

# Le Cours d'Ingénieur Cloud (The Cloud Engineer Course)

De zéro connaissance à ingénieur cloud prêt à l'emploi — un cours B4LCILC

13 août 2026

## Le Cours d'Ingénieur Cloud

**De zéro connaissance à véritablement prêt à l'emploi comme ingénieur cloud.**

### Comment fonctionne ce cours

Vous vous formez pour *être* ingénieur cloud : la personne qui construit des serveurs avec du code, maintient les systèmes en vie à 3 heures du matin, automatise les tâches ennuyeuses, et sait regarder un système en panne et découvrir calmement pourquoi. C'est un métier qui se pratique avec les mains, et ce cours le traite comme tel. Vous passerez bien plus de temps à taper qu'à lire — chaque module comporte des labs (travaux pratiques), et les labs *sont* le cours. Lire sur le cloud construit la reconnaissance ; construire dans le cloud construit une carrière.

Le cours comporte cinq étapes. **Étape 0 (Semaines 1–4) : Fondations** — comment fonctionnent réellement les ordinateurs et les réseaux, et Linux, le système d'exploitation sur lequel tourne le cloud. **Étape 1 (Semaines 5–12) : Le cœur du cloud, en pratique** — les cinq services AWS qui apparaissent dans chaque description de poste, appris sous forme de labs que vous construisez puis démontez. **Étape 2 (Semaines 13–20) : Automatisation** — scripting, Git, Terraform, CI/CD et conteneurs : les compétences qui séparent un ingénieur d'un cliqueur de console. **Étape 3 (Semaines 21–28) : Opérations** — supervision, incidents, sécurité et coûts : les compétences pour lesquelles on vous paie réellement. **Étape 4 (Mois 8–12) : Préparation à l'emploi** — trois projets de portfolio, trois certifications et un entraînement aux entretiens.

Règles pour l'ensemble du cours : étudiez **1,5 à 2 heures par jour, six jours par semaine** — la régularité l'emporte sur l'intensité, et c'est un métier qui s'apprend par la répétition quotidienne, comme un instrument de musique. Chaque module se termine par un **tableau de Vocabulaire** (les mots que vous devez posséder), un exercice « **À dire à voix haute** » (des phrases que vous répétez jusqu'à ce qu'elles deviennent naturelles — les entretiens d'embauche se parlent, ils ne s'écrivent pas), des **Exercices** pratiques, et un **Jalon (Milestone)** qui conditionne la progression : ne passez pas à la suite tant que vous ne savez pas le faire, car chaque étape repose sur la précédente. Dès la Semaine 1, tenez un **Journal d'ingénierie** : chaque lab, chaque message d'erreur, chaque correction, un paragraphe honnête. Au Mois 10, ce journal devient la matière première de votre portfolio et vos histoires d'entretien.

Une promesse en retour : rien dans ce cours ne suppose que vous sachiez quoi que ce soit. Chaque terme est défini la première fois qu'il apparaît. Si vous savez utiliser un navigateur web

et que vous acceptez de taper des commandes qui vous sembleront étranges pendant les deux premières semaines, vous avez tous les prérequis qui existent.

## La description de poste pour laquelle vous vous formez

En août 2026, nous avons extrait de vraies descriptions de poste de Cloud Engineer à partir de modèles d'employeurs et de guides de recrutement (Arc.dev, Wiz, DevsData, XOPA, Betterteam — liste complète dans les Sources). Retirez les noms d'entreprises, et les mêmes exigences reviennent encore et encore. Ce tableau est votre contrat avec le cours — chaque point demandé par un employeur correspond à l'étape qui l'enseigne :

| Ce que demandent les vraies descriptions de poste (quasi mot pour mot)                                                                                             | Où ce cours l'enseigne            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------|
| « Connaissance des systèmes d'exploitation Linux/Unix »                                                                                                            | Étape 0, Module 1                 |
| « Expertise en réseau cloud, y compris VPC, sous-réseaux, équilibreurs de charge, DNS »                                                                            | Étape 0 Module 2 + Étape 1 Lab 3  |
| « Expertise démontrée des services AWS de base, dont EC2, S3, RDS, VPC, IAM »                                                                                      | Étape 1 (Labs 1–5)                |
| « Concevoir, développer et déployer une infrastructure cloud à l'aide d'outils d'infrastructure as code tels que Terraform, CloudFormation »                       | Étape 2, Module 7                 |
| « Maîtrise de langages de script tels que Python, Bash, PowerShell ou Go »                                                                                         | Étape 2, Module 6                 |
| « Construire et maintenir des pipelines CI/CD avec des outils comme Jenkins, GitLab CI ou GitHub Actions »                                                         | Étape 2, Module 8                 |
| « Expérience des technologies de conteneurisation, y compris Docker et Kubernetes »                                                                                | Étape 2, Module 8                 |
| « Surveiller la santé et la performance de l'infrastructure à l'aide d'outils de supervision cloud natifs »                                                        | Étape 3, Module 9                 |
| « Participer à la réponse aux incidents, y compris l'analyse de logs » ; « compétences en dépannage et en analyse »                                                | Étape 3, Module 10                |
| « Mettre en œuvre et faire appliquer des contrôles de sécurité, y compris le chiffrement, la gestion des identités et des accès » ; « accès en moindre privilège » | Étape 1 Lab 5 + Étape 3 Module 10 |
| « Gérer les coûts cloud par le rightsizing des ressources, la mise en place de l'auto-scaling, l'étiquetage des ressources »                                       | Étape 3, Module 10                |

| Ce que demandent les vraies descriptions de poste (quasi mot pour mot)                                                       | Où ce cours l'enseigne                                                      |
|------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------|
| « Maintenir, tester et mettre en œuvre des procédures de reprise après sinistre »                                            | Étape 3 + Projet de portfolio 3                                             |
| « Certifications AWS appréciées »                                                                                            | Calendrier de certifications de l'Étape 4 (CCP – SAA – Terraform Associate) |
| « Fournir des orientations techniques et de la documentation » ; « bonnes compétences en communication et en collaboration » | Journal + runbooks + préparation aux entretiens de l'Étape 4                |

Contexte salarial, pour savoir vers quoi vous travaillez : les salaires des ingénieurs cloud aux États-Unis se concentrent autour d'une **médiane proche de 104 000 –140 K\$**, et des postes AWS seniors/spécialisés affichés bien au-dessus. Les postes débutants existent sous de nombreux intitulés — cloud support associate, junior cloud engineer, cloud operations engineer — et ce cours cible exactement leurs exigences.

## ÉTAPE 0 — FONDATIONS (Semaines 1-4)

### Module 1 (Semaines 1-2) : Comment fonctionnent les ordinateurs, et Linux — la langue des serveurs

**La grande idée** : un **serveur** n'est qu'un ordinateur dont le travail est de servir d'autres ordinateurs. Le laptop devant vous et les machines qui font tourner Netflix diffèrent par la taille et la fiabilité, pas par la nature : les deux sont un **CPU** (la partie qui fait le travail), de la **mémoire/RAM** (l'espace de travail rapide à court terme, effacé au redémarrage), un **disque** (le stockage lent à long terme qui survit aux redémarrages) et une **carte réseau** (la connexion à tout le reste), le tout coordonné par un **système d'exploitation (OS)**. Votre laptop tourne probablement sous Windows ou macOS. Les serveurs tournent massivement sous **Linux** — un OS gratuit et open source, stable, scriptable, et contrôlé entièrement par des commandes tapées. Ce dernier point est l'essentiel : on ne peut pas automatiser des clics de souris, mais on peut automatiser des commandes, et l'ingénierie cloud est de l'automatisation. Votre première fluidité sera donc le **terminal**.

**Le terminal, démystifié.** Le terminal (ou « shell » — le programme à l'intérieur est généralement **Bash**) est une conversation textuelle avec l'ordinateur. Vous tapez une commande ; il répond. C'est tout. Le \$ que vous verrez dans les exemples est le **prompt** (l'invite) — le shell qui dit « à vous ». Sous Linux, tout est fichier, les fichiers vivent dans un arbre unique qui commence à la racine /, et votre dossier personnel est /home/votrenom (surnom : ~). Un **chemin (path)** est l'adresse d'un fichier dans cet arbre : /home/anna/notes.txt.

**Procurez-vous un Linux pour vous entraîner (choisissez-en un, dix minutes)** : sous Windows, installez **WSL** (Windows Subsystem for Linux — un vrai Linux Ubuntu à l'intérieur de Windows : <https://learn.microsoft.com/en-us/windows/wsl/install>) ; sur un Mac,

l'application Terminal intégrée est assez proche pour commencer (macOS est un cousin d'Unix) ; ou attendez la Semaine 5, où vous louerez gratuitement un vrai serveur Linux chez AWS. WSL est la meilleure réponse pour la plupart des gens.

**Les 25 commandes essentielles — ce tableau, c'est les Semaines 1-2. Tapez chacune d'elles, de nombreuses fois :**

| Commande                 | Ce qu'elle fait                                                                                                     | Exemple                            |
|--------------------------|---------------------------------------------------------------------------------------------------------------------|------------------------------------|
| <code>pwd</code>         | Print working directory — « où suis-je ? »                                                                          | <code>pwd</code>                   |
| <code>ls</code>          | Lister les fichiers ici ( <code>-l</code> détail long, <code>-a</code> inclure les cachés)                          | <code>ls -la</code>                |
| <code>cd</code>          | Change directory — se déplacer dans l'arbre                                                                         | <code>cd /var/log</code>           |
| <code>mkdir</code>       | Créer un répertoire (dossier)                                                                                       | <code>mkdir projects</code>        |
| <code>touch</code>       | Créer un fichier vide                                                                                               | <code>touch notes.txt</code>       |
| <code>cp</code>          | Copier un fichier ( <code>-r</code> pour les dossiers)                                                              | <code>cp a.txt backup.txt</code>   |
| <code>mv</code>          | Déplacer ou renommer                                                                                                | <code>mv old.txt new.txt</code>    |
| <code>rm</code>          | Supprimer — définitivement, pas de corbeille. Respectez-la.                                                         | <code>rm notes.txt</code>          |
| <code>cat</code>         | Afficher tout le contenu d'un fichier                                                                               | <code>cat notes.txt</code>         |
| <code>less</code>        | Lire un long fichier page par page ( <code>q</code> pour quitter)                                                   | <code>less /var/log/syslog</code>  |
| <code>head / tail</code> | Premières / dernières lignes d'un fichier. <code>tail -f</code> suit un log en direct — un classique des opérations | <code>tail -f app.log</code>       |
| <code>grep</code>        | Chercher un motif dans du texte — la commande d'exploitation la plus utilisée de toutes                             | <code>grep "ERROR" app.log</code>  |
| <code>find</code>        | Trouver des fichiers par nom/taille/âge                                                                             | <code>find / -name "*.conf"</code> |
| <code>echo</code>        | Afficher du texte (souvent vers des fichiers ou des variables)                                                      | <code>echo "hello"</code>          |
| <code>nano</code>        | Un éditeur de texte convivial dans le terminal                                                                      | <code>nano notes.txt</code>        |
| <code>man</code>         | Le manuel de n'importe quelle commande ( <code>q</code> pour                                                        | <code>man grep</code>              |

| Commande                                 | Ce qu'elle fait                                                                                            | Exemple                               |
|------------------------------------------|------------------------------------------------------------------------------------------------------------|---------------------------------------|
|                                          | quitter)                                                                                                   |                                       |
| <code>sudo</code>                        | Exécuter une commande en tant qu'admin tout-puissant (« root »). Avec respect.                             | <code>sudo apt update</code>          |
| <code>apt</code>                         | Installer/mettre à jour des logiciels (famille Ubuntu/Debian)                                              | <code>sudo apt install htop</code>    |
| <code>chmod</code>                       | Changer les permissions d'un fichier                                                                       | <code>chmod 644 notes.txt</code>      |
| <code>chown</code>                       | Changer le propriétaire d'un fichier                                                                       | <code>sudo chown anna file</code>     |
| <code>ps</code>                          | Lister les processus en cours ( <code>ps aux</code> pour tous)                                             | <code>ps aux</code>                   |
| <code>top</code> (ou <code>htop</code> ) | Tableau de bord en direct CPU/mémoire/processus — « pourquoi le serveur est-il lent ? » commence ici       | <code>top</code>                      |
| <code>df -h</code> / <code>du -sh</code> | Espace disque libre / espace utilisé par un dossier — « le disque est plein » commence ici                 | <code>df -h</code>                    |
| <code>ssh</code>                         | Se connecter au terminal d'une autre machine via le réseau — <i>la</i> porte d'entrée de l'ingénieur cloud | <code>ssh anna@server-ip</code>       |
| <code>curl</code>                        | Faire une requête web depuis le terminal — « le site est-il en ligne ? »                                   | <code>curl https://example.com</code> |

**Les permissions, version 60 secondes.** Chaque fichier a un propriétaire et un mode comme `rwxr-xr--` : trois triplets — propriétaire, groupe, tout le monde — de **r** (read, lecture), **w** (write, écriture), **x** (execute, exécution). En chiffres : `r=4, w=2, x=1`, donc `chmod 755 script.sh` signifie « le propriétaire peut tout faire ( $7=4+2+1$ ) ; tous les autres peuvent le lire et l'exécuter ( $5=4+1$ ) ». Quand un programme « n'a pas la permission », c'est le système qui dit non — et désormais vous savez lire pourquoi.

**SSH, version 60 secondes.** `ssh` ouvre un terminal distant sécurisé sur une autre machine — depuis votre laptop vers un serveur en Virginie comme si vous étiez assis devant. Au lieu de mots de passe, les professionnels utilisent une **paire de clés** : une clé privée (un fichier secret sur votre laptop, jamais partagé) et une clé publique (placée sur le serveur). Elles s'emboîtent comme une clé et une serrure. Chaque serveur AWS que vous lancerez à l'Étape 1 vous remettra exactement cela.

**Processus et services :** un **processus** est un programme en cours d'exécution ; un **service** (ou « daemon », démon) est un processus qui tourne en permanence en arrière-plan — serveurs web, bases de données. `systemctl status nginx` demande à Linux « le service nginx est-il en bonne santé ? » — une phrase que vous taperez professionnellement pendant des années.

### Vocabulaire :

| Terme                        | Définition                                                                                                                            |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| Serveur                      | Un ordinateur dont le travail est de servir d'autres ordinateurs. Dans le cloud, un que vous louez.                                   |
| CPU / RAM / disque           | Le travailleur, l'espace de travail temporaire rapide (effacé au redémarrage), et le stockage permanent lent.                         |
| Système d'exploitation (OS)  | Le logiciel qui fait tourner la machine et héberge les programmes. Les serveurs tournent sous Linux.                                  |
| Linux / distribution         | L'OS serveur gratuit et open source. Une « distro » (Ubuntu, Amazon Linux, Debian) en est une déclinaison packagée.                   |
| Terminal / shell / Bash      | L'interface texte vers l'OS / le programme qui interprète vos commandes / le nom du shell standard.                                   |
| Prompt (invite)              | Le \$ — le shell qui attend votre commande.                                                                                           |
| Répertoire / chemin          | Un dossier / l'adresse complète d'un fichier dans l'arbre unique qui commence à /.                                                    |
| Root (deux sens)             | Le sommet de l'arbre de fichiers (/) et l'utilisateur admin tout-puissant. Le contexte vous dit lequel.                               |
| sudo                         | « Superuser do » — exécuter une commande avec les pouvoirs d'admin.                                                                   |
| Permissions (rwx)            | Règles par fichier sur qui peut lire, écrire, exécuter — affichées en triplets pour propriétaire/groupe/tout le monde.                |
| Processus / service (daemon) | Un programme en cours d'exécution / un qui tourne en permanence en arrière-plan (serveurs web, bases de données).                     |
| SSH / paire de clés          | Connexion distante sécurisée au terminal d'une autre machine / les fichiers de clés privée+publique qui remplacent les mots de passe. |
| Log (journal)                | Un fichier texte où le logiciel écrit ce qui s'est passé — le premier endroit où regarder                                             |

| Terme                   | Définition                                                                                                                  |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| Gestionnaire de paquets | quand quelque chose casse.<br>L'installateur de l'OS (apt, yum) — du logiciel par commande, pas par page de téléchargement. |

### Vidéos pour ce module (liens vérifiés) :

| Vidéo                                               | Chaîne          | Durée | Lien                                                                                                                      |
|-----------------------------------------------------|-----------------|-------|---------------------------------------------------------------------------------------------------------------------------|
| Linux Operating System — Crash Course for Beginners | freeCodeCamp    | ~2 h  | <a href="https://www.youtube.com/watch?v=ROjZy1WbCIA">https://www.youtube.com/watch?v=ROjZy1WbCIA</a>                     |
| WSL install guide (référence, pas une vidéo)        | Microsoft Learn | —     | <a href="https://learn.microsoft.com/en-us/windows/wsl/install">https://learn.microsoft.com/en-us/windows/wsl/install</a> |

**À dire à voix haute jusqu'à ce que ce soit naturel :** “Let me SSH in and check the logs.” (« Je vais me connecter en SSH et vérifier les logs. ») · “Grep the log for the error, then tail -f it while we retry.” (« Greppez le log pour trouver l'erreur, puis faites un tail -f pendant qu'on réessaie. ») · “It's a permissions problem — who owns the file and what's the mode?” (« C'est un problème de permissions — qui possède le fichier et quel est le mode ? ») · “Check top — is it CPU, memory, or disk?” (« Regardez top — c'est le CPU, la mémoire ou le disque ? »)

**Exercices :** (1) Dans votre terminal Linux, construisez une petite arborescence de projet avec `mkdir` et `touch`, copiez et déplacez des choses, puis supprimez le tout — en narrant chaque commande à voix haute. (2) Créez `hello.sh` contenant `echo "hello from $(whoami)"`, rendez-le exécutable avec `chmod +x`, exécutez-le avec `./hello.sh`. (3) Lancez `tail -f` sur un fichier de log (sous Ubuntu : `sudo tail -f /var/log/syslog`) et regardez les lignes arriver. (4) Journal : expliquez à un ami imaginaire pourquoi les serveurs tournent sous Linux, en trois phrases.

**Jalon :** sans notes, vous savez naviguer n'importe où dans l'arbre de fichiers, créer/copier/déplacer/supprimer des fichiers, expliquer `chmod 755`, et utiliser `grep` pour trouver un mot dans un fichier. Si l'un de ces points exige de chercher quelque chose, passez deux jours de plus ici. Ce module est porteur pour tout le reste.

## Module 2 (Semaines 3-4) : Fondamentaux du réseau + votre compte AWS

**La grande idée :** un réseau, ce sont des ordinateurs qui se passent des enveloppes adressées. Chaque machine reçoit une **adresse IP** (comme 172.31.8.14 — une adresse postale numérique). Les données sont découpées en **paquets** (des enveloppes) et acheminées saut par saut vers l'adresse de destination. À l'arrivée, un numéro de **port** indique à quel programme l'enveloppe est destinée — le même immeuble, des milliers de portes numérotées : le port **22** est SSH, **80** le web non chiffré (**HTTP**), **443** le web chiffré (**HTTPS**), **5432** PostgreSQL. « Ouvrir le port 443 » signifie « autoriser les enveloppes adressées à la porte 443 ».

**Privé vs public :** votre domicile et chaque réseau cloud réutilisent des **plages d'IP privées** (10.x.x.x, 172.16–31.x.x, 192.168.x.x) qui ne fonctionnent qu'à l'intérieur du réseau local ; une **IP publique** est joignable depuis tout l'internet. Cette séparation est le socle de la sécurité cloud : ce qui n'a pas besoin de faire face à l'internet ne reçoit aucune adresse publique du tout.

**DNS — l'annuaire téléphonique de l'internet.** Les humains utilisent des noms (exemple.com) ; les paquets ont besoin de numéros. Le **DNS** traduit : votre machine demande à un serveur DNS « quelle est l'IP de exemple.com ? », obtient le numéro, puis se connecte. La moitié des pannes mystérieuses impliquent le DNS ; la blague du métier “it's always DNS” (« c'est toujours le DNS ») existe parce que c'est souvent vrai. `nslookup exemple.com` effectue la résolution à la main.

**HTTP — comment parle le web.** Un **client** (navigateur) envoie une requête — une **méthode** (GET = récupérer, POST = soumettre) plus un chemin — et le **serveur** répond avec un **code de statut** : 200 OK, 301 déplacé, 403 interdit, 404 introuvable, 500 erreur serveur, 502/503 « le serveur derrière moi est cassé/surchargé ». Mémorisez ces six-là ; en tant qu'ingénieur, vous les lirez quotidiennement. `curl -I https://exemple.com` vous montre une ligne de statut et des en-têtes en direct.

**Les pare-feu :** une liste de règles décidant quels paquets peuvent passer, selon la source, la destination et le port — « autoriser 443 depuis partout ; autoriser 22 uniquement depuis le bureau ; refuser le reste ». Dans AWS, le pare-feu par serveur s'appelle un **groupe de sécurité (security group)**, et les groupes mal configurés sont le trou de sécurité n°1 du débutant. La **latence** (le délai, en millisecondes) et la **bande passante** (la capacité par seconde) complètent le vocabulaire : la distance crée de la latence, raison pour laquelle les clouds ont des régions dans le monde entier.

**Maintenant, votre compte AWS — la seconde moitié de ce module.** Rendez-vous sur <https://aws.amazon.com/free> et créez un compte Free Tier (offre gratuite) (e-mail, téléphone, carte de crédit/débit pour l'identité — la carte n'est pas réellement débitée si vous suivez la discipline de teardown de ce cours). Le **Free Tier** donne des allocations mensuelles des services de base (dont 750 heures/mois d'un petit serveur EC2 la première année) et chaque lab de ce cours est conçu pour y tenir. Ensuite, avant toute autre chose, trois étapes de sécurité non négociables — les réaliser est le premier exercice de votre carrière en sécurité :

1. **La MFA sur l'utilisateur root.** L'utilisateur root est la clé maîtresse du compte. Ajoutez l'authentification multifacteur (une application d'authentification sur téléphone) dans IAM → Security credentials. Puis cessez d'utiliser root pour le travail quotidien.
2. **Créez un utilisateur IAM administrateur** (le Lab 5 explique IAM en profondeur ; pour l'instant : console → IAM → Users → créer un utilisateur avec accès admin) et connectez-vous désormais avec cet utilisateur.
3. **Une alerte de facturation.** Console → Billing → Budgets → créez un **budget zéro dépense (zero-spend budget)** (le modèle d'AWS qui vous envoie un e-mail dès que quoi que ce soit est facturé). Activez aussi les alertes d'utilisation du Free Tier. Un ingénieur qui ne sait pas contrôler les coûts est un passif ; vous êtes à 20 minutes de la création de votre compte et déjà en avance sur bien des professionnels. (Référence :

<https://docs.aws.amazon.com/cost-management/latest/userguide/budgets-managing-costs.html>)

## Vocabulaire :

| Terme                          | Définition                                                                                                        |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------|
| Adresse IP                     | L'adresse réseau numérique d'une machine, p. ex. 172 . 31 . 8 . 14.                                               |
| Paquet                         | Une enveloppe de données adressée ; tout le trafic en est un flux.                                                |
| Port                           | Une porte numérotée sur une machine identifiant le programme destinataire du trafic : 22 SSH, 80 HTTP, 443 HTTPS. |
| IP privée / publique           | Une adresse valable uniquement dans un réseau local / une adresse joignable depuis tout l'internet.               |
| DNS                            | Le système qui traduit les noms (example.com) en adresses IP. "It's always DNS."                                  |
| HTTP / HTTPS                   | Le protocole requête-réponse du web / le même, chiffré avec TLS.                                                  |
| Code de statut                 | Le résumé de la réponse du serveur : 200 OK, 404 introuvable, 500 erreur serveur, 503 surchargé.                  |
| Pare-feu                       | La liste de règles décidant quels paquets passent, selon source, destination, port.                               |
| Groupe de sécurité             | Le pare-feu par serveur d'AWS. En configurer un de travers est le trou classique du débutant.                     |
| Latence / bande passante       | Délai (ms) / capacité (par seconde). La distance crée de la latence.                                              |
| Client / serveur (rôles)       | Celui qui demande et celui qui répond dans toute conversation réseau.                                             |
| AWS Free Tier                  | L'allocation mensuelle gratuite d'un nouveau compte AWS — le budget entier de ce cours.                           |
| Utilisateur root               | L'identité maîtresse du compte AWS. Protégez-la par MFA, puis cessez de l'utiliser.                               |
| Alerte de facturation / budget | L'e-mail automatique quand la dépense franchit un seuil. Le vôtre est réglé à 0 \$.                               |

## Vidéos pour ce module :

| Vidéo                                        | Chaîne                         | Durée   | Lien                                                                                                  |
|----------------------------------------------|--------------------------------|---------|-------------------------------------------------------------------------------------------------------|
| What is AWS?                                 | Amazon Web Services (official) | ~2 min  | <a href="https://www.youtube.com/watch?v=a9_D53WsUs">https://www.youtube.com/watch?v=a9_D53WsUs</a>   |
| Top 50+ AWS Services Explained in 10 Minutes | Fireship                       | ~10 min | <a href="https://www.youtube.com/watch?v=JlbiYCM48to">https://www.youtube.com/watch?v=JlbiYCM48to</a> |
| AWS Networking Basics — VPC & Subnets        | KodeKloud                      | ~30 min | <a href="https://www.youtube.com/watch?v=QM63dyA_4Pc">https://www.youtube.com/watch?v=QM63dyA_4Pc</a> |

**À dire à voix haute :** “What’s the IP, and is it public or private?” (« Quelle est l’IP, et est-elle publique ou privée ? ») · “Is port 443 open in the security group?” (« Le port 443 est-il ouvert dans le groupe de sécurité ? ») · “Curl it — what status code do you get?” (« Faites un curl — quel code de statut obtenez-vous ? ») · “Did DNS resolve? Check with nslookup before blaming the server.” (« Le DNS a-t-il résolu ? Vérifiez avec nslookup avant d’accuser le serveur. »)

**Exercices :** (1) `ping google.com` (notez la latence), `nslookup google.com` (notez que le DNS renvoie plusieurs IP), `curl -I https://aws.amazon.com` (lisez la ligne de statut et trois en-têtes, puis cherchez la signification de chacun). (2) Trouvez l’IP privée de votre machine (`ip addr` sous Linux) et votre IP publique (cherchez « what is my IP ») — expliquez dans votre journal pourquoi elles diffèrent. (3) Terminez la configuration du compte AWS : MFA, utilisateur admin, budget zéro dépense — capture d’écran du budget pour votre journal. (4) Dessinez de mémoire : laptop → résolution DNS → requête HTTPS sur le port 443 → pare-feu → serveur web. Trois fois, jusqu’à ce que ce soit automatique.

**Jalon — fin de l’Étape 0 :** vous savez narrer ce qui se passe quand vous tapez une URL et appuyez sur Entrée — DNS, IP, port, pare-feu, requête HTTP, code de statut — en moins de deux minutes, en utilisant chaque terme correctement ; et votre compte AWS existe avec la MFA et une alerte de budget à 0 \$. Vous en savez désormais plus sur le fonctionnement de l’internet que la plupart des gens qui en vivent. L’Étape 1 est celle où vous commencez à construire dessus.

## ÉTAPE 1 — LE CŒUR DU CLOUD, EN PRATIQUE (Semaines 5–12)

**Comment fonctionne cette étape.** Chacun des cinq services ci-dessous est un **Lab** : un objectif, un plan d’étapes (assez détaillé pour être suivi, assez court pour vous obliger à réfléchir — la réflexion, c’est l’apprentissage), ce que vous avez appris, et — toujours — le **teardown** (démontage). La discipline du teardown compte doublement : elle vous maintient dans le Free Tier, et « ne rien laisser tourner qui ne soit pas nécessaire » est un réflexe professionnel que les recruteurs sondent réellement en entretien. Reconstituez chaque lab au moins deux fois : une

fois en suivant le plan, une fois de mémoire. La seconde construction est celle où le savoir passe dans vos mains. Prévoyez environ une semaine et demie par lab ; utilisez la marge pour la casse, car des choses vont casser, et les déboguer est le meilleur enseignement que ce cours ne peut pas scénariser.

D'abord, trois idées qui cadrent tout ce que vous allez construire. **Régions et zones de disponibilité** : une Région est un groupe géographique de centres de données AWS (choisissez-en une proche de vous et restez-y — les ressources d'une région sont invisibles depuis une autre, la confusion n°1 du « où est passé mon serveur ? »). Une **zone de disponibilité (Availability Zone, AZ)** est un centre de données isolé au sein de la région ; les systèmes sérieux tournent dans deux AZ pour que la défaillance d'un bâtiment ne les fasse pas tomber. **La console vs la CLI** : la console (<https://console.aws.amazon.com>) est le panneau de contrôle web d'AWS — parfaite pour apprendre et observer ; la **CLI AWS** (aws dans votre terminal) fait tout ce que fait la console, de façon scriptable. Vous commencerez dans la console et passerez à la CLI, car l'Étape 2 automatise tout ce que vous faites ici à la main.

## Lab 1 (Semaines 5-6) : EC2 — votre premier serveur

**EC2 (Elastic Compute Cloud)** loue des machines virtuelles, appelées **instances**. C'est l'acte primal de l'ingénierie cloud : un vrai serveur Linux, sur l'internet, en 60 secondes, gratuitement.

**Objectif** : lancer un serveur Linux, s'y connecter en SSH, lui faire servir une page web au monde entier, puis le détruire.

**Plan des étapes** : 1. Console → EC2 → Launch instance. Nommez-la. Choisissez **Amazon Linux 2023** comme **AMI** (Amazon Machine Image — le disque modèle depuis lequel votre serveur démarre) et **t2.micro** ou **t3.micro** comme **type d'instance** (la taille ; ceux-ci sont dans le Free Tier). 2. Créez une **paire de clés** ; un fichier de clé privée .pem se télécharge. C'est votre clé SSH du Module 1 — gardez-la précieusement, faites-lui un `chmod 400`. 3. Dans les paramètres réseau, autorisez SSH (port 22) *depuis « My IP » uniquement* — vous savez exactement ce que signifie cette règle de groupe de sécurité désormais — et autorisez HTTP (port 80) depuis partout. 4. Lancez, attendez « running », copiez l'IP publique, puis depuis votre terminal : `ssh -i mykey.pem ec2-user@<public-ip>`. Respirez un coup : vous êtes à l'intérieur d'un ordinateur dans un centre de données AWS. 5. Sur le serveur : `sudo dnf install -y nginx && sudo systemctl start nginx && sudo systemctl enable nginx`. Puis naviguez vers `http://<public-ip>` — c'est *votre* serveur web qui répond au monde. 6. Remplacez la page par défaut : `echo "<h1>Built by me, on EC2</h1>" | sudo tee /usr/share/nginx/html/index.html`. Rafraîchissez. Capture d'écran pour le journal. 7. Fouillez comme un ingénieur : `top`, `df -h`, `sudo tail -f /var/log/nginx/access.log` pendant que vous rafraîchissez la page — regardez vos propres visites arriver dans le log.

**Ce que vous avez appris** : les AMI, les types d'instance, les paires de clés, les groupes de sécurité en conditions réelles, le SSH vers un vrai serveur, l'installation et l'exécution d'un service Linux, la lecture de ses logs — c'est-à-dire les gestes physiques quotidiens du métier.

**Teardown :** EC2 → sélectionner l'instance → Instance state → **Terminate**. Confirmez qu'elle affiche « terminated ». Le Free Tier donne 750 heures/mois d'une instance micro, donc même la laisser tourner n'aurait rien facturé — mais démontez-la quand même. L'habitude est le but.

## Lab 2 (Semaine 7) : S3 — des fichiers qui ne disparaissent jamais

**S3 (Simple Storage Service)** est du stockage objet : un seau sans fond, durable à onze neufs, pour les fichiers. C'est la réponse par défaut à « où met-on les fichiers ? » et, mal configuré, la source des fuites de données les plus célèbres de l'histoire — raison pour laquelle ce lab est à moitié stockage, à moitié sécurité.

**Objectif :** créer un bucket, le manipuler depuis la CLI, héberger un minuscule site web statique, et comprendre exactement ce que signifie « bucket public ».

**Plan des étapes :** 1. Console → S3 → Create bucket (les noms sont uniques au niveau mondial — `yourname-lab-2026` fonctionne). Notez que **Block Public Access** est activé par défaut. Téléversez n'importe quel fichier via la console ; retéléchargez-le. 2. Installez la CLI AWS localement et exécutez `aws configure` avec une **clé d'accès (access key)** que vous créez pour votre utilisateur admin IAM (une clé d'accès est un identifiant+mot de passe programmatique pour l'API — traitez-la comme un mot de passe, ne la mettez jamais dans du code ; vous intérioriserez cette règle au Lab 5). 3. Depuis votre terminal : `aws s3 ls · aws s3 cp notes.txt s3://yourname-lab-2026/ · aws s3 sync ./myfolder s3://yourname-lab-2026/backup/`. Sentez la différence : la console, c'est de la visite ; la CLI, c'est de l'ingénierie. 4. Site web statique : créez un second bucket, activez l'hébergement de site statique, téléversez un `index.html`, et ajoutez la **bucket policy** (politique de bucket) de lecture publique documentée (un document JSON de permissions — lisez-le ligne par ligne : *qui* peut faire *quoi* sur *quel* bucket). Votre page est désormais sur l'internet sans aucun serveur. 5. Explorez les **classes de stockage** (Standard → Infrequent Access → Glacier : moins cher par Go, plus lent/coûteux à récupérer) et définissez une **règle de cycle de vie** (« déplacer les objets vers IA après 30 jours ») — votre premier avant-goût de gestion de coûts automatisée.

**Ce que vous avez appris :** le stockage objet vs les disques, la CLI et les clés d'accès, les bucket policies et l'accès public (délibérément, pas accidentellement), les niveaux de stockage comme levier de coût.

**Teardown :** videz les deux buckets, supprimez les deux buckets, et — important — désactivez toute clé d'accès que vous n'utilisez pas. L'allocation gratuite de S3 est petite (5 Go) mais ces fichiers pèsent des kilo-octets ; la discipline, encore une fois, est le but.

## Lab 3 (Semaines 8-9) : VPC — le réseau qui vous appartient

**VPC (Virtual Private Cloud)** est votre tranche privée et clôturée du réseau d'AWS. Jusqu'ici vous utilisiez celle par défaut sans la regarder ; les ingénieurs construisent la leur, car la phrase « *la base de données se trouve dans un sous-réseau privé sans route vers l'internet* » est la moitié de la sécurité cloud, et vous allez la rendre vraie de vos propres mains.

**Objectif :** construire un réseau à deux niveaux — sous-réseau public pour un serveur web, sous-réseau privé pour une future base de données — et prouver que la moitié privée est injoignable depuis l'internet.

**Plan des étapes :** 1. Console → VPC → Create VPC. Donnez-lui le bloc d'adresses 10.0.0.0/16 — la **notation CIDR**, où /16 signifie « les 16 premiers bits sont fixes, le reste est à moi » : 65 536 adresses privées. 2. Créez deux **sous-réseaux (subnets)** : 10.0.1.0/24 (public, dans l'AZ-a) et 10.0.2.0/24 (privé, dans l'AZ-b). Un sous-réseau est un bloc plus petit au sein du VPC, vivant dans exactement une AZ. 3. Créez une **Internet Gateway** (passerelle internet — la porte du VPC vers l'internet) et attachez-la. Créez une **table de routage** avec la règle 0.0.0.0/0 → internet gateway (« tout ce qui n'est pas local va vers la porte internet ») et associez-la au sous-réseau *public* uniquement. Le sous-réseau privé ne garde que la route locale — cette absence de route est la sécurité. 4. Lancez une instance EC2 micro dans chaque sous-réseau (la publique avec une IP publique, la privée sans). 5. La preuve : SSH vers l'instance publique — ça fonctionne. Essayez l'adresse de l'instance privée depuis votre laptop — ça bloque indéfiniment, et vous savez désormais dire précisément pourquoi. Puis faites SSH *depuis* l'instance publique vers la privée (elle est joignable depuis l'intérieur du VPC) : la machine publique joue le rôle de **bastion (bastion host)**, un motif standard de l'industrie que vous venez de découvrir en le construisant. 6. Concept bonus à rechercher et consigner : une **NAT gateway** (passerelle NAT) permet aux machines privées d'appeler *vers l'extérieur* (pour les mises à jour) tout en restant injoignables de l'extérieur — mais elle facture à l'heure, alors lisez à son sujet, ne la construisez pas.

**Ce que vous avez appris :** CIDR, sous-réseaux, tables de routage, passerelles internet, la séparation public/privé, les bastions — le point réseau exact de chaque description de poste cloud, en mémoire musculaire.

**Teardown :** terminez d'abord les deux instances, puis supprimez le VPC (qui emporte avec lui sous-réseaux, tables de routage et passerelle). Vérifiez dans EC2 que rien n'affiche « running ».

## Lab 4 (Semaine 10) : RDS — une base de données que vous ne dorlotez pas

**RDS (Relational Database Service)** est une base de données managée : AWS fait tourner le moteur de base de données (PostgreSQL, MySQL...) et gère les sauvegardes, les correctifs et le failover (bascule automatique), tandis que vous possédez les données et les requêtes. Le « service managé » est le marché central du cloud — échanger un peu de contrôle contre beaucoup de labeur indifférencié — et ce lab est celui où vous ressentez ce marché.

**Objectif :** lancer une base PostgreSQL dans un sous-réseau privé, s'y connecter depuis une instance EC2, et comprendre les sauvegardes et le multi-AZ — puis la démonter promptement, car RDS est le lab le plus facile à laisser tourner par accident.

**Plan des étapes :** 1. Reconstituez rapidement le VPC du Lab 3 (seconde construction de mémoire — c'est de la répétition espacée délibérée), en ajoutant un second sous-réseau *privé* dans une autre AZ, car RDS exige un **groupe de sous-réseaux (subnet group)** couvrant deux AZ. 2. Console → RDS → Create database → PostgreSQL — **modèle Free tier** (cela présélectionne db.t3.micro/db.t4g.micro, mono-AZ). Définissez le mot de passe maître. Placez-la dans votre VPC, **Public access: No**, dans un groupe de sécurité qui autorise le port 5432 *uniquement depuis le groupe de sécurité du serveur web* — une règle qui référence une autre règle plutôt qu'une IP. Élégant, et standard. 3. Lancez une EC2 micro dans le sous-réseau public, installez le client postgresql, et connectez-vous : `psql -h <rds-endpoint> -U postgres`. L'**endpoint** (point de terminaison) est un nom DNS, pas une IP — AWS peut déplacer la

machine sous-jacente, et le nom la suit. (Le Module 2 paie déjà son loyer.) 4. À l'invite `psql` : créez une table, insérez trois lignes, relisez-les avec un `select`. Vous n'avez pas besoin de profondeur SQL aujourd'hui ; vous avez besoin d'y avoir touché. 5. Visitez sans activer : le réglage des **sauvegardes automatiques** (restauration à un instant donné à partir de snapshots nocturnes + logs), et l'option **Multi-AZ** (une réplique de secours en direct dans une autre AZ avec **failover** automatique — environ le double du coût, raison pour laquelle la prod dit oui et les labs disent non). Prenez un **snapshot** manuel, retrouvez-le dans la console, comprenez que vous pourriez en restaurer un clone. 6. Question de journal qui vaut dix minutes : que fait AWS pour vous ici, exactement, que vous devriez sinon faire vous-même à 2 heures du matin ? (Correctifs, sauvegardes, failover, matériel.) Cette réponse est la réponse d'entretien sur les services managés.

**Ce que vous avez appris** : les bases de données managées, les groupes de sous-réseaux, les règles de groupe de sécurité à groupe de sécurité, les endpoints, les snapshots, le multi-AZ/failover — plus une démonstration en direct du fait de payer en contrôle pour acheter de la fiabilité.

**Teardown** : supprimez l'instance RDS (refusez le snapshot final pour un lab ; notez que la prod en prendrait un), supprimez le snapshot manuel (les snapshots facturent le stockage !), terminez l'EC2, supprimez le VPC. Vérifiez votre tableau de bord de facturation le lendemain — le lire chaque semaine est une habitude de l'Étape 3 qui commence maintenant.

## Lab 5 (Semaines 11–12) : IAM — qui peut faire quoi

**IAM (Identity and Access Management)** décide quelles personnes et quels programmes peuvent faire quoi sur quelles ressources. Il ne facture rien, ne provisionne rien — et c'est le service le plus audité, le plus sondé en entretien et le plus impliqué dans les brèches de tout AWS. Les points sécurité des descriptions de poste (« contrôles d'accès en moindre privilège », « politiques IAM ») signifient ce lab.

**Objectif** : créer des utilisateurs, des groupes, des politiques et — la cruciale — un **rôle**, et interioriser le moindre privilège en vous faisant refuser par AWS.

**Plan des étapes** : 1. Les concepts d'abord, cinq minutes : un **utilisateur** est une identité pour un humain ou un programme ; un **groupe** regroupe des utilisateurs ; une **politique (policy)** est un document JSON accordant des permissions (« `allow s3:GetObject on arn:aws:s3:::my-bucket/*` ») ; un **rôle** est une identité avec des politiques mais *sans mot de passe*, qu'une partie de confiance *assume* temporairement — c'est ainsi que les serveurs et services obtiennent des permissions sans aucun secret stocké. 2. Créez une utilisatrice `readonly-rita` dans un groupe doté de la politique managée `AWS ReadOnlyAccess`. Connectez-vous avec elle dans une fenêtre de navigation privée : elle voit tout, mais chaque bouton de création/suppression échoue avec un refus explicite. Lisez entièrement l'une de ces erreurs — apprendre à décoder « `not authorized to perform X on Y` » est une compétence quotidienne du métier. 3. Écrivez votre première politique personnalisée dans l'éditeur JSON : autorisez `s3:ListBucket` et `s3:GetObject` sur un bucket spécifique. Attachez-la à une nouvelle utilisatrice ; vérifiez qu'elle peut lire ce bucket et *rien d'autre*. Vous venez de mettre en œuvre le moindre privilège, pas seulement de le définir. 4. Le rôle, la récompense : créez un rôle pour EC2 avec `AmazonS3ReadOnlyAccess`, lancez une instance micro avec ce rôle attaché,

connectez-vous en SSH, et exécutez `aws s3 ls` — cela fonctionne avec **aucune clé d'accès nulle part sur la machine**. L'instance assume le rôle et reçoit automatiquement des identifiants de courte durée. C'est le motif de sécurité le plus important d'AWS : *des rôles pour les machines, jamais de clés sur les machines*. Dites-le deux fois. 5. Auditez votre propre compte comme un pro : le root est-il sous MFA (Semaine 3) ? Des clés d'accès de plus de 90 jours ? Un utilisateur a-t-il plus de permissions qu'il n'en utilise ? Lancez le **credential report** (rapport d'identifiants) d'IAM et lisez-le. Ce rituel de dix minutes, mensuel, est la naissance de la checklist d'hygiène IAM de l'Étape 3.

**Ce que vous avez appris** : utilisateurs/groupes/politiques/rôles, lire et écrire du JSON de politique, le moindre privilège imposé par l'expérience, des rôles-pas-des-clés, et votre premier audit de sécurité.

**Teardown** : supprimez les utilisateurs de test et leurs identifiants, terminez l'instance, gardez pour toujours la connaissance des rôles.

### Vocabulaire pour l'Étape 1 (un seul tableau, les cinq labs) :

| Terme                                      | Définition                                                                                                                     |
|--------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| Région / zone de disponibilité             | Groupe géographique de centres de données AWS / un centre de données isolé en son sein. Les systèmes sérieux couvrent deux AZ. |
| EC2 / instance                             | Le service de location de machines virtuelles / un serveur loué.                                                               |
| AMI                                        | Amazon Machine Image — le disque modèle depuis lequel une instance démarre.                                                    |
| Type d'instance                            | La taille/spécification choisie (t3.micro = minuscule, Free Tier).                                                             |
| Groupe de sécurité                         | Pare-feu par ressource : quels ports, depuis quelles sources.                                                                  |
| Paire de clés                              | Les clés SSH privée/publique pour atteindre votre instance.                                                                    |
| S3 / bucket / objet                        | Stockage objet / un conteneur nommé / un fichier stocké.                                                                       |
| Classe de stockage / règle de cycle de vie | Niveau prix-vitesse des objets / règle automatique les déplaçant vers des niveaux moins chers avec l'âge.                      |
| Bucket policy                              | Document JSON énonçant qui peut faire quoi sur un bucket. Les publics font les gros titres.                                    |
| CLI AWS / clé d'accès                      | L'interface terminal vers AWS / les identifiants programmatiques pour elle (à traiter comme un mot de passe).                  |
| VPC / sous-réseau                          | Votre tranche de réseau privée / un bloc plus petit de celle-ci vivant dans une AZ, public ou privé.                           |

| Terme                                       | Définition                                                                                                                                                |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| CIDR                                        | Notation de bloc d'adresses : 10.0.0.0/16<br>= « 16 premiers bits fixes, 65 536 adresses à moi ».                                                         |
| Internet gateway / table de routage         | La porte internet du VPC / les règles décidant où le trafic est envoyé. Pas de route = injoignable = sécurisé.                                            |
| NAT gateway                                 | Permet aux machines privées d'appeler dehors sans être joignables dedans. Facture à l'heure — connaissez-la, ne la laissez pas tourner.                   |
| Bastion (bastion host)                      | La machine publique durcie par laquelle vous passez en SSH pour atteindre les privées.                                                                    |
| RDS / endpoint                              | Service managé de base de données relationnelle / le nom DNS auquel vous vous connectez.                                                                  |
| Snapshot / Multi-AZ / failover              | Copie à un instant donné / réplique de secours en direct dans une seconde AZ / la bascule automatique vers elle.                                          |
| Utilisateur / groupe / politique / rôle IAM | Identité / ensemble d'identités / octroi de permissions en JSON / identité assumable sans mot de passe — comment les machines obtiennent des permissions. |
| Moindre privilège                           | La règle d'or : le minimum de permissions qui fait le travail, rien de plus.                                                                              |
| Service managé                              | AWS exécute le labeur indifférencié (correctifs, sauvegardes, failover) ; vous gardez les données et les décisions.                                       |

### Vidéos pour cette étape :

| Vidéo                                 | Chaîne                                                          | Durée   | Lien                                                                                                  |
|---------------------------------------|-----------------------------------------------------------------|---------|-------------------------------------------------------------------------------------------------------|
| Getting Started with EC2              | AWS Developers (official)                                       | 27 min  | <a href="https://www.youtube.com/watch?v=nJ-djerESW0">https://www.youtube.com/watch?v=nJ-djerESW0</a> |
| Introduction to Amazon S3             | Amazon Web Services (official)                                  | ~5 min  | <a href="https://www.youtube.com/watch?v=ecv-19sYL3w">https://www.youtube.com/watch?v=ecv-19sYL3w</a> |
| AWS Networking Basics — VPC & Subnets | KodeKloud (à revoir après le Lab 3 — elle résonne différemment) | ~30 min | <a href="https://www.youtube.com/watch?">https://www.youtube.com/watch?</a>                           |

| Vidéo                               | Chaîne                               | Durée | Lien                                                                                                                   |
|-------------------------------------|--------------------------------------|-------|------------------------------------------------------------------------------------------------------------------------|
| The AWS Shared Responsibility Model | désormais)<br>Digital Cloud Training | 4 min | v=QM63dyA_4Pc<br><a href="https://www.youtube.com/watch?v=ESPBBEK-cvo">https://www.youtube.com/watch?v=ESPBBEK-cvo</a> |

**À dire à voix haute :** “It’s in a private subnet; there’s no route to the internet gateway.” (« C’est dans un sous-réseau privé ; il n’y a pas de route vers l’internet gateway. ») · “The security group only allows 5432 from the web tier’s security group.” (« Le groupe de sécurité n’autorise 5432 que depuis le groupe de sécurité du niveau web. ») · “The instance uses a role — there are no keys on the box.” (« L’instance utilise un rôle — il n’y a pas de clés sur la machine. ») · “Terminate it when you’re done; nothing idles in this account.” (« Terminez-la quand vous avez fini ; rien ne tourne au ralenti dans ce compte. »)

**Jalon — fin de l’Étape 1 :** la construction du gantelet. En une seule séance, de mémoire : VPC avec sous-réseaux public/privé — serveur web EC2 (public, rôle attaché, servant une page) — RDS PostgreSQL (privée, joignable uniquement depuis le serveur web) — un bucket S3 que l’instance peut lire via son rôle — puis démontez le tout proprement. Moins de trois heures signifie que vous êtes prêt pour l’Étape 2. Aussi : **commencez la préparation du CLF-C02 maintenant** (le cours complet de freeCodeCamp — <https://www.youtube.com/watch?v=7HKot-brXFE> — à 1,25× en révision ; l’édition classique de 14 heures est <https://www.youtube.com/watch?v=NhDYbskXRgc>), et passez l’examen **AWS Cloud Practitioner** vers la Semaine 14–16.

## ÉTAPE 2 — AUTOMATISATION (Semaines 13–20)

**L’idée de cette étape :** tout ce que vous avez cliqué à l’Étape 1, vous allez maintenant le faire avec du code. La légère insulte de l’industrie pour le clic dans la console est le **ClickOps** ; la formule de description de poste pour ce qui le remplace est « *concevoir, développer et déployer une infrastructure cloud à l’aide de l’infrastructure as code* ». Cette étape est la différence entre quelqu’un qui a *utilisé* AWS et quelqu’un qu’une entreprise paiera pour le *faire tourner*.

### Module 6 (Semaines 13–15) : Scripting — Bash et Python pour les opérations, plus Git

**Le scripting Bash**, c’est mettre les commandes du Module 1 dans un fichier pour que l’ordinateur les répète parfaitement. Apprenez, dans cet ordre : les **variables** (NAME="web-1"), la **substitution de commande** (TODAY=\$(date +%F)), le **if** (if [ -f "\$FILE" ]; then ... fi), les **boucles** (for f in \*.log; do gzip "\$f"; done), les **codes de sortie** (chaque commande renvoie 0 en cas de succès, non-zéro en cas d’échec ; && enchaîne sur succès — les scripts prennent leurs décisions avec ça), et la lecture des **arguments** (\$1, \$2). C’est 90 % du Bash d’exploitation réel. Écrivez votre premier script véritablement utile cette semaine : `backup.sh` — archivez un répertoire avec `tar`, nommez l’archive avec la date du jour, envoyez-

la avec `aws s3 cp` vers un bucket, supprimez les archives locales de plus de 7 jours, et écrivez avec `echo` une ligne de succès/échec dans un fichier de log. Ce seul script réunit variables, substitution, conditionnelles, codes de sortie et la CLI en un seul artefact — et un point sur votre CV (« sauvegardes automatisées vers S3 »).

**Python** est le grand frère de Bash : meilleur pour tout ce qui implique de la logique, des données ou des appels d'API. Vous avez besoin d'un Python d'exploitation fonctionnel, pas d'une profondeur de génie logiciel : variables et types, listes et dictionnaires, `if/for`, fonctions, lecture/écriture de fichiers, gestion d'erreurs `try/except`, et installation de bibliothèques avec `pip`. Puis rencontrez **boto3**, la bibliothèque AWS pour Python : `import boto3; ec2 = boto3.client("ec2"); ec2.describe_instances()` — soudain votre savoir de l'Étape 1 devient programmable. Écrivez `audit.py` : listez chaque instance EC2 du compte avec son nom, son type, son état et son heure de lancement, et affichez une ligne d'avertissement pour tout ce qui tourne depuis plus de 24 heures. Félicitations — vous avez écrit un véritable outil de contrôle des coûts pour lequel les employeurs paient.

**Git** est le système de gestion de versions dans lequel vit tout le travail d'ingénierie. Les concepts : un **dépôt (repository)** (un dossier dont tout l'historique est enregistré), un **commit** (un instantané sauvegardé avec un message), une **branche** (une ligne de travail parallèle), un **remote** (la copie sur GitHub), et une **pull request** (demander qu'une branche soit relue et fusionnée). La boucle quotidienne tient en six commandes : `git init / git clone`, `git status`, `git add`, `git commit -m "message"`, `git push`, `git pull`. Créez un compte GitHub aujourd'hui, créez un dépôt nommé `cloud-journey`, et à partir de maintenant, **chaque script et chaque config que vous écrivez dans ce cours y est commité**. Dans dix mois, cet historique de commits est la preuve publique et datée de tout ce que ce cours affirme que vous savez faire.

### Vocabulaire :

| Terme               | Définition                                                                                                       |
|---------------------|------------------------------------------------------------------------------------------------------------------|
| Script              | Un fichier de commandes exécutées de haut en bas — l'atome de l'automatisation.                                  |
| Variable / argument | Une valeur nommée dans un script / une valeur passée à l'exécution (\$1).                                        |
| Code de sortie      | Le signal de succès (0) ou d'échec (non-zéro) de chaque commande — comment les scripts prennent leurs décisions. |
| Cron                | Le planificateur de Linux : exécuter un script chaque nuit à 2 h, pour toujours.                                 |
| Python / pip        | Le langage de programmation préféré du monde des opérations / son installateur de paquets.                       |
| boto3               | La bibliothèque Python pour piloter AWS — la console, en code.                                                   |
| try/except          | Le « tente ceci ; si ça échoue, fais cela à la place » de Python — comment les scripts                           |

| Terme                          | Définition                                                                                                                          |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| Git / dépôt / commit           | échouent avec élégance.<br>La gestion de versions / l'historique enregistré d'un projet / un instantané sauvegardé avec un message. |
| Branche / merge                | Une ligne de travail parallèle / sa réintégration dans la ligne principale.                                                         |
| GitHub / remote / pull request | Le site d'hébergement / la copie cloud de votre dépôt / une demande de fusion d'une branche, relue et discutée.                     |
| README                         | Le document de première page d'un dépôt expliquant ce qu'il est et comment le lancer. Les recruteurs les lisent.                    |

### Vidéos pour ce module :

| Vidéo                                    | Chaîne       | Durée                                  | Lien                                                                                                  |
|------------------------------------------|--------------|----------------------------------------|-------------------------------------------------------------------------------------------------------|
| Learn Python — Full Course for Beginners | freeCodeCamp | ~4,5 h (à regarder sur les 3 semaines) | <a href="https://www.youtube.com/watch?v=rfscVS0vtbw">https://www.youtube.com/watch?v=rfscVS0vtbw</a> |

**À dire à voix haute :** “I’ll script it — it’ll be wrong once, then never again.” (« Je vais le scripter — ce sera faux une fois, puis plus jamais. ») · “Check the exit code before the next step runs.” (« Vérifiez le code de sortie avant que l’étape suivante ne s’exécute. ») · “Commit it with a message that says why, not what.” (« Commitez-le avec un message qui dit pourquoi, pas quoi. ») · “It’s in the repo — pull the latest.” (« C’est dans le dépôt — tirez la dernière version. »)

**Exercices :** (1) `backup.sh`, comme spécifié ci-dessus, planifié chaque nuit avec cron sur une instance EC2 de lab. (2) `audit.py`, comme spécifié ci-dessus. (3) Les deux commits dans `cloud-journey` avec un README expliquant chacun. (4) Cassez votre propre script exprès (un chemin erroné, un bucket manquant) et faites-le échouer *bruyamment et clairement* — la gestion d’erreurs est un langage d’amour des opérations.

**Jalon :** votre script de sauvegarde survit à deux exécutions d’affilée et à un dossier manquant (pas de crash, message clair) ; votre audit Python tourne contre votre vrai compte ; votre dépôt montre une semaine de commits.

## Module 7 (Semaines 16–17) : Terraform — l’infrastructure as code

**La grande idée :** au lieu de faire exister les ressources en cliquant, vous écrivez des fichiers texte *déclarant* ce qui doit exister — « un VPC, deux sous-réseaux, une instance, ces tags » — et **Terraform** fait en sorte qu’AWS corresponde au fichier. Déclaratif, pas impératif : vous énoncez la destination, pas les virages. Pourquoi chaque description de poste le demande : les fichiers vivent dans Git, donc l’infrastructure devient **relue** (des pull requests avant les changements), **répétable** (le même fichier construit dev, staging et prod à l’identique),

**réversible** (retour en arrière en annulant le commit) et **auditable** (l'historique dit qui a changé quoi, quand, pourquoi). La console est un atelier ; Terraform est une usine.

**La boucle centrale que vous exécuterez des centaines de fois :** `terraform init` (télécharger le **provider** AWS — le plugin qui traduit vos fichiers en appels d'API) — `terraform plan` (une répétition à blanc qui affiche exactement ce qui serait créé/modifié/détruit — lisez chaque ligne ; les ingénieurs qui appliquent sans lire les plans causent des pannes) — `terraform apply` (le rendre réel) — `terraform destroy` (le défaire — le teardown en une seule commande, ce que vous trouverez désormais magnifique). Terraform consigne ce qu'il a construit dans un **fichier d'état (state file)** — sa mémoire de la réalité ; perdez-le ou modifiez la réalité à la main dans son dos (le « drift », la dérive) et la douleur suit. Les ressources se déclarent en **HCL**, un langage de configuration lisible :

```
resource "aws_instance" "web" {
  ami          = "ami-0abcdef1234567890"
  instance_type = "t3.micro"
  tags = { Name = "web-1", Project = "lab" }
}
```

**Le lab (c'est le module) :** reconstruisez le gantelet de l'Étape 1 en Terraform, de façon incrémentale. (1) Initialisez un dépôt `terraform-labs` ; écrivez la config d'un simple bucket S3 taggé ; `plan`, lecture, `apply`, vérification dans la console — il est là ; `destroy`. (2) Faites grandir la config : VPC, deux sous-réseaux, internet gateway, table de routage — la construction du Lab 3, désormais ~60 lignes de HCL. (3) Ajoutez le groupe de sécurité et une instance EC2 avec `variables.tf` (région, taille d'instance) et `outputs.tf` (afficher l'IP publique après l'apply). (4) Changez le type d'instance dans le fichier et ré-appliquez — regardez Terraform calculer la différence et ne changer *que cela*. Ce comportement piloté par le diff est toute la magie. (5) Détruisez tout, confirmez que la console est vide, commitez, et taguez le dépôt `v1.0`. La documentation de référence vit sur <https://developer.hashicorp.com/terraform> — mettez-la en favori ; lire la doc des providers est la moitié du vrai travail Terraform.

## Vocabulaire :

| Terme                                                         | Définition                                                                                      |
|---------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| IaC (Infrastructure as Code)                                  | Déclarer l'infrastructure dans des fichiers texte versionnés qu'un outil transforme en réalité. |
| Terraform / HCL                                               | L'outil IaC multi-cloud dominant / son langage de configuration.                                |
| Provider                                                      | Le plugin traduisant vos fichiers en appels d'API d'un cloud (ici : AWS).                       |
| <code>plan</code> / <code>apply</code> / <code>destroy</code> | Le diff à blanc / le rendre réel / tout défaire. Lisez chaque plan.                             |
| Fichier d'état                                                | Le registre de ce que Terraform a construit — sa mémoire de la réalité. Protégez-le.            |
| Drift (dérive)                                                | La réalité a changé dans le dos de Terraform (quelqu'un a cliqué). L'ennemi.                    |

| Terme                      | Définition                                                                                                                                 |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| Variable / output / module | Une entrée de config / un résultat affiché (une IP, une URL) / un morceau de config réutilisable et packagé.                               |
| Déclaratif vs impératif    | Énoncer la destination vs scripter les virages. Terraform est déclaratif.                                                                  |
| CloudFormation             | Le service IaC propre à AWS — même idée, AWS uniquement. Les descriptions de poste acceptent l'un ou l'autre ; apprenez Terraform d'abord. |

### Vidéos pour ce module :

| Vidéo                                                     | Chaîne              | Durée  | Lien                                                                                                  |
|-----------------------------------------------------------|---------------------|--------|-------------------------------------------------------------------------------------------------------|
| Terraform explained in 15 mins                            | TechWorld with Nana | 18 min | <a href="https://www.youtube.com/watch?v=l5klai_GBDE">https://www.youtube.com/watch?v=l5klai_GBDE</a> |
| Terraform Course — Automate your AWS cloud infrastructure | freeCodeCamp        | ~2,5 h | <a href="https://www.youtube.com/watch?v=SLB_c_ayRMo">https://www.youtube.com/watch?v=SLB_c_ayRMo</a> |

**À dire à voix haute :** “Nothing changes in prod except through Terraform.” (« Rien ne change en prod sauf via Terraform. ») · “Show me the plan before you apply.” (« Montrez-moi le plan avant d’appliquer. ») · “That was clicked in manually — that’s drift; let’s import it or remove it.” (« Ça a été cliqué à la main — c’est du drift ; importons-le ou supprimons-le. ») · “It’s in the module; reuse it, don’t rewrite it.” (« C’est dans le module ; réutilisez-le, ne le réécrivez pas. »)

**Exercices :** le lab ci-dessus, plus : (1) créez délibérément du drift — changez le tag de l’instance à la main dans la console, lancez `terraform plan`, et regardez Terraform s’en apercevoir ; consignez ce qu’il a proposé. (2) Écrivez un paragraphe en langage clair dans le README du dépôt : « pourquoi cela bat le clic ». Si vous ne savez pas l’écrire, vous ne le possédez pas encore.

**Jalon :** depuis un répertoire vide, vous savez monter la pile VPC + instance avec `init/plan/apply` et la retirer avec `destroy`, en expliquant à voix haute ce que fait chaque commande — sans notes.

## Module 8 (Semaines 18–20) : CI/CD, Docker, et un premier regard sur Kubernetes

**La CI/CD** (Continuous Integration / Continuous Delivery — intégration et livraison continues) est le tapis roulant robotisé attaché à votre dépôt Git : à chaque push, il vérifie, teste, construit et — quand c’est configuré — déploie automatiquement votre changement. Le but : des mises en production petites, fréquentes et ennuyeuses au lieu de rares et terrifiantes. **GitHub Actions** est le système CI/CD intégré à GitHub : un **workflow** est un fichier YAML dans

.github/workflows/ disant « au push, exécute ces étapes sur un **runner** neuf (une VM temporaire) ». Lab : ajoutez à terraform-labs un workflow qui exécute terraform fmt - check et terraform validate à chaque push — poussez un fichier délibérément malformé et regardez le ✗ rouge, corrigez-le et regardez le ✓ vert. Vous avez désormais un pipeline ; aussi petit soit-il, c'est la même espèce que ceux de chaque description de poste. (Deuxième étape, dans le Projet 1 : un workflow qui exécute terraform plan sur les pull requests, pour que les changements d'infrastructure proposés montrent leur diff dans la revue.)

**Docker** emballe une application avec tout ce dont elle a besoin dans un **conteneur** — une boîte à lunch scellée qui tourne à l'identique sur votre laptop, sur EC2 et partout ailleurs, mettant fin à l'antique fléau du « ça marche sur ma machine ». Une **image** est la recette figée (construite depuis un **Dockerfile**, un court fichier texte d'étapes de build : partir FROM une image de base, COPY votre application, définir la commande de démarrage) ; un **conteneur** est une instance en cours d'exécution d'une image ; un **registre (registry)** (Docker Hub, ou **ECR** d'AWS) est l'endroit où les images sont poussées et tirées. Les conteneurs ne sont *pas* des VM : ils partagent le noyau Linux de l'hôte, donc ils démarrent en une seconde environ et vous pouvez en faire tourner des dizaines sur une instance micro. Lab : sur une instance EC2, installez Docker, docker run hello-world, puis docker run -d -p 80:8080 <a sample web image> et naviguez dessus ; puis écrivez un Dockerfile de cinq lignes servant votre page statique du Lab 2 depuis une image de base nginx, construisez-le, exécutez-le, et poussez-le sur Docker Hub. Apprenez les verbes quotidiens : docker ps, docker logs, docker exec -it <id> bash (un shell à l'intérieur du conteneur), docker stop.

**Kubernetes (K8s)** — pour ce cours, une compétence de niveau lecture, honnêtement étiquetée. Quand une entreprise fait tourner des centaines de conteneurs sur de nombreuses machines, quelque chose doit les planifier, redémarrer ceux qui plantent, faire monter en charge les occupés et router le trafic entre eux : cet orchestrateur est Kubernetes. Apprenez la carte des concepts dès maintenant — un **cluster** de **nœuds (nodes)** exécute des **Pods** (la plus petite unité déployable, généralement un conteneur) ; un **deployment** déclare « garde 3 répliques de ce pod en vie » et le cluster le rend vrai en continu (déclaratif encore — Kubernetes est la philosophie de Terraform appliquée au logiciel en cours d'exécution) ; un **service** donne aux pods une adresse stable. L'offre managée d'AWS est **EKS**. Les descriptions de poste junior veulent exactement cette culture générale plus la fluidité Docker ; la vraie compétence d'exploitation K8s (et la certification CKA) est un solide objectif d'Année 2, pas de Mois 5. Ce cours vous dit honnêtement ce qui relève de l'un et de l'autre.

## Vocabulaire :

| Terme                              | Définition                                                                                                           |
|------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| CI/CD                              | Le pipeline automatisé qui vérifie, construit, teste et livre chaque changement.                                     |
| GitHub Actions / workflow / runner | La CI/CD intégrée de GitHub / le fichier YAML définissant un pipeline / la VM temporaire qui l'exécute.              |
| YAML                               | Le format de config à indentation des pipelines et de Kubernetes. L'indentation, c'est le sens — faites-y attention. |

| Terme                             | Définition                                                                                                                            |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| Docker / image / conteneur        | La boîte à outils des conteneurs / la recette figée en couches / une instance en cours d'exécution de celle-ci.                       |
| Dockerfile                        | Le court fichier texte d'étapes qui construit une image.                                                                              |
| Registre / ECR                    | L'endroit où les images sont poussées et tirées / le registre d'AWS.                                                                  |
| Kubernetes (K8s) / cluster / nœud | L'orchestrateur de conteneurs / son groupe de machines / une machine de ce groupe.                                                    |
| Pod / deployment / service        | La plus petite unité déployable / « garde N répliques en vie », imposé en continu / une adresse stable devant les pods.               |
| EKS                               | Le plan de contrôle Kubernetes managé d'AWS.                                                                                          |
| Rollback                          | Revenir vite à la version précédente quand une mise en production se comporte mal — le filet de sécurité que la CI/CD rend routinier. |

### Vidéos pour ce module :

| Vidéo                                                  | Chaîne              | Durée   | Lien                                                                                                  |
|--------------------------------------------------------|---------------------|---------|-------------------------------------------------------------------------------------------------------|
| DevOps CI/CD Explained in 100 Seconds                  | Fireship            | 2 min   | <a href="https://www.youtube.com/watch?v=scEDHsr3APg">https://www.youtube.com/watch?v=scEDHsr3APg</a> |
| What is DevOps? REALLY understand it                   | TechWorld with Nana | ~15 min | <a href="https://www.youtube.com/watch?v=0yWAtQ6wYNM">https://www.youtube.com/watch?v=0yWAtQ6wYNM</a> |
| Docker in 100 Seconds                                  | Fireship            | 2 min   | <a href="https://www.youtube.com/watch?v=Gjnup-PuquQ">https://www.youtube.com/watch?v=Gjnup-PuquQ</a> |
| Docker Tutorial for Beginners [FULL COURSE in 3 Hours] | TechWorld with Nana | 3 h     | <a href="https://www.youtube.com/watch?v=3c-iBn73dDE">https://www.youtube.com/watch?v=3c-iBn73dDE</a> |
| Kubernetes explained in 15 mins                        | TechWorld with Nana | ~16 min | <a href="https://www.youtube.com/watch?v=VnvRFRk_5lk">https://www.youtube.com/watch?v=VnvRFRk_5lk</a> |

**À dire à voix haute :** “Don’t merge until the pipeline is green.” (« Ne fusionnez pas tant que le pipeline n’est pas vert. ») · “It’s containerized — same image in dev and prod.” (« C’est conteneurisé — la même image en dev et en prod. ») · “Exec into the container and check its logs.” (« Entrez dans le conteneur avec exec et vérifiez ses logs. ») · “K8s keeps three replicas up; kill one and watch it come back.” (« K8s garde trois réplicas en vie ; tuez-en un et regardez-le revenir. »)

**Exercices :** les deux labs ci-dessus, plus : (1) consignez une réponse en un paragraphe à « VM vs conteneur vs serverless — quand utiliser chacun ? » (le Serverless in 100 Seconds de Fireship — [https://www.youtube.com/watch?v=W\\_VV2Fx32\\_Y](https://www.youtube.com/watch?v=W_VV2Fx32_Y) — complète la troisième option). (2) Terminez l’instance du lab Docker ; confirmez que vos images survivent dans le registre — remarquez que l’*artefact* survit désormais au *serveur*, ce qui est toute la vision moderne du déploiement en une seule observation.

**Jalon — fin de l’Étape 2 :** votre GitHub montre : un dépôt de scripts d’exploitation Bash + Python fonctionnels, un dépôt Terraform qui construit et détruit une vraie pile, un workflow Actions vert, et un Dockerfile poussé vers un registre — et vous savez expliquer chaque fichier en entretien. Ce profil GitHub n’est plus celui d’un étudiant. C’est celui d’un ingénieur junior.

---

## ÉTAPE 3 — OPÉRATIONS (Semaines 21-28)

**L’idée de cette étape :** construire des systèmes vous fait embaucher ; les *faire tourner* est le vrai métier. Les points de description de poste auxquels cette étape répond sont la moitié opérationnelle — « surveiller la santé de l’infrastructure », « participer à la réponse aux incidents, y compris l’analyse de logs », « identifier, analyser et résoudre les vulnérabilités de l’infrastructure », « gérer les coûts cloud ». La semaine d’un ingénieur cloud, c’est surtout cette étape.

### Module 9 (Semaines 21-24) : Supervision, logs et incidents

**La supervision — savoir avant les utilisateurs.** CloudWatch est le service d’observabilité intégré d’AWS. Trois primitives : les **métriques** (des nombres dans le temps — % CPU, % disque, nombre de requêtes, nombre d’erreurs), les **alarmes** (une règle surveillant une métrique : « si CPU > 80 % pendant 5 minutes → notifier »), et les **tableaux de bord** (des métriques disposées sur un seul écran). Les notifications passent par **SNS** (Simple Notification Service — un topic sur lequel vous publiez, les abonnés reçoivent e-mails/alertes). L’art est de savoir *sur quoi* alerter : ne réveillez un humain que pour ce qui exige un humain, sinon les gens apprennent à ignorer le pager (la **fatigue d’alerte (alert fatigue)** — le mode de défaillance qui a précédé bien des pannes célèbres). Les quatre signaux d’or à mémoriser : **latence, trafic, erreurs, saturation** — à quel point c’est lent, occupé, cassé, plein.

**Les logs — savoir pourquoi.** Les métriques disent que quelque chose ne va pas ; les logs disent ce qui s’est passé. **CloudWatch Logs** les centralise : un **agent** sur chaque instance expédie des fichiers comme `/var/log/nginx/access.log` vers des **groupes de logs (log groups)**, où **Logs Insights** vous permet de requêter à travers les machines (« compter les

réponses 5xx par minute sur la dernière heure »). La centralisation compte parce que les serveurs sont désormais jetables (vous l'avez prouvé au Module 8) — les logs doivent survivre aux machines qui les ont écrits.

**Lab (Semaines 21–22)** : montez un petit serveur web supervisé, entièrement en Terraform (votre pile du Module 7, étendue) : EC2 + nginx + l'agent CloudWatch ; un topic SNS qui vous envoie des e-mails ; une alarme sur CPU élevé et une autre sur l'échec du status check de l'instance. Puis attaquez-vous vous-même : connectez-vous en SSH et lancez un brûleur de CPU (`yes > /dev/null` & plusieurs fois) ; regardez la métrique grimper, l'alarme se déclencher, l'e-mail arriver ; tuez les processus ; regardez la récupération. Puis requêtez vos propres logs d'accès dans Logs Insights. Vous venez d'assister à la boucle complète détecter—notifier—diagnostiquer—résoudre sur un système que vous avez construit.

**La réponse aux incidents — la moitié humaine.** Un **incident** est un « le système ne va pas bien » non planifié ; les niveaux de **sévérité** (sev-1 = clients à terre, tout le monde sur le pont) fixent l'échelle de la réponse. La boucle professionnelle : **détecter** (l'alarme, pas un e-mail de client, idéalement) — **trier (triage)** (à quel point c'est grave, qui est nécessaire) — **mitiger** (arrêter l'hémorragie *d'abord* — rollback, redémarrage, failover ; la cause racine vient plus tard) — **résoudre** — **post-mortem** : une revue écrite *sans blâme* de ce qui s'est passé, pourquoi, et ce qui empêchera une récurrence. Sans blâme n'est pas de la mollesse ; c'est de l'ingénierie : les gens punis cachent l'information, et l'information cachée cause les pannes récurrentes. **L'astreinte (on-call)** est la rotation de qui porte le pager ; les descriptions de poste la mentionnent, les recruteurs posent des questions dessus, et votre réponse honnête après ce module est « je l'ai simulée et je connais la boucle ».

**Les runbooks — le point documentation, rendu concret.** Un runbook est une recette pas à pas pour une situation opérationnelle, écrite pour qu'une personne stressée à 3 heures du matin puisse la suivre : symptômes — vérifications (commandes exactes) — correctifs (commandes exactes) — escalade (qui réveiller si ça ne marche pas). **Lab (Semaines 23–24)** : écrivez deux runbooks dans votre dépôt — « serveur web à terre » et « disque qui se remplit » — puis *entraînez-les* : cassez la chose, suivez votre propre document à la lettre, et corrigez chaque étape qui s'est révélée vague. Puis lancez un **game day** complet : demandez à un ami (ou à une IA) de casser votre pile de lab en secret ; vous êtes alerté, vous diagnostiquez à partir des métriques et des logs, vous mitigez, et vous rédigez le post-mortem dans votre journal. Ce post-mortem est une histoire d'entretien, et une bonne.

## Vocabulaire :

| Terme                               | Définition                                                                                |
|-------------------------------------|-------------------------------------------------------------------------------------------|
| CloudWatch                          | Le service de supervision d'AWS : métriques, alarmes, tableaux de bord, logs.             |
| Métrique / alarme / tableau de bord | Un nombre dans le temps / une règle qui se déclenche dessus / un écran qui les rassemble. |
| SNS                                 | Le service de notification vers lequel les alarmes publient — e-mail, SMS, pager.         |
| Signaux d'or                        | Latence, trafic, erreurs, saturation — les                                                |

| Terme                          | Définition                                                                                                            |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------|
|                                | quatre nombres qui décrivent la santé de n'importe quel service.                                                      |
| Fatigue d'alerte               | Trop d'alertes non actionnables → alertes ignorées → vraies alertes manquées.<br>L'ennemi de la conception d'alarmes. |
| Groupe de logs / Logs Insights | Là où atterrissent les logs centralisés / le langage de requête au-dessus d'eux.                                      |
| Incident / sévérité            | Une dégradation non planifiée / sa gravité classée (sev-1 = la pire).                                                 |
| Triage / mitiger / résoudre    | Évaluer vite / arrêter l'hémorragie d'abord / réparer vraiment.                                                       |
| Post-mortem                    | La revue écrite sans blâme : quoi, pourquoi, ce qui empêche une récurrence.                                           |
| Runbook                        | La recette à l'épreuve de 3 h du matin pour une situation : symptômes, vérifications, correctifs, escalade.           |
| Astreinte / game day           | La rotation du pager / un incident d'entraînement délibéré.                                                           |
| MTTR                           | Mean time to recovery (temps moyen de rétablissement) — la métrique d'exploitation que les équipes mûres optimisent.  |

**À dire à voix haute :** “Did we find out from the alarm or from a customer?” (« L'avons-nous appris par l'alarme ou par un client ? ») · “Mitigate first — root cause after we're stable.” (« Mitigez d'abord — la cause racine après la stabilisation. ») · “Is there a runbook for this? There will be by tomorrow.” (« Y a-t-il un runbook pour ça ? Il y en aura un d'ici demain. ») · “What did the post-mortem conclude, and what's the prevention item?” (« Qu'a conclu le post-mortem, et quelle est l'action de prévention ? »)

**Jalon :** votre post-mortem de game day existe, est sans blâme, et nomme une prévention concrète que vous avez ensuite réellement mise en œuvre (une alarme ajoutée, un runbook corrigé).

## Module 10 (Semaines 25–28) : Opérations de sécurité et gestion des coûts

**Les opérations de sécurité — de l'hygiène, pas de l'héroïsme. Le modèle de responsabilité partagée** d'abord (à revoir : <https://www.youtube.com/watch?v=ESPBBEK-cvo>) : AWS sécurise le cloud lui-même ; vous sécurisez ce que vous y mettez — et la plupart des vraies brèches sont des erreurs de configuration client, pas des défaillances d'AWS. Votre checklist de sécurité opérationnelle, pratiquée jusqu'à l'ennui :

1. **Hygiène IAM** (mensuelle) : MFA partout ; pas de clés d'accès de longue durée là où un rôle suffirait ; lancez le credential report ; supprimez les utilisateurs et clés inutilisés ;

questionnez chaque \* dans chaque politique. Vous avez construit ce réflexe au Lab 5 — c'est maintenant une entrée de calendrier.

2. **Les correctifs (patching)** : le logiciel non corrigé est le point de départ de la plupart des intrusions. Sur vos instances : appliquez les mises à jour selon un calendrier (et sachez que **SSM Patch Manager** automatise cela à l'échelle du parc) ; dans le monde des conteneurs, patcher signifie souvent reconstruire l'image depuis une base à jour — reliez cela au Module 8.
3. **Des sauvegardes — testées** : une sauvegarde non testée est un espoir, pas un plan. Snapshots RDS, **versioning** S3 (chaque écrasement garde l'ancienne version — le contrôle anti-oups et anti-ransomware). Lab : snapshotez votre base de lab, *restaurez-la* vers une nouvelle instance, vérifiez les lignes, démontez. Vous pouvez désormais dire « testé » en entretien.
4. **Les garde-fous** : activez **CloudTrail** (le journal d'audit de chaque appel d'API du compte — qui a fait quoi, quand) et parcourez une fois les événements d'hier ; activez l'essai gratuit de **GuardDuty** (détection de menaces automatisée) et lisez ce qu'il surveille. Connaissez la formule **chiffrement au repos et en transit** et sachez que dans AWS c'est surtout une case à cocher adossée à **KMS** — le minimum vital, toujours activé.

**La gestion des coûts — la compétence qui fait paraître les juniors seniors.** Le point de description de poste est mot pour mot : « *gérer les coûts cloud par le rightsizing des ressources, la mise en place de l'auto-scaling, l'étiquetage des ressources* ». Comment tourne le compteur : le calcul facture à la seconde où il est *allumé* (inactif ≠ gratuit — « on l'a laissé tourner » est le gaspillage classique), le stockage au Go-mois, et l'**egress** (les données *sortant* d'AWS) au Go tandis que les données entrantes sont gratuites — la célèbre surprise de facture. Les leviers, dans l'ordre où un junior peut les actionner : **éteindre** (les systèmes de dev la nuit ; votre habitude de teardown, industrialisée) → **rightsizer** (la plupart des serveurs sont surdimensionnés ; consultez l'historique CPU de CloudWatch et réduisez) → **tout tagger** (Project, Owner, Environment — la dépense non taguée est une dépense sans responsable ; imposez les tags dans votre Terraform) → **hiérarchiser le stockage** (les règles de cycle de vie du Lab 2) → savoir que la **capacité réservée/les Savings Plans** (engagement 1–3 ans, économie de 30–70 %) et le **Spot** (jusqu'à 90 % de réduction, interruptible) existent respectivement pour les charges stables et les traitements par lots. Les outils : **Cost Explorer** (la dépense du compte, en graphiques — à lire chaque semaine, pour toujours) et **AWS Budgets** (votre alerte à 0 \$ de la Semaine 3, désormais comprise comme le plus petit membre d'une famille sérieuse). La discipline plus large s'appelle le **FinOps** — deux minutes qui en valent la peine : [https://www.youtube.com/watch?v=Y-c\\_xw9bHFW](https://www.youtube.com/watch?v=Y-c_xw9bHFW).

## Vocabulaire :

| Terme                             | Définition                                                                                                         |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------|
| Modèle de responsabilité partagée | AWS sécurise le cloud ; vous sécurisez ce que vous y mettez. La plupart des brèches relèvent de la seconde moitié. |
| CloudTrail                        | Le journal d'audit du compte : chaque appel d'API, par qui, quand.                                                 |

| Terme                                    | Définition                                                                                                                          |
|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| GuardDuty                                | La détection de menaces automatisée d’AWS sur les logs et le trafic.                                                                |
| Patching / SSM                           | Appliquer les mises à jour de sécurité / le Systems Manager d’AWS, qui l’automatise à l’échelle du parc.                            |
| Versioning S3                            | Garder chaque version écrasée — le contrôle anti-oups, anti-ransomware.                                                             |
| KMS / chiffrement au repos et en transit | Le service de clés d’AWS / les données brouillées sur disque et sur le réseau. Toujours activé.                                     |
| Egress                                   | Les données quittant AWS — facturées au Go ; l’entrant est gratuit. La surprise de facture classique.                               |
| Rightsizing                              | Réduire les ressources surdimensionnées au besoin mesuré. De l’argent gratuit.                                                      |
| Tagging (étiquetage)                     | Étiqueter chaque ressource avec propriétaire/projet/environnement pour que toute dépense soit attribuable.                          |
| Savings Plans / Spot                     | Engagement de 1–3 ans pour 30–70 % de réduction sur la charge stable / capacité inutilisée récupérable jusqu’à –90 % pour les lots. |
| Cost Explorer / Budgets                  | Les graphiques de dépense que vous lisez chaque semaine / les alertes qui font que vous n’apprenez jamais par la facture.           |
| FinOps                                   | La discipline qui rend la dépense cloud visible, allouée et continuellement optimisée.                                              |

**À dire à voix haute :** “Who has access to prod, and when did we last review the list?” (« Qui a accès à la prod, et quand avons-nous revu la liste pour la dernière fois ? ») · “When did we last *restore* a backup?” (« Quand avons-nous *restauré* une sauvegarde pour la dernière fois ? ») · “What’s untagged, and what died but is still billing?” (« Qu’est-ce qui n’est pas tagué, et qu’est-ce qui est mort mais facture encore ? ») · “It’s over-provisioned — the CPU history says we can halve it.” (« C’est surprovisionné — l’historique CPU dit qu’on peut le réduire de moitié. »)

**Exercices :** (1) Le lab de test de restauration ci-dessus. (2) Un fichier de checklist de sécurité mensuelle dans votre dépôt, puis exécutez-la réellement contre votre compte. (3) Dans Cost Explorer, trouvez la chose la plus coûteuse de l’historique de votre compte et expliquez-la en une phrase de journal. (4) Ajoutez des tags par défaut à chaque ressource de votre dépôt Terraform.

**Jalon — fin de l’Étape 3 :** menez un auto-audit complet — checklist de sécurité, revue des coûts, test d’alarme, restauration de sauvegarde — et rédigez le rapport d’une page. Vous savez

désormais faire le *métier*, pas seulement la construction. Ce qui reste, c'est le prouver à des inconnus : l'Étape 4.

---

## ÉTAPE 4 — PRÉPARATION À L'EMPLOI (Mois 8-12)

### Module 11 (Mois 8-10) : Trois projets de portfolio

Les certifications disent que vous avez étudié ; les projets prouvent que vous savez construire. Trois, chacun dans son propre dépôt GitHub, chacun avec un diagramme d'architecture, un README écrit pour un responsable du recrutement (quoi, pourquoi, comment le lancer, ce que ça coûte) et un script de teardown. Construire — capturer/enregistrer — *détruire* — le dépôt est l'artefact, pas une facture qui tourne.

#### Projet 1 — Application web trois tiers, entièrement automatisée (la pièce maîtresse).

Terraform construit tout : VPC avec sous-réseaux publics/privés sur deux AZ ; un **Application Load Balancer** (équilibreur de charge applicatif — le dispositif qui répartit le trafic, nouveau pour vous, et la suite naturelle du Lab 3) devant un **Auto Scaling Group** d'instances web (AWS ajoute/retire des instances selon la charge — renseignez-vous, câblez-le) ; RDS dans les sous-réseaux privés ; S3 pour les ressources statiques ; alarmes CloudWatch et un runbook. GitHub Actions exécute `terraform plan` sur chaque pull request et `apply` à la fusion — un vrai pipeline d'infrastructure. Ce seul projet démontre dix des quatorze points de description de poste du tableau de correspondance ; attendez-vous à ce que chaque entretien le parcoure, et répétez sa narration en cinq minutes.

**Projet 2 — Pipeline de données serverless (l'étendue).** Aucun serveur du tout : un fichier atterrissant dans un bucket S3 déclenche une **fonction Lambda** (votre Python, exécuté par AWS par événement, facturé à l'invocation) qui le traite — parser un CSV, le résumer, écrire les résultats dans un second bucket ou une table DynamoDB — avec les échecs interceptés, loggés dans CloudWatch, et alarmés vers votre boîte mail. Déployez-le avec Terraform. Il montre Python, la pensée événementielle, et une envergure au-delà d'EC2. Deux minutes d'orientation d'abord : [https://www.youtube.com/watch?v=W\\_VV2Fx32\\_Y](https://www.youtube.com/watch?v=W_VV2Fx32_Y).

**Projet 3 — Vitrine d'exploitation façon production (le différenciateur).** Prenez le Projet 1 et exploitez-le comme si ça comptait : un tableau de bord CloudWatch des signaux d'or ; des alarmes avec une politique d'alerte documentée ; trois runbooks ; une procédure de sauvegarde-restauration testée avec ses preuves ; une passe de durcissement sécurité (audit IAM, notes de patching, CloudTrail activé) rédigée ; une analyse de coûts (« cette pile coûte X \$/mois ; voici les trois changements qui la réduiraient de moitié ») ; et un post-mortem de game day. Presque aucun candidat junior n'a cela. Il répond à la seule question que posent vraiment les entretiens — « peut-on confier la production à cette personne ? » — avec des documents au lieu d'adjectifs.

## Module 12 (Mois 8–12, en parallèle) : Le calendrier des certifications

Les certifications ouvrent les filtres des recruteurs et structurent votre étude. Le parcours standard 2026 pour ce poste, avec une préparation réaliste au rythme de ce cours — notez comment chaque examen tombe juste après les étapes du cours qui enseignent son contenu, raison pour laquelle ces durées sont plus courtes que celles d'internet :

| Ordre | Certification                                      | Ce qu'elle prouve                                                                                         | Temps de préparation depuis là où vous serez                                                                                                                                                                                                                         | Quand la passer |
|-------|----------------------------------------------------|-----------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| 1     | <b>AWS Certified Cloud Practitioner (CLF-C02)</b>  | Vocabulaire cloud, facturation, responsabilité partagée                                                   | 3–4 semaines (les guides disent la même chose pour les débutants ; les Étapes 0–1 en couvrent l'essentiel)                                                                                                                                                           | ~ Mois 4        |
| 2     | <b>AWS Solutions Architect Associate (SAA-C03)</b> | Concevoir de vraies architectures AWS — le titre sur lequel les descriptions de poste filtrent réellement | 6–8 semaines de préparation ciblée (estimation standard de l'industrie après le CCP)                                                                                                                                                                                 | Mois 8–9        |
| 3     | <b>HashiCorp Terraform Associate (003)</b>         | La fluidité IaC, vérifiée                                                                                 | 2–3 semaines — après le Module 7 et le Projet 1, vous êtes surtout en révision (préparation officielle : <a href="https://developer.hashicorp.com/terraform/tutorials/certification-003">https://developer.hashicorp.com/terraform/tutorials/certification-003</a> ) | Mois 10–11      |

Quatrième optionnelle, si vous visez des postes à intitulé opérationnel : **AWS SysOps Administrator / CloudOps Associate** — son contenu est littéralement l'Étape 3. Pour le CLF-C02, le cours complet de freeCodeCamp (<https://www.youtube.com/watch?v=7HKot-brXFE>, édition classique <https://www.youtube.com/watch?v=NhDYbskXRgc>) plus des examens blancs

est le chemin gratuit bien balisé ; pour le SAA-C03, associez un cours à de nombreux tests blancs chronométrés — l'examen est basé sur des scénarios et l'endurance compte.

## Module 13 (Mois 11-12) : CV, entretiens et les dix questions

**Le CV :** une page. Commencez par les compétences (AWS, Terraform, Python/Bash, Docker, CI/CD, CloudWatch — reprenez les mots mêmes de la description de poste ; les filtres automatiques matchent les mots-clés) et les trois projets, chacun en deux puces de la forme *a fait X avec Y en obtenant Z* : « Déployé une application web trois tiers sur AWS avec Terraform et GitHub Actions ; déploiements sans interruption via ALB + Auto Scaling ». Listez les certifications avec les dates. Mettez le lien GitHub — puis partez du principe qu'ils vont vraiment l'ouvrir, car les bons le font, et le vôtre récompense désormais la visite.

**La préparation aux entretiens :** trois saveurs à répéter — les questions de connaissance (les tableaux de vocabulaire des Étapes), les scénarios (les labs que vous avez réellement faits) et le comportemental (les histoires de votre journal, racontées en forme **STAR** : Situation, Tâche, Action, Résultat). Entraînez-vous à voix *haute*, quotidiennement, pendant deux semaines — idéalement avec un intervieweur IA, sans pitié. Les dix questions ci-dessous couvrent les classiques ; les exemples de réponses sont délibérément compacts — développez chacun avec vos propres détails de lab, car « ...et quand j'ai construit ceci, ce qui s'est réellement passé, c'est... » est la phrase qui vous sépare des candidats qui n'ont fait que lire.

### Les dix questions, avec des réponses solides :

1. **“Explain the difference between a public and a private subnet.”** (« Expliquez la différence entre un sous-réseau public et un sous-réseau privé. ») *La table de routage d'un sous-réseau public a une route vers une internet gateway, donc ses ressources peuvent être atteintes depuis l'internet ; un sous-réseau privé n'a pas cette route. Les niveaux web vont en public, les bases de données en privé, et l'accès admin atteint les machines privées via un bastion ou SSM. Dans mes labs, j'ai prouvé que l'instance privée était injoignable depuis mon laptop mais joignable depuis le niveau public — la route manquante, c'est la sécurité.*
2. **“What is IAM, and what is least privilege?”** (« Qu'est-ce qu'IAM, et qu'est-ce que le moindre privilège ? ») *IAM contrôle quelles identités peuvent effectuer quelles actions sur quelles ressources, via des utilisateurs, des groupes, des rôles et des politiques JSON. Le moindre privilège signifie accorder le minimum qui fait le travail — l'utilisateur de mon script d'audit a un accès en lecture seule à exactement un service, et mes instances EC2 utilisent des rôles au lieu de clés stockées, donc il n'y a aucun identifiant de longue durée à faire fuiter.*
3. **“An EC2 web server stopped responding. Walk me through your troubleshooting.”** (« Un serveur web EC2 a cessé de répondre. Décrivez-moi votre dépannage. ») *D'abord le périmètre : un serveur ou tout — vérifier le tableau de bord CloudWatch et les alarmes. Puis les couches dans l'ordre : les status checks de l'instance ; le groupe de sécurité — 80/443 est-il réellement ouvert ; SSH — nginx tourne-t-il (`systemctl status`), le disque est-il plein (`df -h`), le CPU est-il saturé (`top`) ; puis les logs (`tail` sur le log d'erreurs). Mitiger d'abord — redémarrer le service ou remplacer l'instance — et faire la cause racine une fois stabilisé. C'est l'ordre que je répète dans mes runbooks.*

4. **“Why Terraform instead of clicking in the console?”** (« **Pourquoi Terraform plutôt que cliquer dans la console ?** ») *La console ne laisse pas de trace relisible et ne sait rien reconstruire. Terraform est déclaratif et versionné : les changements passent par des pull requests, la même config construit des environnements identiques, un mauvais changement se rétablit en annulant un commit, et plan montre le diff exact avant que quoi que ce soit n’arrive. Mon pipeline exécute plan sur chaque PR pour que le diff fasse partie de la revue.*
5. **“What’s the difference between a container and a virtual machine?”** (« **Quelle est la différence entre un conteneur et une machine virtuelle ?** ») *Une VM virtualise le matériel et embarque un OS complet, mettant des minutes à démarrer ; un conteneur partage le noyau de l’hôte et n’emballa que l’application et ses dépendances, démarrant en une seconde environ. Les conteneurs donnent un comportement identique d’un environnement à l’autre — la même image tourne sur mon laptop et sur EC2. Les VM comptent encore pour l’isolation et pour les charges non-Linux.*
6. **“How would you reduce our AWS bill?”** (« **Comment réduiriez-vous notre facture AWS ?** ») *Par ordre d’effort : trouver ce qui tourne et ne devrait pas — instances inactives, volumes non attachés, vieux snapshots ; Cost Explorer et les tags rendent cela visible. Puis rightsizer d’après l’historique CloudWatch. Puis programmer l’extinction des environnements de dev hors des heures ouvrées. Puis faire vieillir les données S3 vers des niveaux moins chers. Puis engager les charges stables sur des Savings Plans et mettre les lots interruptibles sur du Spot. Et surveiller l’egress — les données sortantes sont la ligne surprise classique.*
7. **“What happens when you type a URL and press Enter?”** (« **Que se passe-t-il quand vous tapez une URL et appuyez sur Entrée ?** ») *Le DNS résout le nom en IP ; le navigateur ouvre une connexion TCP vers le port 443 et négocie TLS ; il envoie un GET HTTP ; de l’autre côté, cela atteint un équilibreur de charge, qui transmet à une instance saine dans un sous-réseau public ; l’application peut interroger une base dans un sous-réseau privé ; la réponse revient avec un code de statut — 200 si tout va bien — et le navigateur l’affiche. Je peux creuser n’importe quel maillon, et j’ai construit chacun d’eux.*
8. **“A teammate’s change broke production. What happens next?”** (« **Le changement d’un collègue a cassé la production. Que se passe-t-il ensuite ?** ») *Mitiger d’abord — rollback via le pipeline ; sur le moment, la récupération bat le diagnostic. Puis un post-mortem sans blâme : ce qui s’est passé, pourquoi le système l’a permis, et quel garde-fou manquait — un test, une revue de plan, une alarme. Le sans-blâme compte concrètement : les gens qui craignent la punition cachent l’information, et l’information cachée cause la récurrence.*
9. **“What is the shared responsibility model?”** (« **Qu’est-ce que le modèle de responsabilité partagée ?** ») *AWS sécurise le cloud — centres de données, matériel, hyperviseur ; le client sécurise ce qui s’y trouve — données, IAM, règles réseau, correctifs, paramètres de chiffrement. La plupart des vraies brèches cloud sont côté client, des erreurs de configuration comme des buckets publics ou des clés fuitées. Donc « AWS est sécurisé » et « nous sommes sécurisés sur AWS » sont deux phrases différentes, et la seconde est mon travail.*
10. **“Tell me about a problem you debugged.”** (« **Parlez-moi d’un problème que vous avez débogué.** ») *(Le vôtre, tiré du journal, en forme STAR. Squelette d’exemple :)*

*Pendant mon game day de supervision, mon alarme s'est déclenchée pour saturation CPU (S). Je devais trouver et arrêter la cause avant que les « utilisateurs » ne s'en aperçoivent (T). Les métriques montraient le début du pic ; Logs Insights l'a relié à des processus emballés ; je me suis connecté en SSH, j'ai confirmé avec top, je les ai tués, et j'ai suivi la récupération sur le tableau de bord (A). Résolu en onze minutes, et le post-mortem a produit une alarme sur le nombre de processus que j'ai ensuite implémentée dans Terraform (R).*

**La mécanique de la recherche d'emploi :** candidatez dès le Mois 11 — après le SAA, n'attendez pas d'être « prêt », car les entretiens sont de l'entraînement. Intitulés cibles : cloud engineer (junior/associate), cloud support engineer, cloud operations engineer, junior DevOps engineer, AWS support associate. Chaque refus avec entretien est une leçon gratuite ; consignez ce qu'ils ont demandé.

**Vocabulaire :**

| Terme                                       | Définition                                                                                                                 |
|---------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Portfolio                                   | La preuve publique et documentée que vous savez construire — pour ce métier, des dépôts GitHub avec diagrammes et README.  |
| Application Load Balancer (ALB)             | L'équilibreur de charge d'AWS : répartit les requêtes entre instances saines, écarte les malades.                          |
| Auto Scaling Group                          | Maintient N instances en vie et ajuste N selon la charge — auto-réparation et élasticité en un.                            |
| Lambda / serverless                         | Du code qu'AWS exécute par événement, facturé à l'invocation — aucun serveur à gérer du tout.                              |
| DynamoDB                                    | La table NoSQL serverless d'AWS — se marie naturellement avec Lambda.                                                      |
| STAR                                        | Situation, Tâche, Action, Résultat — la forme d'une bonne histoire d'entretien.                                            |
| ATS                                         | Applicant tracking system (système de suivi des candidatures) — le filtre à mots-clés que votre CV d'une page doit passer. |
| CLF-C02 / SAA-C03 / Terraform Associate 003 | Vos trois examens : vocabulaire, architecture, IaC.                                                                        |
| SysOps / CloudOps Associate                 | L'associate AWS optionnel orienté opérations — l'Étape 3, en examen.                                                       |

**Jalon — fin du cours :** trois dépôts qu'un inconnu peut comprendre, trois certifications planifiées ou obtenues, dix réponses répétées à voix haute, des candidatures en cours. Vous n'êtes pas en train d'« espérer entrer dans le cloud ». Vous êtes un ingénieur cloud junior avec des preuves, en entretien.

## Le programme en une page

| Quand          | Focus                                                                    | Preuve pratique                                          | Preuve externe              |
|----------------|--------------------------------------------------------------------------|----------------------------------------------------------|-----------------------------|
| Semaines 1–2   | Comment fonctionnent les ordinateurs ; Linux, terminal, permissions, SSH | Opérations sur fichiers scriptées ; premier script shell | —                           |
| Semaines 3–4   | Réseau : IP, DNS, ports, HTTP, pare-feu ; compte AWS                     | Compte + MFA + budget à 0 \$                             | —                           |
| Semaines 5–6   | Lab 1 : EC2                                                              | Serveur web sur l'internet, démonté                      | —                           |
| Semaine 7      | Lab 2 : S3 + CLI                                                         | Site statique ; fluidité CLI                             | —                           |
| Semaines 8–9   | Lab 3 : VPC                                                              | Réseau public/privé, preuve du bastion                   | —                           |
| Semaine 10     | Lab 4 : RDS                                                              | Base de données privée, snapshot, teardown               | —                           |
| Semaines 11–12 | Lab 5 : IAM ; reconstruction du gantelet de l'Étape 1                    | Pile complète de mémoire, <3 h                           | Planifier le CCP            |
| Semaines 13–15 | Bash, Python/boto3, Git                                                  | backup.sh, audit.py, historique du dépôt                 | <b>Examen CCP (~Mois 4)</b> |
| Semaines 16–17 | Terraform                                                                | Pile en code, plan/apply/destroy                         | —                           |
| Semaines 18–20 | CI/CD, Docker, culture K8s                                               | Pipeline vert ; image dans un registre                   | —                           |
| Semaines 21–24 | CloudWatch, logs, incidents, runbooks                                    | Alarme déclenchée + post-mortem de game day              | —                           |
| Semaines 25–28 | Sécurité opérationnelle ; gestion des coûts                              | Rapport d'auto-audit ; test de restauration              | —                           |
| Mois 8–10      | Projets de portfolio 1–3                                                 | Trois dépôts documentés                                  | <b>SAA-C03 (Mois 8–9)</b>   |
| Mois 10–11     | Préparation Terraform Associate                                          | —                                                        | <b>Terraform Associate</b>  |
| Mois 11–12     | CV, entretiens,                                                          | Dix réponses, à voix                                     | Premières offres            |

| Quand | Focus        | Preuve pratique | Preuve externe |
|-------|--------------|-----------------|----------------|
|       | candidatures | haute           |                |

**Un mot de clôture de votre professeur.** Douze mois, c'est court pour une reconversion et long pour une habitude quotidienne, alors voici le marché honnête : les gens qui terminent ce cours ne sont pas les plus intelligents — ce sont ceux qui ont tapé les commandes les jours de fatigue aussi. Chaque message d'erreur que vous rencontrez est le programme qui fonctionne, pas qui échoue ; l'ingénieur qui a cassé et réparé une centaine de petites choses est exactement ce qu'un responsable du recrutement entend par « expérience ». Tenez le journal, démontez ce que vous construisez, ne prétendez jamais en entretien ce que vos dépôts ne peuvent pas prouver — et dans un an, quand le pager sonnera à 3 heures du matin, vous sentirez quelque chose d'inattendu sous l'adrénaline : de la compétence. Allez construire.

---

## Sources

Recherche sur les descriptions de poste (août 2026) : [Arc.dev — AWS Cloud Engineer Job Description](#) · [Wiz — Cloud Engineer Job Description Guide](#) · [DevsData — AWS Cloud Engineer JD Template](#) · [XOPA — Cloud Engineer JD Template 2026](#) · [Betterteam — Cloud Engineer Job Description](#). Parcours de certification et temps de préparation : [StudyTech — AWS Certification Roadmap 2026](#) · [Cloud Evolvers — Cloud Engineer Roadmap 2026](#) · [HashiCorp — Terraform Associate 003 prep](#). Contexte salarial (médiane US ≈ 104 K–140 K\$) : Wiz, ci-dessus. Tous les liens YouTube vérifiés via les métadonnées YouTube au moment de la rédaction.

*Un cours B4LCILC — volume compagnon du « Cours du Leader Cloud ».*

---

## NOUVELLES LEÇONS — ÉDITION D'AOÛT 2026

*En août 2026, de nouvelles leçons ont été ajoutées aux trois cours interactifs de [b4lcilc.org](#) — les cours **Leader Cloud**, **Ingénieur Cloud** et **Architecte Cloud**. Cet addendum met l'édition de lecture à jour avec des condensés fidèles de ce nouveau matériel, dans la même voix que le reste de ce livre.*

*Un mot sur ce que vous tenez entre les mains. Ces pages sont l'édition de lecture : les idées, les métaphores, et les phrases qui valent la peine d'être dites à voix haute. Les versions interactives complètes en ligne contiennent considérablement plus — chaque nouvelle leçon est livrée avec des quiz qui vérifient votre compréhension avant de vous laisser avancer, des laboratoires pratiques (dont un terminal d'entraînement qui se comporte comme le vrai mais ne peut pas coûter d'argent), des vidéos, des exercices de vocabulaire, et — pour le matériel de synthèse — des exercices de conception guidés avec des listes de contrôle des exigences et des corrigés types. Lisez ces condensés pour installer la forme de chaque idée dans votre tête ; puis, quand vous le pourrez, passez les versions interactives pour ancrer la connaissance.*

*Les ajouts se répartissent en trois groupes. Pour le cours **Leader Cloud** : une introduction en douceur au terminal, la loi thaïlandaise sur la vie privée, un guide du leader face aux propositions d'IA et aux factures d'IA, et un exercice de synthèse qui assemble le cours entier sur*

*une page. Pour le cours **Ingénieur Cloud** : l'informatique serverless — le modèle qui transforme les petits scripts d'exploitation en fonctions qui ne coûtent rien à exécuter. Pour le cours **Architecte Cloud** : l'arithmétique des promesses de disponibilité, le modèle de génération augmentée par récupération (Retrieval-Augmented Generation) sur lequel les clients posent désormais des questions chaque semaine, et un exercice complet de chiffrage — estimer le prix d'une conception avant que quiconque ne la construise.*

---

## Pour le cours Ingénieur Cloud

### Ce qu'est réellement le serverless

**Tout ce que vous avez automatisé jusqu'ici suppose encore une machine** : le cron job du laboratoire de ce module s'exécute deux minutes par jour sur une machine virtuelle que vous payez tout le mois, que vous mettez à jour, que vous surveillez, et qu'un jour vous déboguerez à 3 h du matin. Le serverless est la réponse du cloud à ce décalage. Il y a toujours des serveurs — le nom est du marketing — mais ils sont le problème du fournisseur. Vous remettez une fonction, le fournisseur l'exécute quand quelque chose la déclenche, et vous payez à l'invocation, à la milliseconde. Le modèle s'appelle **FaaS — Function as a Service** (fonction en tant que service) : **Lambda** sur AWS, **Azure Functions** sur Azure. Les concepts se transfèrent nom pour nom.

**L'anatomie tient dans une main.** Un **handler** est la fonction que la plateforme appelle, et il reçoit un **événement (event)** — une description **JSON** (JavaScript Object Notation) de ce qui vient de se passer : un fichier a atterri dans un bucket, une requête HTTP est arrivée, une planification s'est déclenchée. Redressez-vous à la dernière : « une planification s'est déclenchée » est exactement ce que fait cron — sauf qu'une Lambda planifiée n'a besoin d'aucune machine allumée entre les exécutions. Convertir l'un des scripts d'exploitation de ce module est un premier projet serverless classique.

**L'économie est la raison d'être du modèle, et elle coupe dans les deux sens.** Le niveau gratuit d'AWS inclut un million d'invocations Lambda par mois, pour toujours — votre fonction de laboratoire pourrait s'exécuter chaque minute de l'année sans rien coûter, faisant du serverless le foyer le moins cher pour la glu opérationnelle. Mais une tarification à la milliseconde qui est une erreur d'arrondi à faible volume devient de l'argent réel à haut volume ; une fonction qui tourne en permanence est habituellement moins chère en container ou en VM. La compétence professionnelle, c'est de savoir de quel côté de cette ligne se trouve un workload.

**Deux mises en garde honnêtes.** D'abord, le **démarrage à froid (cold start)** : une fonction qui n'a pas tourné récemment met quelques centaines de millisecondes à se réveiller — sans importance pour un rapport nocturne, une considération de conception pour une API sensible à la latence. Ensuite, une fonction s'exécute toujours sous une identité : Lambda assume un rôle IAM (Identity and Access Management — gestion des identités et des accès) exactement comme le fait une instance EC2, et une fonction sur-permissionnée est la même erreur de sécurité dans un emballage plus petit.

---