

Khóa học Kiến trúc sư Cloud (The Cloud Architect Course)

Từ con số không đến việc thiết kế các hệ thống cloud — một khóa học B4LCILC

Ngày 13 tháng 8 năm 2026

Khóa học Kiến trúc sư Cloud (The Cloud Architect Course)

Từ chỗ chưa biết gì đến thiết kế được những hệ thống cloud mà một công ty dám đặt cược cả doanh nghiệp vào đó.

Khóa học này vận hành như thế nào

Bạn không rèn luyện để *vận hành* các hệ thống cloud. Bạn rèn luyện để **thiết kế** chúng: nhận lấy một bài toán kinh doanh bừa bộn (“chúng tôi cần bán cho một triệu khách hàng và không bao giờ mất một đơn hàng”), biến nó thành một bản kế hoạch kỹ thuật được vẽ ra, được định giá, có thể bảo vệ được, rồi thuyết phục cả những kỹ sư sẽ xây nó lẫn những lãnh đạo sẽ trả tiền cho nó. Đó là công việc của một **cloud architect** (kiến trúc sư cloud — cũng được đăng tuyển dưới tên *solutions architect*, *cloud solutions architect*, hay *cloud domain architect*), và nó là một nghề thủ công học được với một giáo trình rõ ràng — chính là giáo trình này.

Khóa học gồm năm giai đoạn. **Giai đoạn 1 (Tuần 1–6): Các khối xây dựng, dưới dạng quyết định.** Mọi thành phần cloud, được dạy lại không phải như một sự kiện mà như một *lựa chọn kèm đánh đổi* — vì đơn vị công việc của kiến trúc sư là quyết định. **Giai đoạn 2 (Tuần 7–14): Các miền thiết kế.** Sáu trụ cột của Well-Architected Framework — độ tin cậy, bảo mật, hiệu năng, chi phí, vận hành, tính bền vững — mỗi trụ được dạy sâu cùng các mẫu hình chuẩn của nó và một bản thiết kế mẫu. **Giai đoạn 3 (Tuần 15–20): Danh mục mẫu hình.** Khoảng một tá kiến trúc giải quyết 95% các bài toán thực tế, và — quan trọng hơn — khi nào *không nên* dùng mỗi cái. **Giai đoạn 4 (Tuần 21–26): Migration và thế giới thực.** Di chuyển các công ty đang tồn tại lên cloud, và những ràng buộc (luật pháp, hệ thống cũ, lock-in) khiến kiến trúc thực tế khó hơn kiến trúc trên bảng trắng. **Giai đoạn 5 (Tháng 7–12): Nghề.** Các tài liệu, sơ đồ, buổi review, bài thuyết trình và chứng chỉ khiến bạn *tuyển được* với vai trò kiến trúc sư, chốt lại bằng ba dự án thiết kế cấp portfolio.

Khóa học này dành cho ai. Nó giả định không có kiến thức trước: mọi thứ nó dựa vào đều được dạy lại ngắn gọn trước khi được dùng. Nếu bạn đã hoàn thành **Khóa học Kỹ sư Cloud (The Cloud Engineer Course)** (hoặc bạn đang làm việc thực hành với cloud hôm nay), Giai đoạn 1 sẽ quen thuộc — *hãy lướt qua nó, nhưng vẫn làm các bài tập của nó*, vì nó tái đóng khung những điều bạn biết như sự kiện thành những điều bạn phải cân nhắc như quyết định, và chính sự tái đóng khung đó là toàn bộ cú chuyển mình từ kỹ sư sang kiến trúc sư.

Quy tắc cho toàn khóa học: học **một giờ mỗi ngày, sáu ngày mỗi tuần** — sự đều đặn thắng sự dồn dập. Mỗi module kết thúc bằng một **bảng Từ vựng** (những từ bạn phải làm chủ), các **Bài tập** (công việc thiết kế, không phải đọc — kiến trúc học được bằng cách vẽ và quyết định), và một **Milestone** (cột mốc — bằng chứng bạn sẵn sàng đi tiếp; đừng nhảy qua một

cột mốc chưa đạt). Và ngay từ Tuần 1, hãy duy trì một **Design Journal** (nhật ký thiết kế): mỗi kiến trúc bạn vẽ, mỗi đánh đổi bạn gọi tên, mỗi dự đoán bạn đưa ra về chi phí hay sự cố. Mười tháng nữa nó trở thành portfolio phỏng vấn của bạn; không gì gây ấn tượng với một hội đồng tuyển dụng bằng một cuốn sổ có ngày tháng với bốn mươi bản thiết kế kèm những ghi chú trung thực về những gì bạn đã sai.

Kiến trúc sư cloud thực sự làm gì

Hãy nghĩ về một kiến trúc sư xây dựng. Cô ấy không xếp gạch — nhưng cô ấy phải biết chính xác gạch làm được và không làm được gì, chúng giá bao nhiêu, và các tòa nhà đổ như thế nào. Cô ấy lắng nghe một khách hàng (“một ngôi nhà gia đình, nhiều năng, trong ngân sách này, trên mảnh đất khó này”), và biến những mong muốn cùng ràng buộc thành *bản vẽ và thông số kỹ thuật* đủ chính xác để thợ xây xây theo và khách hàng ký duyệt. Rồi cô ở lại suốt quá trình thi công, trả lời các câu hỏi và điều chỉnh bản kế hoạch khi nền đất hóa ra mềm hơn kết quả khảo sát.

Đổi gạch thành server và đó chính là công việc. Một kiến trúc sư cloud dành cả tuần cho một hỗn hợp gồm: **lắng nghe** các bên liên quan phía kinh doanh và trích xuất yêu cầu thật từ những mong ước mơ hồ; **thiết kế** — chọn thành phần, vẽ sơ đồ, viết ra các quyết định và lý do; **review** thiết kế của người khác dựa trên sáu trụ cột; **ước tính** một thiết kế sẽ tốn bao nhiêu mỗi tháng và bảo vệ con số đó trước bộ phận tài chính; **lập kế hoạch migration** (di trú) các hệ thống cũ lên cloud; **thuyết trình** — cùng một thiết kế được giải thích một kiểu cho kỹ sư và một kiểu hoàn toàn khác cho lãnh đạo; và **cố vấn (mentoring)** các kỹ sư để các chuẩn mực sống sót qua va chạm với deadline. Ở nhiều đội còn có cả **pre-sales** (tiền bán hàng): ngồi cạnh một nhân viên bán hàng trước một khách hàng tiềm năng, phác họa giải pháp thắng được hợp đồng.

Điều một kiến trúc sư *không* phải là: người viết code giỏi nhất phòng (thường không phải), người gõ phím nhiều nhất (gõ ít nhất), hay một thiên tài đờn độc ban phát bản thiết kế từ trên cao (cách nhanh nhất để bị phốt lờ). Sản phẩm thực sự của kiến trúc sư là **những quyết định tốt, được viết ra, mà người khác tự nguyện xây theo**.

Bản mô tả công việc mà bạn đang rèn luyện để đáp ứng

Vào tháng 8 năm 2026, chúng tôi thu thập các tin tuyển dụng và mẫu mô tả công việc Cloud/Solutions Architect thực tế — một mẫu cloud-architect của nhà tuyển dụng (KORE1), một bản mô tả công việc của công ty nhân sự (4 Corner Resources), định nghĩa của chính Microsoft về vai trò kiến trúc sư trong Azure Well-Architected Framework, một tin tuyển Solutions Architect pre-sales (Arpio, mảng khôi phục thảm họa AWS), một tin tuyển Solution Architect doanh nghiệp (Intel, qua Built In), và một tin tuyển Cloud Domain Architect doanh nghiệp (Halliburton, ưu tiên Azure). Bỏ đi hương vị công ty thì cùng mười hai yêu cầu ấy xuất hiện lặp đi lặp lại. Khóa học này được xây theo danh sách đó — đây chính xác là nơi mỗi yêu cầu được dạy:

Những gì bản mô tả công việc yêu cầu (lời của họ, diễn giải lại)

Nơi khóa học này dạy điều đó

“Design scalable and secure cloud architectures tailored to business and

Giai đoạn 1–3, các capstone ở Giai đoạn 5

Những gì bản mô tả công việc yêu cầu (lời của họ, diễn giải lại)

Nơi khóa học này dạy điều đó

technical requirements” (Thiết kế các kiến trúc cloud mở rộng được và an toàn, may đo theo yêu cầu kinh doanh và kỹ thuật) — thiết kế giải pháp đầu-cuối

“Deep platform expertise” (Chuyên môn nền tảng sâu) về AWS và/hoặc Azure — compute, lưu trữ, mạng, IAM, cấu trúc tài khoản

“Run Well-Architected reviews” (Điều hành các buổi Well-Architected review) dựa trên sáu trụ cột

“Lead migration/modernization initiatives — which workloads move as-is, get rearchitected, or retire” (Dẫn dắt các sáng kiến migration/hiện đại hóa — workload nào chuyển nguyên trạng, được tái kiến trúc, hay nghỉ hưu)

“Design landing zones, account structure, guardrails, and reference patterns teams deploy within” (Thiết kế landing zone, cấu trúc tài khoản, guardrail và các mẫu tham chiếu mà các đội triển khai bên trong)

“Integrate security and compliance into the design rather than bolting it on” (Tích hợp bảo mật và tuân thủ vào thiết kế thay vì gắn thêm sau) — zero trust, danh tính

High availability và disaster recovery — “RTO/RPO gaps, downtime costs, ransomware exposure” (khoảng trống RTO/RPO, chi phí ngừng hoạt động, rủi ro ransomware)

“Own the cloud cost model — tagging, showback, reserved capacity, rightsizing” (Làm chủ mô hình chi phí cloud — tagging, showback, công suất đặt trước, rightsizing); ước tính chi phí giải pháp

Thiết kế kiến trúc mạng — VPC, hub-and-spoke, kết nối hybrid

“Create architectural documentation, diagrams, and standards”; “maintain Architecture Decision Records” (Tạo tài liệu kiến trúc, sơ đồ và chuẩn mực; duy trì các

Giai đoạn 1 + lộ trình chứng chỉ ở Giai đoạn 5

Giai đoạn 2 (các trụ cột), Giai đoạn 5 Module 12 (điều hành review)

Giai đoạn 4 Module 10 (7 chữ R, lập kế hoạch theo đợt)

Giai đoạn 4 Module 10

Giai đoạn 2 Module 5, Giai đoạn 4 Module 11

Giai đoạn 2 Module 4, Giai đoạn 3 Module 9 (multi-region)

Giai đoạn 2 Module 6, Giai đoạn 5 Module 13 (định giá đề xuất)

Giai đoạn 1 Module 2, Giai đoạn 3 Module 9, Giai đoạn 4

Giai đoạn 1 Module 3 (sơ đồ), Giai đoạn 5 Module 12 (ADR, bộ sơ đồ)

Những gì bản mô tả công việc yêu cầu (lời của họ, diễn giải lại)

Nơi khóa học này dạy điều đó

Architecture Decision Record)

Giao tiếp với các bên liên quan — “present technical concepts to C-level and technical audiences”; “defend architectural decisions to security, finance, and engineering” (trình bày khái niệm kỹ thuật cho cấp C và khán giả kỹ thuật; bảo vệ các quyết định kiến trúc trước bảo mật, tài chính và kỹ thuật)

Giai đoạn 5 Module 13

“Mentor the cloud and platform engineers who build against your standards” (Cố vấn các kỹ sư cloud và nền tảng xây theo chuẩn mực của bạn); hỗ trợ pre-sales — demo, POC, RFP

Giai đoạn 5 Module 13

Chứng chỉ, được kiểm chứng trên cùng thị trường đó: bậc thang tiêu chuẩn là **AWS Certified Solutions Architect – Associate (SAA-C03)** — chứng chỉ đơn lẻ được yêu cầu nhiều nhất, thường cần **2–3 tháng** ôn — tiếp theo là **AWS Certified Solutions Architect – Professional (SAP-C02)**, thường thêm **4–8 tháng** nữa; các công ty nặng về Azure đòi hỏi **AZ-305 (Azure Solutions Architect Expert)**; các doanh nghiệp lớn đôi khi thêm **TOGAF** cho phương pháp kiến trúc doanh nghiệp. Kế hoạch đầy đủ nằm ở Giai đoạn 5, Module 14.

GIẢI ĐOẠN 1 — CÁC KHỐI XÂY DỰNG, DƯỚI DẠNG QUYẾT ĐỊNH (Tuần 1–6)

Học viên tốt nghiệp Khóa học Kỹ sư Cloud: lướt qua phân giải thích, nhưng làm mọi bài tập. Sự kiện thì giống nhau; câu hỏi thì mới.

Module 1 (Tuần 1–2): Cloud là gì, và kiến trúc sư tư duy như thế nào

Cloud trong một đoạn văn (phân ôn cho người chưa biết gì). Cloud là *máy tính của người khác, thuê theo giờ, được quản lý bằng phần mềm*. AWS, Microsoft Azure và Google Cloud vận hành những nhà kho chứa server (**data center** — trung tâm dữ liệu) được nhóm thành các **Region** (vùng) theo địa lý (ví dụ “Asia Pacific (Bangkok)”); mỗi region chứa vài **Availability Zone (AZ — vùng khả dụng)** tách biệt — những tòa nhà riêng với nguồn điện độc lập, đủ gần để kết nối nhanh, đủ xa để một trận lụt hay hỏa hoạn không quét sạch được hai cái. Bạn thuê các lát cắt của tất cả những thứ này theo từng giây: máy thô (**IaaS** — bạn quản lý mọi thứ trên đó), nền tảng được quản lý (**PaaS** — bạn chỉ mang ứng dụng của mình), hoặc phần mềm hoàn chỉnh (**SaaS** — bạn chỉ việc dùng). Đó là toàn bộ tầng nền. Mọi thứ một kiến trúc sư thiết kế đều được sắp xếp bên trên nó.

Bây giờ, nước đi của kiến trúc sư: biến mọi sự kiện thành một câu hỏi. Một kỹ sư học “một AZ là một trung tâm dữ liệu tách biệt.” Một kiến trúc sư lập tức hỏi: “*Workload này*

xứng đáng bao nhiêu AZ?” — vì hai AZ tốn hơn một, ba tốn hơn hai, và một trang brochure marketing không xứng đáng với những gì một hệ thống thanh toán xứng đáng. Đây là ý tưởng đầu tiên và quan trọng nhất của khóa học:

Kiến trúc là bộ môn làm cho các đánh đổi trở nên tường minh. Gần như không có thành phần nào sai — chỉ có những thành phần sai *cho workload này, ngân sách này, đội ngũ này, deadline này.*

Tam giác đánh đổi. Mọi thiết kế đều thương lượng giữa ba góc: **nhANH** (hiệu năng — nhanh cho người dùng, nhanh để xây), **rẻ** (hóa đơn tháng thấp, công sức kỹ thuật thấp), và **bền bỉ** (sống sót qua sự cố, mở rộng được, giữ được an toàn). Bạn có thể đẩy về bất kỳ hai góc nào; góc thứ ba trả giá cho điều đó. Bản mẫu thử của một startup nên nhanh và rẻ — sự bền bỉ có thể đợi. Số cái lỗi của một ngân hàng phải bền bỉ và nhanh — nó sẽ không rẻ. Khi một bên liên quan nói “chúng tôi muốn cả ba”, việc của bạn là mỉm cười và hỏi họ muốn cái nào *nhất*, vì bản thiết kế không thể bắt đầu cho đến khi họ trả lời. Hãy vẽ tam giác này ở đâu mỗi bản thiết kế bạn làm trong khóa học này, và đánh dấu workload nằm ở đâu. Nó sẽ giúp bạn tránh được một nghìn cuộc tranh cãi.

Yêu cầu: nguyên liệu thô của thiết kế. Kiến trúc sư tách **functional requirements** (yêu cầu chức năng — hệ thống làm gì: “khách hàng có thể đặt cơm trưa”) khỏi **non-functional requirements (NFR — yêu cầu phi chức năng)** (nó phải làm điều đó tốt đến đâu — “dưới 2 giây, cho 10.000 học sinh đồng thời, 99,9% thời gian, trong khuôn khổ PDPA”). Người mới ám ảnh với danh sách thứ nhất; kiến trúc sư kiếm được lương nhờ danh sách thứ hai, vì NFR mới là thứ thực sự quyết định kiến trúc. Hai câu hỏi thần kỳ trích xuất NFR từ những bên liên quan không biết mình có chúng: “*What happens to the business if this is down for an hour?*” (Điều gì xảy ra với doanh nghiệp nếu cái này sập trong một giờ?) và “*What does success look like at ten times today’s size?*” (Thành công trông như thế nào ở quy mô gấp mười hôm nay?)

Từ vựng:

Thuật ngữ	Định nghĩa
Region / Availability Zone (AZ)	Region: một cụm trung tâm dữ liệu theo địa lý mà bạn chọn để chạy trong đó. AZ: một trung tâm dữ liệu (hoặc nhóm nhỏ) tách biệt bên trong nó — đơn vị của “một tòa nhà có thể hỏng.”
IaaS / PaaS / SaaS	Thuê máy thô / một nền tảng được quản lý / phần mềm hoàn chỉnh. Thanh trượt từ nhiều quyền kiểm soát nhất đến ít việc bảo trì nhất.
Workload	Bất kỳ ứng dụng hay hệ thống nào, được xem như một đơn vị thiết kế — “workload thanh toán.”
Functional requirement	Điều hệ thống phải làm (“khách hàng có thể đặt hàng”).
Non-functional requirement (NFR)	Nó phải làm điều đó tốt đến đâu — tốc độ, quy mô, uptime, bảo mật, tuân thủ, chi phí.

Thuật ngữ	Định nghĩa
	NFR dẫn dắt kiến trúc.
Trade-off	Cái bạn từ bỏ để có được cái bạn chọn. Mọi quyết định thiết kế đều có một; việc của kiến trúc sư là gọi tên nó thành tiếng.
Constraint	Một ranh giới không thể thương lượng: ngân sách, deadline, luật pháp, hệ thống hiện có, kỹ năng đội ngũ. Các ràng buộc không phải chướng ngại của thiết kế — chúng <i>chính là</i> đề bài thiết kế.
Stakeholder	Bất kỳ ai có phần liên quan trong hệ thống: người dùng, kỹ sư, tài chính, bảo mật, lãnh đạo, cơ quan quản lý. Bên liên quan khác nhau, ngôn ngữ khác nhau — bạn nói tất cả các ngôn ngữ đó.
Greenfield / brownfield	Một hệ thống hoàn toàn mới không có lịch sử (greenfield) vs. một hệ thống vướng víu với các hệ thống hiện có (brownfield — phần lớn công việc thực tế).
Managed service	Một thành phần mà nhà cung cấp vận hành cho bạn (đã gồm sao lưu, vá lỗi, failover). Lựa chọn mặc định của kiến trúc sư, trừ khi có một lý do được viết ra.

Video cho module này (liên kết đã kiểm chứng):

Video	Kênh	Thời lượng	Liên kết
Top 50+ AWS Services Explained in 10 Minutes	Fireship	~10 phút	https://www.youtube.com/watch?v=JIbIYCM48to

Xem chuyên tham quan của Fireship hai lần: một lần ngay bây giờ để có tám bản đồ, một lần vào cuối Giai đoạn 1 — bạn sẽ ngạc nhiên vì mình giờ đã *đặt được* bao nhiêu dịch vụ vào một bản thiết kế.

Bài tập: (1) Chọn ba ứng dụng bạn dùng hằng ngày (một ứng dụng ngân hàng, một ứng dụng giao đồ ăn, một ứng dụng video) và, với mỗi cái, viết ba NFR hàng đầu của nó và đánh dấu nó trên tam giác đánh đổi. (2) Phỏng vấn một người bạn về một ý tưởng kinh doanh trong mười phút và trích xuất năm yêu cầu chức năng cùng năm yêu cầu phi chức năng — để ý rằng các NFR chỉ lộ ra khi bạn hỏi hai câu hỏi thân kỳ. (3) Bắt đầu Design Journal của bạn với mục #1: tam giác, và một đoạn văn về vì sao “chúng ta hy sinh góc nào?” là một câu hỏi kinh doanh, không phải một câu hỏi kỹ thuật.

Milestone: với bất kỳ mô tả hệ thống một câu nào, bạn có thể đưa ra các NFR khả dĩ của nó và vị trí của nó trên tam giác trong năm phút, thành tiếng, không cần ghi chú.

Module 2 (Tuần 3-4): Compute, lưu trữ, cơ sở dữ liệu, mạng — bốn quyết định

Mọi bản thiết kế bạn sẽ từng vẽ đều là bốn lựa chọn này, được đưa ra một cách chủ đích. Đây là từng cái, được dạy như một quyết định.

Quyết định 1 — Compute: VM, container, hay serverless? Một **virtual machine (VM — máy ảo)** là một lát thuê của một server hoạt động như một máy tính hoàn chỉnh — quyền kiểm soát tối đa, việc bảo trì tối đa (bạn và nó, bạn mở rộng nó, bạn trả tiền cả khi nó nhàn rỗi). Một **container** đóng gói một ứng dụng cùng mọi thứ nó cần để nó chạy giống hệt nhau ở mọi nơi; một bộ điều phối (**Kubernetes**) chạy và chữa lành các hạm đội container — mật độ và tính di động tuyệt vời, nhưng bạn đã nhận nuôi một nền tảng phức tạp cần người có kỹ năng. **Serverless** (AWS Lambda, Azure Functions) chỉ chạy code của bạn khi được kích hoạt và tính tiền theo mỗi lần gọi — gần như không phải bảo trì và hoàn hảo cho lưu lượng tăng vọt thất thường, nhưng có giới hạn (trên thời gian thực thi, cold start, và một cuộc hôn nhân sâu đậm hơn với một nhà cung cấp). Bảng quyết định mà bạn nên tái tạo được từ trí nhớ:

Chọn	Khi nào	Cẩn thận với
VM	Phần mềm cũ, giấy phép đặc thù, cần toàn quyền kiểm soát hệ điều hành, tải ổn định dự đoán được	Bạn sở hữu việc và lỗi, mở rộng, và 3 giờ sáng; thời gian nhàn rỗi tính tiền đủ
Container + Kubernetes	Nhiều dịch vụ, đội đã có sẵn kỹ năng, tính di động quan trọng	Độ phức tạp nền tảng — K8s là một công việc toàn thời gian; quá tay cho các đội nhỏ
Serverless	Lưu lượng tăng vọt hoặc khó dự đoán, chất keo hướng sự kiện, đội nhỏ, cần ra thị trường nhanh	Giới hạn runtime, cold start, khó dự đoán chi phí ở quy mô ổn định khổng lồ, lock-in

Cái nhìn sâu của người senior: **đây là lựa chọn theo từng workload, không phải tôn giáo của cả công ty.** Các hệ thống thực tế chạy cả ba cạnh nhau, một cách đúng đắn.

Quyết định 2 — Lưu trữ: object, block, hay file — và nóng đến đâu? Object storage (lưu trữ đối tượng — Amazon S3) là một chiếc xô không đáy cho file — rẻ, bền đến phi lý (“mười một số chín”), câu trả lời mặc định cho “file để đâu?” **Block storage** (lưu trữ khối — EBS) là ổ đĩa ảo bắt vít vào một VM. **File storage** (lưu trữ tệp — EFS) là một ổ dùng chung mà nhiều máy cùng gắn kết một lúc. Chiều bổ sung của kiến trúc sư là **nhật độ**: dữ liệu nóng (truy cập liên tục, định giá cho tốc độ) so với các bậc lạnh/lưu trữ dài hạn (Glacier — vài xu, nhưng vài phút đến vài giờ để lấy lại). Thiết kế các lifecycle rule để dữ liệu cũ trôi dạt xuống các bậc lạnh là chiến thắng chi phí rẻ nhất trong cloud; quên làm điều đó là lỗi phổ biến nhất.

Quyết định 3 — Cơ sở dữ liệu: SQL hay NoSQL (và hương vị được quản lý nào)? Cơ sở dữ liệu **relational/SQL** (quan hệ — PostgreSQL, MySQL; được quản lý dưới dạng RDS/Aurora) giữ dữ liệu trong các bảng chặt chẽ với tính nhất quán được bảo đảm — mặc định cho bất cứ thứ gì mà tính chính xác là thiêng liêng: tiền bạc, đơn hàng, tồn kho, người

dùng. Cơ sở dữ liệu **NoSQL** (DynamoDB, MongoDB) đánh đổi cấu trúc chặt chẽ lấy sự linh hoạt và khả năng mở rộng ngang gần như vô hạn — mặc định cho session, danh mục, bảng tin, telemetry. Quy tắc quyết định: *bắt đầu với SQL trừ khi bạn nêu đích danh được lý do cụ thể nó không dùng được* (quy mô cực lớn, schema linh hoạt, đọc toàn câu ở mức mili giây một chữ số). Và trong cloud, “cơ sở dữ liệu” gần như luôn phải nghĩa là “cơ sở dữ liệu **được quản lý**” — nhà cung cấp lo sao lưu, vá lỗi và failover; một đội tự chạy cơ sở dữ liệu trên VM nên có một lý do được viết ra. Thêm các chuyên gia vào vốn từ của bạn: **cache** (Redis — dữ liệu nóng trong bộ nhớ, đọc ở mức micro giây), **warehouse** (kho dữ liệu — phân tích ở quy mô lớn — Giai đoạn 3), **queue** (hàng đợi — không phải cơ sở dữ liệu, nhưng thường là mảnh ghép còn thiếu — Giai đoạn 3).

Quyết định 4 — Mạng: hình dạng của thế giới riêng. Một **VPC (Virtual Private Cloud)** là khu vực được rào chắn của bạn trong mạng của nhà cung cấp. Bên trong nó, **public subnet** chứa những thứ internet được phép chạm tới (load balancer), và **private subnet** chứa mọi thứ còn lại — server ứng dụng và, luôn luôn, cơ sở dữ liệu. “*Cơ sở dữ liệu nằm trong một private subnet*” là câu được lặp lại nhiều nhất trong các buổi review kiến trúc; lập luận — không gì tấn công được thứ không có đường dẫn từ internet — là một nửa của bảo mật mạng. Xung quanh VPC: một **load balancer** trải lượng qua các server và định tuyến vòng qua những cái ốm yếu; **DNS** (Route 53) biến tên thành địa chỉ; một **CDN** (CloudFront) lưu đệm nội dung tại hàng trăm thành phố để nó nhanh ở mọi nơi; một **API gateway** là quầy lễ tân được quản lý cho các API của bạn (xác thực, giới hạn tần suất, ghi log). Kết nối với thế giới cũ: **VPN** (đường hầm mã hóa qua internet) hoặc **Direct Connect** (một đường truyền vật lý riêng) — những cuống rốn của mọi thiết kế hybrid ở Giai đoạn 4.

Từ vựng:

Thuật ngữ	Định nghĩa
Instance / instance type	Một VM đang thuê / kích cỡ của nó (CPU + RAM), thứ quyết định giá theo giờ.
Container / Docker / Kubernetes (K8s)	Một gói di động cho một ứng dụng / công cụ build và chạy chúng / bộ điều phối chạy và chữa lành các hạm đội chúng.
Serverless / Lambda	Code chỉ chạy khi được kích hoạt, tính tiền theo lần chạy, không có server nào phải quản lý. Lambda là phiên bản của AWS.
Cold start	Độ trễ thêm khi một hàm serverless chạy sau thời gian nhàn rỗi — đánh đổi kinh điển của serverless.
S3 / bucket / durability	Object storage của AWS / một thùng chứa file có tên / xác suất dữ liệu sống sót — 99,999999999% của S3 nghĩa là mát mát về cơ bản không bao giờ xảy ra.
Storage tier / lifecycle policy	Hạng giá-tốc độ (nóng — lạnh — lưu trữ dài hạn) / quy tắc tự động chuyển dữ liệu già đi sang các bậc rẻ hơn.

Thuật ngữ	Định nghĩa
RDS / Aurora / DynamoDB	Các dịch vụ SQL được quản lý của AWS / bản SQL hiệu năng cao cloud-native của họ / NoSQL được quản lý chủ lực của họ.
Consistency	Bảo đảm rằng mọi người đọc dữ liệu thấy cùng một sự thật tại cùng một thời điểm — siêu năng lực của SQL, và thứ NoSQL nói lỏng để đổi lấy quy mô.
Cache / Redis	Một bản sao tốc-độ-bộ-nhớ của dữ liệu nóng đặt trước một cơ sở dữ liệu / công cụ tiêu chuẩn cho việc đó.
VPC / subnet (public, private)	Khu mạng riêng của bạn / các phân khu của nó — public đối mặt internet, private thì không. Cơ sở dữ liệu sống ở private. Luôn luôn.
Load balancer / health check	Người điều phối giao thông qua các server / bài kiểm tra nhịp tim nó dùng để ngừng gửi lưu lượng đến những cái đã chết.
CDN / edge	Các bản lưu đệm cấp thành phố của nội dung của bạn trên toàn thế giới / “the edge” (biên) = gần người dùng.
API / API gateway	Cánh cửa có kiểm soát mà một phần mềm mở ra cho phần mềm khác / quầy lễ tân được quản lý cho những cánh cửa đó.
VPN / Direct Connect	Đường hầm mã hóa qua internet / một đường truyền vật lý riêng đến cloud — hai cách on-prem gặp cloud.

Video cho module này (liên kết đã kiểm chứng):

Video	Kênh	Thời lượng	Liên kết
Kubernetes explained in 15 mins	TechWorld with Nana	~16 phút	https://www.youtube.com/watch?v=VnvRFRk_51k
Serverless Computing in 100 Seconds	Fireship	~2 phút	https://www.youtube.com/watch?v=W_VV2Fx32_Y
AWS Networking Basics (VPC & Subnets)	KodeKloud	~30 phút	https://www.youtube.com/watch?v=QM63dyA_4Pc

Bài tập: (1) Với mỗi workload trong năm workload sau, chọn compute, cơ sở dữ liệu và lưu trữ, và viết một câu biện minh cho mỗi cái: một trang đặt cơm trưa cho trường học; sổ cái giao dịch của một ngân hàng; một ứng dụng chia sẻ ảnh; một trình tạo báo cáo hàng đêm chạy 20 phút; một ứng dụng chat cho 5 triệu người dùng. (2) Lấy một trong các lựa chọn của bạn và biện luận cho lựa chọn *ngược lại* một cách thuyết phục nhất có thể — kiến trúc sư không thể “steel-man” phương án thay thế là người chưa hiểu sự đánh đổi. (3) Nhật ký: bảng quyết định compute của bạn, từ trí nhớ.

Milestone: với bất kỳ workload nào trong một câu, bạn có thể nêu tên bốn quyết định của nó kèm biện minh trong dưới ba phút — và với ít nhất một quyết định, nêu được điều gì sẽ khiến bạn đổi ý.

Module 3 (Tuần 5–6): Đọc và vẽ sơ đồ kiến trúc

Một kiến trúc sư không biết vẽ là một nhà tư vấn chỉ biết nói. Sơ đồ là ngôn ngữ làm việc của bạn: module này làm cho bạn đọc chúng trôi chảy và vẽ chúng thành thạo.

Sơ đồ kinh điển — học cái này trước tiên. Three-tier architecture (kiến trúc ba tầng) là “cấu trúc câu” của các sơ đồ cloud; đa số các thiết kế là biến thể của nó:

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

→ [Cache]

→ [Database: primary +

standby, private subnets]

Tầng 1 (**presentation** — trình diện): thứ người dùng chạm vào — nội dung tĩnh từ CDN, các request đi qua load balancer. Tầng 2 (**application** — ứng dụng): hạm đội các server thay thế được cho nhau chạy logic của bạn, trong các private subnet, được mở rộng tự động. Tầng 3 (**data** — dữ liệu): cơ sở dữ liệu, sâu nhất và được bảo vệ nhất, với một bản standby ở AZ thứ hai. Lưu lượng chảy một chiều: người dùng không bao giờ chạm trực tiếp vào tầng ứng dụng, chỉ mình tầng ứng dụng nói chuyện với tầng dữ liệu. Luyện vẽ cái này cho đến khi bàn tay bạn tự làm mà không cần bộ não — nó là màn mở đầu phỏng vấn bảng trắng ở mọi nơi trên trái đất.

Các quy ước ký hiệu khiến bạn trông chuyên nghiệp (vì chúng khiến bạn tư duy chuyên nghiệp): hộp là thành phần — dán nhãn mỗi cái với nó là gì và dịch vụ nào (“App servers — EC2, auto-scaling group”). Mũi tên chỉ hướng một *request* chảy, dán nhãn giao thức khi điều đó quan trọng. Các đường bao nét đứt chỉ ranh giới — VPC, mỗi subnet, mỗi AZ (vẽ ranh giới AZ và multi-AZ trở nên nhìn thấy được thay vì chỉ được tuyên bố). Một người dùng/tác nhân đứng ngoài hệ thống. Các con số trên mũi tên (1, 2, 3...) cho phép bạn thuật lại hành trình của một request. Và mỗi sơ đồ mang một tiêu đề, một ngày tháng, và một chú giải. Quy tắc sâu hơn: **một sơ đồ, một khán giả, một câu hỏi**. Một sơ đồ cho thấy mọi thứ thì không cho thấy gì cả; bạn sẽ học bộ mức thu phóng tiêu chuẩn (context — container — deployment) ở Giai đoạn 5.

Đọc sơ đồ của người khác — tia X của kiến trúc sư. Khi được đưa một sơ đồ, hãy chạy bài quét này thành tiếng: Internet chạm vào hệ thống này ở đâu (mỗi điểm chạm là bề mặt tấn công)? Dữ liệu ở đâu, và nó có nằm trong private subnet không? Cái gì được nhân đôi (bền

bỉ) và cái gì là **single point of failure** (điểm hỏng đơn lẻ) — một chiếc hộp không có bản sinh đôi? Đâu sẽ là chỗ đau ở mức lưu lượng 10×? Mỗi chiếc hộp tốn bao nhiêu mỗi tháng? Năm câu hỏi, ba mươi giây, và bạn đã đọc sơ đồ theo cách một bác sĩ đọc phim X-quang. Duyệt AWS Architecture Center (<https://aws.amazon.com/architecture/>) và chạy bài quét trên ba kiến trúc tham chiếu đã công bố.

Từ vựng:

Thuật ngữ	Định nghĩa
Three-tier architecture	Sự phân tách kinh điển: presentation (lõi vào của người dùng) — application (logic) — data (cơ sở dữ liệu). Hình dạng mặc định của các hệ thống web.
Tier / layer	Một lát cắt ngang của hệ thống với một trách nhiệm, chỉ nói chuyện với hàng xóm của nó.
Single point of failure (SPOF)	Bất kỳ thành phần nào mà cái chết đơn độc của nó kéo sập hệ thống. Thứ đầu tiên phải săn lùng trong bất kỳ sơ đồ nào.
Multi-AZ	Chạy các bản sao trên ít nhất hai Availability Zone để một trung tâm dữ liệu có thể hỏng mà không ai thấy.
Auto-scaling (group)	Máy được thêm vào và bớt đi tự động theo nhu cầu — công suất biết thở.
Stateless / stateful	Một server không giữ dữ liệu độc nhất (bất kỳ bản sinh đôi nào cũng thay được, nên nó mở rộng tự do) vs. một cái giữ dữ liệu không được phép mất. Mục tiêu thiết kế: tầng ứng dụng stateless, trạng thái đẩy xuống cơ sở dữ liệu và cache.
Reference architecture	Một thiết kế mẫu được nhà cung cấp công bố và chứng thực cho một bài toán phổ biến — kiến trúc sư lắp ráp từ những cái này trước khi phát minh.
Context diagram	Mức thu phóng cao nhất: hệ thống của bạn là một chiếc hộp, cộng với người dùng và các hệ thống bên ngoài nó chạm tới.
Attack surface	Mọi điểm nơi thế giới bên ngoài có thể chạm vào hệ thống. Nhỏ hơn là an toàn hơn.
North-south / east-west traffic	Lưu lượng vào/ra hệ thống vs. lưu lượng giữa các thành phần bên trong nó.

Bài tập: (1) Vẽ sơ đồ ba tầng từ trí nhớ, năm ngày liên tiếp, cho đến khi mất dưới bốn phút với mọi ranh giới (VPC, subnet, hai AZ) được vẽ. (2) Lấy năm workload của Module 2 và vẽ từng cái — hai mươi phút mỗi sơ đồ, các quy tắc ký hiệu được thực thi. (3) Tìm bất kỳ sơ đồ

kiến trúc thật nào trên mạng (AWS Architecture Center có hàng trăm), và viết bài quét X-quang năm-câu-hỏi của nó vào nhật ký. (4) Bài luyện thuật lại: sơ đồ trước mặt, thuật lại một cú nhấp của người dùng từ trình duyệt đến cơ sở dữ liệu và quay về, thành tiếng, đánh số các mũi tên khi bạn đi.

Milestone — kết thúc Giai đoạn 1: trên một bảng trắng (hoặc giấy, chụp lại cho nhật ký), bạn có thể vẽ một thiết kế ba tầng đúng, ký hiệu chuẩn, cho một workload một-câu mới lạ trong dưới mười lăm phút, thuật lại một request đi qua nó, và trả lời “cái gì hỏng nếu hộp này chết?” cho mọi chiếc hộp. Màn vẽ-cộng-chất-vấn này chính xác là nửa đầu của một buổi phỏng vấn kiến trúc sư thực thụ — từ đây trở đi, mọi thứ là chiều sâu.

GIAI ĐOẠN 2 — CÁC MIỀN THIẾT KẾ (Tuần 7-14)

Tám bản đồ cho giai đoạn này là AWS Well-Architected Framework — danh sách kiểm tra chung của cả ngành về ý nghĩa của “được thiết kế đúng đắn”, được tổ chức thành sáu trụ cột: **vận hành xuất sắc, bảo mật, độ tin cậy, hiệu quả hiệu năng, tối ưu chi phí, tính bền vững**. (Azure có một framework gần như giống hệt; học sâu một cái là bạn đã học cả hai.) Các bản mô tả công việc đòi hỏi đích danh những kiến trúc sư có thể “run Well-Architected reviews” (điều hành các buổi Well-Architected review), nên chúng ta đi qua từng trụ cột một, và với mỗi trụ bạn học ba thứ: các **câu hỏi then chốt** của trụ cột, các **mẫu hình chuẩn** của nó (những câu trả lời nhằm chán những đã được chứng minh — kiến trúc sư lặp ráp trước khi phát minh), và một **ví dụ mẫu** trên một kịch bản xuyên suốt.

Kịch bản xuyên suốt cho toàn bộ Giai đoạn 2: *ThaiTicket*, một nền tảng bán vé sự kiện hư cấu ở Bangkok. Tải bình thường: 2.000 lượt truy cập/giờ. Nhưng khi vé của một nghệ sĩ nổi tiếng mở bán lúc 10 giờ sáng, nó hứng 400.000 lượt truy cập trong mười phút, khâu thanh toán không được bán trùng ghế, và dữ liệu cá nhân của khách hàng Thái nằm trong phạm vi PDPA. Nhanh, rẻ, bền bỉ — ThaiTicket cần cả ba và không thể có cả ba, đó chính là điều khiến nó thành bệnh nhân thực hành hoàn hảo.

Xem trước Module 4 (liên kết đã kiểm chứng):

Video	Kênh	Thời lượng	Liên kết
The Five Pillars of the AWS Well-Architected Framework	Amazon Web Services (official; trụ cột thứ sáu, Sustainability, được thêm sau)	~4 phút	https://www.youtube.com/watch?v=KvEDbPmha6o
What is the AWS Well-Architected Framework?	Tech With Lucy	~10 phút	https://www.youtube.com/watch?v=MpDJ6TCWKjk

Và bookmark chính framework — <https://aws.amazon.com/architecture/well-architected/> và tài liệu đầy đủ tại

<https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html> — bạn sẽ sống trong đó tám tuần.

Module 4 (Tuần 7–8): Độ tin cậy — thiết kế cho cái ngày mọi thứ hỏng

Tín điều của trụ cột này: mọi thứ hỏng, mọi lúc. Ổ đĩa chết, AZ ngập lụt, các đợt deploy trục trặc, và một certificate ở đâu đó luôn sắp hết hạn. Độ tin cậy không phải là sự vắng mặt của hỏng hóc — nó là sự *vô nghĩa* của hỏng hóc: thiết kế sao cho khi (không phải nếu) một thành phần chết, người dùng không bao giờ nhận ra.

Các câu hỏi then chốt (hỏi những câu này với mọi thiết kế, mãi mãi): Điều gì xảy ra khi mỗi thành phần hỏng — luôn có một “rồi sao nữa” chứ? Hệ thống xử lý tải 10× như thế nào? Làm sao chúng ta *biết* có gì đó hỏng trước khi khách hàng báo? **RTO** và **RPO** của chúng ta là bao nhiêu — và ai trong doanh nghiệp đã ký duyệt chúng? Lần cuối chúng ta *kiểm chứng* một cuộc khôi phục là khi nào?

RTO và RPO — hai con số chính là cuộc trò chuyện về thảm họa. Recovery Time

Objective (mục tiêu thời gian khôi phục): chúng ta được phép sập bao lâu? **Recovery Point Objective** (mục tiêu điểm khôi phục): chúng ta được phép mất bao nhiêu dữ liệu (tức là, bản backup tốt gần nhất được phép cũ đến đâu)? RTO 4 giờ và RPO 1 giờ nghĩa là: trở lại trong vòng bốn giờ, mất nhiều nhất một giờ dữ liệu. Đây là các quyết định *kinh doanh* với bảng giá tăng theo cấp số nhân — RPO 24 giờ là một bản sao lưu hằng đêm (rẻ); RPO ~0 là sao chép liên tục (đắt); RTO tính bằng phút nghĩa là hạ tầng âm chạy chờ sẵn (rất đắt). Việc của kiến trúc sư là làm cho doanh nghiệp chọn các con số *một cách có hiểu biết* — và thiết kế chính xác theo chúng, chứ không lãng mạn vượt quá chúng.

Các mẫu hình chuẩn: redundancy (dự phòng — hai bản của mọi thứ quan trọng — N+1) · **multi-AZ** (các bản sao trải qua các trung tâm dữ liệu; load balancer và failover của cơ sở dữ liệu được quản lý làm nó tự động) · **auto-scaling** (công suất đi theo nhu cầu) · **health check + tự chữa lành** (instance chết được phát hiện và thay thế bởi máy móc, không phải con người) · **backup, đã kiểm chứng** (một bản backup chưa kiểm chứng là một niềm hy vọng, không phải một kế hoạch — hãy lên lịch các buổi diễn tập khôi phục) · **graceful degradation** (suy giảm có kiểm soát — khi quá tải, buông các tính năng ít quan trọng nhất trước: ThaiTicket có thể bỏ bản xem trước sơ đồ ghế và giữ lại khâu checkout) · **queue làm giảm xóc** (Giai đoạn 3) · **tránh hỏng hóc dây chuyền** (timeout, retry với backoff, circuit breaker — để một phụ thuộc chậm chạp không chìm chết cả hạm đội).

Ví dụ mẫu — thiết kế độ tin cậy cho ThaiTicket. Hai AZ trong region Bangkok. Tầng ứng dụng stateless trong một auto-scaling group sau một load balancer, được làm ấm trước theo lịch trước các giờ mở bán đã công bố (auto-scaling phản ứng trong vài phút; một đợt mở bán lúc 10:00 cần công suất lúc 09:45). Cơ sở dữ liệu Aurora, primary ở AZ-a, standby đồng bộ ở AZ-b, failover tự động ≈ dưới một phút. Một queue giữa các cú nhập “mua” và khâu xử lý thanh toán để một sự chậm chạp của nhà cung cấp thanh toán xếp hàng các đơn thay vì làm sập trang. Backup: liên tục, khôi phục theo thời điểm, diễn tập khôi phục hằng tháng. Các con số đã thống nhất với doanh nghiệp: RTO 15 phút, RPO ~0 cho đơn hàng (đó là tiên), RPO 24 giờ cho dữ liệu phân tích (thì không). Bài tập nhật ký: cái này đã lấy đi của chúng ta gì từ góc “rẻ” của tam giác?

Từ vựng:

Thuật ngữ	Định nghĩa
RTO / RPO	Recovery Time Objective: bạn được phép sập bao lâu. Recovery Point Objective: bạn được phép mất bao nhiêu dữ liệu. Hai con số định nghĩa mọi cuộc trò chuyện DR — và chúng là quyết định kinh doanh.
High availability (HA)	Thiết kế sao cho các hỏng hóc thường nhật (một instance, một AZ) không gây gián đoạn nào người dùng thấy được.
Disaster recovery (DR)	Kế hoạch cho những hỏng hóc <i>lớn</i> — cả một region, một sự kiện ransomware — với các mục tiêu RTO/RPO và một runbook đã kiểm chứng.
Redundancy / N+1	Công suất dự phòng để bất kỳ một thành phần nào cũng có thể chết: cần N, chạy N+1.
Failover	Cú chuyển tự động sang bản standby khi bản primary chết.
Health check	Nhịp tim tự động phát hiện các thành phần chết để máy móc định tuyến vòng qua chúng.
Graceful degradation	Buông các tính năng ít quan trọng hơn dưới áp lực để phân lỗi sóng sót.
Timeout / retry with backoff	Bỏ cuộc với một lời gọi chậm sau một giới hạn / thử lại với các quãng nghỉ tăng dần — phép lịch sự ngăn hỏng hóc dây chuyền.
Circuit breaker	Một thành phần ngừng gọi một phụ thuộc đang hỏng trong một khoảng thời gian để nó hồi phục — như cầu chì trong một ngôi nhà.
Chaos engineering	Có ý tiếm hỏng hóc vào hệ thống (kiểu Netflix) để chứng minh các tuyên bố về sự bền bỉ trước khi thực tế kiểm chứng chúng thay bạn.

Bài tập: (1) Lấy thiết kế đặt-cơm-trưa của Giai đoạn 1 và nâng cấp nó để sóng sót qua: một instance chết, mất một AZ, một sự cố cơ sở dữ liệu, và một giờ cao điểm ăn trưa 10× — vẽ bản trước và sau. (2) Viết cuộc trò chuyện RTO/RPO cho ba doanh nghiệp (một blog, một trang thương mại điện tử, một hệ thống hồ sơ bệnh viện) dưới dạng một đoạn hội thoại ngắn giữa kiến trúc sư và chủ doanh nghiệp — luyện cách làm cho chi phí hiện hình trong hội thoại. (3) Nhật ký: liệt kê mọi SPOF trong thiết kế ThaiTicket ở trên. Có ít nhất một cái được để lại một cách chủ ý. (Gợi ý: bao nhiêu region?)

Milestone: bạn có thể chạy màn chat vắn “điều gì xảy ra khi cái này hỏng?” trên bất kỳ sơ đồ nào trong mười phút liên tục mà không cạn lời, và bạn có thể giải thích RTO vs RPO cho một

chủ doanh nghiệp không chuyên kỹ thuật bằng phép so sánh cửa-hàng-bị-ngập trong chín mươi giây.

Module 5 (Tuần 9–10): Bảo mật — thiết kế những hệ thống khó bị tổn thương

Khung nhìn: shared responsibility model (mô hình trách nhiệm chia sẻ). Nhà cung cấp bảo vệ *bản thân cloud* — tòa nhà, phần cứng, hypervisor. Bạn bảo vệ mọi thứ bạn *đặt vào đó* — dữ liệu, danh tính, cấu hình, code. Gần như mọi vụ rò rỉ cloud nổi tiếng đều là một câu hình sai phía khách hàng (một bucket public, một key bị lộ, một role được cấp quyền quá tay), đó là lý do bảo mật là một bài toán *kiến trúc* trước khi là một bài toán công cụ: các bản mô tả công việc nói “integrate security into the design rather than bolting it on” (tích hợp bảo mật vào thiết kế thay vì gắn thêm sau), và module này là cách làm điều đó.

Các câu hỏi then chốt: Ai và cái gì có thể truy cập môi thành phần, và mọi quyền hạn có phải là mức *tối thiểu* cần thiết không? Dữ liệu được mã hóa ở đâu — khi lưu trữ, khi truyền, và ai giữ khóa? Các ranh giới mạng ở đâu, và cái gì băng qua chúng? Chúng ta sẽ *phát hiện* một vụ xâm nhập bằng cách nào — và có kể lại được câu chuyện sau đó từ log không? Luật nào áp dụng cho dữ liệu này, và dữ liệu sống ở đâu về mặt vật lý?

Các mẫu hình chuẩn: **least privilege** (đặc quyền tối thiểu — mỗi con người và chương trình nhận mức truy cập tối thiểu mà công việc của nó cần — quy tắc vàng mà mọi IAM policy bị phân xét theo) · **MFA ở khắp nơi, root cất kỹ · mã hóa khi lưu trữ và khi truyền, luôn bật** (trong cloud nó là một ô đánh dấu; không có lý do bào chữa nào) · **phân đoạn mạng** (public/private subnet, security group làm tường lửa theo-từng-server; bán kính sát thương của một vụ xâm nhập được định nghĩa bởi những bức tường bạn đã vẽ từ trước) · **bí mật trong két, không bao giờ trong code · zero trust** (kiểm chứng mọi request một cách tường minh — danh tính, thiết bị, ngữ cảnh — không tin gì chỉ vì nó “ở bên trong mạng”; các JD gọi tên nó, bạn cũng phải thế) · **defense in depth** (phòng thủ theo chiều sâu — nhiều lớp, để một biện pháp thất bại không phải là hết trận) · **audit logging** (nhật ký kiểm toán kiểu CloudTrail về ai đã làm gì — bất biến, được giám sát) · **bảo mật như guardrail, không phải cổng chặn** (mã hóa các quy tắc thành chính sách tự động làm cho con đường an toàn thành con đường dễ đi, thay vì một cuộc họp review biên bảo mật thành kẻ thù của tiến độ).

Ví dụ mẫu — thiết kế bảo mật cho ThaiTicket. Dữ liệu khách hàng (tên, email, tham chiếu thanh toán) được phân loại là dữ liệu cá nhân theo PDPA — lưu mã hóa trong Aurora ở region Bangkok (data residency — lưu trú dữ liệu), khóa trong KMS. Mạng: chỉ load balancer là public; tầng ứng dụng private; subnet cơ sở dữ liệu chỉ chấp nhận kết nối *duy nhất* từ security group của tầng ứng dụng. Con người: SSO + MFA; kỹ sư mặc định nhận quyền chỉ-đọc trên production và quyền nâng cao có giới hạn thời gian khi yêu cầu (least privilege kèm dấu vết kiểm toán). Việc xử lý thẻ thanh toán được ủy thác cho một nhà cung cấp thanh toán được chứng nhận để số thẻ thô không bao giờ chạm vào hệ thống của chúng ta — một quyết định *thiết kế* gỡ bỏ cả một gánh nặng tuân thủ, đó chính là kiến trúc bảo mật ở dạng đẹp nhất. CloudTrail bật, cảnh báo về truy cập bất thường. Câu hỏi về xâm nhập được trả lời trước: PDPA có nghĩa vụ thông báo trong 72 giờ — log và runbook phải cho phép chúng ta kể lại câu chuyện trong thời gian ngắn hơn thế.

Từ vựng:

Thuật ngữ	Định nghĩa
Shared responsibility model	Nhà cung cấp bảo vệ cloud; bạn bảo vệ những gì ở trong đó. Đa số các vụ rò rỉ nằm ở phía khách hàng của ranh giới.
IAM / role / policy	Identity and Access Management — ai được làm gì. Policy là một danh sách quyền; role là một bó quyền khoác lên được.
Least privilege	Quy tắc vàng: mức truy cập tối thiểu cần thiết, không hơn, được rà soát định kỳ.
MFA	Một bằng chứng danh tính thứ hai ngoài mật khẩu. Không thể thương lượng với con người.
Encryption at rest / in transit / KMS	Dữ liệu được xáo trộn trên đĩa / trên đường truyền / dịch vụ được quản lý giữ các khóa.
Security group	Một danh sách quy tắc tường lửa theo-từng-server — “lưu lượng web vào chỉ từ load balancer.”
Network segmentation / blast radius	Chia mạng thành các khu có tường / kẻ tấn công vườn xa được đến đâu sau một lần xâm nhập. Những bức tường vẽ từ trước định nghĩa nó.
Zero trust	Kiểm chứng mọi request tường minh; không tin gì chỉ vì nó ở “bên trong.” Từ thế mặc định hiện đại.
Defense in depth	Nhiều biện pháp kiểm soát chồng lấn để một thất bại không chí mạng.
Secrets management	Mật khẩu, khóa và token sống trong một dịch vụ kết sắt — không bao giờ trong code, không bao giờ trong bảng tính.
Audit log / CloudTrail	Bản ghi bất biến về ai đã làm gì, khi nào — cách bạn phát hiện rắc rối và tái dựng nó về sau.
Data classification	Dán nhãn dữ liệu theo độ nhạy cảm (công khai / nội bộ / cá nhân / bị quản chế) để các biện pháp kiểm soát khớp với nhãn, không phải với phỏng đoán.
Data residency	Giữ dữ liệu về mặt vật lý trong biên giới một quốc gia — yêu cầu pháp lý ở nhiều ngành, một đầu vào kiến trúc luôn luôn.
PDPA / GDPR	Luật dữ liệu cá nhân của Thái Lan và châu Âu: sự đồng ý, thông báo rò rỉ (72 giờ), quy tắc chuyển dữ liệu xuyên biên giới. Giai

Video cho module này (liên kết đã kiểm chứng): The AWS Shared Responsibility Model — Digital Cloud Training — ~4 phút — <https://www.youtube.com/watch?v=ESPBEEK-cvo>

Bài tập: (1) Vẽ sơ đồ ThaiTicket và phủ lớp bảo mật lên nó: đánh dấu mọi ranh giới tin cậy, mọi điểm mã hóa, mọi nơi thông tin xác thực sinh sống. Thói quen phủ lớp này — cùng một sơ đồ, lăng kính bảo mật — chính xác là điều một buổi review đòi hỏi ở bạn. (2) Đọc một bản post-mortem công khai về một vụ rò rỉ do cấu hình cloud sai (Capital One 2019 là bài giảng kinh điển) và viết vào nhật ký *mẫu hình nào ở trên* lẽ ra đã ngăn được nó. (3) Nhập vai với một AI: nó đóng một nhà sáng lập startup nói “bảo mật để sau” — thuyết phục họ từ bỏ ý định đó bằng phép tính chi-phí-của-một-vụ-rò-rỉ, một cách tử tế.

Milestone: bạn có thể lấy bất kỳ sơ đồ Giai đoạn 1 nào của mình và tạo ra lớp phủ bảo mật của nó trong hai mươi phút, và giải thích least privilege, zero trust và shared responsibility model cho một bên liên quan không chuyên kỹ thuật mà không dùng biệt ngữ.

Module 6 (Tuần 11-12): Hiệu năng và chi phí — tốc độ và tiền bạc, cặp trụ cột song sinh

Hai trụ cột này được dạy cùng nhau vì chúng là cùng một cân gặt bị đẩy theo hai hướng ngược nhau, và kiến trúc sư là người đặt tay trên cân gặt.

Hiệu quả hiệu năng — các câu hỏi then chốt: Người dùng sẽ cảm thấy chậm ở đâu trước tiên? Cái gì được tính toán lặp đi lặp lại mà lẽ ra có thể tính một lần rồi cache? Mỗi thành phần có phải là *công cụ đúng* không (SQL làm việc của một máy tìm kiếm là một lỗi hiệu năng của kiến trúc, không phải của code)? Hiệu năng thay đổi thế nào ở mức 10× — và nút thắt cổ chai đầu tiên ở đâu?

Các mẫu hình hiệu năng: các lớp cache — mẫu hình hiệu năng đòn bẩy cao nhất: cache trình duyệt — **CDN** ở biên (nội dung tĩnh phục vụ từ một thành phố gần người dùng; có thể hấp thụ phần lớn áp đảo của lưu lượng đọc) — cache ứng dụng (**Redis** đứng trước cơ sở dữ liệu cho các lượt đọc nóng: sơ đồ ghé, session, trang sản phẩm) — cache truy vấn cơ sở dữ liệu. Mỗi lớp trả lời các request trước khi chúng chạm tới phần lõi đắt đỏ; các câu hỏi thiết kế luôn là *cái gì được phép cache, trong bao lâu, và nó được vô hiệu hóa thế nào khi sự thật thay đổi*. Tiếp theo: **read replica** (các bản sao của cơ sở dữ liệu phục vụ lượt đọc để bản primary chuyên tâm ghi) · **xử lý bất đồng bộ** (đừng bắt người dùng chờ những việc có thể làm sau — email biên nhận, ảnh thu nhỏ, báo cáo) · **đúng công cụ cho đúng việc** (tìm kiếm — một máy tìm kiếm; phân tích — một warehouse; tra cứu nóng — một kho key-value) · và **đo trước đã** (làm hiệu năng mà không đo lường là mê tín).

Tối ưu chi phí — các câu hỏi then chốt: Thiết kế này tốn bao nhiêu mỗi tháng ở tài hôm nay — và trên mỗi *đơn vị* (mỗi đơn hàng, mỗi khách hàng)? Cái gì đang chạy lúc 3 giờ sáng mà không cần thiết? Mức sử dụng ổn định nào có thể cam kết để được chiết khấu? Bộ phận tài chính sẽ nói xu hướng là gì?

Các mẫu hình chi phí: rightsizing (đa số các hạm đội bị cấp thừa một cách âm thầm; thu nhỏ về đúng nhu cầu đo được là tiền nhặt được) · **chiến lược đặt trước** — thực đơn giá:

on-demand (giá đầy đủ, tự do đầy đủ) cho tải tăng vọt/chứa rõ, **reserved instances / savings plans** (cam kết 1–3 năm, giảm 30–70%) cho phân nền ổn định, **spot** (giảm đến 90%, có thể bị thu hồi trong thời gian ngắn) cho công việc batch chịu được gián đoạn; nước đi của kiến trúc sư là xếp lớp chúng: đặt trước phân sàn, auto-scale phân giữa bằng on-demand, spot cho batch · **vòng đời lưu trữ** (phân bậc của Module 2, tự động hóa) · **tắt mọi thứ đi** (các môi trường phi production ngủ đêm và cuối tuần có thể cắt hai phần ba chi phí của chúng) · **canh chừng egress** (dữ liệu rời khỏi cloud bị tính tiền; các thiết kế xuyên region làm lờn và các lượt tải công khai lớn khiến ai cũng bất ngờ một lần) · **tagging và showback** (mọi tài nguyên dán nhãn chủ sở hữu/dự án để mỗi baht đều truy được trách nhiệm) · và chỉ số vương miện, **unit economics** (kinh tế học đơn vị): không phải “hóa đơn là \$800k/tháng” mà là “chi phí trên mỗi vé bán ra là \$1,90 và đang giảm.” Hóa đơn tăng thì không sao; *chi phí đơn vị* tăng là một mùi kiến trúc. Bộ môn này có một cái tên — **FinOps** — và kiến trúc sư ngồi giữa nó.

Ví dụ mẫu — ThaiTicket, cả hai lăng kính. Hiệu năng: CloudFront phục vụ các trang nghệ sĩ và ảnh sơ đồ ghế (400.000 người hiểu kỳ phần lớn không bao giờ chạm tới các server); Redis cache tình trạng ghế trống với TTL 2 giây — đủ cũ để rẻ, đủ mới để khâu checkout (thứ kiểm tra lại với Aurora, nguồn sự thật) ngăn việc bán trùng; biên nhận và vé được tạo bất đồng bộ sau thanh toán. Chi phí: phân nền ổn định 2.000 lượt truy cập/giờ chạy trên một savings plan (~40% giảm); các cơ sở mở bán chạy on-demand trong đúng cái giờ chúng tồn tại; các job phân tích chạy trên spot vào ban đêm; staging ngủ ngoài giờ làm việc; mọi tài nguyên gắn tag project: thaiticket. Bản đề xuất cho CFO viết: “\$62.000/tháng cho phân nền, ~\$9.000 cho mỗi sự kiện mở bán lớn, chi phí trên mỗi vé ~\$1,90 giảm dần theo sản lượng” — và *chính câu đó* là âm thanh của một kiến trúc sư trôi chảy về chi phí.

Từ vựng:

Thuật ngữ	Định nghĩa
Latency / throughput	Một request mất bao lâu / hệ thống chịu được bao nhiêu request mỗi giây. Liên quan, nhưng không phải một.
Bottleneck	Điểm hẹp nhất quyết định nhịp độ của cả hệ thống. Tối ưu ở bất kỳ chỗ nào khác chỉ là trang trí.
Cache / TTL / invalidation	Một bản sao nhanh của dữ liệu tồn kém khi lấy / nó được tin cậy trong bao lâu / bài toán khó của việc làm mới nó khi sự thật thay đổi.
CDN	Lớp cache ngoài cùng — nội dung của bạn tại hàng trăm thành phố. Câu trả lời đầu tiên cho “làm nó nhanh trên toàn cầu.”
Read replica	Một bản sao cơ sở dữ liệu phục vụ lượt đọc, để bản primary giữ sức cho lượt ghi.
Asynchronous processing	Làm việc không khẩn cấp sau khi đã trả lời người dùng, thường qua một queue.
On-demand / reserved / savings plan / spot	Giá đầy đủ linh hoạt / cam kết 1–3 năm giảm

Thuật ngữ	Định nghĩa
	30–70% / cùng ý tưởng, linh hoạt hơn / giảm đến 90% nhưng có thể bị thu hồi. Kiên trúc sư xếp lớp cả bốn.
Rightsizing	Thu nhỏ tài nguyên bị cấp thừa về đúng nhu cầu đo được. Tiên nhật được trong hầu hết mọi tài khoản.
Egress	Dữ liệu rời khỏi cloud — tính tiền theo GB. Cú bất ngờ hóa đơn nổi tiếng; hãy thiết kế các dòng dữ liệu với nó trong đầu.
Tagging / showback	Dán nhãn mọi tài nguyên với chủ sở hữu và dự án / cho môi đội thấy hóa đơn của chính họ. Trách nhiệm giải trình thay đổi hành vi.
Unit economics	Chi phí trên mỗi đơn vị kinh doanh (mỗi đơn hàng, mỗi người dùng). Chỉ số làm cho một hóa đơn có ý nghĩa — và câu nói ưa thích của kiến trúc sư trước một CFO.
TCO	Total cost of ownership — giá niêm yết cộng con người, giấy phép, migration và chi phí thoát ra trong suốt vòng đời của một lựa chọn.
FinOps	Bộ môn làm cho chi tiêu cloud trở nên hữu hình, được phân bổ, và được tối ưu liên tục.

Video cho module này (liên kết đã kiểm chứng): What is FinOps? — FinOps Foundation (official) — ~2 phút — https://www.youtube.com/watch?v=Y-c_xw9bHFW

Bài tập: (1) Định giá phân nền của ThaiTicket trong AWS pricing calculator (tìm kiếm “AWS pricing calculator” — bài tập liệt kê từng hạng mục của một thiết kế thật một lần sẽ giải mã mọi cuộc trò chuyện chi phí tương lai); so sánh tổng của bạn với con số \$62.000 ở trên và giải thích mọi khoảng chênh. (2) Thêm lớp phủ caching cho hai sơ đồ Giai đoạn 1 của bạn: đánh dấu mỗi cache, TTL của nó, và câu chuyện vô hiệu hóa của nó. (3) Một AI đóng vai một kỹ sư muốn mọi thứ on-demand “cho đơn giản” — thương lượng chiến lược đặt trước, với các con số.

Milestone: với bất kỳ thiết kế nào bạn đã vẽ, bạn có thể nêu ba dòng chi phí lớn nhất của nó và nút thắt cổ chai đầu tiên khả dĩ — và đề xuất một thay đổi cải thiện cả hai cùng lúc (gần như luôn có một cái; caching thường là nó).

Module 7 (Tuần 13–14): Vận hành xuất sắc, tính bền vững — và chạy cả sáu trụ cột cùng nhau

Vận hành xuất sắc là trụ cột đặt câu hỏi: *con người có thực sự vận hành được thứ này, một cách bình tĩnh không?* Các câu hỏi then chốt: Một thay đổi đến production bằng cách nào — qua một pipeline tự động, đã kiểm thử, hay qua một con người anh hùng? Làm sao chúng ta biết hệ thống khỏe mạnh *ngay lúc này*? Khi nó hỏng lúc 3 giờ sáng, kỹ sư on-call thực sự làm

gì? Các mẫu hình: **infrastructure as code** (môi trường được định nghĩa trong văn bản được review, có quản lý phiên bản — Terraform/CloudFormation — nên nó tái tạo được và kiểm toán được; một kiến trúc chỉ tồn tại dưới dạng các cú nhập trong console là truyện dân gian, không phải kỹ thuật) · **các pipeline CI/CD** với chiến lược triển khai an toàn (**blue-green**: dừng phiên bản mới cạnh bản cũ và chuyển; **canary**: cho phiên bản mới 5% lưu lượng và quan sát) · **observability** (khả năng quan sát — log, metric, trace, và các cảnh báo gắn với triệu chứng *người dùng thấy được*, không phải chuyện vặt của máy móc) · **runbook** (bản viết phải-làm-gì cho mỗi kiểu hỏng đã biết) · **post-mortem không đổ lỗi** (sau sự cố: cái gì hỏng, vì sao, cái gì ngăn lặp lại — không có kẻ xấu, nếu không mọi người ngừng nói sự thật). Phần của kiến trúc sư: *thiết kế* cho tính vận hành được — một hệ thống thông minh gấp đôi mà dễ quan sát bằng nửa là một hệ thống tệ hơn.

Tính bền vững, trụ cột thứ sáu, yêu cầu bạn lãng phí thế giới vật lý ít hơn: rightsize (CPU nhàn rỗi đốt điện thật), mở rộng theo nhu cầu thay vì cấp phát theo đỉnh, dùng dịch vụ được quản lý và serverless (hạ tầng chia sẻ là hạ tầng được lắp đầy hơn), phân bậc lưu trữ, và xóa những gì đã chết. Thật tiện, lựa chọn bền vững và lựa chọn rẻ thường là cùng một lựa chọn — nói cả hai trong các buổi review và bạn sẽ thuyết phục cả phòng hai lần.

Chạy các trụ cột cùng nhau — buổi Well-Architected review đầu tiên của bạn. Trong các buổi review thực tế, các trụ cột *xung đột*: độ tin cậy multi-region đánh nhau với chi phí; review bảo mật nghiêm ngặt đánh nhau với tốc độ vận hành; caching hiệu năng đánh nhau với tính đúng đắn của độ tươi dữ liệu. Framework không giải quyết xung đột thay bạn — nó ép chúng ra chỗ sáng, nơi doanh nghiệp có thể lựa chọn. Đó là toàn bộ thiên tài của nó, và là của bạn để vận dụng. Bản thân phương pháp review (các bộ câu hỏi, xếp ưu tiên phát hiện, bản báo cáo) được dạy đầy đủ ở Giai đoạn 5, Module 12; hai tuần này bạn tập dượt kỹ năng thô: chất vấn một thiết kế từ sáu hướng trong một lần ngồi.

Từ vựng:

Thuật ngữ	Định nghĩa
Infrastructure as Code (IaC) / Terraform	Hạ tầng được khai báo trong các file văn bản có quản lý phiên bản mà công cụ biến thành hiện thực / công cụ phổ biến nhất loại đó.
CI/CD pipeline	Băng chuyền tự động build, kiểm thử và triển khai mọi thay đổi.
Blue-green / canary deployment	Hai mẫu hình phát hành an toàn: chuyển lưu lượng giữa bản cũ và bản mới / nhỏ giọt lưu lượng sang phiên bản mới và quan sát.
Observability (logs, metrics, traces)	Khả năng tự giải thích của hệ thống: bản ghi sự kiện, các con số theo thời gian, và bản đồ hành trình theo từng request.
Alert / on-call / runbook	Cuộc gọi tự động khi ngưỡng bị phá / vòng xoay con người trả lời nó / kịch bản viết sẵn họ làm theo.
Blameless post-mortem	Buổi rà soát trung thực, không-kẻ-xấu sau

Thuật ngữ	Định nghĩa
	một sự cố, tạo ra sự phòng ngừa, không phải sự trừng phạt.
Drift	Thực tại tách khỏi định nghĩa IaC vì ai đó đã nhấp chuột. Kẻ thù của tính tái tạo được.
Toil	Công việc vận hành thủ công lặp đi lặp lại mà lẽ ra tự động hóa đã hấp thụ. Kiến trúc sư thiết kế để loại nó ra.
Well-Architected review	Một cuộc chat vắn có cấu trúc một workload theo cả sáu trụ cột, tạo ra các phát hiện được xếp ưu tiên. Giai đoạn 5 dạy bạn điều hành một buổi.

Bài tập: (1) Bài luyện sáu lăng kính: lấy sơ đồ tốt nhất của bạn và dành mười phút cho mỗi trụ cột để viết các phát hiện — sáu mươi phút, một thiết kế, sáu lăng kính; bài luyện này *chính là* Giai đoạn 2 thu nhỏ, lặp lại nó hằng tuần từ giờ. (2) Đọc hai báo cáo hậu-sự-cố của AWS (được công bố trên các trang trạng thái của họ sau các vụ sập lớn) và nhận diện mẫu hình của trụ cột nào đã thất bại. (3) Nhật ký: với ThaiTicket, viết ba xung-đột-trụ-cột bạn sẽ nêu lên cho doanh nghiệp, mỗi cái là một lựa chọn một câu (“Chúng ta có thể có X hoặc Y ở ngân sách này — chọn cái nào?”).

Milestone — kết thúc Giai đoạn 2: buổi review giả lập: một AI tạo ra một kiến trúc đầy lỗi cho một startup cho vay trực tuyến Thái; bạn phải tạo ra các phát hiện viết ra trên cả sáu trụ cột, bao gồm ít nhất: một khoảng trống độ tin cậy (cơ sở dữ liệu single-AZ của họ), một khoảng trống bảo mật (IAM quá rộng), một khoảng trống chi phí (mọi thứ on-demand), một khoảng trống vận hành (không có IaC), và cuộc trò chuyện RTO/RPO họ chưa từng có — mỗi phát hiện được diễn đạt như một câu hỏi mà một đồng nghiệp có thể nghe mà không nhăn mặt. Khi danh sách phát hiện của bạn đọc lên giống một đồng nghiệp senior tận tình hơn là một kiểm toán viên, Giai đoạn 2 hoàn tất.

GIẢI ĐOẠN 3 — DANH MỤC MẪU HÌNH (Tuần 15–20)

Kiến trúc sư không phát minh; họ **lựa chọn**. Giai đoạn này là danh mục các kiến trúc chuẩn của bạn — với mỗi cái: nó là gì, khi nào dùng, khi nào *không*, và hồ sơ chi phí/độ phức tạp của nó. Các cột “khi nào không” là trọng tải thực sự của giai đoạn: khóa học nào cũng nói được cho bạn microservices là gì; biết khi nào chúng sẽ đánh sập một công ty mới là thứ bạn được trả tiền. Trong khi học, hãy tiếp tục duyệt các kiến trúc tham chiếu thực tế tại <https://aws.amazon.com/architecture/> — nhận diện mẫu hình ngoài tự nhiên là bài tập khiến danh mục dính lại.

Module 8 (Tuần 15–17): Các mẫu hình ứng dụng

Monolith vs. microservices — cuộc cãi vã ồn ào nhất của ngành, được phân xử một cách bình thản. Một **monolith** (khối nguyên) là một ứng dụng triển khai duy nhất chứa

mọi tính năng; **microservices** chia hệ thống thành nhiều dịch vụ nhỏ, triển khai độc lập, mỗi cái sở hữu dữ liệu của mình, nói chuyện qua API và sự kiện. Bảng trung thực:

Mẫu hình	Nó là gì	Dùng khi	KHÔNG dùng khi	Chi phí/độ phức tạp
Monolith	Một ứng dụng, một lần triển khai, một cơ sở dữ liệu	Đội nhỏ, sản phẩm non trẻ, ranh giới miền chưa rõ — tức là, đa số các hệ thống mới	Nhiều đội chặn lẫn nhau; các phân cần mở rộng rất khác nhau	Độ phức tạp thấp, chi phí thấp; mở rộng xa hơn mức thời trang thừa nhận — “nhàm chán” là một tính năng
Microservices	Nhiều dịch vụ nhỏ, được build, triển khai và mở rộng độc lập	Nhiều đội cần nhịp phát hành độc lập; mức mở rộng khác nhau dữ dội theo từng phân; ranh giới miền đã được chứng minh, ổn định	Đội nhỏ (“một distributed monolith là một monolith được cộng thêm các sự cố mạng”); miền còn đang dịch chuyển	Độ phức tạp cao: mỗi lời gọi hàm trở thành một lời gọi mạng có thể hỏng; cần CI/CD, observability, on-call trưởng thành

Phán quyết của kiến trúc sư: **bắt đầu bằng monolith, mô-đun hóa bên trong; tách dịch vụ ra khi — và chỉ khi — một cơn đau về mở rộng đội ngũ hoặc mở rộng tải thực sự xuất hiện**. Nói điều này trong phỏng vấn, kèm lý do, đánh dấu bạn là senior; giáo điều theo bất kỳ hướng nào đánh dấu bạn là junior.

Event-driven architecture (kiến trúc hướng sự kiện) — queue và topic, những bộ giảm xóc. Thay vì các thành phần gọi nhau và *chờ đợi* (đồng bộ), các thành phần công bố **event** (sự kiện — “OrderPlaced”) vào một **queue** (hàng đợi — SQS — một consumer lấy mỗi thông điệp, theo nhịp của riêng nó) hoặc một **topic** (SNS — mọi subscriber nhận một bản sao; fan-out). Bên sản xuất không biết và không quan tâm ai đang nghe. Dùng khi: các thành phần nên sống sót qua sự cố của nhau (queue giữ thông điệp trong khi một consumer sập — sự bền bỉ và sự tách rời trong một lần mua); tải tăng vọt (queue hấp thụ cơn sốt; các worker rút cạn nó đều đặn); một sự kiện kích hoạt nhiều phản ứng (đơn hàng được đặt — trừ tiền, email, tồn kho, phân tích — bốn subscriber, không chút ràng buộc nào). KHÔNG dùng khi: người dùng cần câu trả lời *ngay bây giờ* (checkout không thể xác nhận “rời sẽ có”); hoặc đội nhỏ và một lời gọi đồng bộ đơn giản là đủ — mỗi queue thêm ngữ nghĩa giao nhận (at-least-once nghĩa là các consumer phải **idempotent** — an toàn khi chạy hai lần), xử lý dead-letter, và một gánh nặng giám sát. Chi phí/độ phức tạp: các thành phần rẻ, gỡ lỗi đắt hơn — câu chuyện của một request giờ đây rải rác qua các dịch vụ và queue, đó là lý do observability (Module 7) từ đây ngừng là tùy chọn.

Three-tier: đã là của bạn (Module 3). Dùng cho: dải giữa rộng lớn của các ứng dụng web — nó là mẫu hình mà các mẫu khác được đo theo. Không dùng cho: các workload tăng vọt dữ dội

với những quang trũng nhân rồi (serverless rẻ hơn) hoặc các sản phẩm đa-đội thực sự khổng lồ (xem ở trên). Hồ sơ: được hiểu rõ ở mọi nơi, dễ tuyển người, chi phí trung bình.

Serverless-first: ghép các mảnh được quản lý — API Gateway → các Lambda function — DynamoDB, S3 và các queue — không sở hữu server nào cả. Dùng khi: lưu lượng tăng vọt hoặc thấp (scale-to-zero nghĩa là chi phí nhân rồi ~0), đội nhỏ, ra thị trường nhanh, chất keo hướng sự kiện. KHÔNG dùng khi: compute chạy dài hoặc chuyên biệt (giới hạn runtime), các đường đi siêu nhảy độ trễ (cold start), tải Ổn định khổng lồ (giá theo-môi-lần-gọi có thể vượt server đặt trước — hãy làm phép tính ở quy mô), hoặc khi tính di động giữa các cloud là một yêu cầu thực sự (đây là mức lock-in sâu nhất trong mọi mẫu hình — thường đáng giá, nhưng phải nói ra thành tiếng). Hồ sơ: gánh nặng vận hành thấp nhất trong danh mục; chi phí tuyệt hảo ở quy mô thấp/tăng vọt, cần kiểm tra ở quy mô Ổn định cao.

Từ vựng:

Thuật ngữ	Định nghĩa
Monolith / modular monolith	Một ứng dụng triển khai duy nhất với mọi thứ trong đó / phiên bản kỷ luật: một bản triển khai, ranh giới mô-đun nội bộ sạch sẽ — mặc định tốt nhất cho các hệ thống mới.
Microservices	Nhiều dịch vụ nhỏ, triển khai độc lập, mỗi cái sở hữu dữ liệu riêng. Một công cụ mở rộng đội ngũ có thu thuế hệ-thống-phân-tán.
Coupling / decoupling	Mức độ các thành phần phụ thuộc vào sự sẵn sàng và chi tiết của nhau. Kiến trúc phân lớn là nghệ thuật mua đúng lượng decoupling.
Synchronous / asynchronous	Gọi-và-chờ vs. gửi-và-đi-tiếp. Lựa chọn nền tảng trên mỗi mũi tên bạn vẽ.
Event / event-driven	Một sự thật được loan báo cho bất kỳ ai lắng nghe (“OrderPlaced”) / một kiến trúc xây từ những lời loan báo như vậy.
Queue (SQS)	Một hàng thông điệp; mỗi cái được một consumer lấy theo nhịp của nó. Bộ giảm xóc và bộ tách rời.
Topic / pub-sub (SNS)	Một kênh phát sóng; mọi subscriber nhận mỗi thông điệp. Công cụ fan-out.
Dead-letter queue	Nơi các thông điệp thất bại xử lý nhiều lần được để riêng cho con người — lưới an toàn của mẫu hình.
Idempotent	An toàn khi xử lý hai lần với cùng kết quả — bắt buộc với consumer, vì queue có thể giao một thông điệp nhiều hơn một lần.
Serverless-first	Ghép các mảnh được quản lý, scale-to-zero (function, cơ sở dữ liệu được quản lý,

Thuật ngữ	Định nghĩa
Lock-in	queue) thay vì chạy server. Sự phụ thuộc vào các dịch vụ độc quyền của một nhà cung cấp, định giá bằng chi phí chuyển đổi. Không phải một tội lỗi — một thuật ngữ kinh tế cần cân nhắc công khai (Giai đoạn 4).

Bài tập: (1) Thiết kế luồng đặt hàng của ThaiTicket hai lần — ba tầng đồng bộ, rồi hướng sự kiện với SQS/SNS — và viết một đoạn văn về việc mỗi phiên bản làm gì trong một sự cố của nhà cung cấp thanh toán; đoạn văn đó là toàn bộ lập luận cho events. (2) Với bốn công ty (startup 3 người, scale-up 50 kỹ sư, một ngân hàng, một ứng dụng bình chọn truyền hình dùng 4 đêm/năm), chọn một mẫu hình và bảo vệ nó — rồi nêu tên cú hích sẽ khiến mỗi công ty đổi mẫu hình. (3) Tìm một câu chuyện blog kỹ thuật thực tế “chúng tôi chuyển sang microservices và hối hận” và một câu chuyện thành công; ghi nhật ký điểm khác biệt (gần như luôn là kích cỡ đội và độ chín của miền).

Milestone: với một bản mô tả công ty, bạn có thể đề xuất một mẫu hình *kèm lối thoát của nó* — “bắt đầu ở đây; khi X xảy ra, tiến hóa sang Y” — trong năm phút. Các lộ trình tiến hóa, không phải các bản án, mới là cách kiến trúc sư thực sự nói chuyện.

Module 9 (Tuần 18–20): Các mẫu hình dữ liệu, toàn cầu và hybrid

Kiến trúc dữ liệu — lake vs. warehouse. Các cơ sở dữ liệu giao dịch (Giai đoạn 1) vận hành doanh nghiệp; mắng phân tích muốn *đặt câu hỏi* với nó mà không làm nó chậm đi. Một **data warehouse** (kho dữ liệu — Redshift, Snowflake, BigQuery) lưu dữ liệu *có cấu trúc, đã làm sạch*, tối ưu cho phân tích SQL nhanh — dùng khi các câu hỏi đã biết trước và các dashboard phải nhanh; tốn hơn mỗi TB, cần kỷ luật mô hình hóa từ đầu. Một **data lake** (hồ dữ liệu — S3 + catalog + các engine truy vấn như Athena) lưu *mọi thứ, thô, rẻ* — log, clickstream, hình ảnh — và áp cấu trúc lúc đọc; dùng khi bạn muốn giữ mọi thứ ngay bây giờ và quyết định câu hỏi sau; nhưng không được quản trị thì nó suy biến thành trò đùa của ngành, cái **data swamp** (đầm lầy dữ liệu). Sự đồng thuận hiện đại là cả hai, xếp lớp (“lakehouse”): sự thật thô trong lake, các mart tuyển chọn trong warehouse, được nuôi bởi các **pipeline ETL/ELT**. Các quy tắc của kiến trúc sư: mắng phân tích không bao giờ truy vấn cơ sở dữ liệu production (replica hoặc pipeline nuôi nó), và mỗi bộ dữ liệu có một chủ sở hữu, một mục catalog, và một chính sách lưu giữ — dòng quản trị đó là một câu trong bản thiết kế của bạn và là một năm đau khổ nếu bạn bỏ sót.

Multi-region: active-passive vs. active-active. Khi một region là không đủ — vì doanh nghiệp đòi hỏi DR trước các thảm họa cấp region, hoặc người dùng trải dài các châu lục — bạn chọn:

Mẫu hình	Nó là gì	Dùng khi	KHÔNG dùng khi	Chi phí/độ phức tạp
Active-passive	Một region phục vụ; một region standby	RTO/RPO do doanh nghiệp tuyên bố biện	Chưa ai ký RTO/RPO biện minh cho khoản	1,1×–1,7× chi phí theo độ âm của standby; độ

Mẫu hình	Nó là gì	Dùng khi	KHÔNG dùng khi	Chi phí/độ phức tạp
	giữ dữ liệu sao chép, từ “chỉ backup” (lạnh) đến “bản sao thu nhỏ đang chạy” (ấm), được thăng cấp khi có thảm họa	minh cho nó; sự tuân thủ đòi hỏi một câu chuyện DR	chi (như câu trung thực của đa số công ty là multi-AZ tốt + backup xuyên region)	phức tạp vừa — yêu cầu chỉ mạng là <i>failover</i> mà <i>bạn thực sự diễn tập</i> , nếu không standby chỉ là sân khấu
Active-active	Hai region trở lên phục vụ đồng thời; người dùng được định tuyến đến nơi gần nhất; dữ liệu sao chép hai chiều	Cơ sở người dùng toàn cầu muốn độ trễ nội địa; RTO gần-không thực sự bắt buộc (mạng lưới thanh toán, giao dịch tài chính, SaaS lớn)	Gần như tất cả những người còn lại — <i>xung đột ghi</i> xuyên region là một trong những bài toán thực sự khó của ngành máy tính	2×+ chi phí, độ phức tạp cao nhất khóa học này; cần các kho dữ liệu giải quyết xung đột và các đội senior

Câu nói cấp-phỏng-vân: “*Multi-AZ is table stakes; multi-region is a business case.*” (Multi-AZ là điều kiện tối thiểu; multi-region là một bài toán kinh doanh.) Hãy để doanh nghiệp tuyên bố RTO/RPO, định giá các phương án, và để các con số lựa chọn.

Hybrid cloud. Một phần on-prem, một phần cloud, nối bằng VPN hoặc Direct Connect — với đa số doanh nghiệp lâu đời, không phải một mẫu hình mà là một *thực tại kéo dài cả thập kỷ*: những mainframe không thể di dời, các hệ thống nhà máy bị trói bởi độ trễ, dữ liệu bị quản chế ghim tại chỗ, và một cuộc migration (Giai đoạn 4) đang đi ngang qua. Dùng như: một cây cầu có chủ đích với một hướng di chuyển. KHÔNG chấp nhận: “hybrid” như một uyển ngữ cho “chúng tôi chưa bao giờ quyết định.” Ghi chú thiết kế: danh tính phải được hợp nhất trước tiên (một lần đăng nhập xuyên cả hai thế giới — thứ mà các JD doanh nghiệp muốn nói bằng “hybrid identity integration”); nêu đích danh hệ thống nào là nguồn sự thật; canh chừng chi phí egress-quá-đường-truyền; và dự liệu rằng đường mạng sẽ là một SPOF trừ khi được nhân đôi. Độ phức tạp: hai cơ ngơi với đủ thứ của mỗi bên — lý do trung thực khiến kiên trúc sư thúc đẩy thu nhỏ phía on-prem một cách đều đặn.

Từ vựng:

Thuật ngữ	Định nghĩa
OLTP / OLAP	Xử lý giao dịch (nhiều lượt đọc/ghi nhỏ và nhanh — vận hành doanh nghiệp) vs. xử lý phân tích (nhiều lượt quét khổng lồ — nghiên cứu doanh nghiệp). Hãy tách chúng.
Data warehouse	Kho lưu trữ có cấu trúc, được tuyển chọn, tối ưu cho phân tích SQL nhanh (Redshift, Snowflake, BigQuery).

Thuật ngữ	Định nghĩa
Data lake / data swamp	Kho rẻ chứa mọi thứ, thô, cấu trúc hóa lúc đọc (S3 + Athena) / phiên bản không được quản trị, nơi dữ liệu đến để bị thất lạc.
ETL / ELT	Các pipeline chuyển dữ liệu từ hệ thống nguồn vào lake/warehouse (Extract, Transform, Load — thứ tự thay đổi).
Data governance / catalog / retention	Quy tắc về quyền sở hữu, tài liệu và vòng đời cho mỗi bộ dữ liệu — một câu thiết kế tiết kiệm một năm đau khổ.
Replication (sync / async)	Sao chép dữ liệu liên tục sang một cơ sở dữ liệu hoặc region khác — nhất-quán-tức-thì-nhưng-giới-hạn-khoảng-cách vs. hơi-trễ-nhưng-đi-bất-cứ-đâu. Độ trễ async là nơi RPO sinh ra.
Active-passive / failover drill	DR bằng region standby / buổi diễn tập theo lịch chứng minh nó hoạt động. Một failover chưa diễn tập là một niềm hy vọng.
Active-active	Nhiều region phục vụ cùng lúc với sao chép hai chiều. Mạnh mẽ; thực sự khó; thường không cần thiết.
Write conflict	Hai region thay đổi cùng một dữ liệu cùng lúc — lý do kỹ thuật khiến active-active là lãnh địa chuyên gia.
Hybrid cloud / hybrid identity	On-prem + cloud nối thành một cơ ngơi / một lần đăng nhập xuyên cả hai — thứ đầu tiên phải hợp nhất.
Direct Connect	Đường truyền vật lý riêng nối trung tâm dữ liệu với cloud — cuống rốn của hybrid, nhân đôi nếu nó quan trọng.

Bài tập: (1) ThaiTicket vươn ra khu vực: thiết kế cuộc mở rộng sang Singapore hai lần — active-passive (Bangkok là primary) và active-active — với bội số chi phí và RTO/RPO cho mỗi cái; viết bản khuyến nghị một trang và *ra một quyết định*. (2) Một nhà bán lẻ có 15 năm dữ liệu bán hàng trong một cơ sở dữ liệu SQL production và muốn “phân tích sẵn sàng cho AI”; phác họa thiết kế lake + warehouse và viết ba câu quản trị. (3) Nhận diện mẫu hình: chọn ba kiến trúc tham chiếu từ <https://aws.amazon.com/architecture/> và nêu tên mọi mẫu hình danh mục có mặt trong mỗi cái.

Milestone — kết thúc Giai đoạn 3: màn thử thách danh mục: một AI đưa bạn sáu kịch bản đôn dập; với mỗi cái bạn nêu tên mẫu hình, lý do, lời cảnh báo khi-KHÔNG, và hồ sơ chi phí/độ phức tạp — dưới năm phút mỗi cái. Chính cuộc trao đổi này, ở chính tốc độ này, là một phân ba giữa của một buổi phỏng vấn kiến trúc sư thực thụ.

GIẢI ĐOẠN 4 — MIGRATION VÀ THẾ GIỚI THỰC (Tuần 21–26)

Thiết kế greenfield là phân thiêu số của công việc. Đa số kiến trúc diễn ra trong các công ty đã tồn tại sẵn — với những phòng server, phần mềm cổ xưa mà doanh nghiệp phụ thuộc vào, các hợp đồng, và luật pháp. Giai đoạn này là thế giới đó, và là nơi “lead migration and modernization initiatives” (dẫn dắt các sáng kiến migration và hiện đại hóa) — một gạch đầu dòng trong gần như mọi JD chúng tôi nghiên cứu — được dạy.

Module 10 (Tuần 21–23): Migration — đưa một công ty lên cloud

7 chữ R — vốn từ chung của mọi cuộc trò chuyện migration. Với môi workload trong cơ ngơi, bạn chọn một:

R	Ý nghĩa	Khi nào	Công sức / thành quả
Retire	Tắt nó đi — thực ra chẳng ai dùng	Mọi cơ ngơi có 10–20% loại này; tìm chúng trước tiên	Không đáng kể / tiết kiệm tức thì — chữ R tốt nhất
Retain	Để lại on-prem, tạm thời	Các hệ thống bị trói bởi độ trễ, bị ghim bởi tuân thủ, hoặc sắp hết đời	Không / hoãn chi phí — một lời “chưa phải lúc” trung thực
Rehost (“lift and shift”)	Chuyển lên VM cloud nguyên trạng	Tốc độ quan trọng, ứng dụng ổn định, kỹ năng còn mỏng	Thấp / thoát nhanh khỏi trung tâm dữ liệu, nhưng chưa có mấy lợi ích cloud — một bước đầu, không phải đích đến
Relocate	Chuyển ở tầng hypervisor (ví dụ hạm đội VMware sang VMware-on-cloud)	Các cơ ngơi ảo hóa lớn với một deadline	Thấp / cú chuyển hàng loạt nhanh nhất; một điểm dừng chân
Repurchase (“drop and shop”)	Thay thế bằng SaaS	Các ứng dụng không tạo khác biệt — email, HR, CRM	Thấp-vừa / cả hạ tầng rời khỏi cơ ngơi của bạn
Replatform (“lift, tinker, shift”)	Nâng cấp nhỏ trong lúc chuyển — cơ sở dữ liệu tự quản — RDS, ứng dụng — container	Khoảng giữa thực dụng: lợi ích thật, rủi ro có giới hạn	Vừa / chữ R chủ lực của đa số các cuộc migration
Refactor / re-architect	Viết lại cho cloud-native (các mẫu hình Giai đoạn 3)	Các hệ thống lõi tạo khác biệt mà giới hạn của chúng làm doanh nghiệp đau	Cao / thành quả cao nhất — dành nó cho số ít workload xứng đáng

Phương pháp migration: assess – mobilize – migrate (đánh giá – chuẩn bị – di trú).

Assess: kiểm kê mọi thứ (công cụ khám phá cộng với ngành khảo cổ của việc hỏi con người), và chấm điểm mỗi workload theo giá trị kinh doanh và độ khó migration — đầu ra: một bản kiểm kê ứng dụng với một chữ R mỗi hàng, và một **business case** (TCO của việc ở lại vs. chuyển đi; hãy trung thực rằng hóa đơn *tăng lên* trong giai đoạn chông lún khi cả hai cơ ngơi cùng chạy — lãnh đạo không được cảnh báo về cái bong bóng migration sẽ trở thành lãnh đạo hủy migration giữa chừng). **Mobilize:** xây **landing zone** — nền móng cloud được quản trị, dựng sẵn *trước khi* workload đầu tiên hạ cánh: cấu trúc đa tài khoản (tài khoản riêng cho mỗi môi trường và đội, để bán kính sát thương giữ được nhỏ), danh tính và logging tập trung, hub mạng (các VPC hub-and-spoke, đường Direct Connect về on-prem), và **guardrails** — các chính sách tự động khiến con đường an toàn, có tag, tuân thủ trở thành mặc định (AWS Control Tower là bộ khởi đầu được quản lý). Các JD nói “design the landing zone every workload inherits” (thiết kế landing zone mà mọi workload thừa hưởng) — chính là cái này. Bỏ qua nó để “cứ bắt đầu migrate đi” là tái tạo cái trung tâm dữ liệu bừa bộn trong cloud với chi phí cao hơn; đó là chữ ký kinh điển của migration thất bại. **Migrate theo đợt (wave):** nhóm các workload thành từng đợt vài cái, xếp dễ-trước: Đợt 1 cố ý chọn rủi ro thấp (học bộ máy ở nơi sai lầm còn rẻ), các đợt sau đưa những viên ngọc quý đi với các cuộc cutover đã tập dượt, mỗi đợt có một kế hoạch rollback và một giai đoạn **hypercare** — thời kỳ canh chừng tăng cường. Với mỗi cú chuyển cơ sở dữ liệu, hai câu hỏi quan trọng chính là các con số của Module 4 trá hình: cuộc cutover được phép mất bao nhiêu downtime (RTO) — và sao chép liên tục với một cú chuyển ngăn tồn tại cho trường hợp câu trả lời là “gần như không.”

Từ vựng:

Thuật ngữ	Định nghĩa
7 Rs	Retire, retain, rehost, relocate, repurchase, replatform, refactor — thực đơn migration theo từng workload. (Bạn cũng sẽ nghe “6 Rs” — danh sách cũ hơn không có relocate.)
Discovery / application inventory	Tìm ra cái gì thực sự đang chạy (công cụ + phỏng vấn) / danh sách kết quả, một hàng mỗi workload, với chủ sở hữu, các phụ thuộc, và chữ R của nó.
Dependency mapping	Vẽ sơ đồ cái gì nói chuyện với cái gì — lý do các workload di trú theo nhóm, và các cuộc migration thiếu nó thất bại ngay ngày đầu.
Business case / migration bubble	Lập luận TCO ở-lại-vs-chuyển-đi / cái bước chỉ phí tạm thời khi cả hai cơ ngơi cùng chạy. Cảnh báo về nó hoặc bị nó phục kích.
Landing zone	Nền móng cloud được quản trị, dựng sẵn — tài khoản, danh tính, mạng, logging, guardrail — mà mọi workload thừa hưởng. Xây <i>trước</i> migration.
Multi-account strategy	Các tài khoản cloud riêng cho mỗi môi

Thuật ngữ	Định nghĩa
	trường/đội, để hóa đơn truy được trách nhiệm và bán kính sát thương được khoanh vùng.
Guardrail	Một chính sách tự động ngăn chặn hoặc ngăn cản các hành động không tuân thủ — quản trị bằng máy móc, không phải bằng công văn.
Control Tower	Dịch vụ được quản lý của AWS để dựng một landing zone đa tài khoản với các guardrail.
Wave plan	Lịch migration theo các nhóm nhỏ, dễ-trước, các phụ thuộc đi cùng nhau.
Cutover / rollback / hypercare	Khoảnh khắc chuyển sang bản sao cloud / cú hoàn tác đã tập dượt / cửa sổ hỗ trợ tăng cường sau đó.

Bài tập: (1) Cuộc migration trên giấy: một AI tạo ra bản kiểm kê một công ty hư cấu 40 server (bạn sẽ gặp lại nó ở Capstone 2); gán một chữ R cho mỗi hàng và bảo vệ mười quyết định khó nhất. (2) Phác họa một landing zone: sơ đồ tài khoản, luồng danh tính, mạng hub-and-spoke, năm guardrail bạn sẽ cưỡng chế từ ngày đầu. (3) Viết lời cảnh báo “migration bubble” hai đoạn văn cho một CFO — luyện báo-tin-xấu-sớm là bài thể dục tim mạch của kiến trúc sư.

Milestone: với một bản kiểm kê ~20 workload kèm mô tả, bạn tạo ra một bảng R-môi-workload bảo vệ được, một kế hoạch ba đợt kèm lý do, và một bản phác landing zone — trong một lần ngồi.

Module 11 (Tuần 24–26): Các ràng buộc — luật pháp, hệ thống cũ, và lock-in

Data residency và luật riêng tư như các đầu vào kiến trúc. Các luật dữ liệu cá nhân — PDPA của Thái Lan, GDPR của châu Âu, và họ hàng của chúng trên toàn thế giới — chia sẻ một hình dạng mà kiến trúc sư phải thuộc lòng: dữ liệu cá nhân cần một cơ sở hợp pháp (thường là sự đồng ý); các cá nhân có quyền (truy cập, sửa chữa, xóa bỏ — thiết kế của bạn phải tìm được và xóa được dữ liệu của một người, điều rất khó nếu bạn đã rải nó qua các bản sao không được quản trị); các vụ rò rỉ mang nghĩa vụ **thông báo trong 72 giờ** (hệ thống logging của bạn phải cho phép kể lại câu chuyện trong thời gian ngắn hơn); và **các quy tắc chuyển xuyên biên giới** ràng buộc nơi dữ liệu được phép sống về mặt vật lý — đó là một ràng buộc *chọn region*, một ràng buộc *thiết kế sao chép* (bản sao phân tích ở Singapore kia có thể là một sự kiện pháp lý), và một lý do các region trong nước tồn tại. Phương pháp của kiến trúc sư, theo thứ tự: phân loại dữ liệu (cái gì là cá nhân?), vẽ bản đồ các hành trình của nó (mọi kho, mọi bản sao, mọi biên giới bị băng qua — những dòng chảy không ai vẽ là nơi các vi phạm sinh sống), rồi thiết kế các biện pháp kiểm soát: các region tuân thủ residency, mã hóa, lịch lưu giữ, và một đường xóa thực sự chạm đến kỳ hết hạn của các bản backup. Hãy nói “data protection by design” (bảo vệ dữ liệu ngay từ thiết kế) — đó là cụm từ cả hai luật dùng, và nó đúng nghĩa đen là chức danh của bạn trong một câu.

Tích hợp hệ thống cũ (legacy). Hệ thống hóa đơn trên mainframe sẽ không di dời trong năm nay, và ứng dụng cloud bóng bẩy phải nói chuyện với nó. Các mẫu hình: một **anti-corruption layer** (lớp chống nhiễm bẩn) — một dịch vụ phiên dịch giữa mới và cũ, để các hình dạng dữ liệu kỳ quặc của hệ thống cũ không rò rỉ vào và làm hỏng thiết kế mới của bạn; **strangler fig** (cây đa bóp nghẹt) — định tuyến lưu lượng qua một mặt tiền, rồi bóc từng chức năng khỏi hệ thống cũ từng cái một cho đến khi, nhiều năm sau, nó có thể được tắt đi (đặt tên theo loài đa dân dần bao trùm cây chủ; nó thắng các cuộc viết lại big-bang, thứ thất bại với tỷ lệ huyền thoại); các câu batch và vòi sự kiện cho dữ liệu phải chảy giữa hai thế giới; và sự tôn trọng vật lý độ trễ của các lời gọi cloud-đến-on-prem làm lời (Direct Connect giúp ích; tốt hơn nữa, thiết kế để loại sự làm lời đó ra). Quy tắc: **khoanh vùng hệ thống cũ, đừng nhiễm nó** — mỗi thành phần mới nên được xây như thể hệ thống cũ đã biến mất rồi.

Kinh tế học vendor lock-in. Mỗi dịch vụ được quản lý tiện lợi làm sâu thêm cuộc hôn nhân của bạn với một nhà cung cấp; tính di động (các lớp trừu tượng đa-cloud, tự-quản-mọi-thứ) là một lựa chọn thật có giá thật bằng độ phức tạp và tốc độ bị từ bỏ. Không cái nào là tội lỗi — chế độ thất bại không phải là *chọn* lock-in, mà là không *nhận ra*. Công cụ của kiến trúc sư là một ước tính chi phí thoát được viết vào bản thiết kế: “Dùng DynamoDB tiết kiệm ≈2 năm-kỹ-sư ngay bây giờ; chuyển đổi về sau ≈ một cuộc viết lại 6 tháng của tầng dữ liệu — chúng ta chấp nhận điều này, và đây là ranh giới interface sẽ thu nhỏ cuộc viết lại.” Ba câu, định giá trung thực, quyết định được ghi lại (các ADR của Giai đoạn 5) — đó là quản lý lock-in trưởng thành. Multi-cloud *như một chiến lược* (chạy cùng một workload di động trên hai cloud) thường là câu trả lời đắt nhất có thể và được mua bởi chính xác những tổ chức mà quy mô hoặc cơ quan quản lý của họ đòi hỏi nó; multi-cloud *như một thực tế* (các workload khác nhau trên các cloud khác nhau do lịch sử hoặc do độ phù hợp) là cuộc sống bình thường.

Từ vựng:

Thuật ngữ	Định nghĩa
PDPA / GDPR	Luật bảo vệ dữ liệu cá nhân của Thái Lan và châu Âu — cặp khuôn mẫu cho các ràng buộc luật-riêng-tư trên toàn thế giới.
Data protection by design	Xây các biện pháp kiểm soát riêng tư vào kiến trúc ngay từ bản phác đầu tiên — cụm từ pháp lý và bốn phần của kiến trúc sư.
Lawful basis / consent	Sự biện minh pháp lý bắt buộc cho việc xử lý dữ liệu cá nhân.
Right to erasure	Quyền của một cá nhân được xóa dữ liệu của mình — điều mà thiết kế của bạn phải làm cho <i>khả thi</i> .
Data flow mapping	Vẽ sơ đồ mọi kho, bản sao và làn băng-quá-biên-giới của một hạng mục dữ liệu. Nơi các dòng chảy chưa được vẽ, các vi phạm sinh sống.
Cross-border transfer	Dữ liệu cá nhân rời khỏi đất nước — bị quản chế; biến thiết kế sao chép thành một câu hỏi pháp lý.

Thuật ngữ	Định nghĩa
Anti-corruption layer	Một thành phần phiên dịch giữ cho các thói tật của hệ thống cũ không rò rỉ vào một thiết kế mới.
Strangler fig	Hiện đại hóa bằng cách định tuyến qua một mặt tiền và thay thế hệ thống cũ từng mảnh cho đến khi có thể tắt nó.
Big-bang rewrite	Thay thế một hệ thống cùng một lúc, cutover trong một ngày. Tỷ lệ thất bại huyền thoại; strangler fig tồn tại vì nó.
Exit cost / switching cost	Chi phí được định giá thực tế của việc rời một nhà cung cấp hay dịch vụ — con số biên lock-in từ nỗi sợ thành kinh tế học.
Multi-cloud (chiến lược vs. thực tế)	Chủ đích chạy di động trên nhiều cloud (đắt, hiếm khi được biện minh) vs. đơn giản là có các workload trên vài cloud (bình thường).

Bài tập: (1) Vẽ bản đồ dòng dữ liệu khách hàng của ThaiTicket: mọi kho, mọi bản sao (đừng quên log, backup, pipeline phân tích, các bản xuất của đội hỗ trợ), mọi biên giới; rồi viết đường xóa. Cảm nhận cách tám bản đồ tìm ra những vấn đề mà sơ đồ che giấu. (2) Thiết kế kế hoạch strangler-fig cho một hệ thống tồn kho on-prem 20 năm tuổi, nêu tên ba lần bóc đầu tiên. (3) Viết đoạn văn lock-in kiểu DynamoDB (lợi ích bây giờ, chi phí thoát về sau, chấp nhận hay giảm nhẹ) cho ba dịch vụ bạn sẽ thực sự dùng.

Milestone — kết thúc Giai đoạn 4: màn thử thách ràng buộc: một đề bài thiết kế cài đủ cả ba (“công ty bảo hiểm Thái, dữ liệu khách hàng, hệ thống hợp đồng trên mainframe, hội đồng quản trị lo lắng về sự phụ thuộc AWS”) — bạn tạo ra một hướng tiếp cận một trang chạm đến residency, tích hợp và lock-in, mỗi cái với một mẫu hình được nêu tên và một chi phí trung thực. Khi các ràng buộc làm bạn phấn khích hơn cả greenfield — vì ràng buộc là nơi kiến trúc sư kiếm nhiều hơn những người-vẽ-sơ-đồ — Giai đoạn 4 hoàn tất.

GIAI ĐOẠN 5 — NGHỀ CỦA KIẾN TRÚC SƯ (Tháng 7-12)

Mọi thứ đến giờ làm cho bạn có khả năng *thiết kế*. Giai đoạn này làm cho bạn có khả năng *làm việc như một kiến trúc sư* — các tài liệu, buổi review, bài thuyết trình và bằng chứng mà vai trò này thực sự được cấu thành từ đó, cộng với các chứng chỉ đưa bạn qua cửa HR, và ba capstone trở thành portfolio của bạn.

Module 12 (Tháng 7-8): Các sản phẩm bàn giao — ADR, bộ sơ đồ, và điều hành review

Architecture Decision Records (ADR — hồ sơ quyết định kiến trúc). Một ADR là một tài liệu một trang ghi lại một quyết định quan trọng: chúng ta đã chọn gì, đã không chọn gì, và vì

sao — được viết *ngay khi quyết định được đưa ra*, được đánh số, và được giữ trong repository của dự án mãi mãi. Vì sao các JD gọi tên chúng: hai năm nữa, ai đó sẽ hỏi “trời ạ vì sao cái này lại là DynamoDB?” và ADR trả lời trong ba mươi giây — với bối cảnh, các ràng buộc, và những phương án thay thế đã được cân nhắc trung thực. Các đội có ADR không xử lại vụ án nào; các đội không có thì cãi vòng quanh hằng năm. Mẫu — học thuộc nó:

```
# 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.
```

Viết một ADR cho mỗi quyết định quan trọng trong mọi capstone. Một portfolio phỏng vấn chứa các ADR thật là thứ hiếm hoi và hiệu quả đến tàn khốc.

Bộ sơ đồ — một hệ thống, ba mức thu phóng. Tài liệu kiến trúc thực thụ là một bộ (mô hình C4 đã phổ biến hóa kỷ luật này): **context diagram** (sơ đồ bối cảnh) — hệ thống là một chiếc hộp, với người dùng và các hệ thống bên ngoài nó chạm tới; góc nhìn cho lãnh đạo/người mới, và là cái bạn sẽ trình bày cho ban điều hành. **Container diagram** (sơ đồ container) — phóng to: các mảnh đang chạy chính (web app, API, cơ sở dữ liệu, queue, cache) và cách chúng nói chuyện; góc nhìn kỹ thuật hằng ngày, đại khái là các bản vẽ Giai đoạn 1–3 của bạn. **Deployment diagram** (sơ đồ triển khai) — tất cả chạy ở đâu về mặt vật lý: region, các AZ, VPC, subnet, các nhóm scaling; góc nhìn cho review, bảo mật và vận hành. Một hệ thống, ba khán giả, ba sơ đồ — được giữ cập nhật, có ngày tháng, trong hệ thống quản lý phiên bản cạnh các ADR. Một sơ đồ không tìm được hoặc không tin được là một sơ đồ không tồn tại.

Điều hành một buổi Well-Architected review. Bạn đã biết sáu trụ cột (Giai đoạn 2); đây là bản thân cuộc họp. **Trước:** chọn workload và phạm vi, cập nhật bộ sơ đồ, mời những người *vận hành* thứ đó (không chỉ những người thiết kế nó), và đặt tông giọng — đây là một buổi kiểm tra sức khỏe vì lợi ích của đội, không phải một cuộc kiểm toán cho hồ sơ của ai. **Trong (nửa ngày):** đi qua các trụ cột với các bộ câu hỏi của framework (AWS công bố chúng, kèm một Well-Architected Tool miễn phí trong console cấu trúc hóa toàn bộ bài tập — <https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html>); với mỗi câu trả lời, ghi lại phát hiện, và *xếp ưu tiên ngay khi đi* — một nhóm vấn đề rủi ro cao, không phải một trăm lời bắt bẻ. Tông giọng đặt câu hỏi của bạn chính là buổi review: “help me understand what happens when the payment provider times out” (giúp tôi hiểu điều gì xảy ra khi nhà cung cấp thanh toán hết giờ chờ) mở những cánh cửa mà “you don’t handle timeouts?” (các anh không xử lý timeout à?) đóng sập. **Sau:** một bản báo cáo ngắn — các phát hiện hàng đầu, mỗi cái với rủi ro, công sức và một khuyến nghị — và các hạng mục cải thiện đáp xuống backlog thật của đội với người phụ trách, nếu không buổi review chỉ là sân khấu. Thực hành: chạy một buổi review đầy đủ trên chính thiết kế Capstone 1 của bạn; tìm ra khuyết điểm của chính mình một cách có cấu trúc là kỹ năng lái-kép-nhanh-nhất trong khóa học này.

Bài tập: (1) Viết bù ADR cho năm quyết định lớn trong các thiết kế nhật ký Giai đoạn 2–3 của bạn — bạn sẽ tìm thấy ít nhất một cái mình không còn biện minh được, và đó chính là bài học. (2) Tạo bộ ba-sơ-đồ đầy đủ cho ThaiTicket. (3) Chạy buổi review giả lập ở trên, tạo bản báo cáo phát hiện một trang.

Milestone: một người lạ (hoặc một AI đóng vai) có thể cầm lấy tập hồ sơ ThaiTicket của bạn — ba sơ đồ + năm ADR — và trả lời đúng “đây là gì, nó chạy thế nào, và vì sao nó được xây như vậy?” mà không có bạn trong phòng. Tập hồ sơ đó *chính là* sản phẩm bàn giao của nghề kiến trúc sư.

Module 13 (Tháng 8–9): Nghề con người — thuyết trình, định giá, đối mặt phản bác, và cố vấn

Thuyết trình cho lãnh đạo vs. kỹ sư — một thiết kế, hai ngôn ngữ. Lãnh đạo mua kết quả; kỹ sư mua cơ chế. Với lãnh đạo: mở đầu bằng quyết định và con số (“Thiết kế này đáp ứng mục tiêu một triệu khách hàng ở mức \$1,90 mỗi đơn, và đây là hai quyết định tối cần từ các vị”); một slide, sơ đồ *context*, các rủi ro đóng khung như rủi ro kinh doanh (doanh thu, tuân thủ, danh tiếng), và đừng bao giờ nói “Kubernetes” khi “nền tảng” là đủ. Chuẩn bị cho ba câu hỏi lãnh đạo luôn hỏi: *cái này tốn bao nhiêu, cái gì có thể hỏng, vì sao không chọn phương án rẻ hơn?* Với kỹ sư: mở đầu bằng bài toán và các ràng buộc *trước* giải pháp (kỹ sư cảm được bài toán sẽ chấp nhận giải pháp; kỹ sư bị trao một bản án sẽ sẵn khuyết điểm theo nguyên tắc), cho xem sơ đồ container và deployment, tự mình nêu tên các đánh đổi và những phương án bị loại (uy tín đến từ những gì bạn thừa nhận), và chừa chỗ thật để thay đổi thiết kế — các câu hỏi trong phòng là review miễn phí. Thói quen kiến trúc sư tệ nhất là một bộ slide cho cả hai khán giả; thói quen tốt nhất là viết bản tóm tắt cho lãnh đạo *trước tiên*, vì nếu thiết kế không sống sót qua sự nén xuống năm câu, nó chưa hoàn thành.

Ước tính chi phí cho một bản đề xuất. Phương pháp: phân rã thiết kế thành ~10 thành phần mang chi phí; định giá từng cái ở mức tải kỳ vọng trong pricing calculator; **viết các giả định của bạn ra giấy** (request/ngày, tăng trưởng dữ liệu, egress — các giả định chính là bản ước tính; khi chúng thay đổi, con số di chuyển theo với một lượng tâm trong sạch); áp chiến lược đặt trước cho các phần ổn định; thêm các kịch bản tăng trưởng (hôm nay, 2×, 10× — lãnh đạo nhớ con số 10×); và trình bày dưới dạng một *khoảng kèm các động lực* (“\$55–70k/tháng, chủ yếu do egress — đây là cần gạt”), không bao giờ là một con số đơn chính-xác-giả-tạo. Bao gồm cả migration bubble nếu có. Ước tính thấp đi để được phê duyệt là tội kinh điển của kiến trúc sư non tay; dự án sẽ nhớ.

Đối mặt phản bác. Bạn sẽ bị ép về chi phí (“đắt quá” — quay về tam giác: “chúng ta có thể cắt \$30k bằng cách chấp nhận single-AZ — đây là phép toán về sự có; quyền quyết định là của anh, và tôi sẽ ghi lại” — làm cho sự đánh đổi tưởng mình và *được ghi lại* biến đa số phản bác thành hoặc sự đồng thuận hoặc một rủi ro được chấp nhận có hiểu biết, cả hai đều ổn); và khẩu vị (“kỹ sư thích một stack khác” — steel-man nó thành tiếng, rồi đưa nó về tiêu chí, không phải sở thích: độ khớp NFR, kỹ năng đội, hệ sinh thái, chi phí thoát — và khi thực sự sít sao, hãy để sở thích của kỹ sư thực thi thắng, mua được sự cam kết với giá rẻ); và về quyền uy (“bạn của CTO bảo dùng X” — đừng bao giờ đầu ý kiến bằng ý kiến; hỏi xin các tiêu chí, chạy X qua cùng một cuộc đánh giá công khai như mọi thứ khác, để ma trận trả lời một cách lịch sự). Và khi bạn sai — bạn sẽ thiết kế thứ gì đó thất bại; kiến trúc sư nào cũng vậy — nước đi là post-mortem không đổ lỗi áp lên chính mình, công khai, với ADR được cập nhật.

Không gì xây uy tín thập-kỷ nhanh hơn; không gì phá hủy nó nhanh hơn việc bảo vệ một xác chết.

Cô vấn kỹ sư. Cụm từ trong JD là “mentor the engineers who build against your standards” (cô vấn các kỹ sư xây theo chuẩn mực của bạn), và các hình thức thực tế là: **giờ tiếp chuyện design review** (một cánh cửa thường trực, để sự dẫn dắt diễn ra trước code, không phải sau); **review thiết kế của họ một cách tử tế** — câu hỏi trước phán quyết, và luôn kèm cái vì sao đằng sau chuẩn mực (một guardrail được giải thích chiều mộ một đồng minh; một guardrail bị áp đặt chiều mộ một đường vòng); **ủy quyền các quyết định thật** kèm lưới an toàn (“bạn sở hữu thiết kế cache; đây là các ràng buộc; tôi sẽ review, và tôi sẽ đứng sau quyết định của bạn”); và **dạy công khai** — mỗi ADR, mỗi bản viết review, mỗi buổi brown-bag nhân bạn lên vượt quá số giờ của chính bạn. Sự thật nghề nghiệp thẳng thừng: một kiến trúc sư thiên tài đơn độc sẽ chậm trễ; một kiến trúc sư đã nuôi năm kỹ sư thành đồng nghiệp biết thiết kế sẽ điều hành cả nghiệp vụ.

Bài tập: (1) Thuyết trình ThaiTicket hai lần — bản 5 phút cho lãnh đạo và bản 20 phút cho kỹ thuật — ghi âm cả hai, và nghe xem biệt ngữ có rõ ràng vào bản đầu không. (2) Tạo bản đề xuất chi phí viết đầy đủ (giả định, khoảng, động lực, kịch bản 2×/10×). (3) Sân khấu phản bác với một AI: ba hiệp — đòn tấn công chi phí từ một CFO, sở thích stack từ một kỹ sư senior, màn quyền uy bạn-của-CTO — ghi nhật ký cái gì hiệu quả. (4) Review thiết kế của một junior (một AI có thể tạo một bản đầy lỗi) bằng văn bản, câu-hỏi-trước, và để AI chấm điểm tông giọng của bạn.

Milestone: trận đấu kép: trình bày bản pitch cho lãnh đạo và sòng sót qua hai mươi phút hỏi-đáp hôn hợp thù-địch-nhưng-công-bằng (hội đồng AI: một CFO, một kỹ sư hoài nghi) mà không lẫn biệt ngữ, không phòng thủ, và không một đánh đổi nào chưa được viết ra.

Module 14 (Tháng 9–12): Chứng chỉ và ba capstone

Lộ trình chứng chỉ, với các mốc thời gian trung thực. Chứng chỉ không làm bạn thành kiến trúc sư — mười ba module trước đó mới làm — nhưng chúng mang lại cho bạn các buổi phỏng vấn, và việc ôn thi cho chúng khắc các dịch vụ của nhà cung cấp vào ngón tay bạn. Bậc thang, ở nhịp một-giờ-mỗi-ngày: **AWS Certified Solutions Architect – Associate (SAA-C03)** — 2–3 tháng; chứng chỉ kiến trúc sư được yêu cầu nhiều nhất trong các tin tuyển dụng, và sau Giai đoạn 1–3, phần lớn nó sẽ có cảm giác như ôn lại kèm tên dịch vụ gắn vào; thi nó trước. **AWS Certified Solutions Architect – Professional (SAP-C02)** — 4–8 tháng sau Associate; dựa trên tình huống, thực sự khó, và là dòng résumé mạnh nhất trong lĩnh vực này; phần migration và multi-account của Giai đoạn 4 là một nửa giáo trình của nó. **Azure AZ-305 (Azure Solutions Architect Expert)** — thêm vào nếu thị trường của bạn nặng về Microsoft (đa số thị trường doanh nghiệp ít nhất là song ngữ); dự trù 2–3 tháng với kiến thức AWS chuyển giao được với chiết khấu lớn (bắt đầu tại <https://learn.microsoft.com/en-us/credentials/certifications/azure-solutions-architect/>).

TOGAF Foundation — tùy chọn, 2–4 tuần; một số doanh nghiệp lớn đòi hỏi nó; nó chứng nhận *phương pháp* kiến trúc doanh nghiệp và vốn từ chứ không phải kỹ năng cloud. Trình tự cho khóa học này: ôn SAA song song với Tháng 7–8, thi vào ~Tháng 8; sau đó hoặc SA Pro (con đường kỹ thuật sâu) hoặc AZ-305 (con đường chiều rộng) đến hết Tháng 12, với SA Pro hoàn thành vào Tháng 14–16 theo nhịp trung thực.

Ba capstone. Mỗi cái là một tập hồ sơ hoàn chỉnh, cấp portfolio: bộ sơ đồ ba mức, 5+ ADR, ước tính chi phí kèm giả định, và một buổi Well-Architected review tự-điều-hành kèm các phát hiện. Dành 3–4 tuần cho mỗi cái. Ba sản phẩm này, cộng với nhật ký của bạn, *chính là* bằng chứng phổng ván của bạn cho “end-to-end solution design” (thiết kế giải pháp đầu-cuối).

Capstone 1 — greenfield: một nền tảng thương mại điện tử Thái cho 1 triệu người dùng. Đề bài: marketplace, 1 triệu người dùng đăng ký, 50k đồng thời ở đỉnh flash-sale, ưu tiên di động, tích hợp thanh toán kiểu PromptPay, tuân thủ PDPA, khả dụng 99,9%, tiết kiệm ngân sách kiểu vòng hạt giống. Phải bao gồm: lựa chọn mẫu hình kèm lộ trình tiên hóa, chiến lược caching, cuộc trò chuyện RTO/RPO viết dưới dạng hội thoại, ước tính unit economics, bản đồ dòng dữ liệu cho dữ liệu cá nhân.

Capstone 2 — brownfield: di trú một công ty on-prem 40 server. Đề bài: một công ty logistics Thái; 40 server (ERP trên Oracle, ứng dụng quản lý kho 12 năm tuổi, các file share, AD, các ứng dụng phòng ban linh tinh — tạo bản kiểm kê đầy đủ với một AI và đóng băng nó); hợp đồng thuê trung tâm dữ liệu hết hạn sau 14 tháng; hội đồng quản trị muốn thoát ra. Phải bao gồm: bảng kiểm kê 7-R đầy đủ, kế hoạch ba đợt với lý do về phụ thuộc, thiết kế landing zone, dòng thời gian chi phí migration-bubble, và bản ghi nhớ cho CFO.

Capstone 3 — bài khó: DR multi-region cho một fintech. Đề bài: một công ty thanh toán bị quản chế ở mức RTO ≤ 15 phút / RPO ≈ 0 cho sổ cái, với các ràng buộc data-residency về dữ liệu khách hàng và một cuộc kiểm tra failover hằng năm có cơ quan quản lý chứng kiến. Phải bao gồm: phân tích active-passive vs. active-active kèm chi phí, thiết kế sao chép kèm bản đồ residency, runbook failover, và kế hoạch diễn tập.

Thang tự đánh giá (áp cho mỗi capstone, một cách trung thực, trong nhật ký của bạn):

Chiều	1 — Chưa được	3 — Vững	5 — Tuyển người này
Yêu cầu	Nhảy thẳng vào giải pháp	NFR được nêu và truy được đến các lựa chọn thiết kế	Tam giác đánh đổi được định vị, các xung đột được nêu lên cho “doanh nghiệp”, các quyết định được trích xuất
Chất lượng thiết kế	SPOF còn sót; mẫu hình lệch pha	Mẫu hình đúng, multi-AZ, các hỏng hóc được xử lý	Lập luận khi-KHÔNG được thể hiện; lộ trình tiên hóa; thiết kế đơn giản nhất đáp ứng yêu cầu
Sáu trụ cột	Các trụ cột bị bỏ qua	Môi trụ cột được xử lý thấy được	Buổi review tự-điều-hành tìm ra khuyết điểm thật — và thiết kế được sửa để đáp lại
Chi phí	Không có con số	Ước tính từng hạng mục, giả định được	Khoảng kèm động lực, unit economics,

Chiều	1 — Chưa được	3 — Vững nêu	5 — Tuyển người này
Tài liệu	Chỉ có sơ đồ	Bộ ba mức + ADR, ký hiệu đúng	kịch bản 10×, chiến lược đặt trước Một người lạ trả lời được “cái gì/thế nào/vì sao” chỉ từ tập hồ sơ
Giao tiếp	Một sản phẩm cho mọi khán giả	Có phiên bản cho lãnh đạo và cho kỹ sư	Bản tóm tắt năm câu sống sót; phần bác được trả lời trước bằng văn bản

Chăm mọi chiều 4+ trên cả ba capstone — sửa cho đến khi đạt — và bạn đã hoàn thành khóa học.

Mười câu hỏi phỏng vấn kiến trúc sư (kèm câu trả lời mạnh)

Luyện những câu này thành tiếng; sức mạnh nằm ở *cấu trúc* của mỗi câu trả lời, thứ giờ đây bạn đã sở hữu.

1. “Design a URL shortener / ticketing site / photo app for a million users.” (Thiết kế một trình rút gọn URL / trang bán vé / ứng dụng ảnh cho một triệu người dùng.)

Khuôn hình mạnh: yêu cầu trước tiên, thành tiếng (“Đọc áp đảo ghi? Mục tiêu khả dụng? Ngân sách?”) → định vị trên tam giác → bộ khung ba tầng hoặc serverless-first với multi-AZ, cache, CDN → tự nêu tên các đánh đổi không cần ai nhắc → kết thúc bằng bậc độ lớn chi phí và lộ trình tiến hóa 10×. Người phỏng vấn chăm chú *những câu hỏi bạn hỏi*, không phải những chiếc hộp bạn vẽ.

2. “When would you choose NoSQL over a relational database?” (Khi nào bạn chọn NoSQL thay vì cơ sở dữ liệu quan hệ?) “Tôi mặc định SQL — các bảo đảm về tính đúng đắn và kỹ năng phổ quát — cho đến khi một lý do được nêu đích danh lần át: mở rộng ngang cực đại, schema linh hoạt, hoặc truy cập key-value ở mức mili giây một chữ số. Khi đó tôi chọn hương vị NoSQL khớp với mẫu truy cập, và tôi viết ADR bao gồm cả những gì chúng ta từ bỏ: các phép join, và một phân ngữ nghĩa nhất quán.”

3. “Explain RTO and RPO, and how they drive design.” (Giải thích RTO và RPO, và cách chúng dẫn dắt thiết kế.) Định nghĩa cả hai một cách sắc gọn — “chúng là các quyết định kinh doanh với bảng giá tăng theo cấp số nhân” — bậc thang: backup hằng đêm — sao chép liên tục — standby âm — active-active, với chi phí tăng ở mỗi nấc — “việc của tôi là làm cho doanh nghiệp chọn một cách có hiểu biết, rồi thiết kế chính xác theo con số — và diễn tập nó, vì một failover chưa kiểm chứng là một niềm hy vọng.”

4. “Monolith or microservices?” (Monolith hay microservices?) Phán quyết của Module 8, nguyên văn: bắt đầu bằng modular-monolith; tách ra khi cơn đau mở rộng đội ngũ hoặc mở

rộng rãi thực sự xuất hiện; microservices là một công cụ mở rộng đội ngũ có thu thuế hệ-thống-phân-tán. Điểm cộng cho câu “I’d rather run a good monolith than a bad distributed system” (Tôi thà vận hành một monolith tốt còn hơn một hệ thống phân tán tồi).

5. “How do you handle a large cloud bill / cost optimization?” (Bạn xử lý một hóa đơn cloud lớn / tối ưu chi phí như thế nào?) “Sự hữu hình trước tiên — tagging và showback; rồi vụ gặt theo thứ tự công sức: giết tài nguyên chết, rightsize từ số đo, phân bậc lưu trữ, lập lịch ngủ cho non-prod, rồi đặt trước phân nền ổn định. Và tôi báo cáo unit economics, không phải tổng số — một hóa đơn tăng với chi-phí-mỗi-đơn giảm là thành công, không phải vấn đề.”

6. “How would you migrate a legacy on-prem application?” (Bạn sẽ di trú một ứng dụng on-prem cũ như thế nào?) “Đánh giá trước khi chuyển — kiểm kê, các phụ thuộc, và một chữ R mỗi workload từ 7 chữ R; xây landing zone trước đợt một; các đợt đề-trước với những cuộc cutover đã tập dượt và kế hoạch rollback; và với cơ sở dữ liệu ngọc quý, cutover dựa trên sao chép được đo theo mức downtime doanh nghiệp đã ký. Thêm nữa: lời cảnh báo migration-bubble trung thực ngay từ đầu.”

7. “How do you secure a cloud architecture?” (Bạn bảo mật một kiến trúc cloud như thế nào?) Đi qua các lớp: danh tính (least privilege, MFA, không dùng root hằng ngày) — mạng (private subnet, phân đoạn, bề mặt tấn công tối thiểu) — dữ liệu (mã hóa khi lưu trữ/khi truyền, phân loại, residency) — phát hiện (nhật ký kiểm toán, cảnh báo) — “và ngay từ thiết kế, không gán thêm sau — biện pháp bảo mật rẻ nhất là quyết định kiến trúc gỡ bỏ hoàn toàn rủi ro, như không bao giờ chạm vào dữ liệu thẻ thô.”

8. “Tell me about a design decision you got wrong.” (Kể cho tôi về một quyết định thiết kế bạn đã sai.) Họ đang kiểm tra cái tôi, không phải lịch sử. Khuôn hình: ví dụ thật (từ capstone) — bạn đã tin gì — thực tế đã nói gì — bản post-mortem, ADR được cập nhật, mẫu hình bạn giờ luôn kiểm tra. Một kiến trúc sư không đưa ra được câu trả lời này là một kiến trúc sư chưa từng được review.

9. “How do you explain a complex technical decision to a non-technical executive?” (Bạn giải thích một quyết định kỹ thuật phức tạp cho một lãnh đạo không chuyên kỹ thuật như thế nào?) “Quyết định và con số kinh doanh trước, cơ chế chỉ khi được hỏi; sơ đồ context, không phải container; rủi ro theo ngôn ngữ doanh-thu-tuân-thủ-danh-tiếng; và tôi mang theo hai quyết định tôi cần từ họ, viết dưới dạng các lựa chọn kèm bảng giá — lãnh đạo quyết định giữa các phương án; họ không phê duyệt những điều bí ẩn.”

10. “An engineer strongly disagrees with your design. What do you do?” (Một kỹ sư phản đối mạnh mẽ thiết kế của bạn. Bạn làm gì?) “Trước tiên tôi steel-man họ thành tiếng — họ có thể đúng, và cuộc review làm thay đổi thiết kế của tôi là cuộc review đang hoạt động. Nếu thực sự sít sao, tiêu chí quyết định (độ khớp NFR, kỹ năng đội, chi phí thoát), không phải thâm niên — và tôi để sở thích của người thực thi thắng khi hòa, vì sự cam kết là món hời ở cái giá đó. Dù thế nào, quyết định và phương án bị loại đều vào ADR, để chúng tôi không bao giờ cãi lại lần hai.”

Giáo trình một trang

Khi nào	Module	Trọng tâm	Bảng chứng bên ngoài
Tuần 1–2	1	Ôn cloud; tam giác đánh đổi; NFR	Design Journal khởi động
Tuần 3–4	2	Compute/lưu trữ/cơ sở dữ liệu/mạng dưới dạng quyết định	—
Tuần 5–6	3	Đọc & vẽ sơ đồ; trôi chảy 3-tier	Cột mốc bảng trắng 15 phút
Tuần 7–8	4	Độ tin cậy: multi-AZ, auto-scaling, RTO/RPO	—
Tuần 9–10	5	Bảo mật: least privilege, zero trust, phân đoạn	—
Tuần 11–12	6	Hiệu năng & chi phí: caching, CDN, đặt trước, unit economics	Thiết kế được định giá trong calculator
Tuần 13–14	7	Vận hành xuất sắc & bền vững; bài luyện sáu trụ cột	Buổi review sáu-trụ-cột giả lập
Tuần 15–17	8	Mẫu hình: monolith/microservices, hướng sự kiện, serverless	—
Tuần 18–20	9	Mẫu hình: data lake/warehouse, multi-region, hybrid	Màn thử thách danh mục
Tuần 21–23	10	Migration: 7 chữ R, các đợt, landing zone	Cuộc migration trên giấy
Tuần 24–26	11	Ràng buộc: PDPA/GDPR, hệ thống cũ, kinh tế học lock-in	Màn thử thách ràng buộc
Tháng 7–8	12	ADR, bộ sơ đồ, điều hành Well-Architected review	Tập hồ sơ ThaiTicket; thi SAA-C03 (ôn 2–3 tháng)
Tháng 8–9	13	Thuyết trình, định giá đề xuất, phản	Trận đấu kép được ghi âm

Khi nào	Module	Trọng tâm	Bảng chứng bên ngoài
Tháng 9–12	14	bác, cổ vấn Capstone 1–3	Portfolio hoàn tất; SA Pro (4–8 tháng) hoặc AZ-305 đang tiến hành

Lời kết từ người thầy của bạn. Sau hai mươi sáu tuần, bạn có thể thiết kế; sau mười hai tháng, bạn có thể *hành nghề* — có một sự khác biệt, và đó là sự khác biệt mà khóa học này được xây để khép lại. Những thói quen giờ đây chính là sự nghiệp: vẽ trước khi tranh luận, viết ADR ngay ngày bạn quyết định, định giá những gì bạn đề xuất, gọi tên sự đánh đổi trước khi ai kịp hỏi, và nuôi lớn những kỹ sư quanh bạn cho đến khi chuẩn mực của bạn sống lâu hơn sự hiện diện của bạn trong phòng. Kiến trúc là nghề kỹ thuật hiểm hoi trở nên *tốt hơn* khi bạn già đi cùng nó, vì nguyên liệu thô của nó là phán đoán, và phán đoán sinh lãi kép. Hãy tiếp tục cuốn nhật ký. Bốn mươi bản thiết kế nữa, bạn sẽ không cần khóa học này — bạn sẽ đang sửa lại nó.

Nguồn tham khảo

Các bản mô tả công việc và định nghĩa vai trò dùng cho bảng ánh xạ yêu cầu (thu thập tháng 8 năm 2026): mẫu [KORE1 Cloud Architect job description template](#) · bản [4 Corner Resources Cloud Architect job description](#) · [Solution Architect responsibilities in the Azure Well-Architected Framework](#) của Microsoft · tin tuyển [Pre-Sales Solutions Architect – AWS](#) của Arpio · tin tuyển [Solution Architect – Enterprise \(qua Built In\)](#) của Intel · tin tuyển [Cloud Domain Architect](#) của Halliburton. Ước tính thời gian ôn chứng chỉ: [khảo sát thời gian học SAA-C03](#) của CBT Nuggets và [hướng dẫn ôn SAP-C02](#) của Whizlabs; định nghĩa chứng chỉ từ [Microsoft Learn \(AZ-305\)](#) và [AWS Well-Architected Framework](#). Một tập sách đồng hành của *The Cloud Leader Course* trong bộ B4LCILC.

CÁC BÀI HỌC MỚI — ẤN BẢN THÁNG 8 NĂM 2026

Vào tháng 8 năm 2026, các bài học mới đã được bổ sung vào ba khóa học tương tác tại [b4lcilc.org](#) — khóa *Cloud Leader (Lãnh đạo Cloud)*, *Cloud Engineer (Kỹ sư Cloud)* và *Cloud Architect (Kiến trúc sư Cloud)*. Phần phụ lục này cập nhật ấn bản đọc cho theo kịp, với những bản cô đọng trung thành của tài liệu mới đó, cùng một giọng văn với phần còn lại của cuốn sách này.

Đôi lời về thứ bạn đang cầm trên tay. Những trang này là ấn bản đọc: các ý tưởng, các phép ẩn dụ, và những câu văn đáng nói thành tiếng. Phiên bản tương tác đầy đủ trên mạng mang theo nhiều hơn đáng kể — mỗi bài học mới đều đi kèm các bài quiz kiểm tra mức độ hiểu trước khi cho bạn đi tiếp, các bài thực hành hands-on (bao gồm một terminal luyện tập hoạt động y như terminal thật nhưng không thể tổn tiền), video, bài luyện từ vựng, và — với phần tài liệu

capstone — các bài tập thiết kế có hướng dẫn kèm danh sách kiểm tra yêu cầu và đáp án mẫu được giải chi tiết. Hãy đọc những bản cô đọng này để nắm được hình hài của từng ý tưởng trong đầu; rồi khi có điều kiện, hãy ngồi làm phiên bản tưởng tác để kiến thức bám rễ.

Các phần bổ sung chia thành ba nhóm. Cho khóa **Cloud Leader**: một phần giới thiệu nhẹ nhàng về terminal, luật riêng tư của chính Thái Lan, cẩm nang dành cho người lãnh đạo về các đề xuất AI và hóa đơn AI, và một bài tập capstone gom toàn bộ khóa học vào một trang giấy. Cho khóa **Cloud Engineer**: điện toán serverless — mẫu hình biên những script vận hành nhỏ thành các function chạy mà gần như không tốn đồng nào. Cho khóa **Cloud Architect**: phép tính số học của các lời hứa về độ sẵn sàng, mẫu hình Retrieval-Augmented Generation mà khách hàng giờ đây hỏi đến hằng tuần, và một bài tập tính chi phí trọn vẹn — định giá một bản thiết kế trước khi bất kỳ ai xây nó.

Cho khóa Cloud Architect

Phép số học của những con số 9 — SLA, SLO và ngân sách lỗi

Độ sẵn sàng được báo giá bằng những con số 9, và việc đầu tiên một kiến trúc sư làm với một con số 9 là quy đổi nó ra phút — bởi phút là thứ doanh nghiệp cảm nhận được. Một tháng 30 ngày chứa 43.200 phút:

Độ sẵn sàng	Thời gian ngừng hoạt động cho phép mỗi tháng
99,9% (“ba số 9”)	khoảng 43 phút
99,99% (“bốn số 9”)	khoảng 4,3 phút
99,999% (“năm số 9”)	khoảng 26 giây

Năm số 9 — con số mà các lãnh đạo cấp cao thích nói thành tiếng — nằm ngoài tầm với của mọi quy trình khôi phục do con người vận hành; việc chuyển đổi dự phòng (failover) phải là tự động, nếu không lời hứa chỉ là hư cấu. Mỗi con số 9 thêm vào tốn khoảng gấp mười lần công sức kỹ thuật; quy đổi ra phút chính là cách bạn khiến một bên liên quan lựa chọn trong sự hiểu biết.

Ba từ viết tắt gánh cả cuộc trò chuyện. Một **Service-Level Indicator (SLI — chỉ số mức dịch vụ)** là một phép đo. Một **Service-Level Objective (SLO — mục tiêu mức dịch vụ)** là chỉ tiêu nội bộ đặt trên phép đo đó. Một **Service-Level Agreement (SLA — thỏa thuận mức dịch vụ)** là bản hợp đồng đối ngoại, có tiền gắn vào mỗi lần lỡ hẹn. Kỷ luật ở đây: SLO luôn nghiêm ngặt hơn SLA — bạn nhắm vào phía trong lẫn ranh mà bạn được trả tiền để giữ — và SLI đo cái người dùng trải nghiệm: một server đang bật nhưng trả về lỗi thì đã sập theo mọi nghĩa đáng kể.

Ngân sách lỗi (error budget) biến SLO thành thứ vận hành được: khoảng cách giữa sự hoàn hảo và mục tiêu, được đối xử như một nguồn lực để chi tiêu. Một SLO 99,9% cấp cho bạn 43 phút hỏng hóc mỗi tháng; các đợt triển khai và thử nghiệm chi tiêu từ đó; khi nó cạn, cả đội ngừng ra tính năng mới và tập trung vào độ tin cậy — chấm dứt cuộc tranh cãi lâu đời nhất của ngành phần mềm, ra hàng hay ổn định, bằng cách thay ý kiến bằng phép số học.

Giờ đến cái bẫy: độ sẵn sàng nhân lên dọc theo một chuỗi. Ba dịch vụ nối tiếp nhau, mỗi cái trung thực ở mức 99,9%, hợp lại thành $0,999 \times 0,999 \times 0,999 \approx 99,7\%$ — chừng 130 phút mỗi tháng, gấp ba lần con số mà bất kỳ dịch vụ đơn lẻ nào thừa nhận. Sự dư thừa (redundancy) hoạt động theo chiều ngược lại: hai instance độc lập cùng ở mức 99,9% chạy song song chỉ cùng hỏng $0,1\% \times 0,1\%$ thời gian — đó là lý do multi-zone đáng đồng tiền đến vậy. Câu nói ở đẳng cấp phỏng vấn: *bạn không thể hứa nhiều độ sẵn sàng hơn tích số của các phụ thuộc đồng bộ của mình, vậy nên hoặc rút ngắn chuỗi, hoặc hạ thấp lời hứa.*

Kiến trúc trong thời đại AI — RAG và họ hàng của nó

Đến năm 2026, danh mục mẫu hình có thêm một mục mới mà khách hàng hỏi đến hằng tuần: nó là gì, khi nào nó xứng đáng với chi phí, và khi nào nó sẽ khiến bạn bẽ mặt. Một API Mô hình Ngôn ngữ Lớn (Large Language Model — LLM) — Amazon Bedrock, Azure OpenAI Service — về mặt kiến trúc là *một phụ thuộc đắt đỏ, chậm chạp và không lưu trạng thái*. Nó được đo đếm theo **token** — một mảnh từ chừng bốn ký tự. Nó trả lời trong vài giây, không phải vài mili-giây, không nhớ gì giữa các lần gọi, và thỉnh thoảng sai một cách đầy tự tin — một thành phần với những kiểu hỏng hóc khác thường, xứng đáng nhận cùng những timeout, retry, circuit breaker và chuông báo ngân sách như bất kỳ phụ thuộc nào, bởi một vòng lặp gọi mất kiểm soát đột tiền thật theo từng giờ.

Mô hình không biết tài liệu của bạn; câu trả lời tiêu chuẩn là **Retrieval-Augmented Generation (RAG — sinh nội dung có truy xuất tăng cường)**: thay vì dạy dữ liệu của bạn cho mô hình, bạn *tra cứu* các sự kiện liên quan vào lúc đặt câu hỏi. Tài liệu được cắt thành các đoạn (chunk), mỗi đoạn được chuyển thành một **embedding** — một dãy số định vị ý nghĩa của nó trong không gian — lưu trong một **vector database** (cơ sở dữ liệu vector: pgvector trên AWS, AI Search trên Azure), nơi trả về các đoạn gần nhất. Những đoạn đó được dán vào prompt: mô hình không tra cứu dữ liệu của bạn; nó đang tóm tắt các trích đoạn mà bạn đã đi lấy về cho nó.

Trên bảng trắng, RAG là hai đường ống: một **đường nạp (ingestion path)** chạy trước từ sớm (tài liệu được cắt đoạn, chuyển thành embedding, lưu trữ) và một **đường truy vấn (query path)** chạy theo từng yêu cầu — hai mẫu hình sẵn có trong danh mục, và đó là lý do các kiến trúc sư nắm được nó chỉ trong một buổi chiều.

Các đánh đổi vẫn là bộ ba tiêu chuẩn, được mài sắc thêm: chi phí (mỗi truy vấn tốn tiền thật — hãy cache các câu lặp lại, dẫn các câu hỏi dễ sang một mô hình rẻ hơn), độ trễ (vài giây, không phải vài mili-giây — hãy stream câu trả lời), và quyền riêng tư (tài liệu của bạn rời khỏi ranh giới của bạn; nếu vector database phớt lờ kiểm soát truy cập ở cấp tài liệu, con chatbot sẽ vui vẻ tóm tắt bảng lương cho bất kỳ ai hỏi đến). Hãy biết khi nào *không nên* xây nó: nhiều yêu cầu “chúng tôi cần một chatbot” chỉ là một ô tìm kiếm khoác áo thời trang, và những câu trả lời chính xác, kiểm toán được — các phép cộng, số dư, sổ cái — là việc sai chỗ cho một hệ thống chỉ đúng 97% thời gian. Câu nói ở đẳng cấp phỏng vấn: *RAG là một bài toán tìm kiếm được ghim vào một API tóm tắt.*

Tính chi phí một bản thiết kế trước khi bất kỳ ai xây nó

Module Trụ cột Hiệu năng và Chi phí của khóa học giờ đây khép lại bằng một bài tập thiết kế trọn vẹn: ước lượng dưới áp lực. Đề bài: ban lãnh đạo muốn một bản ước tính chi phí hằng tháng cho một sản phẩm quản lý tài liệu trước khi phê duyệt việc xây dựng.

Workload — 200.000 người dùng đã đăng ký, 40.000 hoạt động mỗi ngày, mỗi người thực hiện khoảng 25 yêu cầu API mỗi ngày và tải lên trung bình 2 megabyte tài liệu mỗi tháng, lưu giữ trong bảy năm. Bản thiết kế: một tầng API có cân bằng tải trải trên hai zone, một cơ sở dữ liệu PostgreSQL managed đa zone, kho lưu trữ đối tượng (object storage), một đợt xuất 50 gigabyte hằng đêm cho một đối tác, và các môi trường development và staging riêng biệt. Nhiệm vụ: một bản ước tính hằng tháng dài một trang bằng số tròn, với các giả định được viết ra trước khi bất kỳ con số giá nào xuất hiện, phân số học công suất được trình bày rõ (người dùng — số yêu cầu mỗi giây — số lượng instance), và các kịch bản hôm nay / gấp 2 lần / gấp 10 lần.

Danh sách kiểm tra của bài tập dạy nên kỷ luật: định giá compute theo tải đỉnh, không phải tải trung bình; đếm cả bản standby của cơ sở dữ liệu; mô hình hóa lưu trữ như sự tăng trưởng qua bảy năm với các tầng lifecycle; tính egress thay vì đoán nó — dòng chi mà hầu hết các bản ước tính bỏ quên và cũng là dòng thường xuyên chiếm phần lớn nhất; nhớ đến các NAT (Network Address Translation — biên dịch địa chỉ mạng) gateway, các môi trường ngoài production, sao lưu, giám sát, và gói hỗ trợ; chỉ áp dụng chiết khấu cam kết cho phần nền ổn định; và kết thúc bằng một khoảng giá kèm các yếu tố dân dốt được nêu tên, cùng một phép kiểm tra hợp lý theo kinh tế đơn vị — chi phí trên mỗi người dùng hoạt động mỗi tháng. Một con số đơn lẻ chính xác là sai ngay từ ngày nó được nói ra.