Description des étapes de mise en place de la solution API\_DataLake

|  |  |  |
| --- | --- | --- |
| Date | Modification | Auteur |
| 2025-05-12 | Création | Valentin Le Lay |
| 2025-05-23 | Passage d’une Ip publique éphémère (auto-assign) à une Elastic IP | Valentin Le Lay |

# Instance EC2 (Staging)

Pour l’instant aucun déploiement d’API Staging n’a été effectué.

# Instance EC2 (Prod)

L’instance EC2 est une machine virtuelle sur laquelle l’API fonctionne.

### Création de l’instance EC2 (Prod)

Name : API\_DataLake-Prod   
AMI : Ubuntu Server 24.04 LTS (HVM), SSD Volume Type  
Instance Type : t2.micro (free tier)  
Key Pair Login (SSH) : API\_DataLake-Prod-key.pem (voir dans Dashlane)  
VPC : DataLake-vpc  
Subnet : DataLake-public-subnet-a (10.0.0.0/20)  
Auto-Assign public ip : ~~Enable~~ Disable (il faudra lui associer manuellement une Elastic IP, 13.36.203.126 en l’occurence)  
Security Group : DataLake-custom-ec2-prod-sg  
EBS storage : 8GiB gp3  
Termination & Stop Protection : Enable  
Tags : Name = API\_DataLake-Prod ; Environment = Prod ; Application = API\_DataLake  
User Data :

Contrairement au déploiement des machines prod et staging pour le calcul des LTV j’ai ajouté l’argument set -e qui permet d’arrêter tout le processus si il rencontre la moindre erreur dans le script User Data. Cela évite de continuer le déploiement de l’instance sur de mauvaises bases et permet de pin-point parfaitement l’origine de l’erreur.

#!/bin/bash

set -e

# 1) Update package lists

apt-get update -y

# 2) Install agent prerequisites

apt-get install -y ruby-full wget

# 3) Download & install CodeDeploy agent’s installation script

cd /home/ubuntu

wget https://aws-codedeploy-eu-west-3.s3.eu-west-3.amazonaws.com/latest/install

chmod +x install

./install auto > /tmp/logfile

# 4) Verify the agent is running

systemctl status codedeploy-agent --no-pager

# Elastic IP address

An Elastic IP address is a static IPv4 address. You can remap easily the address from an instance to an another.

Créer une nouvelle allocation d’IP fixe : EC2 > Network & Security > Elastic IPs > Allocate Elastic IP address > Allocate.  
Allocated (fixed) IPv4 address : 13.36.203.126  
Name : API-DataLake-Prod-eip  
Allocation ID : eipalloc-0b4610d72b9fd1174  
(Chaque instance EC2 a automatiquement un ENI principal créé à son lancement)

# Security Group associé à l’instance EC2 (Prod)

Name : DataLake-custom-ec2-prod-sg

| **Port Range** | **Source** | **Description** |
| --- | --- | --- |
| 5000 | ::/0 | IPv6 from anywhere on port 5000 |
| 5000 | 0.0.0.0/0 | IPv4 from anywhere on port 5000 |
| 22 | ::/0 | IPv6 from anywhere on port 22 |
| 22 | 0.0.0.0/0 | IPv4 from anywhere on port 22 |

# Rôle associé à l’instance EC2 (Prod et Staging)

Permet au CodeDeploy de la machine EC2 de récupérer le code github dans le bucket S3. Ce rôle donne des accès limités mais suffisants par mesure de sécurité contrairement au rôle associé aux machines Prod et Staging du priceur LTV qui inclut des permissions trop larges (AutoScaling, ELB, CloudWatch, etc.).

Role Name : API\_DataLake-EC2-Role   
Policy : AmazonEC2RoleforAWSCodeDeploy & AmazonSSMManagedInstanceCore

{

"Version": "2012-10-17",

"Statement": [

{

"Action": [

"s3:GetObject",

"s3:GetObjectVersion",

"s3:ListBucket"

],

"Effect": "Allow",

"Resource": "\*"

}

]

}

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"ssm:DescribeAssociation",

"ssm:GetDeployablePatchSnapshotForInstance",

"ssm:GetDocument",

"ssm:DescribeDocument",

[…]

# Application CodeDeploy

Une **Application** dans CodeDeploy est simplement un conteneur logique pour tous tes déploiements d’une même codebase. Cette application ne fait rien toute seule : elle référence ton repo GitHub ou ton bucket S3 comme source d’artefacts, et regroupe ensuite un ou plusieurs **Deployment Groups**.

Name : API\_DataLake  
Compute platform : EC2/on-premises

# Deployment Group

Un Deployment Group (DG) relie l’Application CodeDeploy à un ou plusieurs serveurs cibles, et définit comment et où déployer.

Name : API\_DataLake-Prod-DG  
Service Role : CodeDeployServiceRole (on réutilise le même que pour API\_Reverse\_Mortgage)  
Tag Group : Name = API\_DataLake-Prod ; Environment = Prod ; Application = API\_DataLake  
Deployment settings : CodeDeployDefault.AllAtOnce  
Load Balancer : disabled (c’est un overkill pour cette api et c’est payant)

# Bucket S3

Le bucket S3 contiendra les artefacts de déploiement issus de GitHub et qui seront transférés sur l’instance EC2.

Name : api-datalake-deployments-971375  
Public Acess : Block all public access  
Bucket Versioning : Disabled  
Encryption : Oui, options par défaut

Life cycle rule : RemoveAllOldFilesAfter90Days. It removes ALL elements older than 90 days.

# CodePipeline

**CodePipeline** prend le code depuis GitHub et le déploie via **CodeDeploy** sur l’instance EC2.

Name : API\_DataLake-Prod-Pipeline  
Bucket location : api-datalake-deployments-971375  
Source Stage : GitHub App  
Connection : GitHub-ValentinLeLay  
Repository name : valentinllay/API\_DataLake  
Default branch : main  
Webhook events : Push (et pas une simple Pull Request)  
Build Stage : skipped   
Test Stage : skipped  
Deploy Stage : Deploy provider : AWS CodeDeploy ; Application Name : API\_DataLake, Deployment group : API\_DataLake-Prod-DG

## Explications de A à Z : du git push à l’API redémarrée

| **Étape** | **Service impliqué** | **Description** |
| --- | --- | --- |
| **1. Push GitHub** | GitHub | Vous poussez (git push origin main) votre dernier commit sur la branche main de valentinllay/API\_DataLake. |
| **2. Webhook GitHub** | GitHub → CodePipeline | Le webhook configuré sur l’app GitHub déclenche immédiatement CodePipeline dès qu’un push est détecté. |
| **3. Source Stage** | AWS CodePipeline | CodePipeline récupère le code source via la connexion GitHub-App, puis stocke le zip d’artefacts dans le bucket S3 api-datalake-deployments-971375. |
| **4. (Build/Test absents)** | — | Vous avez choisi de ne pas inclure d’étapes de build ou de test dans votre pipeline, donc CodePipeline passe directement à la phase déploiement. |
| **5. Deploy Stage** | AWS CodePipeline → CodeDeploy | CodePipeline invoque CodeDeploy en lui fournissant l’Application (API\_DataLake) et le Deployment Group (API\_DataLake-Prod-DG), plus l’URL du zip S3. |
| **6. Pré-déploiement** | AWS CodeDeploy | CodeDeploy récupère l’archive depuis S3 et, selon votre configuration de DG, distribue les fichiers sur l’instance EC2 ciblée (tag Name=API\_DataLake-Prod). |
| **7. Lifecycle hook : ApplicationStop** | CodeDeploy agent sur EC2 | Sur l’instance, le CodeDeploy Agent exécute votre script ApplicationStop.sh. Typiquement il tue l’ancien process Python et efface les caches APT pour repartir au propre. |
| **8. Synchronisation de fichiers** | CodeDeploy agent sur EC2 | Sur l’instance, le CodeDeploy Agent décompresse le contenu du zip (votre repo) dans le dossier /home/ubuntu/API\_DataLake/ |
| **9. Lifecycle hook : ApplicationStart** | CodeDeploy agent sur EC2 | Votre script ApplicationStart.sh est lancé. Il : • installe Python 3.12 et le module venv si besoin • crée/active le venv • installe les dépendances • démarre l’API |
| **10. Validation / Health Check (facultatif)** | CodeDeploy | Si vous aviez configuré un health-check (ex. sur un load-balancer), CodeDeploy ne marquerait le déploiement comme réussi qu’après avoir reçu un status “healthy”. |
| **11. Fin du déploiement** | CodeDeploy agent | Si tout s’est bien passé (retours 0 de vos scripts), CodeDeploy enregistre le succès, notifie dans la console/AWS SNS, et votre API tourne désormais sur la version mise à jour. |

## Une image contenant texte, capture d’écran, ligne, Police Le contenu généré par l’IA peut être incorrect.