task-service --desired-count 0 --region eu-west-3aws ecs delete-service --cluster
cluster-ecs-app \--service app-ecr-task-service --region eu-west-3# Supprimer le clusteraws
ecs delete-cluster --cluster cluster-ecs-app --region eu-west-3# Supprimer le repository
ECRaws ecr delete-repository --repository-name app-ecr --force --region eu-west-3
```

11 - Job 9 : Orchestrer les workflows avec Step Functions

Objectif

Automatiser une série de tâches de calcul sans serveur pour un flux de travail d'application complexe en utilisant Step Functions pour orchestrer les fonctions Lambda.

Ressources AWS Utilisées

- Step Functions
- Lambda
- CloudWatch

Étapes Détaillées

Étape 1 : Créer des fonctions Lambda

Créez 3 fonctions Lambda simples :

127. validate-input : valide les données d'entrée
128. process-data : traite les données
129. send-notification : envoie une notification

Exemple de fonction Lambda :

```
def lambda_handler(event, context):    if 'data' in event:        return {
'statusCode': 200,            'data': event['data'],            'processed': True        }
else:        raise Exception('Invalid input')
```

Étape 2 : Créer une machine d'état Step Functions

130. Allez à Step Functions > Create a state machine
131. Choisissez 'Write your own state machine'
132. Utilisez le langage Amazon States Language

Exemple de définition JSON :

```
{  "Comment": "Workflow de traitement de données",  "StartAt": "ValidateInput",  "States": {
"ValidateInput": {      "Type": "Task",      "Resource":
"arn:aws:lambda:...:function:validate-input",      "Next": "ProcessData"    },
"ProcessData": {      "Type": "Task",      "Resource":
"arn:aws:lambda:...:function:process-data",      "Next": "SendNotification"    },
"SendNotification": {      "Type": "Task",      "Resource":
"arn:aws:lambda:...:function:send-notification",      "End": true    }  }}
```

Étape 3 : Configurer les rôles IAM

133. Créez un rôle IAM pour Step Functions
134. Ajoutez les permissions d'invoquer Lambda
135. Testez que les permissions sont correctes

Étape 4 : Tester la machine d'état

136. Lancez une exécution avec des données de test
137. Vérifiez que chaque étape s'exécute correctement
138. Visualisez le flux d'exécution dans la console

Étape 5 : Ajouter la gestion des erreurs

139. Ajoutez des états 'Catch' pour gérer les erreurs
140. Définissez les états de fallback (ex: envoyer une alerte)
141. Testez les scénarios d'erreur

Étape 6 : Surveiller avec CloudWatch

142. Activez la journalisation CloudWatch
143. Configurez les alarmes pour les exécutions échouées
144. Analysez les exécutions dans la console

Coûts

- 4 000 transitions gratuites par mois avec Step Functions
- À partir de la 2e année : 0.000025$ par transition

Résumé des 9 Jobs

Compétences Acquises

- Administrer et sécuriser les infrastructures virtualisées (EC2, RDS, VPC)
- Mettre en production des évolutions de l'infrastructure de manière automatisée
- Participer à la détection et au traitement des incidents de sécurité avec CloudWatch et alarmes
- Déployer des applications web avec auto-scaling
- Utiliser le cloud pour l'hébergement statique et distribué
- Créer et gérer des bases de données relationnelles
- Construire des APIs serverless scalables
- Automatiser les pipelines de données et l'ingestion
- Analyser les données avec SQL et créer des tableaux de bord
- Déployer des applications conteneurisées sans gérer l'infrastructure
- Orchestrer des workflows complexes avec des fonctions sans serveur

Prochaines Étapes

- Obtenir la certification AWS Solutions Architect Associate
- Explorer les services avancés : Machine Learning, Kubernetes (EKS)
- Implémenter la sécurité avec AWS Secrets Manager et KMS
- Découvrir Infrastructure as Code (IaC) avec CloudFormation et Terraform
- Maîtriser le CI/CD avec CodePipeline et CodeBuild

Ressources Utiles

- Documentation AWS officielle : https://docs.aws.amazon.com
- AWS Free Tier : https://aws.amazon.com/free
- AWS Academy : formations gratuites en ligne
- LocalStack : simuler AWS en local
- Slack communauté AWS francophone

Conseils de Sécurité

- Activez MFA (Multi-Factor Authentication) sur votre compte AWS
- Utilisez IAM pour limiter les permissions par service
- Configurez CloudTrail pour auditer toutes les actions
- Gardez vos clés AWS privées et ne les commitez jamais dans Git
- Utilisez VPC pour isoler vos ressources
- Chiffrez vos données en transit (SSL/TLS) et au repos (KMS)
- Mettez en place une stratégie de rotation des credentials

Gestion des Coûts

- Vérifiez votre facture AWS mensuellement via Billing Dashboard
- Utilisez AWS Cost Explorer pour analyser vos dépenses
- Supprimez les ressources non utilisées (instances, volumes EBS, snapshots)
- Configurez des budget alerts pour être notifié si vos dépenses dépassent un seuil
- Envisagez d'acheter des Reserved Instances pour les charges long terme (jusqu'à 40% de réduction)