

Le Cours d'Architecte Cloud (The Cloud Architect Course)

De zéro connaissance à la conception de systèmes cloud — un cours B4LCILC

13 août 2026

Le Cours d'Architecte Cloud

De zéro connaissance à la conception de systèmes cloud sur lesquels une entreprise miserait son activité.

Comment fonctionne ce cours

Vous ne vous formez pas pour *exploiter* des systèmes cloud. Vous vous formez pour les **concevoir** : prendre un problème métier confus (« nous devons vendre à un million de clients et ne jamais perdre une commande »), le transformer en un plan technique dessiné, chiffré et défendable, puis convaincre à la fois les ingénieurs qui le construiront et les dirigeants qui le paieront. C'est le métier d'**architecte cloud** (aussi affiché comme *solutions architect*, *cloud solutions architect* ou *cloud domain architect*), et c'est un artisanat qui s'apprend avec un programme clair — celui-ci.

Le cours comporte cinq étapes. **Étape 1 (Semaines 1–6) : Les briques de base, en tant que décisions.** Chaque composant du cloud, ré-enseigné non comme un fait mais comme un *choix avec des compromis* — car l'unité de travail de l'architecte est la décision. **Étape 2 (Semaines 7–14) : Les domaines de conception.** Les six piliers du Well-Architected Framework — fiabilité, sécurité, performance, coûts, opérations, durabilité — chacun enseigné en profondeur avec ses motifs standards et une conception travaillée. **Étape 3 (Semaines 15–20) : Le catalogue de patterns.** La douzaine d'architectures qui résolvent 95 % des problèmes réels, et — plus important encore — quand ne *pas* utiliser chacune. **Étape 4 (Semaines 21–26) : La migration et le monde réel.** Faire migrer des entreprises existantes vers le cloud, et les contraintes (la loi, le legacy, le lock-in) qui rendent l'architecture réelle plus difficile que l'architecture de tableau blanc. **Étape 5 (Mois 7–12) : L'artisanat.** Les documents, diagrammes, revues, présentations et certifications qui vous rendent *employable* comme architecte, couronnés par trois projets de conception dignes d'un portfolio.

À qui s'adresse ce cours. Il suppose zéro connaissance préalable : tout ce dont il dépend est brièvement ré-enseigné avant d'être utilisé. Si vous avez déjà terminé **Le Cours d'Ingénieur Cloud** (ou si vous travaillez aujourd'hui dans le cloud de façon pratique), l'Étape 1 vous sera familière — *parcourez-la, mais faites quand même ses exercices*, car elle reformule des choses que vous connaissez comme des faits en choses que vous devez peser comme des décisions, et cette reformulation est tout le passage d'ingénieur à architecte.

Règles pour l'ensemble du cours : étudiez **une heure par jour, six jours par semaine** — la régularité l'emporte sur l'intensité. Chaque module se termine par un **tableau de Vocabulaire** (les mots que vous devez posséder), des **Exercices** (du travail de conception, pas de la lecture

— l'architecture s'apprend en dessinant et en décidant) et un **Jalon (Milestone)** (la preuve que vous êtes prêt à avancer ; ne sautez pas au-delà d'un jalon non atteint). Et dès la Semaine 1, tenez un **Journal de conception** : chaque architecture que vous dessinez, chaque compromis que vous tranchez, chaque prédiction que vous faites sur les coûts ou les pannes. Dans dix mois, il devient votre portfolio d'entretien ; rien n'impressionne plus un jury de recrutement qu'un carnet daté de quarante conceptions avec des notes honnêtes sur ce que vous avez raté.

Ce que fait réellement un architecte cloud

Pensez à une architecte de bâtiment. Elle ne pose pas les briques — mais elle doit savoir exactement ce que les briques peuvent et ne peuvent pas faire, ce qu'elles coûtent, et comment les bâtiments s'effondrent. Elle écoute un client (« une maison familiale, ensoleillée, sous ce budget, sur ce terrain difficile ») et transforme les envies et les contraintes en *plans et spécifications* assez précis pour que les constructeurs puissent construire à partir d'eux et que le client puisse les valider. Puis elle reste présente pendant le chantier, répondant aux questions et ajustant le plan quand le sol se révèle plus meuble que l'étude ne le disait.

Remplacez les briques par des serveurs et c'est le métier. Un architecte cloud passe sa semaine à faire un mélange de : **écouter** les parties prenantes métier et extraire de vraies exigences de souhaits vagues ; **concevoir** — choisir les composants, dessiner les diagrammes, consigner les décisions et leurs raisons ; **relire** les conceptions des autres à l'aune des six piliers ; **estimer** ce qu'une conception coûtera par mois et défendre ce chiffre devant la finance ; **planifier les migrations** de vieux systèmes vers le cloud ; **présenter** — la même conception expliquée d'une façon aux ingénieurs et d'une façon complètement différente aux dirigeants ; et **mentorer** les ingénieurs pour que les standards survivent au contact des échéances. Dans beaucoup d'équipes, il y a aussi de l'**avant-vente (pre-sales)** : s'asseoir à côté d'un commercial devant un client potentiel et esquisser la solution qui remporte l'affaire.

Ce qu'un architecte n'est *pas* : le meilleur codeur de la pièce (rarement), la personne qui tape le plus (elle tape le moins), ou un génie solitaire qui distribue des plans (le chemin le plus rapide pour être ignoré). Le véritable produit de l'architecte, ce sont **de bonnes décisions, écrites, que les autres construisent de leur plein gré.**

La description de poste pour laquelle vous vous formez

En août 2026, nous avons extrait de vraies offres et de vrais modèles de postes de Cloud/Solutions Architect — un modèle d'architecte cloud de recruteur (KORE1), une description de poste d'un cabinet de recrutement (4 Corner Resources), la définition par Microsoft du rôle d'architecte dans l'Azure Well-Architected Framework, une offre de Solutions Architect avant-vente (Arpio, reprise après sinistre AWS), une offre de Solution Architect en entreprise (Intel, via Built In) et une offre de Cloud Domain Architect en entreprise (Halliburton, Azure d'abord). Retirez la couleur de chaque entreprise, et les douze mêmes exigences reviennent encore et encore. Ce cours est construit sur cette liste — voici exactement où chacune est enseignée :

Ce que demandent les descriptions de poste
(leurs mots, paraphrasés)

Où ce cours l'enseigne

« Concevoir des architectures cloud scalables

Étapes 1–3, capstones à l'Étape 5

Ce que demandent les descriptions de poste (leurs mots, paraphrasés)	Où ce cours l'enseigne
et sécurisées, adaptées aux exigences métier et techniques » — conception de solution de bout en bout	
« Expertise plateforme approfondie » sur AWS et/ou Azure — calcul, stockage, réseau, IAM, structure de comptes	Étape 1 + parcours de certification à l'Étape 5
« Mener des revues Well-Architected » à l'aune des six piliers	Étape 2 (les piliers), Étape 5 Module 12 (mener les revues)
« Piloter les initiatives de migration/modernisation — quels workloads bougent tels quels, sont ré-architecturés ou retirés »	Étape 4 Module 10 (les 7 R, la planification par vagues)
« Concevoir les landing zones, la structure de comptes, les garde-fous et les patterns de référence dans lesquels les équipes déploient »	Étape 4 Module 10
« Intégrer la sécurité et la conformité dans la conception plutôt que de les ajouter après coup » — zero trust, identité	Étape 2 Module 5, Étape 4 Module 11
Haute disponibilité et reprise après sinistre — « écarts RTO/RPO, coûts d'indisponibilité, exposition au ransomware »	Étape 2 Module 4, Étape 3 Module 9 (multi-région)
« Porter le modèle de coûts cloud — tagging, showback, capacité réservée, rightsizing » ; estimer les coûts des solutions	Étape 2 Module 6, Étape 5 Module 13 (chiffrage de proposition)
Conception d'architecture réseau — VPC, hub-and-spoke, connectivité hybride	Étape 1 Module 2, Étape 3 Module 9, Étape 4
« Créer la documentation d'architecture, les diagrammes et les standards » ; « maintenir des Architecture Decision Records »	Étape 1 Module 3 (diagrammes), Étape 5 Module 12 (ADR, jeux de diagrammes)
Communication avec les parties prenantes — « présenter des concepts techniques à des audiences de direction et techniques » ; « défendre les décisions d'architecture devant la sécurité, la finance et l'ingénierie »	Étape 5 Module 13
« Mentorner les ingénieurs cloud et plateforme qui construisent selon vos standards » ; soutien avant-vente — démos, POC, appels d'offres (RFP)	Étape 5 Module 13

Les certifications, vérifiées sur le même marché : l'échelle standard est **AWS Certified Solutions Architect – Associate (SAA-C03)** — le titre le plus demandé, typiquement **2–3 mois** de préparation — suivie d'**AWS Certified Solutions Architect – Professional (SAP-C02)**, typiquement **4–8 mois** de plus ; les entreprises très Azure demandent **AZ-305 (Azure Solutions Architect Expert)** ; les grandes entreprises ajoutent parfois **TOGAF** pour la méthode d'architecture d'entreprise. Le plan complet est à l'Étape 5, Module 14.

ÉTAPE 1 — LES BRIQUES DE BASE, EN TANT QUE DÉCISIONS (Semaines 1–6)

Diplômés du Cours d'Ingénieur Cloud : parcourez les explications, mais faites chaque exercice. Les faits sont les mêmes ; les questions sont nouvelles.

Module 1 (Semaines 1–2) : Ce qu'est le cloud, et comment pensent les architectes

Le cloud en un paragraphe (le rappel pour partir de zéro). Le cloud, ce sont *les ordinateurs de quelqu'un d'autre, loués à l'heure, gérés par du logiciel*. AWS, Microsoft Azure et Google Cloud exploitent des entrepôts de serveurs (des **data centres**, centres de données) regroupés en **Régions** géographiques (p. ex. « Asia Pacific (Bangkok) ») ; chaque région contient plusieurs **zones de disponibilité (Availability Zones, AZ)** isolées — des bâtiments séparés avec une alimentation indépendante, assez proches pour des connexions rapides, assez éloignés pour qu'une inondation ou un incendie ne puisse pas en emporter deux. Vous louez des tranches de tout cela à la seconde : des machines brutes (**IaaS** — vous gérez tout ce qui s'y trouve), des plateformes managées (**PaaS** — vous n'apportez que votre application) ou du logiciel fini (**SaaS** — vous l'utilisez, c'est tout). C'est là tout le substrat. Tout ce qu'un architecte conçoit s'arrange par-dessus.

Maintenant, le geste de l'architecte : transformer chaque fait en question. Un ingénieur apprend « une AZ est un centre de données isolé ». Un architecte demande immédiatement : « *Combien d'AZ ce workload mérite-t-il ?* » — car deux AZ coûtent plus qu'une, trois plus que deux, et un site vitrine marketing ne mérite pas ce que mérite un système de paiement. C'est la première et la plus importante idée du cours :

L'architecture est la discipline qui rend les compromis explicites. Il n'y a presque pas de composants faux — seulement des composants faux *pour ce workload, ce budget, cette équipe, cette échéance*.

Le triangle des compromis. Toute conception négocie entre trois sommets : **rapide** (la performance — rapide pour les utilisateurs, rapide à construire), **pas cher** (facture mensuelle basse, effort d'ingénierie faible) et **résilient** (survit aux pannes, monte en charge, reste sécurisé). Vous pouvez pousser vers deux sommets au choix ; le troisième en paie le prix. Le prototype d'une startup doit être rapide et pas cher — la résilience peut attendre. Le grand livre comptable d'une banque doit être résilient et rapide — il ne sera pas bon marché. Quand une partie prenante dit « nous voulons les trois », votre travail est de sourire et de demander lequel

elle veut *le plus*, car la conception ne peut pas commencer tant qu'elle n'a pas répondu. Dessinez ce triangle en haut de chaque conception que vous faites dans ce cours, et marquez où se situe le workload. Cela vous épargnera mille disputes.

Les exigences : la matière première de la conception. Les architectes séparent les **exigences fonctionnelles** (ce que fait le système — « les clients peuvent commander leur déjeuner ») des **exigences non fonctionnelles (NFR)** (à quel point il doit bien le faire — « en moins de 2 secondes, pour 10 000 étudiants simultanés, 99,9 % du temps, dans le respect du PDPA »). Les débutants s'obsèdent sur la première liste ; les architectes gagnent leur salaire sur la seconde, car ce sont les NFR qui déterminent réellement l'architecture. Deux questions magiques extraient les NFR de parties prenantes qui ne savent pas qu'elles en ont : « *Qu'arrive-t-il à l'activité si c'est en panne pendant une heure ?* » et « *À quoi ressemble le succès à dix fois la taille d'aujourd'hui ?* »

Vocabulaire :

Terme	Définition
Région / zone de disponibilité (AZ)	Région : un groupe géographique de centres de données où vous choisissez de tourner. AZ : un centre de données isolé (ou petit groupe) en son sein — l'unité du « un bâtiment peut tomber ».
IaaS / PaaS / SaaS	Louer des machines brutes / une plateforme managée / du logiciel fini. Le curseur du plus de contrôle vers le moins de maintenance.
Workload (charge de travail)	Toute application ou tout système, traité comme une unité de conception — « le workload des paiements ».
Exigence fonctionnelle	Ce que le système doit faire (« les clients peuvent commander »).
Exigence non fonctionnelle (NFR)	À quel point il doit bien le faire — vitesse, échelle, disponibilité, sécurité, conformité, coût. Les NFR pilotent l'architecture.
Compromis (trade-off)	Ce que vous abandonnez pour obtenir ce que vous avez choisi. Chaque décision de conception en a un ; le travail de l'architecte est de le nommer à voix haute.
Contrainte	Une limite non négociable : budget, échéance, loi, systèmes existants, compétences de l'équipe. Les contraintes ne sont pas des obstacles à la conception — elles <i>sont</i> le cahier des charges.
Partie prenante (stakeholder)	Quiconque a un intérêt dans le système : utilisateurs, ingénieurs, finance, sécurité, dirigeants, régulateurs. Différentes parties

Terme	Définition
	prenantes, différentes langues — vous les parlez toutes.
Greenfield / brownfield	Un système tout neuf sans histoire (greenfield) vs un système enchevêtré dans l'existant (brownfield — la majorité du travail réel).
Service managé	Un composant que le fournisseur exploite pour vous (sauvegardes, correctifs, failover inclus). Le choix par défaut de l'architecte, sauf raison écrite du contraire.

Vidéos pour ce module (liens vérifiés) :

Vidéo	Chaîne	Durée	Lien
Top 50+ AWS Services Explained in 10 Minutes	Fireship	~10 min	https://www.youtube.com/watch?v=JlBIYCM48to

Regardez le tour d'horizon de Fireship deux fois : une fois maintenant pour la carte, une fois à la fin de l'Étape 1 — vous serez surpris du nombre de services que vous saurez alors *placer* dans une conception.

Exercices : (1) Choisissez trois applications que vous utilisez quotidiennement (une appli bancaire, une appli de livraison de repas, une appli vidéo) et, pour chacune, écrivez ses trois principales NFR et positionnez-la sur le triangle des compromis. (2) Interviewez un ami sur une idée d'entreprise pendant dix minutes et extrayez cinq exigences fonctionnelles et cinq non fonctionnelles — remarquez comme les NFR ne sortent que lorsque vous posez les deux questions magiques. (3) Ouvrez votre Journal de conception avec l'entrée n°1 : le triangle, et un paragraphe sur pourquoi « quel sommet sacrifions-nous ? » est une question métier, pas une question technique.

Jalon : face à n'importe quelle description de système en une phrase, vous savez produire ses NFR probables et sa position sur le triangle en cinq minutes, à voix haute, sans notes.

Module 2 (Semaines 3–4) : Calcul, stockage, base de données, réseau — les quatre décisions

Chaque conception que vous dessinerez jamais, ce sont ces quatre choix, faits délibérément. Voici chacun enseigné comme une décision.

Décision 1 — Le calcul : VM, conteneur ou serverless ? Une **machine virtuelle (VM)** est une tranche louée d'un serveur qui se comporte comme un ordinateur entier — contrôle maximal, maintenance maximale (vous la patchez, vous la scalez, vous payez pendant qu'elle est inactive). Un **conteneur** emballe une application avec tout ce dont elle a besoin pour tourner à l'identique partout ; un orchestrateur (**Kubernetes**) fait tourner et répare des flottes

de conteneurs — excellente densité et portabilité, mais vous avez adopté une plateforme complexe qui exige des gens compétents. Le **serverless** (AWS Lambda, Azure Functions) n'exécute votre code que lorsqu'il est déclenché et facture à l'invocation — maintenance quasi nulle et parfait pour le trafic en pics, mais avec des limites (plafonds de durée d'exécution, cold starts, et un mariage plus profond avec un fournisseur). Le tableau de décision que vous devez savoir reproduire de mémoire :

Choisissez	Quand	Attention à
VM	Logiciel legacy, licences particulières, besoin du contrôle complet de l'OS, charge stable et prévisible	Les correctifs, la montée en charge et les 3 h du matin sont pour vous ; le temps inactif facture plein pot
Conteneurs + Kubernetes	Beaucoup de services, équipe qui a déjà les compétences, la portabilité compte	Complexité de plateforme — K8s est un travail à temps plein ; surdimensionné pour les petites équipes
Serverless	Trafic en pics ou imprévisible, glu événementielle, petites équipes, mise sur le marché rapide	Limites d'exécution, cold starts, coût plus dur à prédire à très grande échelle stable, lock-in

L'intuition senior : **c'est un choix par workload, pas une religion d'entreprise**. Les vrais parcs font tourner les trois côte à côte, à juste titre.

Décision 2 — Le stockage : objet, bloc ou fichier — et à quelle température ? Le **stockage objet** (Amazon S3) est un seau sans fond pour les fichiers — bon marché, absurde durable (« onze neufs »), la réponse par défaut à « où vont les fichiers ? ». Le **stockage bloc** (EBS) est le disque virtuel boulonné à une VM. Le **stockage fichier** (EFS) est un lecteur partagé que de nombreuses machines montent en même temps. La dimension supplémentaire de l'architecte est la **température** : données chaudes (accédées en permanence, tarifées pour la vitesse) versus niveaux froids/archives (Glacier — quelques centimes, mais des minutes à des heures pour récupérer). Concevoir des règles de cycle de vie qui font dériver les vieilles données vers les niveaux froids est le gain de coût le moins cher du cloud ; l'oublier est le plus courant.

Décision 3 — La base de données : SQL ou NoSQL (et quelle saveur managée) ? Les bases **relationnelles/SQL** (PostgreSQL, MySQL ; managées comme RDS/Aurora) gardent les données dans des tables strictes avec une cohérence garantie — le choix par défaut pour tout ce où la justesse est sacrée : l'argent, les commandes, les stocks, les utilisateurs. Les bases **NoSQL** (DynamoDB, MongoDB) échangent la structure stricte contre de la flexibilité et une montée en charge horizontale quasi illimitée — le choix par défaut pour les sessions, les catalogues, les fils d'actualité, la télémétrie. L'heuristique de décision : *commencez avec SQL sauf si vous pouvez nommer la raison précise pour laquelle cela ne marchera pas* (échelle extrême, schéma flexible, lectures mondiales à la milliseconde). Et dans le cloud, « base de données » devrait presque toujours vouloir dire « base de données **managée** » — le fournisseur gère les sauvegardes, les correctifs et le failover ; une équipe qui fait tourner sa propre base sur des VM devrait avoir une

raison écrite. Ajoutez les spécialistes à votre vocabulaire : le **cache** (Redis — les données chaudes en mémoire, lectures en microsecondes), l'**entrepôt (warehouse)** (l'analytique à grande échelle — Étape 3), la **file (queue)** (pas une base de données, mais souvent la pièce manquante — Étape 3).

Décision 4 — Le réseau : la forme du monde privé. Un **VPC (Virtual Private Cloud)** est votre section clôturée du réseau du fournisseur. À l'intérieur, les **sous-réseaux publics** contiennent ce que l'internet peut atteindre (les équilibreur de charge), et les **sous-réseaux privés** contiennent tout le reste — les serveurs d'application et, toujours, les bases de données. « *La base de données se trouve dans un sous-réseau privé* » est la phrase la plus répétée des revues d'architecture ; le raisonnement — rien ne peut attaquer ce qui n'a aucun chemin depuis l'internet — est la moitié de la sécurité réseau. Autour du VPC : un **équilibreur de charge (load balancer)** répartit le trafic entre les serveurs et contourne les malades ; le **DNS** (Route 53) transforme les noms en adresses ; un **CDN** (CloudFront) met le contenu en cache dans des centaines de villes pour qu'il soit rapide partout ; une **API gateway** est l'accueil managé de vos API (authentification, limites de débit, journalisation). Pour se connecter à l'ancien monde : le **VPN** (tunnel chiffré via l'internet) ou **Direct Connect** (une ligne physique privée) — les cordons ombilicaux de toute conception hybride de l'Étape 4.

Vocabulaire :

Terme	Définition
Instance / type d'instance	Une VM louée / sa taille (CPU + RAM), qui fixe son prix horaire.
Conteneur / Docker / Kubernetes (K8s)	Un paquet portable pour une application / l'outil qui les construit et les exécute / l'orchestrateur qui fait tourner et répare des flottes entières.
Serverless / Lambda	Du code qui ne s'exécute que déclenché, facturé à l'exécution, sans serveurs à gérer. Lambda est la version d'AWS.
Cold start (démarrage à froid)	Le délai supplémentaire quand une fonction serverless s'exécute après une période d'inactivité — le compromis serverless classique.
S3 / bucket / durabilité	Le stockage objet d'AWS / un conteneur nommé de fichiers / la probabilité que les données survivent — les 99,99999999 % de S3 signifient que la perte n'arrive pour ainsi dire jamais.
Niveau de stockage / politique de cycle de vie	Classe prix-vitesse (chaud → froid → archive) / la règle automatique qui déplace les données vieillissantes vers des niveaux moins chers.
RDS / Aurora / DynamoDB	Les services SQL managés d'AWS / son SQL

Terme	Définition
	cloud natif haute performance / son NoSQL managé phare.
Cohérence	La garantie que tous ceux qui lisent les données voient la même vérité au même moment — le superpouvoir du SQL, et ce que le NoSQL relâche pour gagner en échelle.
Cache / Redis	Une copie à la vitesse de la mémoire des données chaudes, placée devant une base de données / l'outil standard pour cela.
VPC / sous-réseau (public, privé)	Votre section de réseau privée / ses subdivisions — le public fait face à l'internet, le privé non. Les bases de données vivent en privé. Toujours.
Équilibreur de charge / health check	Le directeur du trafic entre serveurs / le test de pouls qu'il utilise pour cesser d'envoyer du trafic aux morts.
CDN / edge	Des caches de votre contenu à l'échelle des villes, dans le monde entier / « l'edge » = près des utilisateurs.
API / API gateway	La porte contrôlée qu'un logiciel offre à un autre / l'accueil managé de ces portes.
VPN / Direct Connect	Tunnel chiffré via l'internet / une ligne physique privée vers le cloud — les deux façons dont le on-prem rencontre le cloud.

Vidéos pour ce module (liens vérifiés) :

Vidéo	Chaîne	Durée	Lien
Kubernetes explained in 15 mins	TechWorld with Nana	~16 min	https://www.youtube.com/watch?v=VnvRFRk_51k
Serverless Computing in 100 Seconds	Fireship	~2 min	https://www.youtube.com/watch?v=W_VV2Fx32_Y
AWS Networking Basics (VPC & Subnets)	KodeKloud	~30 min	https://www.youtube.com/watch?v=QM63dyA_4Pc

Exercices : (1) Pour chacun de ces cinq workloads, choisissez le calcul, la base de données et le stockage, et écrivez une phrase de justification pour chacun : un site de commande de déjeuners

scolaires ; le grand livre des transactions d'une banque ; une appli de partage de photos ; un générateur de rapports nocturne qui tourne 20 minutes ; une appli de chat pour 5 millions d'utilisateurs. (2) Prenez l'un de vos choix et défendez le choix *opposé* aussi convainamment que possible — les architectes qui ne savent pas plaider au mieux l'alternative n'ont pas compris le compromis. (3) Journal : votre tableau de décision du calcul, de mémoire.

Jalon : face à n'importe quel workload en une phrase, vous savez nommer ses quatre décisions avec leurs justifications en moins de trois minutes — et pour au moins une décision, nommer ce qui vous ferait changer d'avis.

Module 3 (Semaines 5-6) : Lire et dessiner des diagrammes d'architecture

Un architecte qui ne sait pas dessiner est un consultant qui ne sait que parler. Les diagrammes sont votre langue de travail : ce module vous rend fluide en lecture et compétent en dessin.

Le diagramme canonique — apprenez celui-ci en premier. L'architecture trois tiers est la « structure de phrase » des diagrammes cloud ; la plupart des conceptions en sont des variations :

Users → DNS → CDN → Load Balancer → [Web/App servers, ×N, multi-AZ, auto-scaled]

→ [Cache]

→ [Database: primary +

standby, private subnets]

Tiers 1 (**présentation**) : ce que touchent les utilisateurs — le contenu statique depuis le CDN, les requêtes via l'équilibreur de charge. Tiers 2 (**application**) : la flotte de serveurs interchangeables qui exécutent votre logique, dans des sous-réseaux privés, scalée automatiquement. Tiers 3 (**données**) : la base de données, la plus profonde et la plus protégée, avec une réplique de secours dans une seconde AZ. Le trafic circule dans un seul sens : les utilisateurs ne touchent jamais directement le tiers application, seul le tiers application parle au tiers données. Entraînez-vous à dessiner cela jusqu'à ce que votre main le fasse sans votre cerveau — c'est l'ouverture de l'entretien au tableau blanc partout sur terre.

Les conventions de notation qui vous font paraître professionnel (parce qu'elles vous font penser professionnellement) : les boîtes sont des composants — étiquetez chacune avec ce qu'elle est et quel service (« App servers — EC2, auto-scaling group »). Les flèches montrent la direction dans laquelle circule une *requête*, étiquetées avec le protocole quand cela compte. Les enceintes en pointillés montrent les frontières — le VPC, chaque sous-réseau, chaque AZ (dessinez les frontières d'AZ et le multi-AZ devient visible au lieu d'être affirmé). Un utilisateur/acteur se tient hors du système. Des numéros sur les flèches (1, 2, 3...) vous permettent de narrer le voyage d'une requête. Et chaque diagramme porte un titre, une date et une légende. La règle plus profonde : **un diagramme, une audience, une question**. Un diagramme qui montre tout ne montre rien ; vous apprendrez le jeu standard de niveaux de zoom (contexte → conteneur → déploiement) à l'Étape 5.

Lire les diagrammes des autres — la radiographie de l'architecte. Quand on vous tend un diagramme, exécutez ce scan à voix haute : où l'internet touche-t-il ce système (chaque point de contact est une surface d'attaque) ? Où sont les données, et sont-elles dans un sous-réseau privé

? Qu'est-ce qui est dupliqué (résilient) et qu'est-ce qui est un **point de défaillance unique** — une boîte sans jumelle ? Où cela ferait-il mal à 10× le trafic ? Combien coûte chaque boîte par mois ? Cinq questions, trente secondes, et vous avez lu le diagramme comme un médecin lit une radio. Parcourez l'AWS Architecture Centre (<https://aws.amazon.com/architecture/>) et exécutez le scan sur trois architectures de référence publiées.

Vocabulaire :

Terme	Définition
Architecture trois tiers	La séparation classique : présentation (l'entrée des utilisateurs) – application (la logique) – données (la base). La forme par défaut des systèmes web.
Tiers / couche	Une tranche horizontale du système avec une seule responsabilité, ne parlant qu'à ses voisines.
Point de défaillance unique (SPOF)	Tout composant dont la mort solitaire fait tomber le système. La première chose à traquer dans tout diagramme.
Multi-AZ	Faire tourner des duplicatas sur au moins deux zones de disponibilité pour qu'un centre de données puisse tomber invisiblement.
Auto-scaling (groupe d')	Des machines ajoutées et retirées automatiquement selon la demande — une capacité qui respire.
Stateless / stateful (sans état / avec état)	Un serveur ne détenant aucune donnée unique (n'importe quel jumeau peut le remplacer, donc il scale librement) vs un serveur détenant des données à ne pas perdre. Objectif de conception : tiers application stateless, l'état poussé vers la base de données et le cache.
Architecture de référence	Un exemple de conception publié et béni par le fournisseur pour un problème courant — les architectes assemblent à partir de celles-ci avant d'inventer.
Diagramme de contexte	Le niveau de zoom le plus haut : votre système en une boîte, plus les utilisateurs et systèmes externes qu'il touche.
Surface d'attaque	Chaque point où le monde extérieur peut toucher le système. Plus petit, plus sûr.
Trafic nord-sud / est-ouest	Le trafic qui entre/sort du système vs le trafic entre composants à l'intérieur.

Exercices : (1) Dessinez le diagramme trois tiers de mémoire, cinq jours d'affilée, jusqu'à ce qu'il prenne moins de quatre minutes avec toutes les frontières (VPC, sous-réseaux, deux AZ) dessinées. (2) Prenez les cinq workloads du Module 2 et dessinez chacun — vingt minutes par diagramme, règles de notation imposées. (3) Trouvez n'importe quel vrai diagramme d'architecture en ligne (l'AWS Architecture Centre en a des centaines) et écrivez son scan-radiographie en cinq questions dans votre journal. (4) L'exercice de narration : diagramme sous les yeux, narrez le clic d'un utilisateur du navigateur à la base de données et retour, à voix haute, en numérotant les flèches au passage.

Jalon — fin de l'Étape 1 : sur un tableau blanc (ou du papier, photographié pour votre journal), vous savez dessiner une conception trois tiers correcte et correctement notée pour un workload inédit d'une phrase en moins de quinze minutes, y narrer une requête, et répondre à « que se passe-t-il si cette boîte meurt ? » pour chaque boîte. Ce dessin-plus-interrogatoire est précisément la première moitié d'un vrai entretien d'architecte — à partir d'ici, tout est profondeur.

ÉTAPE 2 — LES DOMAINES DE CONCEPTION (Semaines 7-14)

La carte de cette étape est l'AWS Well-Architected Framework — la checklist partagée de l'industrie sur ce que « conçu correctement » veut dire, organisée en six piliers : **excellence opérationnelle, sécurité, fiabilité, efficacité de la performance, optimisation des coûts, durabilité**. (Azure a un cadre quasi identique ; apprenez-en un en profondeur et vous avez appris les deux.) Les descriptions de poste demandent nommément des architectes capables de « mener des revues Well-Architected », nous prenons donc les piliers un par un, et pour chacun vous apprenez trois choses : les **questions clés** du pilier, ses **motifs standards** (les réponses ennuyeuses et éprouvées — les architectes assemblent avant d'inventer), et un **exemple travaillé** sur un scénario fil rouge.

Le scénario fil rouge de toute l'Étape 2 : ThaiTicket, une plateforme fictive de billetterie événementielle à Bangkok. Charge normale : 2 000 visiteurs/heure. Mais quand les billets d'un artiste célèbre sortent à 10 h 00, elle encaisse 400 000 visiteurs en dix minutes, les paiements ne doivent pas vendre deux fois le même siège, et les données personnelles des clients thaïlandais relèvent du PDPA. Rapide, pas cher, résilient — ThaiTicket a besoin des trois et ne peut pas les avoir, ce qui en fait le patient d'entraînement parfait.

À regarder avant le Module 4 (liens vérifiés) :

Vidéo	Chaîne	Durée	Lien
The Five Pillars of the AWS Well-Architected Framework	Amazon Web Services (official; un sixième pilier, Sustainability, a été ajouté plus tard)	~4 min	https://www.youtube.com/watch?v=KvEDbPmha6o
What is the AWS Well-Architected	Tech With Lucy	~10 min	https://www.youtube.com/

Vidéo	Chaîne	Durée	Lien
Framework?			watch?v=MpDJ6TCWKjk

Et mettez le cadre lui-même en favori — <https://aws.amazon.com/architecture/well-architected/> et le document complet sur <https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html> — vous allez y vivre pendant huit semaines.

Module 4 (Semaines 7-8) : La fiabilité — concevoir pour le jour où les choses cassent

Le credo du pilier : tout tombe en panne, tout le temps. Les disques meurent, les AZ sont inondées, les déploiements tournent mal, et quelque part un certificat est toujours sur le point d'expirer. La fiabilité n'est pas l'absence de panne — c'est l'*insignifiance* de la panne : concevoir pour que quand (pas si) un composant meurt, les utilisateurs ne s'aperçoivent de rien.

Questions clés (posez-les à chaque conception, pour toujours) : Que se passe-t-il quand chaque composant tombe — y a-t-il toujours un « et ensuite » ? Comment le système gère-t-il 10× la charge ? Comment *savons-nous* qu'une chose a échoué avant que les clients ne nous le disent ? Quels sont nos **RTO** et **RPO** — et qui, côté métier, les a validés ? Quand avons-nous *testé* une restauration pour la dernière fois ?

RTO et RPO — les deux nombres qui sont la conversation du sinistre. **Recovery Time Objective** : combien de temps pouvons-nous rester à terre ? **Recovery Point Objective** : combien de données pouvons-nous perdre (c'est-à-dire quel âge peut avoir la dernière bonne sauvegarde) ? Un RTO de 4 heures et un RPO de 1 heure signifient : de retour en quatre heures, avec au plus une heure de données perdues. Ce sont des décisions *métier* aux étiquettes de prix exponentielles — un RPO de 24 heures, c'est une sauvegarde nocturne (pas cher) ; un RPO de ~zéro, c'est de la réplication continue (cher) ; un RTO de quelques minutes, c'est une infrastructure tiède qui tourne en attente (très cher). Le travail de l'architecte est de faire choisir les chiffres au métier *en connaissance de cause* — et de concevoir exactement pour eux, pas romantiquement au-delà.

Les motifs standards : la **redondance** (deux exemplaires de tout ce qui compte — N+1) · le **multi-AZ** (des duplicatas entre centres de données ; l'équilibreur de charge et le failover de la base managée le rendent automatique) · l'**auto-scaling** (la capacité suit la demande) · **health checks + auto-réparation** (les instances mortes détectées et remplacées par la machinerie, pas par des humains) · **des sauvegardes, testées** (une sauvegarde non testée est un espoir, pas un plan — planifiez des exercices de restauration) · la **dégradation gracieuse** (en surcharge, déléster d'abord les fonctionnalités les moins importantes : ThaiTicket peut abandonner les aperçus de plan de salle et garder le checkout) · **les files comme amortisseurs** (Étape 3) · **éviter la défaillance en cascade** (timeouts, retries avec backoff, circuit breakers — pour qu'une dépendance lente ne noie pas la flotte).

Exemple travaillé — la conception fiabilité de ThaiTicket. Deux AZ dans la région de Bangkok. Tiers application stateless dans un groupe d'auto-scaling derrière un équilibreur de charge, préchauffé sur planning avant les heures de mise en vente annoncées (l'auto-scaling

réagit en minutes ; une sortie à 10 h 00 exige de la capacité à 9 h 45). Base Aurora, primaire en AZ-a, réplique synchrone en AZ-b, failover automatique ≈ moins d'une minute. Une file entre les clics « acheter » et le traitement des paiements, pour qu'un ralentissement du prestataire de paiement mette les commandes en file au lieu de faire tomber le site. Sauvegardes : continues, restauration à un instant donné, exercice de restauration mensuel. Chiffres convenus avec le métier : RTO 15 minutes, RPO ~ zéro pour les commandes (c'est de l'argent), RPO 24 heures pour les données analytiques (ça n'en est pas). Exercice de journal : qu'est-ce que cela nous a coûté du sommet « pas cher » du triangle ?

Vocabulaire :

Terme	Définition
RTO / RPO	Recovery Time Objective : combien de temps vous pouvez rester à terre. Recovery Point Objective : combien de données vous pouvez perdre. Les deux nombres qui définissent toute conversation DR — et ce sont des décisions métier.
Haute disponibilité (HA)	Concevoir pour que les pannes de routine (une instance, une AZ) ne causent aucune interruption visible pour l'utilisateur.
Reprise après sinistre (DR)	Le plan pour les <i>grandes</i> pannes — une région entière, un ransomware — avec des cibles RTO/RPO et un runbook testé.
Redondance / N+1	De la capacité de réserve pour que n'importe quel composant puisse mourir : N nécessaires, N+1 en marche.
Failover	La bascule automatique vers une réplique de secours quand le primaire meurt.
Health check	Le pouls automatisé qui détecte les composants morts pour que la machinerie les contourne.
Dégradation gracieuse	Délester les fonctionnalités moins importantes sous stress pour que le cœur survive.
Timeout / retry avec backoff	Abandonner un appel lent après une limite / réessayer avec des pauses croissantes — les bonnes manières qui préviennent les défaillances en cascade.
Circuit breaker (disjoncteur)	Un composant qui cesse d'appeler une dépendance défaillante pendant un temps pour qu'elle récupère — comme le fusible d'une maison.
Chaos engineering	Injecter délibérément des pannes (façon

Terme	Définition
	Netflix) pour prouver les prétentions de résilience avant que la réalité ne les teste pour vous.

Exercices : (1) Prenez votre conception de commande de déjeuners de l'Étape 1 et améliorez-la pour survivre à : la mort d'une instance, la perte d'une AZ, une panne de base de données, et une ruée du déjeuner à 10× — dessinez l'avant et l'après. (2) Écrivez la conversation RTO/RPO pour trois activités (un blog, un site e-commerce, un système de dossiers hospitaliers) sous forme de court dialogue entre l'architecte et le propriétaire — entraînez-vous à rendre le coût visible dans le dialogue. (3) Journal : listez chaque SPOF de la conception de ThaiTicket ci-dessus. Il en reste au moins un, délibérément. (Indice : combien de régions ?)

Jalon : vous savez mener l'interrogatoire « que se passe-t-il quand ceci tombe ? » sur n'importe quel diagramme pendant dix minutes d'affilée sans vous tarir, et vous savez expliquer RTO vs RPO à un propriétaire non technique avec une analogie de boutique inondée en quatre-vingt-dix secondes.

Module 5 (Semaines 9–10) : La sécurité — concevoir des systèmes difficiles à blesser

Le cadre : le modèle de responsabilité partagée. Le fournisseur sécurise le *cloud lui-même* — bâtiments, matériel, hyperviseur. Vous sécurisez tout ce que vous y mettez — données, identités, configurations, code. Presque toutes les brèches cloud célèbres sont des erreurs de configuration côté client (un bucket public, une clé fuitée, un rôle surdoté), raison pour laquelle la sécurité est un problème d'*architecture* avant d'être un problème d'outillage : les descriptions de poste disent « intégrer la sécurité dans la conception plutôt que de l'ajouter après coup », et ce module est le comment.

Questions clés : Qui et quoi peut accéder à chaque composant, et chaque permission est-elle le *minimum* nécessaire ? Où les données sont-elles chiffrées — au repos, en transit, et qui détient les clés ? Où sont les frontières réseau, et qu'est-ce qui les traverse ? Comment *détecterions-nous* une brèche — et pourrions-nous raconter l'histoire après coup à partir des logs ? Quelle loi s'applique à ces données, et où les données vivent-elles physiquement ?

Les motifs standards : le **moindre privilège** (chaque humain et chaque programme reçoit l'accès minimum que son travail exige — la règle d'or à l'aune de laquelle chaque politique IAM est jugée) · la **MFA partout, le root sous clé · le chiffrement au repos et en transit, toujours activé** (c'est une case à cocher dans le cloud ; il n'y a aucune excuse) · la **segmentation réseau** (sous-réseaux publics/privés, groupes de sécurité comme pare-feu par serveur ; le rayon d'explosion d'une brèche est défini par les murs que vous avez dessinés à l'avance) · les **secrets dans un coffre, jamais dans le code** · le **zero trust** (vérifier chaque requête explicitement — identité, appareil, contexte — ne rien croire pour la simple raison d'être « à l'intérieur du réseau » ; les descriptions de poste le nomment, vous devez le nommer aussi) · la **défense en profondeur** (des couches, pour qu'un contrôle défaillant ne soit pas la fin de partie) · la **journalisation d'audit** (des traces façon CloudTrail de qui a fait quoi — immuables, surveillées) · la **sécurité en garde-fous, pas en barrières** (encoder les règles en

politiques automatisées qui font du chemin sécurisé le chemin facile, plutôt qu'une réunion de revue qui fait de la sécurité l'ennemie de la livraison).

Exemple travaillé — la conception sécurité de ThaiTicket. Les données clients (noms, e-mails, références de paiement) classées données personnelles sous le PDPA → stockées chiffrées dans Aurora dans la région de Bangkok (résidence des données), clés dans KMS. Réseau : seul l'équilibreur de charge est public ; tiers application privé ; le sous-réseau de la base n'accepte de connexions *que* depuis le groupe de sécurité du tiers application. Les humains : SSO + MFA ; les ingénieurs ont par défaut un accès production en lecture seule et un accès élevé limité dans le temps sur demande (moindre privilège avec piste d'audit). Le traitement des cartes de paiement est délégué à un prestataire de paiement certifié pour que les numéros de carte bruts ne touchent jamais nos systèmes — une décision de *conception* qui supprime un fardeau de conformité entier, ce qui est l'architecture de sécurité à son meilleur. CloudTrail activé, alertes sur accès inhabituels. La question de la brèche pré-répondue : le PDPA impose une notification sous 72 heures — les logs et le runbook doivent nous permettre de raconter l'histoire en moins.

Vocabulaire :

Terme	Définition
Modèle de responsabilité partagée	Le fournisseur sécurise le cloud ; vous sécurisez ce qui s'y trouve. La plupart des brèches sont du côté client de la ligne.
IAM / rôle / politique	Identity and Access Management — qui peut faire quoi. Une politique est une liste de permissions ; un rôle en est un lot que l'on peut endosser.
Moindre privilège	La règle d'or : l'accès minimum nécessaire, rien de plus, revu régulièrement.
MFA	Une seconde preuve d'identité au-delà du mot de passe. Non négociable pour les humains.
Chiffrement au repos / en transit / KMS	Les données brouillées sur disque / sur le réseau / le service managé qui détient les clés.
Groupe de sécurité	Une liste de règles de pare-feu par serveur — « trafic web entrant depuis l'équilibreur de charge uniquement ».
Segmentation réseau / rayon d'explosion	Diviser le réseau en zones murées / jusqu'où un attaquant peut aller après une brèche. Les murs dessinés à l'avance le définissent.
Zero trust	Vérifier chaque requête explicitement ; ne rien croire pour être « à l'intérieur ». La posture moderne par défaut.
Défense en profondeur	Plusieurs contrôles qui se recouvrent, pour qu'une défaillance ne soit pas fatale.

Terme	Définition
Gestion des secrets	Les mots de passe, clés et jetons vivent dans un service de coffre — jamais dans le code, jamais dans un tableur.
Journal d'audit / CloudTrail	La trace immuable de qui a fait quoi, quand — comment on détecte les ennuis et comment on les reconstitue après.
Classification des données	Étiqueter les données par sensibilité (publiques / internes / personnelles / réglementées) pour que les contrôles suivent l'étiquette, pas une intuition.
Résidence des données	Garder les données physiquement à l'intérieur des frontières d'un pays — une exigence légale dans beaucoup d'industries, une donnée d'architecture toujours.
PDPA / GDPR (RGPD)	Les lois thaïlandaise et européenne sur les données personnelles : consentement, notification de brèche (72 heures), règles de transfert transfrontalier. L'Étape 4 approfondit.

Vidéo pour ce module (lien vérifié) : The AWS Shared Responsibility Model — Digital Cloud Training — ~4 min — <https://www.youtube.com/watch?v=ESPBBEK-cvo>

Exercices : (1) Dessinez le diagramme de ThaiTicket et superposez-y sa sécurité : marquez chaque frontière de confiance, chaque point de chiffrement, chaque endroit où vivent des identifiants. Cette habitude de calque — même diagramme, lentille sécurité — est exactement ce qu'une revue exige de vous. (2) Lisez un post-mortem public d'une brèche par mauvaise configuration cloud (Capital One 2019 est le classique pédagogique) et écrivez, dans votre journal, quel *motif ci-dessus* l'aurait empêchée. (3) Jeu de rôle avec une IA : elle joue un fondateur de startup qui dit « on ajoutera la sécurité plus tard » — dissuadez-le avec l'arithmétique du coût d'une brèche, gentiment.

Jalon : vous savez prendre n'importe lequel de vos diagrammes de l'Étape 1 et produire son calque de sécurité en vingt minutes, et expliquer le moindre privilège, le zero trust et le modèle de responsabilité partagée à une partie prenante non technique sans jargon.

Module 6 (Semaines 11-12) : Performance et coûts — la vitesse et l'argent, les piliers couplés

Ces deux piliers sont enseignés ensemble parce qu'ils sont le même levier poussé dans des directions opposées, et l'architecte est la personne qui a la main sur le levier.

Efficacité de la performance — questions clés : Où les utilisateurs sentiront-ils la lenteur en premier ? Qu'est-ce qui est calculé de façon répétée et pourrait être calculé une fois puis mis en cache ? Chaque composant est-il le *bon outil* (du SQL qui fait le travail d'un moteur de

recherche est un bug de performance de l'architecture, pas du code) ? Comment la performance change-t-elle à 10× — et où est le premier goulot d'étranglement ?

Motifs de performance : les couches de cache — le motif de performance au levier le plus élevé : cache navigateur — **CDN** en bordure (le contenu statique servi depuis une ville proche de l'utilisateur ; peut absorber l'immense majorité du trafic en lecture) — cache applicatif (**Redis** devant la base de données pour les lectures chaudes : plans de salle, sessions, pages produit) — cache de requêtes de la base. Chaque couche répond aux requêtes avant qu'elles n'atteignent le cœur coûteux ; les questions de conception sont toujours *que peut-on mettre en cache, pour combien de temps, et comment est-il invalidé quand la vérité change*. Puis : **les répliques de lecture** (des copies de la base qui servent les lectures pour que le primaire continue d'écrire) · **le traitement asynchrone** (ne faites pas attendre les utilisateurs pour un travail qui peut arriver après — reçus par e-mail, miniatures, rapports) · **le bon outil pour chaque tâche** (la recherche — un moteur de recherche ; l'analytique — un entrepôt ; les consultations chaudes — un magasin clé-valeur) · et **mesurer d'abord** (le travail de performance sans mesure est de la superstition).

Optimisation des coûts — questions clés : Combien cette conception coûte-t-elle par mois à la charge actuelle — et par *unité* (par commande, par client) ? Qu'est-ce qui tourne à 3 h du matin sans en avoir besoin ? Quel usage stable pourrait être engagé à prix réduit ? Que dirait la finance de la tendance ?

Motifs de coûts : le rightsizing (la plupart des flottes sont silencieusement surprovisionnées ; réduire au besoin mesuré est de l'argent gratuit) · **la stratégie de réservation** — le menu de tarification : **on-demand** (plein prix, pleine liberté) pour la charge en pics/inconnue, **instances réservées / savings plans** (engagement 1–3 ans, 30–70 % de réduction) pour la base stable, **spot** (jusqu'à -90 %, récupérable à bref préavis) pour le travail par lots interruptible ; le geste de l'architecte est de les empiler : réserver le plancher, auto-scaler le milieu en on-demand, mettre les lots sur spot · **le cycle de vie du stockage** (la hiérarchisation du Module 2, automatisée) · **éteindre les choses** (des environnements hors production qui dorment les nuits et week-ends peuvent réduire leur coût de deux tiers) · **surveiller l'egress** (les données *quittant* le cloud sont facturées ; les conceptions bavardes inter-régions et les gros téléchargements publics surprennent tout le monde une fois) · **le tagging et le showback** (chaque ressource étiquetée avec propriétaire/projet pour que chaque baht soit attribuable) · et la métrique reine, **l'économie unitaire (unit economics)** : pas « la facture est de \$800k/mois » mais « le coût par billet vendu est de \$1,90 et il baisse ». Des factures qui grossissent, c'est bien ; des *coûts unitaires* qui grossissent, c'est une odeur d'architecture. Cette discipline a un nom — le **FinOps** — et les architectes en occupent le centre.

Exemple travaillé — ThaiTicket, sous les deux lentilles. Performance : CloudFront sert les pages d'artistes et les images de plans de salle (les 400 000 curieux ne touchent pour la plupart jamais les serveurs) ; Redis met en cache la disponibilité des sièges avec un TTL de 2 secondes — assez périmé pour être bon marché, assez frais pour que le checkout (qui révérifie contre Aurora, la source de vérité) empêche la double vente ; les reçus et billets sont générés de façon asynchrone après paiement. Coûts : la base stable de 2 000 visiteurs/heure tourne sur un savings plan (≈40 % de réduction) ; les pics de mise en vente tournent en on-demand pendant l'heure où ils existent ; les jobs analytiques tournent sur spot la nuit ; le staging dort hors heures ouvrées ; chaque ressource est taguée `project:thaiticket`. La proposition au CFO se

lit : « \$62 000/mois en base, ≈\$9 000 par grand événement de mise en vente, coût par billet ≈\$1,90 en baisse avec le volume » — et *cette phrase* est ce à quoi ressemble un architecte fluide en coûts.

Vocabulaire :

Terme	Définition
Latence / débit	Combien de temps prend une requête / combien de requêtes par seconde le système soutient. Liés, pas identiques.
Goulot d'étranglement	Le point le plus étroit qui fixe le rythme de tout le système. Optimiser ailleurs est de la décoration.
Cache / TTL / invalidation	Une copie rapide de données coûteuses à obtenir / combien de temps on peut s'y fier / le problème difficile de la rafraîchir quand la vérité change.
CDN	Le cache le plus extérieur — votre contenu dans des centaines de villes. Première réponse à « rendez-le rapide mondialement ».
Réplica de lecture	Une copie de la base servant les lectures, pour que le primaire garde ses forces pour les écritures.
Traitement asynchrone	Faire le travail non urgent après avoir répondu à l'utilisateur, généralement via une file.
On-demand / réservé / savings plan / spot	Plein prix flexible / engagement 1–3 ans à –30–70 % / même idée, plus flexible / jusqu'à –90 % mais récupérable. L'architecte empile les quatre.
Rightsizing	Réduire les ressources surprovisionnées au besoin mesuré. De l'argent gratuit dans presque chaque compte.
Egress	Les données quittant le cloud — facturées au Go. La célèbre surprise de facture ; concevez les flux de données en la gardant à l'esprit.
Tagging / showback	Étiqueter chaque ressource avec propriétaire et projet / montrer à chaque équipe sa propre facture. La responsabilisation change les comportements.
Économie unitaire	Le coût par unité métier (par commande, par utilisateur). La métrique qui donne du sens à une facture — et la phrase préférée de

Terme	Définition
	l'architecte devant un CFO.
TCO	Total cost of ownership (coût total de possession) — le prix affiché plus les gens, les licences, la migration et les coûts de sortie sur toute la vie d'un choix.
FinOps	La discipline qui rend la dépense cloud visible, allouée et continuellement optimisée.

Vidéo pour ce module (lien vérifié) : What is FinOps? — FinOps Foundation (official) — ~2 min — https://www.youtube.com/watch?v=Y-c_xw9bHFw

Exercices : (1) Chiffrez la base de ThaiTicket dans le calculateur de prix AWS (cherchez « AWS pricing calculator » — l'exercice de détailler une vraie conception une fois démystifie toutes les futures conversations de coûts) ; comparez votre total aux \$62 000 ci-dessus et expliquez tout écart. (2) Ajoutez le calque de cache à deux de vos diagrammes de l'Étape 1 : marquez chaque cache, son TTL et son histoire d'invalidation. (3) Une IA joue un ingénieur qui veut tout en on-demand « pour rester simple » — négociez la stratégie de réservation, chiffres à l'appui.

Jalon : pour toute conception que vous avez dessinée, vous savez énoncer ses trois plus grosses lignes de coûts et son premier goulot d'étranglement probable — et proposer un changement qui améliore les deux à la fois (il y en a presque toujours un ; c'est généralement le cache).

Module 7 (Semaines 13–14) : Excellence opérationnelle, durabilité — et faire tourner les six piliers ensemble

L'excellence opérationnelle est le pilier qui demande : *des humains peuvent-ils réellement faire tourner cette chose, calmement ?* Questions clés : Comment un changement atteint-il la production — par un pipeline automatisé et testé, ou par un humain héroïque ? Comment savons-nous que le système est en bonne santé *en ce moment même* ? Quand ça casse à 3 h du matin, que fait concrètement l'ingénieur d'astreinte ? Motifs : **l'infrastructure as code** (l'environnement défini dans du texte relu et versionné — Terraform/CloudFormation — donc reproductible et auditable ; une architecture qui n'existe que sous forme de clics de console est du folklore, pas de l'ingénierie) · **les pipelines CI/CD** avec des stratégies de déploiement sûres (**blue-green** : monter la nouvelle version à côté de l'ancienne et basculer ; **canary** : donner à la nouvelle version 5 % du trafic et observer) · **l'observabilité** (logs, métriques, traces, et des alertes liées à des symptômes *visibles par l'utilisateur*, pas à des trivialités de machines) · **les runbooks** (le quoi-faire écrit pour chaque panne connue) · **les post-mortems sans blâme** (après les incidents : ce qui a échoué, pourquoi, ce qui empêche une récurrence — pas de coupables, sinon les gens cessent de dire la vérité). La part de l'architecte : *concevoir* pour l'exploitabilité — un système deux fois plus malin et moitié moins observable est un système pire.

La durabilité, sixième pilier, vous demande de gaspiller moins le monde physique : rightsizez (les CPU inactifs brûlent de l'électricité réelle), scalez à la demande au lieu de provisionner au pic, utilisez les services managés et serverless (une infrastructure partagée est une infrastructure mieux remplie), hiérarchisez le stockage, et supprimez ce qui est mort.

Commodément, le choix durable et le choix économique sont généralement le même choix — dites les deux en revue et vous emporterez la salle deux fois.

Faire tourner les piliers ensemble — votre première revue Well-Architected. Dans les vraies revues, les piliers *se contredisent* : la fiabilité multi-région combat le coût ; la revue de sécurité stricte combat la vélocité opérationnelle ; le cache de performance combat la justesse de fraîcheur des données. Le cadre ne résout pas les conflits pour vous — il les force au grand jour, où le métier peut choisir. C'est là tout son génie, et le vôtre à manier. La méthode de revue elle-même (jeux de questions, priorisation des constats, le rapport) est enseignée entièrement à l'Étape 5, Module 12 ; cette quinzaine, vous répétez la compétence brute : interroger une conception sous six angles en une seule séance.

Vocabulaire :

Terme	Définition
Infrastructure as Code (IaC) / Terraform	L'infrastructure déclarée dans des fichiers texte versionnés que des outils transforment en réalité / l'outil le plus populaire du genre.
Pipeline CI/CD	Le convoyeur automatisé qui construit, teste et déploie chaque changement.
Déploiement blue-green / canary	Deux motifs de mise en production sûre : basculer le trafic entre l'ancienne et la nouvelle copie / envoyer un filet de trafic vers la nouvelle version et observer.
Observabilité (logs, métriques, traces)	La capacité du système à s'expliquer : traces d'événements, nombres dans le temps, et cartes du voyage de chaque requête.
Alerte / astreinte / runbook	L'appel automatique quand les seuils cassent / la rotation humaine qui y répond / le script écrit qu'elle suit.
Post-mortem sans blâme	La revue honnête et sans coupables après un incident, qui produit de la prévention, pas des punitions.
Drift (dérive)	La réalité qui diverge de la définition IaC parce que quelqu'un a cliqué quelque chose. L'ennemi de la reproductibilité.
Toil	Le travail opérationnel manuel et répétitif que l'automatisation aurait dû absorber. Les architectes le conçoivent hors du système.
Revue Well-Architected	Un interrogatoire structuré d'un workload à l'aune des six piliers, produisant des constats priorités. L'Étape 5 vous apprend à en mener une.

Exercices : (1) L'exercice des six lentilles : prenez votre meilleur diagramme et passez dix minutes par pilier à écrire des constats — soixante minutes, une conception, six lentilles ; cet exercice est l'Étape 2 en miniature, répétez-le chaque semaine désormais. (2) Lisez deux rapports post-incident d'AWS (publiés sur leurs pages de statut après les grandes pannes) et identifiez quel motif de quel pilier a échoué. (3) Journal : pour ThaiTicket, écrivez les trois conflits de piliers que vous remonteriez au métier, chacun comme un choix en une phrase (« Nous pouvons avoir X ou Y à ce budget — lequel ? »).

Jalon — fin de l'Étape 2 : la revue simulée : une IA génère une architecture défaillante pour une startup thaïlandaise de prêt en ligne ; vous devez produire des constats écrits sur les six piliers, incluant au moins : une faille de fiabilité (leur base mono-AZ), une faille de sécurité (IAM trop large), une faille de coûts (tout en on-demand), une faille opérationnelle (pas d'IaC), et la conversation RTO/RPO qu'ils n'ont jamais eue — chaque constat formulé comme une question qu'un collègue pourrait entendre sans tressaillir. Quand votre liste de constats se lit comme celle d'un collègue senior serviable plutôt que d'un auditeur, l'Étape 2 est terminée.

ÉTAPE 3 — LE CATALOGUE DE PATTERNS (Semaines 15–20)

Les architectes n'inventent pas ; ils **sélectionnent**. Cette étape est votre catalogue des architectures standards — pour chacune : ce qu'elle est, quand l'utiliser, quand ne *pas* l'utiliser, et son profil coût/complexité. Les colonnes « quand ne pas » sont la vraie cargaison de l'étape : n'importe quel cours peut vous dire ce que sont les microservices ; savoir quand ils vont démolir une entreprise est ce pour quoi on vous paie. Pendant votre étude, continuez à parcourir de vraies architectures de référence sur <https://aws.amazon.com/architecture/> — repérer les patterns dans la nature est l'exercice qui fait tenir le catalogue.

Module 8 (Semaines 15–17) : Les patterns applicatifs

Monolithe vs microservices — la dispute la plus bruyante de l'industrie, réglée calmement. Un **monolithe** est une application déployable unique contenant toutes les fonctionnalités ; les **microservices** découpent le système en de nombreux petits services déployés indépendamment, chacun possédant ses données, parlant par API et par événements. Le tableau honnête :

Pattern	Ce que c'est	À utiliser quand	À ne PAS utiliser quand	Coût/complexité
Monolithe	Une application, un déploiement, une base de données	Petite équipe, produit jeune, frontières de domaine floues — c'est-à-dire la plupart des nouveaux systèmes	Plusieurs équipes se bloquent mutuellement ; des parties doivent scaler très différemment	Complexité basse, coût bas ; scale plus loin que la mode ne l'admet — « ennuyeux » est une qualité

Pattern	Ce que c'est	À utiliser quand	À ne PAS utiliser quand	Coût/complexité
Microservices	Beaucoup de petits services, construits, déployés et scalés indépendamment	Beaucoup d'équipes exigeant des cadences de mise en production indépendantes ; montée en charge très différente par partie ; frontières de domaine éprouvées et stables	Petite équipe (« un monolithe distribué est un monolithe avec des pannes réseau en plus ») ; domaine encore mouvant	Complexité élevée : chaque appel de fonction devient un appel réseau qui peut échouer ; exige CI/CD mûre, observabilité, astreinte

Le verdict de l'architecte : **commencez monolithique, modulaire à l'intérieur ; extrayez des services quand — et seulement quand — une douleur de montée en charge d'équipe ou de charge arrive réellement.** Dire cela en entretien, avec des raisons, vous marque comme senior ; l'idéologie dans un sens ou dans l'autre vous marque comme junior.

L'architecture événementielle — files et topics, les amortisseurs. Au lieu de composants qui s'appellent et *attendent* (synchrone), les composants publient des **événements** (« OrderPlaced ») vers une **file (queue)** (SQS — un consommateur prend chaque message, à son rythme) ou un **topic** (SNS — chaque abonné reçoit une copie ; le fan-out). Le producteur ne sait pas et ne se soucie pas de qui écoute. À utiliser quand : les composants doivent survivre aux pannes les uns des autres (la file retient les messages pendant qu'un consommateur est à terre — résilience et découplage en un seul achat) ; la charge est en pics (la file absorbe le pic ; les workers la vident régulièrement) ; une occurrence déclenche plusieurs réactions (commande passée — débit, e-mail, stock, analytique — quatre abonnés, zéro couplage). À ne PAS utiliser quand : l'utilisateur a besoin de la réponse *maintenant* (le checkout ne peut pas confirmer « à terme ») ; ou l'équipe est petite et un simple appel synchrone suffirait — chaque file ajoute des sémantiques de livraison (at-least-once signifie que les consommateurs doivent être **idempotents** — sûrs à exécuter deux fois), la gestion des dead-letters, et un fardeau de supervision. Coût/complexité : composants bon marché, débogage plus cher — l'histoire d'une requête est désormais éparpillée entre services et files, raison pour laquelle l'observabilité (Module 7) cesse ici d'être optionnelle.

Trois tiers : déjà à vous (Module 3). À utiliser pour : le large milieu des applications web — c'est le pattern à l'aune duquel les autres se mesurent. Pas pour : les workloads très en pics avec des creux inactifs (le serverless est moins cher) ou les très grands produits multi-équipes (voir ci-dessus). Profil : compris partout, recrute facilement, coût moyen.

Serverless-first : composer des pièces managées — API Gateway — fonctions Lambda — DynamoDB, S3 et des files — sans posséder aucun serveur. À utiliser quand : trafic en pics ou

faible (le scale-to-zero signifie que l'inactivité ne coûte ~rien), petites équipes, mise sur le marché rapide, glu événementielle. À ne PAS utiliser quand : calcul long ou spécialisé (limites d'exécution), chemins ultrasensibles à la latence (cold starts), charge *stable* massive (la tarification à l'invocation peut dépasser les serveurs réservés — faites l'arithmétique à l'échelle), ou quand la portabilité entre clouds est une exigence réelle (c'est le lock-in le plus profond de tout le catalogue — souvent rentable, mais dites-le à voix haute). Profil : le fardeau opérationnel le plus bas du catalogue ; coût superbe à échelle faible/en pics, à vérifier à grande échelle stable.

Vocabulaire :

Terme	Définition
Monolithe / monolithe modulaire	Une application déployable avec tout dedans / la version disciplinée : un déployable, des frontières de modules internes propres — le meilleur défaut pour les nouveaux systèmes.
Microservices	Beaucoup de petits services déployés indépendamment, chacun possédant ses propres données. Un outil de montée en charge d'équipe qui prélève une taxe de systèmes distribués.
Couplage / découplage	À quel point les composants dépendent de la disponibilité et des détails les uns des autres. L'architecture est largement l'art d'acheter la bonne quantité de découplage.
Synchrone / asynchrone	Appeler-et-attendre vs envoyer-et-continuer. Le choix fondamental sur chaque flèche que vous dessinez.
Événement / événementiel	Un fait annoncé à qui veut l'entendre (« OrderPlaced ») / une architecture construite à partir de telles annonces.
File (SQS)	Une file de messages ; chacun pris par un consommateur à son rythme. Amortisseur et découpleur.
Topic / pub-sub (SNS)	Un canal de diffusion ; chaque abonné reçoit chaque message. L'outil du fan-out.
Dead-letter queue	Là où les messages qui échouent de façon répétée sont mis de côté pour les humains — le filet de sécurité du pattern.
Idempotent	Sûr à traiter deux fois avec le même résultat — exigé des consommateurs, car les files peuvent livrer un message plus d'une fois.
Serverless-first	Composer des pièces managées à scale-to-

Terme	Définition
	zero (fonctions, bases managées, files) au lieu de faire tourner des serveurs.
Lock-in	La dépendance aux services propriétaires d'un fournisseur, tarifiée en coût de changement. Pas un péché — un terme économique à peser au grand jour (Étape 4).

Exercices : (1) Concevez le flux de commande de ThaiTicket deux fois — trois tiers synchrone, puis événementiel avec SQS/SNS — et écrivez un paragraphe sur ce que fait chaque version pendant une panne du prestataire de paiement ; ce paragraphe est tout l'argument des événements. (2) Pour quatre entreprises (une startup de 3 personnes, une scale-up de 50 ingénieurs, une banque, une appli de vote TV utilisée 4 soirs/an), choisissez un pattern et défendez-le — puis nommez le déclencheur qui ferait changer de pattern chaque entreprise. (3) Trouvez une vraie histoire de blog d'ingénierie « nous avons migré vers les microservices et nous l'avons regretté » et une histoire de succès ; consignez la différence (c'est presque toujours la taille d'équipe et la maturité du domaine).

Jalon : face à la description d'une entreprise, vous savez recommander un pattern *avec sa sortie de secours* — « commencez ici ; quand X arrive, évoluez vers Y » — en cinq minutes. Des chemins d'évolution, pas des verdicts : c'est ainsi que parlent réellement les architectes.

Module 9 (Semaines 18–20) : Les patterns de données, mondiaux et hybrides

Les architectures de données — lake vs warehouse. Les bases transactionnelles (Étape 1) font tourner l'entreprise ; l'analytique veut lui *poser des questions* sans la ralentir. Un **data warehouse** (entrepôt de données — Redshift, Snowflake, BigQuery) stocke des données *structurées et nettoyées*, optimisées pour l'analytique SQL rapide — à utiliser quand les questions sont connues et que les tableaux de bord doivent être rapides ; coûte plus cher au To, exige une discipline de modélisation en amont. Un **data lake** (lac de données — S3 + catalogue + moteurs de requête comme Athena) stocke *tout, brut, pour pas cher* — logs, flux de clics, images — et applique la structure au moment de la lecture ; à utiliser quand vous voulez tout garder maintenant et décider des questions plus tard ; mais sans gouvernance, il dégénère en la blague de l'industrie, le **data swamp** (marécage de données). Le consensus moderne est les deux, en couches (le « lakehouse ») : la vérité brute dans le lake, des marts organisés dans le warehouse, alimentés par des **pipelines ETL/ELT**. Les règles de l'architecte : l'analytique n'interroge jamais la base de production (des répliques ou des pipelines la nourrissent), et chaque jeu de données a un propriétaire, une entrée de catalogue et une politique de rétention — cette ligne de gouvernance est une phrase dans votre conception et une année de douleur si vous l'omettez.

Le multi-région : actif-passif vs actif-actif. Quand une région ne suffit pas — parce que le métier exige une DR face aux désastres à l'échelle d'une région, ou que les utilisateurs couvrent des continents — vous choisissez :

Pattern	Ce que c'est	À utiliser quand	À ne PAS utiliser quand	Coût/complexité
Actif-passif	Une région sert ; une région de secours détient les données répliquées, de « sauvegardes seulement » (froide) à « copie réduite en marche » (tiède), promue en cas de sinistre	Des RTO/RPO énoncés par le métier le justifient ; la conformité exige une histoire de DR	Personne n'a signé les RTO/RPO qui justifient la dépense (le besoin honnête de la plupart des entreprises est un bon multi-AZ + des sauvegardes inter-régions)	1,1×–1,7× le coût selon la tiédeur du secours ; complexité modérée — l'exigence tueuse est <i>un failover que vous répétez réellement</i> , sinon le secours est du théâtre
Actif-actif	Deux régions ou plus servent simultanément ; les utilisateurs routés vers la plus proche ; données répliquées dans les deux sens	Base d'utilisateurs mondiale voulant une latence locale ; RTO quasi nul réellement requis (réseaux de paiement, trading, gros SaaS)	Presque tous les autres — les <i>conflits d'écriture</i> inter-régions sont l'un des problèmes véritablement difficiles de l'informatique	2×+ le coût, la complexité la plus élevée de ce cours ; exige des magasins de données à résolution de conflits et des équipes seniors

La phrase de niveau entretien : “*Multi-AZ is table stakes; multi-region is a business case.*” (« Le multi-AZ est la mise de départ ; le multi-région est un dossier métier. ») Faites énoncer les RTO/RPO par le métier, chiffrez les options, et laissez les nombres choisir.

Le cloud hybride. Une partie sur site, une partie dans le cloud, reliées par VPN ou Direct Connect — pour la plupart des entreprises établies, pas un pattern mais une *réalité de dix ans* : des mainframes qui ne peuvent pas bouger, des systèmes d'usine contraints par la latence, des données réglementées épinglées sur site, et une migration (Étape 4) qui passe par là. À utiliser comme : un pont délibéré avec une direction de voyage. À ne PAS accepter : « hybride » comme euphémisme de « nous n'avons jamais décidé ». Notes de conception : l'identité doit être unifiée d'abord (une seule connexion à travers les deux mondes — ce que les descriptions de poste d'entreprise entendent par « intégration d'identité hybride ») ; nommez quels systèmes sont la source de vérité ; surveillez les coûts d'egress à travers le lien ; et attendez-vous à ce que le lien réseau *soit* un SPOF s'il n'est pas doublé. Complexité : deux parcs entiers de tout — la raison honnête pour laquelle les architectes poussent à rétrécir régulièrement le côté on-prem.

Vocabulaire :

Terme	Définition
OLTP / OLAP	Le traitement transactionnel (beaucoup de

Terme	Définition
	petites lectures/écritures rapides — fait tourner l'entreprise) vs le traitement analytique (d'immenses balayages — étudie l'entreprise). Séparez-les.
Data warehouse	Le stockage structuré et organisé, optimisé pour l'analytique SQL rapide (Redshift, Snowflake, BigQuery).
Data lake / data swamp	Le stockage bon marché de tout, brut, structuré à la lecture (S3 + Athena) / la version sans gouvernance, où les données vont pour se perdre.
ETL / ELT	Les pipelines qui déplacent les données des systèmes sources vers le lake/warehouse (Extract, Transform, Load — l'ordre varie).
Gouvernance / catalogue / rétention des données	Les règles de propriété, de documentation et de durée de vie de chaque jeu de données — une phrase de conception qui épargne une année de douleur.
Réplication (sync / async)	Copier les données en continu vers une autre base ou région — instantanément cohérente mais limitée par la distance vs légèrement en retard mais n'importe où. Le retard async est d'où vient le RPO.
Actif-passif / exercice de failover	La DR en région de secours / la répétition planifiée qui prouve que ça marche. Un failover non répété est un espoir.
Actif-actif	Plusieurs régions servant en même temps avec réplication bidirectionnelle. Puissant ; véritablement difficile ; généralement inutile.
Conflit d'écriture	Deux régions modifiant les mêmes données en même temps — la raison technique pour laquelle l'actif-actif est un territoire d'experts.
Cloud hybride / identité hybride	On-prem + cloud réunis en un seul parc / une seule connexion à travers les deux — la première chose à unifier.
Direct Connect	La ligne physique privée reliant le centre de données au cloud — l'ombilical hybride, doublé si ça compte.

Exercices : (1) ThaiTicket devient régional : concevez l'expansion à Singapour deux fois — actif-passif (Bangkok primaire) et actif-actif — avec les multiples de coût et les RTO/RPO de chacun ; écrivez la recommandation d'une page et *tranchez*. (2) Un détaillant a 15 ans de ventes

dans une base SQL de production et veut une « analytique prête pour l'IA » ; esquissez la conception lake + warehouse et écrivez les trois phrases de gouvernance. (3) Repérage de patterns : choisissez trois architectures de référence sur <https://aws.amazon.com/architecture/> et nommez chaque pattern du catalogue présent dans chacune.

Jalon — fin de l'Étape 3 : le gantelet du catalogue : une IA vous donne six scénarios en rafale ; pour chacun vous nommez le pattern, la raison, l'avertissement quand-NE-PAS, et le profil coût/complexité — en moins de cinq minutes chacun. Cet échange exact, à cette vitesse exacte, est le tiers central d'un vrai entretien d'architecte.

ÉTAPE 4 — LA MIGRATION ET LE MONDE RÉEL (Semaines 21–26)

La conception greenfield est la minorité du métier. La plupart de l'architecture se passe dans des entreprises qui existent déjà — avec des salles serveurs, du logiciel ancien dont l'activité dépend, des contrats et des lois. Cette étape est ce monde-là, et c'est là que « piloter les initiatives de migration et de modernisation » — un point de presque chaque description de poste que nous avons étudiée — s'enseigne.

Module 10 (Semaines 21–23) : La migration — déplacer une entreprise vers le cloud

Les 7 R — le vocabulaire partagé de toute conversation de migration. Pour chaque workload du parc, vous en choisissez un :

R	Signification	Quand	Effort / bénéfice
Retire (retirer)	L'éteindre — personne ne l'utilise vraiment	Chaque parc en compte 10–20 % ; trouvez-les d'abord	Trivial / économies immédiates — le meilleur R
Retain (conserver)	Laisser sur site, pour l'instant	Systèmes contraints par la latence, épinglés par la conformité, ou bientôt en fin de vie	Aucun / diffère le coût — un « pas encore » honnête
Rehost (« lift and shift »)	Déplacer vers des VM cloud tel quel	La vitesse compte, les applications sont stables, les compétences sont minces	Bas / sortie rapide du centre de données, mais peu de bénéfice cloud encore — un premier pas, pas une destination
Relocate (relocaliser)	Déplacer au niveau de l'hyperviseur (p. ex. flotte VMware vers VMware-sur-	Grands parcs virtualisés avec une échéance	Bas / le déplacement en masse le plus rapide ; une étape

R	Signification	Quand	Effort / bénéfice
	cloud)		intermédiaire
Repurchase (« drop and shop »)	Remplacer par du SaaS	Applications non différenciantes — e-mail, RH, CRM	Bas-moyen / des catégories entières quittent votre parc
Replatform (« lift, tinker, shift »)	De petites améliorations en cours de route — base autogérée — RDS, application — conteneurs	Le milieu pragmatique : bénéfice réel, risque borné	Moyen / le R bête de somme de la plupart des migrations
Refactor / re-architect (réarchitecturer)	Réécrire pour le cloud natif (patterns de l'Étape 3)	Les systèmes cœur différenciants dont les limites font mal à l'activité	Élevé / le bénéfice le plus élevé — dépensez-le sur les quelques workloads qui le méritent

La méthode de migration : évaluer – mobiliser – migrer. Évaluer : inventoriez tout (outils de découverte plus l'archéologie qui consiste à interroger les gens), et notez chaque workload sur sa valeur métier et sa difficulté de migration — livrables : un inventaire applicatif avec un R par ligne, et un **dossier métier (business case)** (TCO de rester vs bouger ; soyez honnête sur le fait que la facture *monte* pendant la période de chevauchement où les deux parcs tournent — les dirigeants qu'on n'a pas prévenus de la bulle de migration deviennent des dirigeants qui annulent les migrations à mi-chemin). **Mobiliser :** construisez la **landing zone** — la fondation cloud préconstruite et gouvernée *avant* que le premier workload n'atterrisse : structure multi-comptes (des comptes séparés par environnement et par équipe, pour que les rayons d'explosion restent petits), identité et journalisation centralisées, le hub réseau (VPC en hub-and-spoke, le Direct Connect vers le on-prem) et les **garde-fous (guardrails)** — des politiques automatisées faisant du chemin sécurisé, tagué et conforme le chemin par défaut (AWS Control Tower est le kit de démarrage managé). Les descriptions de poste disent « concevoir la landing zone dont hérite chaque workload » — c'est cela. La sauter pour « juste commencer à migrer » recrée le centre de données désordonné dans le cloud à coût plus élevé ; c'est la signature classique de la migration ratée. **Migrer par vagues :** groupez les workloads en vagues de quelques-uns, ordonnées facile-d'abord : la Vague 1 est délibérément à faibles enjeux (apprendre la machinerie là où les erreurs sont bon marché), les vagues suivantes prennent les bijoux de la couronne avec des bascules répétées, chacune avec un plan de retour arrière et une période d'**hypercare** de vigilance renforcée. Pour chaque déplacement de base de données, les deux questions qui comptent sont les nombres du Module 4 déguisés : combien d'indisponibilité la bascule peut-elle prendre (RTO) — et la réplication continue avec une bascule courte existe quand la réponse est « presque aucune ».

Vocabulaire :

Terme	Définition
7 R	Retire, retain, rehost, relocate, repurchase, replatform, refactor — le menu de migration

Terme	Définition
Découverte / inventaire applicatif	par workload. (Vous entendrez aussi « 6 R » — la liste plus ancienne sans relocate.) Découvrir ce qui tourne réellement (outils + entretiens) / la liste qui en résulte, une ligne par workload, avec propriétaire, dépendances et son R.
Cartographie des dépendances	Tracer ce qui parle à quoi — la raison pour laquelle les workloads migrent en groupes, et pour laquelle les migrations sans elle échouent dès le premier jour.
Dossier métier / bulle de migration	L'argumentaire TCO rester-vs-bouger / la bosse de coût temporaire pendant que les deux parcs tournent. Prévenez, ou soyez pris en embuscade.
Landing zone	La fondation cloud gouvernée et préconstruite — comptes, identité, réseau, journalisation, garde-fous — dont hérite chaque workload. Construite <i>avant</i> la migration.
Stratégie multi-comptes	Des comptes cloud séparés par environnement/équipe, pour que la facturation soit attribuable et le rayon d'explosion contenu.
Garde-fou (guardrail)	Une politique automatisée qui empêche ou signale les actions non conformes — la gouvernance en machinerie, pas en notes de service.
Control Tower	Le service managé d'AWS pour monter une landing zone multi-comptes avec garde-fous.
Plan par vagues	Le calendrier de migration en petits groupes, facile-d'abord, dépendances ensemble.
Cutover / rollback / hypercare	Le moment de la bascule vers la copie cloud / l'annulation répétée / la fenêtre de soutien renforcé après.

Exercices : (1) La migration sur papier : une IA génère l'inventaire d'une entreprise fictive de 40 serveurs (vous la retrouverez dans le Capstone 2) ; assignez un R à chaque ligne et défendez les dix choix les plus difficiles. (2) Esquissez une landing zone : diagramme de comptes, flux d'identité, réseau hub-and-spoke, cinq garde-fous que vous imposeriez dès le premier jour. (3) Écrivez l'avertissement « bulle de migration » en deux paragraphes à un CFO — s'entraîner à annoncer tôt les mauvaises nouvelles est le cardio de l'architecte.

Jalon : face à un inventaire d'environ 20 workloads avec descriptions, vous produisez un tableau R-par-workload défendable, un plan en trois vagues avec son raisonnement, et une esquisse de landing zone — en une seule séance.

Module 11 (Semaines 24-26) : Les contraintes — la loi, le legacy et le lock-in

La résidence des données et le droit de la vie privée comme données d'entrée d'architecture. Les lois sur les données personnelles — le PDPA thaïlandais, le GDPR (RGPD) européen et leurs cousines dans le monde — partagent une forme qu'un architecte doit connaître sur le bout des doigts : les données personnelles exigent une base légale (souvent le consentement) ; les individus ont des droits (accès, rectification, effacement — votre conception doit pouvoir *trouver et supprimer les données d'une seule personne*, ce qui est difficile si vous les avez éparpillées dans des copies non gouvernées) ; les brèches portent une obligation de **notification sous 72 heures** (votre journalisation doit vous permettre de raconter l'histoire en moins) ; et les **règles de transfert transfrontalier** contraignent où les données peuvent physiquement vivre — ce qui est une contrainte de *sélection de région*, une contrainte de *conception de réplication* (cette copie analytique à Singapour peut être un événement juridique) et une raison d'être des régions locales. La méthode de l'architecte, dans l'ordre : classifiez les données (qu'est-ce qui est personnel ?), cartographiez leurs voyages (chaque magasin, chaque copie, chaque frontière traversée — les flux que personne n'a dessinés sont là où vivent les violations), puis concevez les contrôles : des régions conformes à la résidence, le chiffrement, des calendriers de rétention, et un chemin de suppression qui atteint réellement l'expiration des sauvegardes. Dites « data protection by design » (protection des données dès la conception) — c'est la formule que les deux lois utilisent, et c'est littéralement votre intitulé de poste en une phrase.

L'intégration du legacy. Le système de facturation sur mainframe ne bougera pas cette année, et la brillante application cloud doit lui parler. Les patterns : une **anti-corruption layer** (couche anti-corruption) — un service de traduction entre le neuf et le vieux, pour que les formes de données étranges du système legacy ne s'infiltrant pas dans votre nouvelle conception et ne la corrompent pas ; le **strangler fig** (figuier étrangleur) — routez le trafic à travers une façade, puis détachez les fonctions du système legacy une par une jusqu'à ce que, des années plus tard, il puisse être éteint (nommé d'après le figuier qui enveloppe lentement son arbre hôte ; il bat les réécritures big-bang, qui échouent à un taux légendaire) ; des ponts batch et des dérivations d'événements pour les données qui doivent circuler entre les mondes ; et le respect de la physique de latence des appels bavards cloud-vers-on-prem (Direct Connect aide ; mieux encore, concevez le bavardage hors du système). La règle : **contenez le legacy, ne l'attrapez pas** — chaque nouveau composant devrait être construit comme si le système legacy était déjà parti.

L'économie du vendor lock-in. Chaque service managé commode approfondit votre mariage avec un fournisseur ; la portabilité (abstractions multi-cloud, tout autogéré) est une option réelle qui coûte de l'argent réel en complexité et en vitesse sacrifiée. Ni l'un ni l'autre n'est un péché — le mode d'échec n'est pas de *choisir* le lock-in, c'est de ne pas le *remarquer*. L'outil de l'architecte est une estimation du coût de sortie écrite dans la conception : « Utiliser DynamoDB économise ≈2 années-ingénieur maintenant ; changer plus tard ≈ une réécriture de 6 mois de la couche données — nous l'acceptons, et voici la frontière d'interface qui réduirait la réécriture. » Trois phrases, honnêtement chiffrées, décision enregistrée (les ADR de l'Étape 5) — c'est cela,

la gestion mûre du lock-in. Le multi-cloud *comme stratégie* (faire tourner le même workload de façon portable sur deux clouds) est généralement la réponse la plus chère possible et n'est acheté que par les organisations dont l'échelle ou le régulateur l'exige ; le multi-cloud *comme fait* (des workloads différents sur des clouds différents par histoire ou par adéquation) est la vie normale.

Vocabulaire :

Terme	Définition
PDPA / GDPR (RGPD)	Les lois thaïlandaise et européenne de protection des données personnelles — la paire modèle des contraintes de droit de la vie privée dans le monde.
Data protection by design	Construire les contrôles de vie privée dans l'architecture dès la première esquisse — la formule légale et le devoir de l'architecte.
Base légale / consentement	La justification juridique requise pour traiter des données personnelles.
Droit à l'effacement	Le droit d'un individu à voir ses données supprimées — que votre conception doit rendre <i>possible</i> .
Cartographie des flux de données	Tracer chaque magasin, copie et traversée de frontière d'une catégorie de données. Là où sont les flux non dessinés vivent les violations.
Transfert transfrontalier	Des données personnelles quittant le pays — réglementé ; fait de la conception de réplique une question juridique.
Anti-corruption layer	Un composant de traduction qui empêche les bizarreries d'un système legacy de s'infiltrer dans une nouvelle conception.
Strangler fig	Moderniser en routant à travers une façade et en remplaçant le système legacy pièce par pièce jusqu'à pouvoir l'éteindre.
Réécriture big-bang	Remplacer un système d'un coup, basculé en un jour. Taux d'échec légendaire ; le strangler fig existe à cause d'elle.
Coût de sortie / coût de changement	Le coût, chiffré de façon réaliste, de quitter un fournisseur ou un service — le nombre qui transforme le lock-in de peur en économie.
Multi-cloud (stratégie vs fait)	Tourner délibérément de façon portable sur plusieurs clouds (cher, rarement justifié) vs simplement avoir des workloads sur

Terme	Définition
	plusieurs (normal).

Exercices : (1) Cartographiez les flux des données clients de ThaiTicket : chaque magasin, chaque copie (n'oubliez pas les logs, les sauvegardes, le pipeline analytique, les exports de l'équipe support), chaque frontière ; puis écrivez le chemin d'effacement. Sentez comme la carte trouve des problèmes que le diagramme cachait. (2) Concevez le plan strangler-fig pour un système d'inventaire on-prem de 20 ans, les trois premiers détachements nommés. (3) Écrivez le paragraphe de lock-in façon DynamoDB (bénéfice maintenant, coût de sortie plus tard, accepté ou atténué) pour trois services que vous utiliseriez réellement.

Jalon — fin de l'Étape 4 : le gantelet des contraintes : un cahier des charges truffé des trois (« assureur thaïlandais, données clients, système de polices sur mainframe, conseil d'administration nerveux quant à la dépendance à AWS ») — vous produisez une approche d'une page touchant la résidence, l'intégration et le lock-in, chacune avec un pattern nommé et un coût honnête. Quand les contraintes vous enthousiasment plus que ne le ferait le greenfield — car les contraintes sont là où les architectes gagnent plus que les dessinateurs de diagrammes — l'Étape 4 est terminée.

ÉTAPE 5 — L'ARTISANAT DE L'ARCHITECTE (Mois 7-12)

Tout ce qui précède vous a rendu capable de *concevoir*. Cette étape vous rend capable de *travailler comme architecte* — les documents, revues, présentations et preuves dont le rôle est réellement fait, plus les certifications qui vous font passer les RH, et trois capstones qui deviennent votre portfolio.

Module 12 (Mois 7-8) : Les livrables — ADR, jeux de diagrammes et conduite de revues

Les Architecture Decision Records (ADR). Un ADR est un document d'une page capturant une décision significative : ce que nous avons choisi, ce que nous n'avons pas choisi, et pourquoi — écrit *au moment où la décision est prise*, numéroté, et gardé pour toujours dans le dépôt du projet. Pourquoi les descriptions de poste les nomment : dans deux ans, quelqu'un demandera « pourquoi diable est-ce du DynamoDB ? » et l'ADR répond en trente secondes — avec le contexte, les contraintes et les alternatives honnêtement considérées. Les équipes avec des ADR ne rejugent rien ; les équipes sans eux se disputent en rond chaque année. Le modèle — mémorisez-le :

```
# ADR-014: Use DynamoDB for the session store
Status: Accepted           Date: 2026-08-13
Context:   What situation forced a decision? (Load, constraints,
deadlines – the facts.)
Decision:   What we chose, in one sentence, active voice: "We will..."
Options considered: 2–3 real alternatives, each with honest pros/cons
– including the one you rejected reluctantly.
```

Consequences: What becomes easier; what becomes harder; the risks we accept; the exit cost.

Écrivez un ADR par décision significative dans chaque capstone. Un portfolio d'entretien contenant de vrais ADR est rare et d'une efficacité dévastatrice.

Le jeu de diagrammes — un système, trois niveaux de zoom. La vraie documentation d'architecture est un *jeu* (le modèle C4 a popularisé cette discipline) : le **diagramme de contexte** — le système en une boîte, avec ses utilisateurs et les systèmes externes qu'il touche ; la vue du dirigeant et du nouvel arrivant, et celle que vous présenterez à la direction. Le **diagramme de conteneurs** — on zoome : les grandes pièces en fonctionnement (application web, API, base de données, file, cache) et comment elles se parlent ; la vue d'ingénierie du quotidien, en gros vos dessins des Étapes 1–3. Le **diagramme de déploiement** — où tout tourne physiquement : région, AZ, VPC, sous-réseaux, groupes de scaling ; la vue pour les revues, la sécurité et les opérations. Un système, trois audiences, trois diagrammes — tenus à jour, datés, en gestion de versions à côté des ADR. Un diagramme qu'on ne peut ni trouver ni croire est un diagramme qui n'existe pas.

Mener une revue Well-Architected. Vous connaissez les six piliers (Étape 2) ; voici la réunion elle-même. **Avant** : choisissez le workload et le périmètre, mettez le jeu de diagrammes à jour, invitez les gens qui *exploitent* la chose (pas seulement ses concepteurs), et donnez le ton — c'est un bilan de santé au bénéfice de l'équipe, pas un audit pour le dossier de quiconque. **Pendant (une demi-journée)** : parcourez les piliers avec les jeux de questions du cadre (AWS les publie, avec un Well-Architected Tool gratuit dans la console qui structure tout l'exercice — <https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html>) ; pour chaque réponse, notez le constat, et *priorisez au fil de l'eau* — une poignée de risques élevés, pas une centaine de chipotages. Votre ton de questionnement est la revue : « aidez-moi à comprendre ce qui se passe quand le prestataire de paiement expire » ouvre des portes que « vous ne gérez pas les timeouts ? » claque. **Après** : un rapport court — les principaux constats, chacun avec risque, effort et recommandation — et les actions d'amélioration atterrissent dans le vrai backlog de l'équipe avec des propriétaires, sinon la revue était du théâtre. Entraînement : menez une revue complète contre votre propre conception du Capstone 1 ; trouver ses propres failles de façon structurée est la compétence à l'effet composé le plus rapide de ce cours.

Exercices : (1) Rédigez rétroactivement les ADR de cinq grandes décisions de vos conceptions de journal des Étapes 2–3 — vous en trouverez au moins une que vous ne savez plus justifier, et c'est la leçon. (2) Produisez le jeu complet de trois diagrammes pour ThaiTicket. (3) Menez la revue simulée ci-dessus, produisez le rapport de constats d'une page.

Jalon : un inconnu (ou une IA qui en joue un) peut prendre votre dossier ThaiTicket — trois diagrammes + cinq ADR — et répondre correctement à « qu'est-ce que c'est, comment ça tourne, et pourquoi est-ce construit ainsi ? » sans vous dans la pièce. Ce dossier *est* le livrable du métier d'architecte.

Module 13 (Mois 8-9) : L'artisanat humain — présenter, chiffrer, encaisser les objections et mentorer

Présenter aux dirigeants vs aux ingénieurs — une conception, deux langues. Les dirigeants achètent des résultats ; les ingénieurs achètent des mécanismes. Aux dirigeants : commencez par la décision et le chiffre (« Cette conception soutient l'objectif du million de clients à \$1,90 par commande, et voici les deux décisions dont j'ai besoin de vous ») ; une diapositive, le diagramme de *contexte*, les risques formulés comme des risques métier (chiffre d'affaires, conformité, réputation), et ne dites jamais « Kubernetes » quand « la plateforme » suffit. Préparez-vous aux trois questions que les dirigeants posent toujours : *combien ça coûte, qu'est-ce qui pourrait mal tourner, pourquoi pas l'option moins chère ?* Aux ingénieurs : commencez par le problème et les contraintes *avant* la solution (les ingénieurs qui ressentent le problème acceptent la solution ; les ingénieurs à qui l'on tend un verdict chassent les failles par principe), montrez les diagrammes de conteneurs et de déploiement, nommez vous-même les compromis et les options rejetées (la crédibilité vient de ce que vous admettez), et laissez une vraie place au changement de la conception — les questions dans la salle sont de la revue gratuite. La pire habitude d'architecte est un seul jeu de diapositives pour les deux audiences ; la meilleure habitude est d'écrire le résumé exécutif *d'abord*, car si la conception ne survit pas à la compression en cinq phrases, elle n'est pas terminée.

Estimer les coûts pour une proposition. La méthode : décomposez la conception en ses ~10 composants porteurs de coûts ; chiffrez chacun à la charge attendue dans le calculateur de prix ; **énoncez vos hypothèses par écrit** (requêtes/jour, croissance des données, egress — les hypothèses sont l'estimation ; quand elles changent, le chiffre bouge la conscience tranquille) ; appliquez la stratégie de réservation aux parties stables ; ajoutez des scénarios de croissance (aujourd'hui, 2x, 10x — les dirigeants retiennent le chiffre du 10x) ; et présentez une *fourchette avec ses moteurs* (« \$55-70k/mois, tirés principalement par l'egress — voici le levier »), jamais un chiffre unique à la fausse précision. Incluez la bulle de migration s'il y en a une. Sous-estimer pour gagner l'approbation est le péché classique du jeune architecte ; le projet s'en souvient.

Encaisser les objections. On vous poussera sur le coût (« trop cher » — retournez au triangle : « nous pouvons couper \$30k en acceptant le mono-AZ — voici le calcul de la panne ; à vous de trancher, et je le consigne » — rendre l'échange explicite et *enregistré* convertit la plupart des objections soit en accord, soit en risque accepté en connaissance de cause, les deux très bien) ; sur le goût (« un ingénieur préfère une autre stack » — plaidez-la au mieux à voix haute, puis ramenez à des critères, pas à des préférences : adéquation aux NFR, compétences de l'équipe, écosystème, coût de sortie — et quand c'est vraiment serré, laissez gagner la préférence de l'ingénieur qui implémente, achetant l'engagement à bas prix) ; et sur l'autorité (« l'ami du CTO dit d'utiliser X » — ne combattez jamais l'opinion par l'opinion ; demandez les critères, faites passer X par la même évaluation publique que tout le reste, laissez la matrice répondre poliment). Et quand vous avez tort — vous concevrez quelque chose qui échoue ; tous les architectes le font — le geste est le post-mortem sans blâme appliqué à vous-même, en public, avec l'ADR mis à jour. Rien ne construit plus vite une crédibilité de dix ans ; rien ne la détruit plus vite que de défendre un cadavre.

Mentorer les ingénieurs. La formule des descriptions de poste est « mentorer les ingénieurs qui construisent selon vos standards », et les formes concrètes sont : **des permanences de revue de conception** (une porte ouverte fixe, pour que le conseil arrive avant le code, pas après) ; **relire leurs conceptions avec bienveillance** — des questions avant les verdicts, et toujours le *pourquoi* derrière le standard (un garde-fou expliqué recrute un allié ; un garde-fou imposé recrute un contournement) ; **déléguer de vraies décisions** avec un filet de sécurité (« tu portes la conception du cache ; voici les contraintes ; je relirai, et je soutiendrai ton choix ») ; et **enseigner en public** — chaque ADR, chaque compte rendu de revue, chaque session brown-bag vous démultiplie au-delà de vos propres heures. La vérité de carrière, sans détour : un architecte génie solitaire plafonne ; un architecte qui a fait grandir cinq ingénieurs en collègues capables de concevoir dirige la pratique.

Exercices : (1) Présentez ThaiTicket deux fois — une version exécutive de 5 minutes et une version ingénierie de 20 minutes — enregistrez les deux, et écoutez le jargon qui fuit dans la première. (2) Produisez la proposition de coûts écrite complète (hypothèses, fourchette, moteurs, scénarios 2×/10×). (3) Théâtre d'objections avec une IA : trois manches — attaque sur les coûts par un CFO, préférence de stack par un ingénieur senior, coup d'autorité de l'ami-du-CTO — consignez ce qui a marché. (4) Relisez la conception d'un junior (une IA peut en générer une défaillante) par écrit, questions d'abord, et faites noter votre ton par l'IA.

Jalon : le doublé : livrez le pitch exécutif et survivez à vingt minutes de questions-réponses mixtes hostiles-mais-justes (panel d'IA : un CFO, un ingénieur sceptique) sans mélange de jargon, sans défensive, et sans un seul compromis non écrit.

Module 14 (Mois 9-12) : Les certifications et les trois capstones

Le parcours de certification, avec des délais honnêtes. Les certifications ne font pas de vous un architecte — les treize modules précédents le font — mais elles vous obtiennent des entretiens, et les préparer câble les services du fournisseur dans vos doigts. L'échelle, au rythme d'une heure par jour : **AWS Certified Solutions Architect – Associate (SAA-C03)** — 2-3 mois ; le titre d'architecte le plus demandé dans les annonces, et après les Étapes 1-3, une grande partie ressemblera à de la révision avec des noms de services attachés ; passez-le en premier.

AWS Certified Solutions Architect – Professional (SAP-C02) — 4-8 mois après l'Associate ; basé sur des scénarios, véritablement difficile, et la ligne de CV la plus forte de ce domaine ; la matière migration et multi-comptes de l'Étape 4 est la moitié de son programme. **Azure AZ-305 (Azure Solutions Architect Expert)** — à ajouter si votre marché est très Microsoft (la plupart des marchés d'entreprise sont au moins bilingues) ; comptez 2-3 mois, la connaissance d'AWS se transférant à fort rabais (commencez sur <https://learn.microsoft.com/en-us/credentials/certifications/azure-solutions-architect/>).

TOGAF Foundation — optionnel, 2-4 semaines ; certaines grandes entreprises le demandent ; il certifie la *méthode* et le vocabulaire d'architecture d'entreprise plutôt que la compétence cloud. La séquence pour ce cours : étude du SAA en parallèle des Mois 7-8, examen vers le Mois 8 ; puis soit le SA Pro (voie technique profonde), soit l'AZ-305 (voie de largeur) jusqu'au Mois 12, avec le SA Pro achevé au Mois 14-16 au rythme honnête.

Les trois capstones. Chacun est un dossier complet, digne d'un portfolio : jeu de diagrammes à trois niveaux, 5+ ADR, estimation de coûts avec hypothèses, et une revue Well-Architected auto-

menée avec ses constats. Comptez 3–4 semaines chacun. Ces trois artefacts, plus votre journal, *sont* votre preuve d’entretien de la « conception de solution de bout en bout ».

Capstone 1 — le greenfield : une plateforme e-commerce thaïlandaise pour 1 M d’utilisateurs. Cahier des charges : une marketplace, 1 M d’utilisateurs inscrits, 50 k simultanés au pic de vente flash, mobile-first, intégration de paiement type PromptPay, conforme PDPA, disponibilité 99,9 %, attentive au budget d’amorçage. Doit inclure : choix de pattern avec chemin d’évolution, stratégie de cache, conversation RTO/RPO écrite en dialogue, estimation d’économie unitaire, carte des flux de données personnelles.

Capstone 2 — le brownfield : migrer une entreprise on-prem de 40 serveurs. Cahier des charges : une entreprise de logistique thaïlandaise ; 40 serveurs (ERP sur Oracle, application de gestion d’entrepôt vieille de 12 ans, partages de fichiers, AD, applications diverses par service — générez l’inventaire complet avec une IA et gelez-le) ; le bail du centre de données expire dans 14 mois ; le conseil veut partir. Doit inclure : le tableau complet d’inventaire en 7 R, le plan en trois vagues avec le raisonnement des dépendances, la conception de la landing zone, la chronologie de coûts de la bulle de migration, et le mémo au CFO.

Capstone 3 — le difficile : la DR multi-région d’une fintech. Cahier des charges : une entreprise de paiements réglementée à $RTO \leq 15$ minutes / $RPO \approx 0$ pour le grand livre, avec des contraintes de résidence sur les données clients et un test de failover annuel observé par le régulateur. Doit inclure : l’analyse actif-passif vs actif-actif avec les coûts, la conception de réplication avec la carte de résidence, le runbook de failover, et le plan d’exercice.

La grille d’auto-évaluation (appliquez-la à chaque capstone, honnêtement, dans votre journal) :

Dimension	1 — Pas encore	3 — Solide	5 — Embauchez cette personne
Exigences	A sauté à la solution	NFR énoncées et tracées jusqu’aux choix de conception	Triangle des compromis positionné, conflits remontés au « métier », décisions extraites
Qualité de conception	Des SPOF restent ; pattern inadapté	Patterns corrects, multi-AZ, pannes gérées	Raisonnement quand-NE-PAS montré ; chemin d’évolution ; la conception la plus simple qui satisfait les exigences
Six piliers	Piliers ignorés	Chaque pilier visiblement traité	La revue auto-menée a trouvé de vraies failles — et la conception a été révisée en réponse
Coûts	Pas de chiffres	Estimation détaillée,	Fourchette avec

Dimension	1 — Pas encore	3 — Solide	5 — Embauchez cette personne
		hypothèses énoncées	moteurs, économie unitaire, scénario 10×, stratégie de réservation
Documentation	Diagramme seul	Jeu à trois niveaux + ADR, notation correcte	Un inconnu pourrait répondre à « quoi/comment/pour quoi » à partir du seul dossier
Communication	Un artefact pour toutes les audiences	Les versions dirigeant et ingénieur existent	Un résumé en cinq phrases qui survit ; les objections pré-répondues par écrit

Notez chaque dimension 4+ sur les trois capstones — en révisant jusqu'à y parvenir — et vous avez terminé le cours.

Dix questions d'entretien d'architecte (avec des réponses solides)

Entraînez-les à voix haute ; la force réside dans la *structure* de chaque réponse, que vous possédez désormais.

1. “Design a URL shortener / ticketing site / photo app for a million users.” (« Concevez un raccourcisseur d'URL / un site de billetterie / une appli photo pour un million d'utilisateurs. ») La forme solide : les exigences d'abord, à voix haute (« Les lectures dominent les écritures ? Cible de disponibilité ? Budget ? ») → position sur le triangle → squelette trois tiers ou serverless-first avec multi-AZ, cache, CDN → nommer les compromis sans y être invité → finir avec l'ordre de grandeur des coûts et le chemin d'évolution à 10×. Les intervieweurs notent les *questions que vous posez*, pas les boîtes que vous dessinez.

2. “When would you choose NoSQL over a relational database?” (« Quand choisiriez-vous NoSQL plutôt qu'une base relationnelle ? ») « Je pars par défaut sur SQL — garanties de justesse et compétences universelles — jusqu'à ce qu'une raison nommée l'emporte : échelle horizontale extrême, schéma flexible, ou accès clé-valeur à la milliseconde. Alors je choisis la saveur NoSQL qui colle au motif d'accès, et j'écris l'ADR en incluant ce que nous abandonnons : les jointures, et certaines sémantiques de cohérence. »

3. “Explain RTO and RPO, and how they drive design.” (« Expliquez RTO et RPO, et comment ils pilotent la conception. ») Définir les deux nettement → « ce sont des décisions métier aux étiquettes de prix exponentielles » → l'échelle : sauvegardes nocturnes → réplication continue → secours tiède → actif-actif, avec un coût qui monte à chaque barreau → « mon

travail est de faire choisir le métier en connaissance de cause, puis de concevoir exactement pour le chiffre — et de le répéter, car un failover non testé est un espoir ».

4. “Monolith or microservices?” (« Monolithe ou microservices ? ») Le verdict du Module 8, mot pour mot : commencez en monolithe modulaire ; extrayez quand une douleur de montée en charge d’équipe ou de charge arrive réellement ; les microservices sont un outil de montée en charge d’équipe qui prélève une taxe de systèmes distribués. Points bonus pour « je préfère faire tourner un bon monolithe qu’un mauvais système distribué ».

5. “How do you handle a large cloud bill / cost optimisation?” (« Comment gérez-vous une grosse facture cloud / l’optimisation des coûts ? ») « La visibilité d’abord — tagging et showback ; puis la récolte par ordre d’effort : tuer les ressources mortes, rightsizer à partir des mesures, hiérarchiser le stockage, faire dormir le hors-prod, puis réserver la base stable. Et je rapporte l’économie unitaire, pas les totaux — une facture qui grossit avec un coût par commande qui baisse est un succès, pas un problème. »

6. “How would you migrate a legacy on-prem application?” (« Comment migreriez-vous une application legacy on-prem ? ») « Évaluer avant de bouger — inventaire, dépendances, et un R par workload parmi les 7 R ; construire la landing zone avant la première vague ; des vagues facile-d’abord avec des bascules répétées et des plans de retour arrière ; et pour la base de données joyau de la couronne, une bascule par réplication dimensionnée à l’indisponibilité que le métier a signée. Aussi : l’avertissement honnête de la bulle de migration dès le départ. »

7. “How do you secure a cloud architecture?” (« Comment sécurisez-vous une architecture cloud ? ») Parcourez les couches : identité (moindre privilège, MFA, pas de root au quotidien) — réseau (sous-réseaux privés, segmentation, surface d’attaque minimale) — données (chiffrement au repos/en transit, classification, résidence) — détection (journaux d’audit, alertes) — « et dès la conception, pas ajouté après coup — le contrôle de sécurité le moins cher est la décision d’architecture qui supprime entièrement le risque, comme ne jamais toucher aux données de carte brutes ».

8. “Tell me about a design decision you got wrong.” (« Parlez-moi d’une décision de conception que vous avez ratée. ») Ils testent l’ego, pas l’histoire. La forme : un exemple réel (de capstone) — ce que vous croyiez — ce que la réalité a dit — le post-mortem, l’ADR mis à jour, le motif que vous vérifiez désormais. Un architecte qui ne peut pas produire cette réponse est un architecte qui n’a jamais été relu.

9. “How do you explain a complex technical decision to a non-technical executive?” (« Comment expliquez-vous une décision technique complexe à un dirigeant non technique ? ») « La décision et le chiffre métier d’abord, le mécanisme seulement sur demande ; le diagramme de contexte, pas celui de conteneurs ; les risques en termes de chiffre d’affaires-conformité-réputation ; et j’apporte les deux décisions dont j’ai besoin d’eux, écrites comme des choix avec leurs étiquettes de prix — les dirigeants décident entre des options ; ils n’approuvent pas des mystères. »

10. “An engineer strongly disagrees with your design. What do you do?” (« Un ingénieur est en profond désaccord avec votre conception. Que faites-vous ? ») « D’abord je plaide sa position au mieux, à voix haute — il a peut-être raison, et la revue qui change ma conception est la revue qui fonctionne. Si c’est vraiment serré, les critères décident

(adéquation aux NFR, compétences de l'équipe, coût de sortie), pas l'ancienneté — et je laisse la préférence de l'implémenteur gagner les égalités, car l'engagement n'est pas cher à ce prix. Dans tous les cas, la décision et l'option rejetée vont dans l'ADR, pour que nous n'en débattions jamais deux fois. »

Le programme en une page

Quand	Module	Focus	Preuve externe
Semaines 1–2	1	Rappel cloud ; triangle des compromis ; NFR	Journal de conception ouvert
Semaines 3–4	2	Calcul/stockage/base de données/réseau en tant que décisions	—
Semaines 5–6	3	Lire et dessiner des diagrammes ; fluidité trois tiers	Jalon du tableau blanc en 15 minutes
Semaines 7–8	4	Fiabilité : multi-AZ, auto-scaling, RTO/RPO	—
Semaines 9–10	5	Sécurité : moindre privilège, zero trust, segmentation	—
Semaines 11–12	6	Performance et coûts : cache, CDN, réservations, économie unitaire	Conception chiffrée dans le calculateur
Semaines 13–14	7	Excellence opérationnelle et durabilité ; exercice des six piliers	Revue simulée des six piliers
Semaines 15–17	8	Patterns : monolithe/microservices, événementiel, serverless	—
Semaines 18–20	9	Patterns : data lake/warehouse, multi-région, hybride	Gantelet du catalogue
Semaines 21–23	10	Migration : 7 R, vagues, landing zones	Migration sur papier
Semaines 24–26	11	Contraintes : PDPA/GDPR, legacy,	Gantelet des

Quand	Module	Focus	Preuve externe
Mois 7–8	12	économie du lock-in ADR, jeux de diagrammes, conduite de revues Well-Architected	contraintes Dossier ThaiTicket ; examen SAA-C03 (2–3 mois de préparation)
Mois 8–9	13	Présenter, chiffrer les propositions, objections, mentorat	Doublé enregistré
Mois 9–12	14	Capstones 1–3	Portfolio complet ; SA Pro (4–8 mois) ou AZ-305 en cours

Un mot de clôture de votre professeur. À vingt-six semaines, vous saviez concevoir ; à douze mois, vous savez *exercer* — il y a une différence, et c'est la différence que ce cours a été construit pour combler. Les habitudes sont désormais la carrière : dessinez avant d'argumenter, écrivez l'ADR le jour où vous décidez, chiffrez ce que vous proposez, nommez le compromis avant qu'on ne le demande, et faites grandir les ingénieurs autour de vous jusqu'à ce que vos standards survivent à votre présence dans la pièce. L'architecture est ce rare métier technique qui devient *meilleur* à mesure qu'on y vieillit, car sa matière première est le jugement, et le jugement se compose. Continuez le journal. Dans quarante conceptions, vous n'aurez plus besoin de ce cours — vous le corrigerez.

Sources

Descriptions de poste et définitions de rôle utilisées pour le tableau de correspondance des exigences (consultées en août 2026) : le [KORE1 Cloud Architect job description template](#) · la [4 Corner Resources Cloud Architect job description](#) · les [Solution Architect responsibilities in the Azure Well-Architected Framework](#) de Microsoft · l'offre [Pre-Sales Solutions Architect – AWS](#) d'Arpio · l'offre [Solution Architect – Enterprise \(via Built In\)](#) d'Intel · l'offre [Cloud Domain Architect](#) de Halliburton. Estimations des temps de préparation aux certifications : l'[enquête de temps d'étude SAA-C03](#) de CBT Nuggets et le [guide de préparation SAP-C02](#) de Whizlabs ; définitions des certifications d'après [Microsoft Learn \(AZ-305\)](#) et l'[AWS Well-Architected Framework](#). Volume compagne du « Cours du Leader Cloud » dans la série B4LCILC.

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 Architecte Cloud

L'arithmétique des neufs — SLA, SLO et budgets d'erreur

La disponibilité se cite en neufs, et la première chose qu'un architecte fait avec un neuf est de le convertir en minutes — parce que les minutes sont ce que le business peut ressentir. Un mois de 30 jours contient 43 200 minutes :

Disponibilité	Temps d'arrêt autorisé par mois
99,9 % (« trois neufs »)	environ 43 minutes
99,99 % (« quatre neufs »)	environ 4,3 minutes
99,999 % (« cinq neufs »)	environ 26 secondes

Cinq neufs — le nombre que les dirigeants aiment prononcer à voix haute — est hors de portée de toute récupération opérée par un humain ; le basculement doit être automatique, sinon la promesse est une fiction. Chaque neuf supplémentaire coûte environ dix fois plus à ingénierer ; convertir en minutes est la façon de faire choisir une partie prenante en connaissance de cause.

Trois acronymes portent la conversation. Un **Service-Level Indicator (SLI)** — indicateur de niveau de service — est une mesure. Un **Service-Level Objective (SLO)** — objectif de niveau de service — est la cible interne sur cette mesure. Un **Service-Level Agreement (SLA)** — accord de niveau de service — est le contrat externe, avec de l'argent attaché à un manquement. La discipline : le SLO est toujours plus strict que le SLA — vous visez à l'intérieur de la ligne que vous êtes payé pour tenir — et le SLI mesure ce que les utilisateurs vivent : un serveur qui est allumé tout en renvoyant des erreurs est en panne dans tous les sens qui comptent.

Le budget d'erreur rend les SLO opérationnels : l'écart entre la perfection et l'objectif, traité comme une ressource à dépenser. Un SLO de 99,9 % accorde 43 minutes de défaillance par mois ; les déploiements et les expérimentations puisent dedans ; quand il est épuisé, l'équipe arrête de livrer et travaille sur la fiabilité — mettant fin à la plus vieille dispute du logiciel, livrer contre stabilité, en remplaçant l'opinion par l'arithmétique.

Maintenant le piège : la disponibilité se multiplie le long d'une chaîne. Trois services en série, chacun honnêtement à 99,9 %, se composent en $0,999 \times 0,999 \times 0,999 \approx 99,7\%$ — environ 130 minutes par mois, trois fois ce qu'aucun service pris isolément n'admet. La redondance fonctionne dans l'autre sens : deux instances indépendantes à 99,9 % en parallèle ne tombent ensemble que $0,1\% \times 0,1\%$ du temps — voilà pourquoi le multi-zone est un si bon investissement. La phrase de niveau entretien d'embauche : *vous ne pouvez pas promettre plus de disponibilité que le produit de vos dépendances synchrones, donc soit vous raccourcissez la chaîne, soit vous abaissez la promesse.*

L'architecture à l'ère de l'IA — le RAG et ses cousins

En 2026, le catalogue de patterns a une nouvelle entrée sur laquelle les clients posent des questions chaque semaine : ce que c'est, quand cela vaut son prix, et quand cela vous fera honte. Une API de **Large Language Model (LLM)** — grand modèle de langage : Amazon Bedrock, Azure OpenAI Service — est, architecturalement, *une dépendance chère, lente et sans état*. Elle est facturée au **token** — un fragment de mot d'environ quatre caractères. Elle répond en secondes, pas en millisecondes, ne se souvient de rien entre les appels, et se trompe occasionnellement avec assurance — un composant aux modes de défaillance inhabituels, méritant les mêmes timeouts, les mêmes réessais, les mêmes disjoncteurs (circuit breakers) et les mêmes alarmes budgétaires que n'importe quelle dépendance, parce qu'une boucle d'appels hors de contrôle brûle de l'argent réel à l'heure.

Le modèle ne connaît pas vos documents ; la réponse standard est le **RAG — Retrieval-Augmented Generation** (génération augmentée par récupération) : au lieu d'enseigner vos données au modèle, vous *cherchez* les faits pertinents au moment de la question. Les documents sont découpés en morceaux (chunks), chacun converti en un **embedding** — une liste de nombres positionnant son sens dans l'espace — stocké dans une **base de données vectorielle** (pgvector sur AWS, AI Search sur Azure), qui renvoie les morceaux les plus proches. Ces morceaux sont collés dans le prompt : le modèle ne consulte pas vos données ; il résume les extraits que vous êtes allé lui chercher.

Au tableau blanc, le RAG, c'est deux pipelines : un **chemin d'ingestion** qui tourne à l'avance (documents découpés, convertis en embeddings, stockés) et un **chemin de requête** qui tourne à chaque demande — deux patterns déjà au catalogue, ce qui explique pourquoi les architectes assimilent cela en un après-midi.

Les compromis sont les trois habituels, aiguisés : le coût (chaque requête coûte de l'argent réel — mettez en cache les répétitions, routez les questions faciles vers un modèle moins cher), la latence (des secondes, pas des millisecondes — diffusez la réponse en streaming), et la vie privée (vos documents quittent votre périmètre ; si la base de données vectorielle ignore le contrôle d'accès au niveau du document, le chatbot résumera joyeusement le tableur des salaires pour quiconque le demande). Sachez quand ne *pas* le construire : beaucoup de

demandes « il nous faut un chatbot » sont une boîte de recherche déguisée à la mode, et les réponses exactes et auditables — sommes, soldes, grands livres — sont le mauvais travail pour un système qui a raison 97 % du temps. La phrase de niveau entretien d'embauche : *le RAG est un problème de recherche agrafé à une API de résumé.*

Chiffrer une conception avant que quiconque ne la construise

Le module Piliers de la performance et du coût du cours se clôt désormais par un exercice de conception complet : l'estimation sous le feu. Le brief : la direction veut une estimation de coût mensuel pour un produit de gestion documentaire avant d'approuver la construction. La charge de travail — 200 000 utilisateurs inscrits, 40 000 actifs par jour, chacun effectuant environ 25 requêtes API par jour et téléversant en moyenne 2 mégaoctets de documents par mois, conservés sept ans. La conception : un étage d'API réparti par un répartiteur de charge sur deux zones, une base de données PostgreSQL managée multi-zone, du stockage objet, un export nocturne de 50 gigaoctets vers un partenaire, et des environnements de développement et de staging séparés. La tâche : une estimation mensuelle d'une page en chiffres ronds, avec les hypothèses énoncées par écrit avant qu'aucun prix n'apparaisse, l'arithmétique de capacité montrée (utilisateurs → requêtes par seconde → nombre d'instances), et des scénarios aujourd'hui / 2× / 10×.

La liste de contrôle de l'exercice enseigne la discipline : chiffrer le calcul à partir de la charge de pointe, pas de la moyenne ; compter le serveur de secours de la base de données ; modéliser le stockage comme une croissance sur sept ans avec des tiers de cycle de vie ; calculer l'egress (le trafic sortant) plutôt que le deviner — la ligne que la plupart des estimations oublient et celle qui domine le plus souvent ; ne pas oublier les passerelles NAT (Network Address Translation — traduction d'adresses réseau), les environnements hors production, les sauvegardes, la supervision et le plan de support ; n'appliquer les remises d'engagement qu'à la base stable ; et finir par une fourchette avec ses moteurs nommés, plus une vérification de bon sens en économie unitaire — le coût par utilisateur actif et par mois. Un chiffre unique et précis est faux le jour même où il est prononcé.