AWS CodeDeploy est un service de déploiement entièrement géré qui automatise le déploiement des applications sur des plateformes variées :

- **Amazon EC2** (instances virtuelles).
- **AWS Lambda** (applications serverless).
- **Amazon ECS** (services conteneurisés).
- **Serveurs locaux (On-premises)**.

L'objectif principal de CodeDeploy est d'assurer des déploiements fiables, rapides, et sans interruption (**zéro downtime**).

Les Composants Essentiels d’AWS CodeDeploy

1. Application

L'application représente le **groupe logique** des ressources que vous souhaitez déployer.

- **Exemple** : Une application web en Python/Django ou PHP pour un site e-commerce.

---

2. Plateforme de Calcul

C’est la **destination** où votre application sera déployée. AWS CodeDeploy prend en charge les plateformes suivantes :

1. **Amazon EC2** : Instances virtuelles.
2. **AWS Lambda** : Fonctionnalités serverless.
3. **Amazon ECS** : Conteneurs (Docker ou Kubernetes).
4. **Serveurs locaux (On-premises)** : Vos propres serveurs physiques.

---

3. IAM Instance Profile

Le profil d’instance IAM est un rôle attaché aux instances EC2 ou serveurs locaux. Il permet à CodeDeploy d'accéder aux ressources nécessaires pour gérer les déploiements (par ex., récupérer la révision depuis S3).

- **Exemple** : IAM role avec les permissions pour accéder à des buckets S3 et logs CloudWatch.

---

4. Service Role

Un **rôle de service IAM** donne à AWS CodeDeploy les autorisations nécessaires pour effectuer des actions, telles que :

- Mettre à jour les instances EC2.
- Modifier les services ECS.
- Exécuter des commandes sur des fonctions Lambda.

Exemple : Rôle IAM avec la politique `AWSCodeDeployRole`.

---

5. Revision

C’est le **contenu de déploiement**, c'est-à-dire les fichiers qui définissent ce qui doit être déployé. Les révisions peuvent inclure :

- Code de l'application (ex. Python, Node.js).
- Fichiers de configuration (JSON, YAML).
- Scripts (ex. Bash).

Les révisions sont stockées dans :

- **Amazon S3**.
- **GitHub** ou un autre dépôt.

---

6. Target Revision

La **révision cible** est la version spécifique de votre application que vous souhaitez déployer. Cela permet de déployer des versions précises, comme `v1.2.3`.

---

7. Application Specification File

Un **AppSpec file** est utilisé pour définir comment le déploiement doit être effectué sur chaque plateforme. Par exemple :

- Où copier les fichiers.
- Quelles commandes exécuter avant, pendant ou après le déploiement.
- Comment gérer les hooks de cycle de vie.

Format :

- **YAML** pour EC2 et on-premises.
- **JSON** pour AWS Lambda.

---

8. CodeDeploy Agent 🤖

L'agent CodeDeploy est un logiciel installé sur des instances EC2 ou serveurs locaux. Il gère les étapes suivantes :

- Récupère la révision à partir de S3/GitHub.
- Exécute les commandes définies dans le fichier AppSpec.
- Signale l’état du déploiement (réussi ou échoué).

---

9. Instance Health 💓

L’état de santé des instances est essentiel pendant un déploiement. AWS CodeDeploy surveille les cibles pour s'assurer qu'elles sont fonctionnelles. Si une instance échoue, le déploiement peut être annulé.

Les Types de Déploiement 🚦

1. AllAtOnce 🚀

- **Description** : Toutes les cibles reçoivent la mise à jour en même temps.
- **Avantage** : Rapide.
- **Inconvénient** : Risque élevé si le déploiement échoue.
- **Utilisé avec** : EC2, on-premises, Lambda.

---

2. Rolling 🔚

- **Description** : Mise à jour progressive par petits groupes d'instances (ex. 25% à la fois).
- **Avantage** : Moins de risque d’interruption.
- **Inconvénient** : Plus lent que AllAtOnce.
- **Utilisé avec** : EC2, ECS.

---

3. Rolling with Additional Batch ➕

- **Description** : Ajoute une ou plusieurs instances supplémentaires au pool avant de commencer le déploiement, puis les retire à la fin.
- **Avantage** : Minimise les risques de surcharge.
- **Utilisé avec** : EC2, ECS.

---

4. Blue/Green □ □

- **Description** : Déploie la nouvelle version dans un environnement distinct. Une fois validée, le trafic est redirigé vers la nouvelle version.
- **Avantage** : Zéro interruption.
- **Inconvénient** : Coûte plus cher.
- **Utilisé avec** : EC2, ECS, Lambda.

---

5. Canary

- **Description** : Déploie sur un petit pourcentage d'instances (ex. 10%) avant de déployer sur le reste.
- **Avantage** : Surveillance facile des problèmes.
- **Utilisé avec** : Lambda, ECS.

---

6. Linear

- **Description** : Déploie des lots d'instances à intervalles réguliers.
- **Avantage** : Similaire à Rolling mais avec un contrôle temporel.
- **Utilisé avec** : Lambda, ECS.

---

7. Immutable

- **Description** : Crée de nouvelles instances pour déployer la mise à jour, puis détruit les anciennes.
- **Avantage** : Très sécurisé.
- **Utilisé avec** : EC2.

Quel Type de Déploiement pour Quel Serveur ?

Plateforme	Déploiement Recommandé
EC2	Rolling, Immutable, Blue/Green
Lambda	Canary, AllAtOnce, Linear
ECS	Blue/Green, Rolling
On-premises	Rolling, AllAtOnce

Exemple de Scénario Complet

Scénario 1 : Déploiement d'une Application Django

1. **Plateforme** : Instances EC2.
2. **Fichiers nécessaires** :
    - Code de l'application stocké dans S3.
    - Fichier AppSpec (YAML) pour définir le déploiement.
3. **Type de déploiement** : Rolling (25 % des instances à la fois).
4. **Étapes** :
    - Créer un groupe de déploiement avec les instances EC2 taguées `Environment=Production`.
    - Configurer un rôle IAM avec accès à S3 et CodeDeploy.
    - Lancer le déploiement via AWS CodeDeploy.

Questions pour Révision ❓

1. **Qu’est-ce qu’un IAM Instance Profile ?**
    - **Réponse** : Un rôle IAM attaché aux instances EC2 pour permettre l’accès aux ressources (ex. S3).
2. **À quoi sert le fichier AppSpec ?**
    - **Réponse** : Définir comment et où les fichiers doivent être déployés, et quelles commandes exécuter.
3. **Quel type de déploiement garantit zéro interruption ?**
    - **Réponse** : Le déploiement Blue/Green.
4. **Quelles plateformes sont prises en charge par AWS CodeDeploy ?**
    - **Réponse** : EC2, Lambda, ECS, on-premises.

Conclusion ✨

- AWS CodeDeploy est un outil puissant pour automatiser les déploiements.
- Les rôles IAM, AppSpec files, et types de déploiements doivent être bien configurés.
- Le choix du type de déploiement dépend de la plateforme et de vos besoins en termes de disponibilité.

Comparatif et Explicatif : Les Serveurs AWS 🌐

AWS propose plusieurs types de services de calcul adaptés aux différents besoins des développeurs, des startups et des grandes entreprises. Ces services incluent **Amazon EC2**, **AWS Lambda**, **Amazon ECS**, **Amazon EKS**, et les **Serveurs On-premises**. Ce cours va détailler chaque serveur, ses cas d'utilisation, ses avantages et inconvénients, et ses déploiements adaptés.

---

1. Amazon EC2 (Elastic Compute Cloud) 💻

Description :

Amazon EC2 offre des serveurs virtuels dans le cloud. Vous choisissez le système d'exploitation, la configuration matérielle (CPU, RAM, stockage), et gérez entièrement le serveur.

Cas d’utilisation :

- Hébergement de sites web et applications.
- Déploiement d'environnements de développement/test.
- Applications nécessitant un contrôle total sur l’infrastructure.

Avantages :

- **Personnalisation complète** : Vous configurez le serveur comme vous le souhaitez.
- **Choix des types d'instances** : Optimisées pour le calcul, la mémoire ou le stockage.
- **Autoscaling** : Capacité à ajouter ou supprimer automatiquement des instances selon le trafic.

Inconvénients :

- **Gestion manuelle** : Vous devez installer, configurer et mettre à jour les serveurs.
- **Coût** : Peut devenir élevé si mal optimisé.

Type de déploiement adapté :

- **Rolling** : Mettre à jour progressivement les instances.
- **Immutable** : Crée de nouvelles instances, puis détruit les anciennes après validation.
- **Blue/Green** : Pour zéro interruption pendant les mises à jour.

---

2. AWS Lambda (Serverless)

Description :

AWS Lambda permet d'exécuter du code sans gérer de serveurs. Vous chargez votre code, Lambda s'occupe de tout le reste (mise à l'échelle, maintenance).

Cas d'utilisation :

- Automatisation de tâches simples (ex. traitement de fichiers).
- Création d'API backend sans serveur.
- Applications événementielles (répondant à des événements comme les téléchargements S3 ou les requêtes API Gateway).

Avantages :

- **Aucune gestion d'infrastructure**.
- **Économie** : Paiement uniquement pour le temps d'exécution du code.
- **Mise à l'échelle automatique**.

Inconvénients :

- **Limitations** : Durée maximale d'exécution (15 minutes), mémoire limitée.
- **Cold start** : L'exécution initiale peut être plus lente.

Type de déploiement adapté :

- **Canary** : Testez avec un petit pourcentage d'invocations.
- **Linear** : Augmentation progressive des invocations.
- **Blue/Green** : Pour tester et basculer le trafic.

---

3. Amazon ECS (Elastic Container Service) 

Description :

ECS est un service pour exécuter et gérer des conteneurs Docker. Il peut être utilisé avec EC2 ou **AWS Fargate** (serverless pour conteneurs).

Cas d'utilisation :

- Applications conteneurisées nécessitant une orchestration simple.
- Microservices.
- Migration de charges de travail Docker vers le cloud.

Avantages :

- **Flexibilité** : Exécution sur EC2 ou Fargate.
- **Intégration facile** avec d'autres services AWS.
- **Mise à l'échelle automatique**.

Inconvénients :

- **Configuration complexe** : Nécessite une compréhension des conteneurs et de leur orchestration.
- **Coût élevé avec Fargate** pour de gros volumes.

Type de déploiement adapté :

- **Rolling** : Déployez les mises à jour conteneur par conteneur.
- **Blue/Green** : Pour garantir que les mises à jour n'affectent pas les utilisateurs.

---

4. Amazon EKS (Elastic Kubernetes Service) 🛠️

Description :

EKS est une solution gérée pour Kubernetes. Il permet d’exécuter des charges de travail conteneurisées avec la puissance de Kubernetes pour l’orchestration.

Cas d’utilisation :

- Gestion avancée de microservices avec Kubernetes.
- Applications nécessitant une grande scalabilité.
- Workloads hybrides (cloud + on-premises).

Avantages :

- **Puissance de Kubernetes** : Gestion avancée des conteneurs.
- **Flexibilité multi-cloud** : Compatible avec Kubernetes natif.

Inconvénients :

- **Complexité** : Kubernetes nécessite une expertise.
- **Coût élevé** pour de petites applications.

Type de déploiement adapté :

- **Rolling** : Mettez à jour les pods progressivement.
- **Blue/Green** : Déployez une nouvelle version de pods, puis redirigez le trafic.

---

5. Serveurs On-Premises (locaux) 🏠

Description :

AWS CodeDeploy peut être utilisé pour gérer le déploiement sur vos propres serveurs locaux, tout en profitant de la gestion automatisée.

Cas d'utilisation :

- Applications nécessitant une gestion locale (ex. pour des raisons de conformité).
- Applications ayant des dépendances physiques (ex. matériel spécifique).

Avantages :

- **Flexibilité** : Utilisation de l'infrastructure existante.
- **Intégration hybride** avec AWS.

Inconvénients :

- **Maintenance** : Vous devez toujours gérer la maintenance et la mise à jour des serveurs.
- **Pas de scalabilité automatique**.

Type de déploiement adapté :

- **Rolling** : Mettre à jour les serveurs par lots.
- **AllAtOnce** : Pour des environnements simples.

Comparaison des Serveurs AWS

Critères	EC2	Lambda	ECS	EKS	On-premises
Gestion Serveurs	Manuelle	Aucune	Aucune	Aucune	Manuelle
Mise à l'échelle	Auto Scaling	Automatique	Automatique	Automatique	Manuelle
Coût	Payez pour le serveur	Payez à l'usage	Payez pour les conteneurs	Payez pour Kubernetes	Basé sur l'infrastructure
Cas d'utilisation	Sites web et bases de données	Automatisation, API	Microservices	Microservices complexes	Besoins locaux
Type de Déploiement	Rolling, Blue/Green	Canary, Linear	Rolling, Blue/Green	Rolling, Blue/Green	Rolling, AllAtOnce

---

Exemples de Cas Pratiques

1. EC2 pour un Site Web

- **Objectif** : Héberger un site web avec une base de données.
- **Type de Déploiement** : Rolling avec mise à jour des instances 50% à la fois.
- **Outils AWS** : EC2 + RDS.

2. Lambda pour un Backend API

- **Objectif** : Créer une API REST serverless.
- **Type de Déploiement** : Canary avec 10% des invocations routées initialement.
- **Outils AWS** : API Gateway + Lambda + DynamoDB.

3. ECS pour une Application Conteneurisée

- **Objectif** : Déployer une application Docker.
- **Type de Déploiement** : Blue/Green avec AWS Fargate.
- **Outils AWS** : ECS + ALB (Load Balancer).

Conclusion

Chaque serveur AWS répond à des besoins spécifiques :

1. **EC2** : Pour un contrôle total.
2. **Lambda** : Pour des tâches ponctuelles et serverless.
3. **ECS/EKS** : Pour les microservices conteneurisés.
4. **On-premises** : Pour des environnements locaux.

Le choix du serveur et du type de déploiement dépend de la complexité de votre application, du besoin de scalabilité, et des contraintes de maintenance.