Cours : Introduction à AWS CodePipeline (Introduction to AWS CodePipeline)

---

1. Présentation de AWS CodePipeline

AWS CodePipeline est un service qui automatise l'ensemble du processus de développement et de déploiement d'une application. Il prend en charge les étapes de **construction (Build)**, **test (Test)** et **déploiement (Deploy)** en facilitant l'intégration et la livraison continue (**CI/CD - Continuous Integration / Continuous Delivery**).

---

2. Comprendre le CI/CD

(CI - Continuous Integration : Intégration Continue)

L'intégration continue permet aux développeurs de fusionner régulièrement leur code dans un référentiel centralisé où des tests automatisés sont exécutés. Cela permet de détecter rapidement les erreurs et d'améliorer la qualité du code.

(CD - Continuous Delivery : Livraison Continue)

La livraison continue permet d'automatiser l'envoi des nouvelles versions de l'application jusqu'à un environnement de préproduction ou de production. **Une intervention humaine est nécessaire** pour valider et déployer les mises à jour.

(CD - Continuous Deployment : Déploiement Continu)

Le déploiement continu est une extension de la livraison continue, où chaque mise à jour validée est directement déployée en production **sans intervention humaine**.

---

3. Pourquoi utiliser AWS CodePipeline ?

L'utilisation d'AWS CodePipeline présente plusieurs avantages :

- **Gain de temps** : Automatisation complète du cycle de développement.
- **Rapidité et fiabilité** : Réduction des délais entre le développement et la mise en production.
- **Moins d'erreurs** : Grâce à l'automatisation, les erreurs humaines sont minimisées.

---

4. Termes importants dans AWS CodePipeline

A) (Pipelines : Pipelines)

Un **pipeline** est un workflow défini qui contient plusieurs étapes menant à l'automatisation du déploiement.

1. **(Stage : Étape)**
    - Un pipeline est divisé en plusieurs **étapes (stages)**, chacune représentant une phase du SDLC (Software Development Life Cycle).
    - Exemples : `Source`, `Build`, `Test`, `Deploy`.
2. **(Action : Action)**
    - Chaque **étape (stage)** contient une ou plusieurs **actions** qui exécutent des tâches spécifiques.
    - Exemples : récupération du code source, exécution de tests, déploiement.

B) (Pipeline Execution : Exécution du Pipeline)

Une exécution de pipeline correspond à une exécution complète du workflow défini.

- **(Stopped Executions : Exécutions arrêtées)** : Une exécution interrompue volontairement.
- **(Failed Executions : Exécutions échouées)** : Une exécution qui a rencontré une erreur à une étape.
- **(Superseded Execution : Exécution remplacée)** : Une exécution remplacée par une version plus récente.

C) (Stage Execution : Exécution des Étapes)

Chaque **étape (stage)** du pipeline est exécutée indépendamment, et son état peut être suivi en temps réel.

D) (Action Executions : Exécution des Actions)

Chaque action définie dans un **stage** est exécutée de manière séquentielle ou parallèle en fonction de la configuration du pipeline.

E) (Artifacts : Artéfacts)

Les **artéfacts** sont des fichiers générés et stockés tout au long du pipeline.

- **(Amazon S3 : AWS S3)** : Stockage des fichiers temporaires et permanents.
- **(AWS CodeArtifact)** : Gestion des paquets et dépendances.
- **(AWS ECR : Elastic Container Registry)** : Stockage des images Docker.

F) (Source Revision : Révision de la source)

La révision de la source correspond à une **version spécifique du code source** récupérée depuis un dépôt Git (AWS CodeCommit, GitHub, Bitbucket) et qui déclenche le pipeline.

Conclusion

AWS CodePipeline est un outil puissant pour automatiser l'ensemble du processus **CI/CD** en réduisant le temps de livraison, en améliorant la fiabilité et en minimisant les erreurs. Grâce à ses différents composants (stages, actions, exécutions, artefacts), il permet de simplifier et d'optimiser le cycle de vie du développement logiciel.