

El Curso de Arquitecto Cloud (The Cloud Architect Course)

De cero a diseñar sistemas cloud — un curso B4LCILC

13 de agosto de 2026

El Curso de Arquitecto Cloud (The Cloud Architect Course)

De cero conocimiento a diseñar sistemas cloud a los que una empresa apostaría su negocio.

Cómo funciona este curso

Usted no se está formando para *operar* sistemas cloud. Se está formando para **diseñarlos**: para tomar un problema de negocio desordenado (“necesitamos venderle a un millón de clientes y no perder nunca un pedido”), convertirlo en un plan técnico dibujado, costado y defendible, y luego persuadir tanto a los ingenieros que lo construirán como a los ejecutivos que lo pagarán. Ese es el trabajo de un **arquitecto cloud** (también anunciado como *solutions architect*, *cloud solutions architect* o *cloud domain architect*), y es un oficio aprendible con un plan de estudios claro — este.

El curso tiene cinco etapas. **Etapas 1 (Semanas 1–6): Los componentes básicos, como decisiones.** Cada componente cloud, re-enseñado no como un hecho sino como una *elección con trade-offs (compromisos)* — porque la unidad de trabajo de un arquitecto es la decisión. **Etapas 2 (Semanas 7–14): Los dominios de diseño.** Los seis pilares del Well-Architected Framework — confiabilidad, seguridad, rendimiento, costos, operaciones, sostenibilidad — cada uno enseñado a fondo con sus patrones estándar y un diseño trabajado. **Etapas 3 (Semanas 15–20): El catálogo de patrones.** La docena de arquitecturas que resuelven el 95% de los problemas reales, y — más importante — cuándo *no* usar cada una. **Etapas 4 (Semanas 21–26): Migración y el mundo real.** Mover empresas existentes a la nube, y las restricciones (ley, sistemas heredados, lock-in) que hacen la arquitectura real más difícil que la de pizarra. **Etapas 5 (Meses 7–12): El oficio.** Los documentos, diagramas, revisiones, presentaciones y certificaciones que lo hacen *empleable* como arquitecto, coronado por tres proyectos de diseño de calidad de portafolio.

Para quién es este curso. Asume cero conocimiento previo: todo aquello de lo que depende se re-enseña brevemente antes de usarse. Si usted ya completó **El Curso de Ingeniero Cloud** (o trabaja hoy con las manos en el cloud), la Etapa 1 le resultará familiar — *hojéela, pero haga sus ejercicios de todos modos*, porque re-enmarca cosas que usted conoce como hechos en cosas que debe sopesar como decisiones, y ese re-enmarcado es todo el cambio de ingeniero a arquitecto.

Reglas para todo el curso: estudie **una hora al día, seis días a la semana** — la constancia vence a la intensidad. Cada módulo termina con una **tabla de Vocabulario** (las palabras que debe dominar), **Ejercicios** (trabajo de diseño, no lectura — la arquitectura se aprende dibujando y decidiendo) y un **Hito** (la prueba de que está listo para avanzar; no se salte un hito

sin cumplirlo). Y desde la Semana 1, lleve un **Diario de Diseño**: cada arquitectura que dibuje, cada trade-off que señale, cada predicción que haga sobre costos o fallas. Dentro de diez meses se convierte en su portafolio de entrevistas; nada impresiona más a un panel de contratación que un cuaderno fechado con cuarenta diseños y notas honestas sobre lo que salió mal.

Qué hace realmente un arquitecto cloud

Piense en una arquitecta de edificios. Ella no coloca ladrillos — pero debe saber exactamente qué pueden y no pueden hacer los ladrillos, cuánto cuestan y cómo fallan los edificios. Escucha a un cliente (“una casa familiar, soleada, dentro de este presupuesto, en este terreno complicado”) y convierte deseos y restricciones en *planos y especificaciones* lo bastante precisos como para que los constructores construyan a partir de ellos y el cliente los apruebe. Luego se queda durante la construcción, respondiendo preguntas y ajustando el plan cuando el suelo resulta más blando de lo que decía el estudio.

Cambie ladrillos por servidores y ese es el trabajo. Un arquitecto cloud pasa su semana haciendo alguna mezcla de: **escuchar** a los interesados del negocio y extraer requisitos reales de deseos vagos; **diseñar** — elegir componentes, dibujar diagramas, dejar por escrito las decisiones y el porqué; **revisar** los diseños de otros contra los seis pilares; **estimar** cuánto costará un diseño al mes y defender ese número ante finanzas; **planificar migraciones** de sistemas viejos hacia la nube; **presentar** — el mismo diseño explicado de una manera a los ingenieros y de una manera completamente distinta a los ejecutivos; y **mentorar** a ingenieros para que los estándares sobrevivan al contacto con los plazos. En muchos equipos también hay **pre-ventas**: sentarse junto a un vendedor frente a un cliente potencial, esbozando la solución que gana el trato.

Lo que un arquitecto *no* es: el mejor programador de la sala (normalmente no lo es), la persona que más teclea (es la que menos teclea), ni un genio solitario que baja planos desde el olimpo (la forma más rápida de ser ignorado). El producto real del arquitecto son **buenas decisiones, escritas, que otros construyen de buena gana**.

La descripción de puesto para la que se está formando

En agosto de 2026 recopilamos publicaciones y plantillas reales de puestos de Cloud/Solutions Architect — la plantilla de arquitecto cloud de un reclutador (KORE1), la descripción de puesto de una firma de staffing (4 Corner Resources), la definición del propio Microsoft del rol de arquitecto en el Azure Well-Architected Framework, una publicación de Solutions Architect de pre-ventas (Arpio, recuperación ante desastres en AWS), una publicación de Solution Architect empresarial (Intel, vía Built In) y una publicación empresarial de Cloud Domain Architect (Halliburton, Azure-first). Quite el sabor de cada empresa y los mismos doce requisitos aparecen una y otra vez. Este curso está construido según esa lista — aquí está exactamente dónde se enseña cada uno:

Lo que piden las descripciones de puesto (sus palabras, parafraseadas)	Dónde lo enseña este curso
“Diseñar arquitecturas cloud escalables y seguras a la medida de los requisitos técnicos y de negocio” — diseño de soluciones de	Etapas 1–3, capstones en la Etapa 5

Lo que piden las descripciones de puesto (sus palabras, parafraseadas)	Dónde lo enseña este curso
extremo a extremo	
“Experiencia profunda de plataforma” en AWS y/o Azure — cómputo, almacenamiento, redes, IAM, estructura de cuentas	Etapa 1 + ruta de certificaciones en la Etapa 5
“Dirigir revisiones Well-Architected” contra los seis pilares	Etapa 2 (los pilares), Etapa 5 Módulo 12 (dirigir revisiones)
“Liderar iniciativas de migración/modernización — qué workloads se mueven tal cual, se rearquitectan o se retiran”	Etapa 4 Módulo 10 (las 7 R, planificación por olas)
“Diseñar landing zones, estructura de cuentas, guardrails y patrones de referencia dentro de los cuales despliegan los equipos”	Etapa 4 Módulo 10
“Integrar la seguridad y el cumplimiento en el diseño en lugar de añadirlos al final” — zero trust, identidad	Etapa 2 Módulo 5, Etapa 4 Módulo 11
Alta disponibilidad y recuperación ante desastres — “brechas de RTO/RPO, costos de caída, exposición a ransomware”	Etapa 2 Módulo 4, Etapa 3 Módulo 9 (multi-región)
“Ser dueño del modelo de costos cloud — etiquetado, showback, capacidad reservada, rightsizing”; estimar costos de soluciones	Etapa 2 Módulo 6, Etapa 5 Módulo 13 (costeo de propuestas)
Diseño de arquitectura de red — VPCs, hub-and-spoke, conectividad híbrida	Etapa 1 Módulo 2, Etapa 3 Módulo 9, Etapa 4
“Crear documentación arquitectónica, diagramas y estándares”; “mantener Architecture Decision Records”	Etapa 1 Módulo 3 (diagramas), Etapa 5 Módulo 12 (ADRs, conjuntos de diagramas)
Comunicación con interesados — “presentar conceptos técnicos a audiencias de nivel C y técnicas”; “defender decisiones arquitectónicas ante seguridad, finanzas e ingeniería”	Etapa 5 Módulo 13
“Mentorar a los ingenieros cloud y de plataforma que construyen según sus estándares”; soporte de pre-ventas — demos, POCs, RFPs	Etapa 5 Módulo 13

Certificaciones, verificadas contra el mismo mercado: la escalera estándar es **AWS Certified Solutions Architect – Associate (SAA-C03)** — la credencial individual más solicitada, típicamente **2-3 meses** de preparación — seguida por **AWS Certified Solutions Architect –**

Professional (SAP-C02), típicamente **4–8 meses** más; los entornos con mucho Azure piden **AZ-305 (Azure Solutions Architect Expert)**; las grandes empresas a veces añaden **TOGAF** para el método de arquitectura empresarial. El plan completo está en la Etapa 5, Módulo 14.

ETAPA 1 — LOS COMPONENTES BÁSICOS, COMO DECISIONES (Semanas 1–6)

Graduados del Curso de Ingeniero Cloud: hojeen las explicaciones, pero hagan todos los ejercicios. Los hechos son los mismos; las preguntas son nuevas.

Módulo 1 (Semanas 1–2): Qué es la nube, y cómo piensan los arquitectos

La nube en un párrafo (el repaso desde cero). La nube es *las computadoras de otro, alquiladas por hora, gestionadas por software*. AWS, Microsoft Azure y Google Cloud operan almacenes de servidores (**data centers**, centros de datos) agrupados en **Regiones** geográficas (p. ej., “Asia Pacific (Bangkok)”); cada región contiene varias **Availability Zones (AZs, zonas de disponibilidad)** aisladas — edificios separados con energía independiente, lo bastante cercanos para conexiones rápidas, lo bastante lejanos para que una inundación o un incendio no pueda tumbar dos. Usted alquila porciones de todo esto por segundo: máquinas en bruto (**IaaS** — usted gestiona todo lo que corre en ellas), plataformas gestionadas (**PaaS** — usted solo trae su aplicación) o software terminado (**SaaS** — usted solo lo usa). Ese es todo el sustrato. Todo lo que un arquitecto diseña se acomoda encima de él.

Ahora el movimiento del arquitecto: convertir cada hecho en una pregunta. Un ingeniero aprende “una AZ es un centro de datos aislado”. Un arquitecto pregunta de inmediato: “¿Cuántas AZ merece este workload?” — porque dos AZ cuestan más que una, tres más que dos, y un sitio de folletos de marketing no merece lo que merece un sistema de pagos. Esta es la primera y más importante idea del curso:

La arquitectura es la disciplina de hacer explícitos los trade-offs. Casi no hay componentes incorrectos — solo componentes incorrectos *para este workload, este presupuesto, este equipo, este plazo*.

El triángulo de los trade-offs. Todo diseño negocia entre tres esquinas: **rápido** (rendimiento — rápido para los usuarios, rápido de construir), **barato** (factura mensual baja, poco esfuerzo de ingeniería) y **resiliente** (sobrevive fallas, escala, se mantiene seguro). Usted puede empujar hacia cualesquiera dos esquinas; la tercera lo paga. El prototipo de una startup debe ser rápido y barato — la resiliencia puede esperar. El libro mayor central de un banco debe ser resiliente y rápido — no será barato. Cuando un interesado dice “queremos las tres”, su trabajo es sonreír y preguntar cuál quiere *más*, porque el diseño no puede empezar hasta que respondan. Dibuje este triángulo en la parte superior de cada diseño que haga en este curso, y marque dónde se sitúa el workload. Le ahorrará mil discusiones.

Requisitos: la materia prima del diseño. Los arquitectos separan los **requisitos funcionales** (qué hace el sistema — “los clientes pueden pedir el almuerzo”) de los **requisitos**

no funcionales (NFRs) (qué tan bien debe hacerlo — “en menos de 2 segundos, para 10,000 estudiantes simultáneos, el 99.9% del tiempo, dentro de la PDPA”). Los principiantes se obsesionan con la primera lista; los arquitectos se ganan su salario con la segunda, porque los NFRs son lo que realmente determina la arquitectura. Dos preguntas mágicas extraen NFRs de interesados que no saben que los tienen: “¿Qué le pasa al negocio si esto está caído una hora?” y “¿Cómo se ve el éxito a diez veces el tamaño de hoy?”

Vocabulario:

Término	Definición
Región / Availability Zone (AZ)	Región: un grupo geográfico de centros de datos donde usted elige correr. AZ: un centro de datos aislado (o pequeño grupo) dentro de ella — la unidad de “un edificio puede fallar”.
IaaS / PaaS / SaaS	Alquilar máquinas en bruto / una plataforma gestionada / software terminado. El deslizador de más control a menos mantenimiento.
Workload (carga de trabajo)	Cualquier aplicación o sistema, tratado como unidad de diseño — “el workload de pagos”.
Requisito funcional	Lo que el sistema debe hacer (“los clientes pueden pedir”).
Requisito no funcional (NFR)	Qué tan bien debe hacerlo — velocidad, escala, uptime, seguridad, cumplimiento, costo. Los NFRs dirigen la arquitectura.
Trade-off	Lo que usted cede para obtener lo que eligió. Toda decisión de diseño tiene uno; el trabajo del arquitecto es nombrarlo en voz alta.
Restricción	Un límite no negociable: presupuesto, plazo, ley, sistemas existentes, habilidades del equipo. Las restricciones no son obstáculos para el diseño — <i>son</i> el encargo de diseño.
Interesado (stakeholder)	Cualquiera con algo en juego en el sistema: usuarios, ingenieros, finanzas, seguridad, ejecutivos, reguladores. Distintos interesados, distintos idiomas — usted los habla todos.
Greenfield / brownfield	Un sistema completamente nuevo sin historia (greenfield) vs. uno enredado con sistemas existentes (brownfield — la mayor parte del trabajo real).
Servicio gestionado	Un componente que el proveedor opera por usted (respaldos, parcheo, failover incluidos). El predeterminado del arquitecto, salvo que haya una razón por escrito para lo contrario.

Videos de este módulo (enlaces verificados):

Video	Canal	Duración	Enlace
Top 50+ AWS Services Explained in 10 Minutes	Fireship	~10 min	https://www.youtube.com/watch?v=JlbIYCM48to

Vea el recorrido de Fireship dos veces: una ahora para el mapa, otra al final de la Etapa 1 — se sorprenderá de cuántos servicios ya puede *colocar* en un diseño.

Ejercicios: (1) Elija tres apps que use a diario (una app bancaria, una de entrega de comida, una de video) y, para cada una, escriba sus tres NFRs principales y márkela en el triángulo de trade-offs. (2) Entreviste a un amigo sobre una idea de negocio durante diez minutos y extraiga cinco requisitos funcionales y cinco no funcionales — note cómo los NFRs solo salen cuando hace las dos preguntas mágicas. (3) Comience su Diario de Diseño con la entrada #1: el triángulo, y un párrafo sobre por qué “¿qué esquina sacrificamos?” es una pregunta de negocio, no técnica.

Hito: dada cualquier descripción de sistema en una frase, usted puede producir sus NFRs probables y su posición en el triángulo en cinco minutos, en voz alta, sin notas.

Módulo 2 (Semanas 3-4): Cómputo, almacenamiento, base de datos, red — las cuatro decisiones

Todo diseño que usted dibuje en su vida son estas cuatro elecciones, hechas deliberadamente. Aquí está cada una enseñada como decisión.

Decisión 1 — Cómputo: ¿VM, container o serverless? Una **máquina virtual (VM)** es una porción alquilada de un servidor que actúa como una computadora completa — máximo control, máximo mantenimiento (usted la parchea, usted la escala, usted paga mientras está ociosa). Un **container** (contenedor) empaqueta una aplicación con todo lo que necesita para que corra de forma idéntica en cualquier lugar; un orquestador (**Kubernetes**) ejecuta y repara flotas de ellos — gran densidad y portabilidad, pero usted ha adoptado una plataforma compleja que necesita gente capacitada. **Serverless** (sin servidor; AWS Lambda, Azure Functions) ejecuta su código solo cuando se dispara y factura por invocación — mantenimiento casi cero y perfecto para tráfico con picos, pero con límites (topes de tiempo de ejecución, cold starts, y un matrimonio más profundo con un solo proveedor). La tabla de decisión que debe poder reproducir de memoria:

Elija	Cuándo	Cuidado con
VM	Software heredado, licenciamiento especial, se necesita control total del OS, carga estable y predecible	Usted es dueño del parcheo, el escalado y las 3 a.m.; el tiempo ocioso factura completo
Containers + Kubernetes	Muchos servicios, el equipo ya tiene las habilidades, la	Complejidad de plataforma — K8s es un trabajo de

Elija	Cuándo	Cuidado con
Serverless	portabilidad importa	tiempo completo; excesivo para equipos pequeños
	Tráfico con picos o impredecible, pegamento orientado a eventos, equipos pequeños, rapidez de salida al mercado	Límites de runtime, cold starts, predicción de costos más difícil a gran escala estable, lock-in

La perspectiva senior: **esta es una elección por workload, no una religión de empresa**. Los parques reales corren las tres a la vez, correctamente.

Decisión 2 — Almacenamiento: ¿objetos, bloques o archivos — y qué tan caliente? El **almacenamiento de objetos** (Amazon S3) es un balde sin fondo para archivos — barato, absurdamente durable (“once nueves”), la respuesta por defecto a “¿a dónde van los archivos?” El **almacenamiento de bloques** (EBS) es el disco virtual atornillado a una VM. El **almacenamiento de archivos** (EFS) es una unidad compartida que muchas máquinas montan a la vez. La dimensión extra del arquitecto es la **temperatura**: datos calientes (accedidos constantemente, con precio de velocidad) versus niveles fríos/de archivo (Glacier — centavos, pero de minutos a horas para recuperar). Diseñar reglas de ciclo de vida que deriven los datos viejos hacia niveles fríos es la victoria de costos más barata del cloud; olvidarlo es lo más común.

Decisión 3 — Base de datos: ¿SQL o NoSQL (y cuál variante gestionada)? Las bases de datos **relacionales/SQL** (PostgreSQL, MySQL; gestionadas como RDS/Aurora) mantienen los datos en tablas estrictas con consistencia garantizada — el predeterminado para todo aquello donde la corrección es sagrada: dinero, pedidos, inventario, usuarios. Las bases de datos **NoSQL** (DynamoDB, MongoDB) intercambian estructura estricta por flexibilidad y escala horizontal casi ilimitada — el predeterminado para sesiones, catálogos, feeds, telemetría. La heurística de decisión: *empiece con SQL a menos que pueda nombrar la razón específica por la que no funcionará* (escala extrema, esquema flexible, lecturas globales de un dígito de milisegundos). Y en la nube, “base de datos” debería significar casi siempre “base de datos **gestionada**” — el proveedor se encarga de respaldos, parches y failover; un equipo que opera su propia base de datos en VMs debería tener una razón por escrito. Añada los especialistas a su vocabulario: **caché** (Redis — datos calientes en memoria, lecturas de microsegundos), **warehouse** (almacén de datos; analítica a escala — Etapa 3), **cola (queue)** (no es una base de datos, pero a menudo es la pieza faltante — Etapa 3).

Decisión 4 — Red: la forma del mundo privado. Una **VPC (Virtual Private Cloud, nube privada virtual)** es su sección cercada de la red del proveedor. Dentro de ella, las **subredes públicas** contienen lo que internet puede alcanzar (balanceadores de carga), y las **subredes privadas** contienen todo lo demás — servidores de aplicación y, siempre, las bases de datos. “La base de datos está en una subred privada” es la frase más repetida en las revisiones de arquitectura; el razonamiento — nada puede atacar lo que no tiene camino desde internet — es la mitad de la seguridad de redes. Alrededor de la VPC: un **balanceador de carga (load balancer)** reparte el tráfico entre servidores y esquiva a los enfermos; el **DNS** (Route 53) convierte nombres en direcciones; una **CDN** (CloudFront) almacena en caché el contenido en

cientos de ciudades para que sea rápido en todas partes; un **API gateway** es la recepción gestionada de sus APIs (autenticación, límites de tasa, registro). Conectando con el mundo viejo: **VPN** (túnel cifrado sobre internet) o **Direct Connect** (una línea física privada) — los cordones umbilicales de todo diseño híbrido de la Etapa 4.

Vocabulario:

Término	Definición
Instancia / tipo de instancia	Una VM alquilada / su tamaño (CPU + RAM), que fija su precio por hora.
Container / Docker / Kubernetes (K8s)	Un paquete portátil para una app / la herramienta que los construye y ejecuta / el orquestador que ejecuta y repara flotas de ellos.
Serverless / Lambda	Código que corre solo cuando se dispara, facturado por ejecución, sin servidores que gestionar. Lambda es la versión de AWS.
Cold start	El retraso extra cuando una función serverless corre tras estar ociosa — el trade-off clásico del serverless.
S3 / bucket / durabilidad	El almacenamiento de objetos de AWS / un contenedor de archivos con nombre / la probabilidad de que los datos sobrevivan — el 99.999999999% de S3 significa que la pérdida esencialmente nunca ocurre.
Tier de almacenamiento / política de ciclo de vida	Clase precio-velocidad (caliente → frío → archivo) / la regla automática que mueve los datos que envejecen a niveles más baratos.
RDS / Aurora / DynamoDB	Los servicios SQL gestionados de AWS / su SQL nativo de la nube de alto rendimiento / su NoSQL gestionado insignia.
Consistencia	La garantía de que todos los que leen los datos ven la misma verdad al mismo tiempo — el superpoder de SQL, y lo que NoSQL relaja para ganar escala.
Caché / Redis	Una copia a velocidad de memoria de los datos calientes colocada delante de una base de datos / la herramienta estándar para ello.
VPC / subred (pública, privada)	Su sección de red privada / sus subdivisiones — la pública mira a internet, la privada no. Las bases de datos viven en privadas. Siempre.
Balanceador de carga / health check	El director de tráfico entre servidores / la prueba de latido que usa para dejar de enviar

Término	Definición
CDN / edge	tráfico a los muertos. Cachés a nivel de ciudad de su contenido en todo el mundo / “the edge” (el borde) = cerca de los usuarios.
API / API gateway	La puerta controlada que una pieza de software ofrece a otra / la recepción gestionada de esas puertas.
VPN / Direct Connect	Túnel cifrado sobre internet / una línea física privada hacia la nube — las dos maneras en que on-prem se encuentra con la nube.

Videos de este módulo (enlaces verificados):

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

Ejercicios: (1) Para cada uno de estos cinco workloads, elija cómputo, base de datos y almacenamiento, y escriba una frase de justificación para cada uno: un sitio escolar de pedidos de almuerzo; el libro mayor de transacciones de un banco; una app para compartir fotos; un generador de informes nocturno que corre 20 minutos; una app de chat para 5 millones de usuarios. (2) Tome una de sus elecciones y defienda la elección *opuesta* tan persuasivamente como pueda — los arquitectos que no pueden defender con fuerza la alternativa no han entendido el trade-off. (3) Diario: su tabla de decisión de cómputo, de memoria.

Hito: dado cualquier workload en una frase, usted puede nombrar sus cuatro decisiones con justificaciones en menos de tres minutos — y para al menos una decisión, nombrar qué le haría cambiar de opinión.

Módulo 3 (Semanas 5-6): Leer y dibujar diagramas de arquitectura

Un arquitecto que no puede dibujar es un consultor que solo puede hablar. Los diagramas son su idioma de trabajo: este módulo lo hace fluido en leerlos y competente en dibujarlos.

El diagrama canónico — aprenda este primero. La **arquitectura de tres capas (three-tier)** es la “estructura de oración” de los diagramas cloud; la mayoría de los diseños son variaciones de ella:

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

→ [Cache]

→ [Database: primary +

standby, private subnets]

Capa 1 (**presentación**): lo que los usuarios tocan — contenido estático desde la CDN, solicitudes a través del balanceador de carga. Capa 2 (**aplicación**): la flota de servidores intercambiables que ejecutan su lógica, en subredes privadas, escalada automáticamente. Capa 3 (**datos**): la base de datos, la más profunda y protegida, con una copia en espera en una segunda AZ. El tráfico fluye en un solo sentido: los usuarios nunca tocan la capa de aplicación directamente, y solo la capa de aplicación habla con la capa de datos. Practique dibujar esto hasta que su mano lo haga sin su cerebro — es el arranque de la entrevista de pizarra en cualquier lugar del planeta.

Convenciones de notación que lo hacen ver profesional (porque lo hacen pensar profesionalmente): las cajas son componentes — etiquete cada una con lo que es y con qué servicio (“App servers — EC2, auto-scaling group”). Las flechas muestran la dirección en que fluye una *solicitud*, etiquetadas con el protocolo cuando importa. Los recuadros punteados muestran fronteras — la VPC, cada subred, cada AZ (dibuje las fronteras de AZ y el multi-AZ se vuelve visible en lugar de declarado). Un usuario/actor queda fuera del sistema. Los números en las flechas (1, 2, 3...) le permiten narrar el viaje de una solicitud. Y todo diagrama lleva título, fecha y leyenda. La regla más profunda: **un diagrama, una audiencia, una pregunta**. Un diagrama que muestra todo no muestra nada; aprenderá el conjunto estándar de niveles de zoom (contexto → contenedores → despliegue) en la Etapa 5.

Leer los diagramas de otros — los rayos X del arquitecto. Cuando le entreguen un diagrama, ejecute este escaneo en voz alta: ¿Dónde toca internet a este sistema (cada punto de contacto es superficie de ataque)? ¿Dónde están los datos, y están en una subred privada? ¿Qué está duplicado (resiliente) y qué es un **punto único de falla** — una caja sin gemela? ¿Dónde dolería a 10× el tráfico? ¿Cuánto cuesta cada caja al mes? Cinco preguntas, treinta segundos, y usted ha leído el diagrama como un médico lee una radiografía. Navegue por el AWS Architecture Center (<https://aws.amazon.com/architecture/>) y ejecute el escaneo sobre tres arquitecturas de referencia publicadas.

Vocabulario:

Término	Definición
Arquitectura de tres capas	La separación clásica: presentación (la entrada de los usuarios) — aplicación (lógica) — datos (base de datos). La forma por defecto de los sistemas web.
Capa (tier / layer)	Una rebanada horizontal del sistema con una responsabilidad, que habla solo con sus

Término	Definición
	vecinas.
Punto único de falla (SPOF)	Cualquier componente cuya muerte solitaria tumba el sistema. Lo primero que hay que cazar en cualquier diagrama.
Multi-AZ	Correr duplicados en al menos dos Availability Zones para que un centro de datos pueda fallar de forma invisible.
Auto-scaling (group)	Máquinas añadidas y retiradas automáticamente según la demanda — capacidad que respira.
Stateless / stateful	Un servidor que no guarda datos únicos (cualquier gemelo puede reemplazarlo, así que escala libremente) vs. uno que guarda datos que no deben perderse. Meta de diseño: capa de aplicación stateless, el estado empujado hacia la base de datos y la caché.
Arquitectura de referencia	Un diseño de ejemplo publicado y bendecido por el proveedor para un problema común — los arquitectos ensamblan a partir de estos antes de inventar.
Diagrama de contexto	El nivel de zoom más alto: su sistema como una sola caja, más los usuarios y sistemas externos que toca.
Superficie de ataque	Cada punto donde el mundo exterior puede tocar el sistema. Más pequeña es más segura.
Tráfico norte-sur / este-oeste	Tráfico que entra/sale del sistema vs. tráfico entre componentes dentro de él.

Ejercicios: (1) Dibuje el diagrama de tres capas de memoria, cinco días seguidos, hasta que tome menos de cuatro minutos con todas las fronteras (VPC, subredes, dos AZ) dibujadas. (2) Tome los cinco workloads del Módulo 2 y dibuje cada uno — veinte minutos por diagrama, con las reglas de notación aplicadas. (3) Encuentre cualquier diagrama de arquitectura real en línea (el AWS Architecture Center tiene cientos) y escriba en su diario su escaneo de rayos X de cinco preguntas. (4) El drill de narración: con el diagrama al frente, narre el clic de un usuario desde el navegador hasta la base de datos y de vuelta, en voz alta, numerando las flechas sobre la marcha.

Hito — fin de la Etapa 1: en una pizarra (o papel, fotografiado para su diario), usted puede dibujar un diseño de tres capas correcto y bien anotado para un workload nuevo de una frase en menos de quince minutos, narrar una solicitud a través de él, y responder “¿qué falla si esta caja muere?” para cada caja. Este dibujo-más-interrogatorio es precisamente la primera mitad de una entrevista real de arquitecto — de aquí en adelante, todo es profundidad.

ETAPA 2 — LOS DOMINIOS DE DISEÑO (Semanas 7–14)

El mapa de esta etapa es el **AWS Well-Architected Framework** — la lista compartida de la industria de lo que significa “diseñado correctamente”, organizada en seis pilares: **excelencia operativa, seguridad, confiabilidad, eficiencia de rendimiento, optimización de costos, sostenibilidad**. (Azure tiene un marco casi idéntico; aprenda uno a fondo y habrá aprendido ambos.) Las descripciones de puesto piden por nombre arquitectos que puedan “dirigir revisiones Well-Architected”, así que tomamos los pilares uno a la vez, y para cada uno usted aprende tres cosas: las **preguntas clave** del pilar, sus **patrones estándar** (las respuestas aburridas y probadas — los arquitectos ensamblan antes de inventar) y un **ejemplo trabajado** sobre un escenario recurrente.

El escenario recurrente de toda la Etapa 2: *ThaiTicket*, una plataforma ficticia de venta de boletos para eventos en Bangkok. Carga normal: 2,000 visitantes/hora. Pero cuando los boletos de un artista famoso salen a las 10:00 a.m., recibe 400,000 visitantes en diez minutos, los pagos no deben vender dos veces el mismo asiento, y los datos personales de los clientes tailandeses caen bajo la PDPA. Rápido, barato, resiliente — ThaiTicket necesita las tres cosas y no puede tenerlas, que es lo que lo convierte en el paciente de práctica perfecto.

Vea antes del Módulo 4 (enlaces verificados):

Video	Canal	Duración	Enlace
The Five Pillars of the AWS Well-Architected Framework	Amazon Web Services (oficial; un sexto pilar, Sustainability, fue añadido después)	~4 min	https://www.youtube.com/watch?v=KvEDbPmha6o
What is the AWS Well-Architected Framework?	Tech With Lucy	~10 min	https://www.youtube.com/watch?v=MpDJ6TCWKjk

Y guarde en marcadores el marco mismo — <https://aws.amazon.com/architecture/well-architected/> y el documento completo en <https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html> — vivirá en él durante ocho semanas.

Módulo 4 (Semanas 7–8): Confiabilidad — diseñar para el día en que las cosas se rompen

El credo del pilar: todo falla, todo el tiempo. Los discos mueren, las AZ se inundan, los despliegues salen mal, y siempre hay un certificado en algún lugar a punto de expirar. La confiabilidad no es la ausencia de fallas — es la *irrelevancia* de las fallas: diseñar de modo que cuando (no si) un componente muera, los usuarios nunca lo noten.

Preguntas clave (hágaselas a todo diseño, para siempre): ¿Qué pasa cuando falla cada componente — hay siempre un “y entonces qué”? ¿Cómo maneja el sistema 10× la carga?

¿Cómo *sabemos* que algo falló antes de que nos lo digan los clientes? ¿Cuáles son nuestros **RTO** y **RPO** — y quién en el negocio los aprobó? ¿Cuándo fue la última vez que *probamos* una recuperación?

RTO y RPO — los dos números que son la conversación del desastre. **Recovery Time Objective** (objetivo de tiempo de recuperación): ¿cuánto tiempo podemos estar caídos? **Recovery Point Objective** (objetivo de punto de recuperación): ¿cuántos datos podemos perder (es decir, qué tan viejo puede ser el último respaldo bueno)? Un RTO de 4 horas y un RPO de 1 hora significa: de vuelta en cuatro horas, habiendo perdido como máximo una hora de datos. Estas son decisiones de *negocio* con etiquetas de precio exponenciales — un RPO de 24 horas es un respaldo nocturno (barato); un RPO de ~cero es replicación continua (caro); un RTO de minutos significa infraestructura tibia ociosa en espera (muy caro). El trabajo del arquitecto es hacer que el negocio elija los números *a sabiendas* — y diseñar exactamente para ellos, no románticamente más allá de ellos.

Los patrones estándar: **redundancia** (dos de todo lo que importa — N+1) · **multi-AZ** (duplicados entre centros de datos; el balanceador de carga y el failover de la base de datos gestionada lo hacen automático) · **auto-scaling** (la capacidad sigue a la demanda) · **health checks + autorreparación** (instancias muertas detectadas y reemplazadas por maquinaria, no por humanos) · **respaldos, probados** (un respaldo no probado es una esperanza, no un plan — agende simulacros de restauración) · **degradación elegante** (bajo sobrecarga, suelte primero las funciones menos importantes: ThaiTicket puede dejar caer las vistas previas del mapa de asientos y conservar el checkout) · **colas como amortiguadores** (Etapa 3) · **evitar fallas en cascada** (timeouts, reintentos con backoff, circuit breakers — para que una dependencia lenta no ahogue a la flota).

Ejemplo trabajado — diseño de confiabilidad de ThaiTicket. Dos AZ en la región de Bangkok. Capa de aplicación stateless en un grupo de auto-scaling detrás de un balanceador de carga, precalentada por calendario antes de los horarios de venta anunciados (el auto-scaling reacciona en minutos; un lanzamiento a las 10:00 necesita capacidad a las 09:45). Base de datos Aurora, primaria en la AZ-a, copia síncrona en espera en la AZ-b, failover automático ≈ menos de un minuto. Una cola entre los clics de “comprar” y el procesamiento de pagos, para que una lentitud del proveedor de pagos encole pedidos en lugar de tumbar el sitio. Respaldos: continuos, restauración a un punto en el tiempo, simulacro de restauración mensual. Números acordados con el negocio: RTO 15 minutos, RPO ~cero para los pedidos (es dinero), RPO 24 horas para los datos de analítica (no lo es). Ejercicio de diario: ¿qué nos costó esto desde la esquina “barato” del triángulo?

Vocabulario:

Término	Definición
RTO / RPO	Recovery Time Objective: cuánto tiempo puede estar caído. Recovery Point Objective: cuántos datos puede perder. Los dos números que definen toda conversación de DR — y son decisiones de negocio.
Alta disponibilidad (HA)	Diseñar para que las fallas rutinarias (una

Término	Definición
	instancia, una AZ) no causen ninguna caída visible para el usuario.
Recuperación ante desastres (DR)	El plan para las fallas <i>grandes</i> — una región entera, un evento de ransomware — con objetivos de RTO/RPO y un runbook probado.
Redundancia / N+1	Capacidad de sobra para que cualquier componente pueda morir: N necesarios, N+1 corriendo.
Failover	El cambio automático a una copia en espera cuando la primaria muere.
Health check	El latido automatizado que detecta componentes muertos para que la maquinaria los esquivе.
Degradación elegante	Soltar funciones menos importantes bajo estrés para que el núcleo sobreviva.
Timeout / reintento con backoff	Rendirse ante una llamada lenta después de un límite / reintentar con pausas crecientes — modales que previenen fallas en cascada.
Circuit breaker	Un componente que deja de llamar a una dependencia que falla durante un rato para que se recupere — como el fusible de una casa.
Chaos engineering	Inyectar fallas deliberadamente (al estilo Netflix) para demostrar las afirmaciones de resiliencia antes de que la realidad las pruebe por usted.

Ejercicios: (1) Tome su diseño de pedidos de almuerzo de la Etapa 1 y mejórelo para sobrevivir: la muerte de una instancia, la pérdida de una AZ, una falla de base de datos y una hora pico de almuerzo de 10× — dibuje el antes y el después. (2) Escriba la conversación de RTO/RPO para tres negocios (un blog, un sitio de e-commerce, un sistema de expedientes hospitalarios) como un diálogo corto entre arquitecto y dueño — practique hacer visible el costo en el diálogo. (3) Diario: liste cada SPOF en el diseño de ThaiTicket de arriba. Queda al menos uno, dejado deliberadamente. (Pista: ¿cuántas regiones?)

Hito: usted puede ejecutar el interrogatorio de “¿qué pasa cuando esto falla?” sobre cualquier diagrama durante diez minutos seguidos sin quedarse seco, y puede explicar RTO vs RPO a un dueño no técnico usando una analogía de la inundación de una tienda en noventa segundos.

Módulo 5 (Semanas 9–10): Seguridad — diseñar sistemas difíciles de lastimar

El marco: el modelo de responsabilidad compartida. El proveedor asegura la *nube en sí* — edificios, hardware, el hipervisor. Usted asegura todo lo que *pone en ella* — datos, identidades,

configuraciones, código. Casi todas las brechas cloud famosas son errores de configuración del lado del cliente (un bucket público, una clave filtrada, un rol con permisos excesivos), y por eso la seguridad es un problema de *arquitectura* antes que un problema de herramientas: las descripciones de puesto dicen “integrar la seguridad en el diseño en lugar de añadirla al final”, y este módulo es el cómo.

Preguntas clave: ¿Quién y qué puede acceder a cada componente, y cada permiso es el *mínimo* necesario? ¿Dónde se cifran los datos — en reposo, en tránsito, y quién guarda las llaves? ¿Dónde están las fronteras de red, y qué las cruza? ¿Cómo *detectaríamos* una brecha — y podríamos contar la historia después a partir de los logs? ¿Qué ley aplica a estos datos, y dónde viven físicamente?

Los patrones estándar: mínimo privilegio (least privilege) (cada humano y programa recibe el acceso mínimo que su trabajo necesita — la regla de oro contra la que se juzga cada política de IAM) · **MFA en todas partes, root bajo llave · cifrado en reposo y en tránsito, siempre activo** (en la nube es una casilla; no hay excusa) · **segmentación de red** (subredes públicas/privadas, security groups como firewalls por servidor; el radio de daño de una brecha lo definen las paredes que usted dibujó por adelantado) · **secretos en una bóveda, nunca en el código · zero trust** (confianza cero; verificar cada solicitud explícitamente — identidad, dispositivo, contexto — no confiar en nada solo por estar “dentro de la red”; las descripciones de puesto lo nombran, usted también debe) · **defensa en profundidad** (capas, para que un control fallido no sea el fin del juego) · **registro de auditoría** (registros estilo CloudTrail de quién hizo qué — inmutables, monitoreados) · **la seguridad como barandillas, no como puertas** (codifique las reglas como política automatizada que hace del camino seguro el camino fácil, en lugar de una reunión de revisión que convierte a la seguridad en enemiga de la entrega).

Ejemplo trabajado — diseño de seguridad de ThaiTicket. Los datos de clientes (nombres, correos, referencias de pago) clasificados como datos personales bajo la PDPA → almacenados cifrados en Aurora en la región de Bangkok (residencia de datos), llaves en KMS. Red: solo el balanceador de carga es público; la capa de aplicación privada; la subred de la base de datos acepta conexiones *solo* desde el security group de la capa de aplicación. Humanos: SSO + MFA; los ingenieros obtienen acceso de solo lectura a producción por defecto y acceso elevado con límite de tiempo bajo solicitud (mínimo privilegio con rastro de auditoría). El manejo de tarjetas de pago delegado a un proveedor de pagos certificado para que los números de tarjeta en bruto nunca toquen nuestros sistemas — una decisión de *diseño* que elimina una carga entera de cumplimiento, que es la arquitectura de seguridad en su mejor forma. CloudTrail activado, alertas por accesos inusuales. La pregunta de la brecha, pre-respondida: la PDPA tiene un deber de notificación de 72 horas — los logs y el runbook deben permitirnos contar la historia en menos.

Vocabulario:

Término	Definición
Modelo de responsabilidad compartida	El proveedor asegura la nube; usted asegura lo que está en ella. La mayoría de las brechas están del lado del cliente de la línea.

Término	Definición
IAM / rol / política	Identity and Access Management — quién puede hacer qué. Una política es una lista de permisos; un rol es un paquete de ellas que se puede asumir.
Mínimo privilegio	La regla de oro: el acceso mínimo necesario, nada más, revisado regularmente.
MFA	Una segunda prueba de identidad además de la contraseña. No negociable para los humanos.
Cifrado en reposo / en tránsito / KMS	Datos codificados en disco / en la red / el servicio gestionado que guarda las llaves.
Security group	Una lista de reglas de firewall por servidor — “tráfico web entra solo desde el balanceador de carga”.
Segmentación de red / radio de daño (blast radius)	Dividir la red en zonas amuralladas / qué tan lejos puede llegar un atacante tras una brecha. Las paredes dibujadas por adelantado lo definen.
Zero trust	Verificar cada solicitud explícitamente; no confiar en nada por estar en el “interior”. La postura moderna por defecto.
Defensa en profundidad	Múltiples controles superpuestos para que una falla no sea fatal.
Gestión de secretos	Las contraseñas, llaves y tokens viven en un servicio de bóveda — nunca en el código, nunca en una hoja de cálculo.
Log de auditoría / CloudTrail	El registro inmutable de quién hizo qué, cuándo — cómo usted detecta problemas y los reconstruye después.
Clasificación de datos	Etiquetar los datos por sensibilidad (públicos / internos / personales / regulados) para que los controles correspondan a la etiqueta, no a una suposición.
Residencia de datos	Mantener los datos físicamente dentro de las fronteras de un país — un requisito legal en muchas industrias, un insumo de arquitectura siempre.
PDPA / GDPR	Las leyes de datos personales de Tailandia y Europa: consentimiento, notificación de brechas (72 horas), reglas de transferencia transfronteriza. La Etapa 4 profundiza.

Video de este módulo (enlace verificado): The AWS Shared Responsibility Model — Digital Cloud Training — ~4 min — <https://www.youtube.com/watch?v=ESPBBEK-cvo>

Ejercicios: (1) Dibuje el diagrama de ThaiTicket y superponga su seguridad: marque cada frontera de confianza, cada punto de cifrado, cada lugar donde viven credenciales. Este hábito de superposición — el mismo diagrama, la lente de seguridad — es exactamente lo que una revisión le pide. (2) Lea un post-mortem público de una brecha por mala configuración cloud (Capital One 2019 es el clásico de enseñanza) y escriba, en su diario, cuál *patrón de arriba* la habría prevenido. (3) Juego de roles con una IA: interpreta a un fundador de startup que dice “añadiremos la seguridad después” — convénzalo de lo contrario con la aritmética del costo de una brecha, con amabilidad.

Hito: usted puede tomar cualquiera de sus diagramas de la Etapa 1 y producir su superposición de seguridad en veinte minutos, y explicar el mínimo privilegio, el zero trust y el modelo de responsabilidad compartida a un interesado no técnico sin jerga.

Módulo 6 (Semanas 11-12): Rendimiento y costos — velocidad y dinero, los pilares acoplados

Estos dos pilares se enseñan juntos porque son la misma palanca empujada en direcciones opuestas, y el arquitecto es la persona con la mano en la palanca.

Eficiencia de rendimiento — preguntas clave: ¿Dónde sentirán los usuarios la lentitud primero? ¿Qué se calcula repetidamente que podría calcularse una vez y guardarse en caché? ¿Es cada componente la *herramienta correcta* (SQL haciendo el trabajo de un motor de búsqueda es un bug de rendimiento de la arquitectura, no del código)? ¿Cómo cambia el rendimiento a 10× — y dónde está el primer cuello de botella?

Patrones de rendimiento: capas de caché — el patrón de rendimiento de mayor palanca: caché del navegador — **CDN** en el borde (contenido estático servido desde una ciudad cercana al usuario; puede absorber la gran mayoría del tráfico de lectura) — caché de aplicación (**Redis** delante de la base de datos para lecturas calientes: mapas de asientos, sesiones, páginas de producto) — caché de consultas de la base de datos. Cada capa responde solicitudes antes de que lleguen al núcleo costoso; las preguntas de diseño son siempre *qué puede guardarse en caché, por cuánto tiempo y cómo se invalida cuando la verdad cambia*. Luego: **réplicas de lectura** (copias de la base de datos que sirven lecturas para que la primaria siga escribiendo) · **procesamiento asíncrono** (no haga esperar a los usuarios por trabajo que puede ocurrir después — recibos por correo, miniaturas, informes) · **la herramienta correcta para cada trabajo** (búsqueda — un motor de búsqueda; analítica — un warehouse; consultas calientes — un almacén clave-valor) · y **medir primero** (el trabajo de rendimiento sin medición es superstición).

Optimización de costos — preguntas clave: ¿Cuánto cuesta este diseño al mes con la carga de hoy — y por *unidad* (por pedido, por cliente)? ¿Qué está corriendo a las 3 a.m. que no necesita estarlo? ¿Qué uso estable podría comprometerse con descuento? ¿Qué diría finanzas que es la tendencia?

Patrones de costos: rightsizing (la mayoría de las flotas están silenciosamente sobredimensionadas; encoger a la necesidad medida es dinero gratis) · **estrategia de reservas**

— el menú de precios: **on-demand** (precio completo, libertad completa) para carga con picos/desconocida, **instancias reservadas / savings plans** (compromiso de 1–3 años, 30–70% de descuento) para la línea base estable, **spot** (hasta 90% de descuento, reclamable con poco aviso) para trabajo por lotes interrumpible; el movimiento del arquitecto es apilarlos: reservar el piso, auto-escalar el medio en on-demand, y poner los lotes en spot · **ciclo de vida de almacenamiento** (los niveles del Módulo 2, automatizados) · **apagar cosas** (los entornos no productivos durmiendo noches y fines de semana pueden recortar su costo en dos tercios) · **vigilar el egress** (los datos que *salen* de la nube se facturan; los diseños parlanchines entre regiones y las grandes descargas públicas sorprenden a todos una vez) · **etiquetado y showback** (cada recurso etiquetado con dueño/proyecto para que cada baht sea atribuible) · y la métrica corona, la **unit economics** (economía unitaria): no “la factura es \$800k/mes” sino “el costo por boleto vendido es \$1.90 y va bajando”. Las facturas crecientes están bien; los *costos unitarios* crecientes son un mal olor de arquitectura. Esta disciplina tiene nombre — **FinOps** — y los arquitectos se sientan en su centro.

Ejemplo trabajado — ThaiTicket, ambas lentes. Rendimiento: CloudFront sirve las páginas de artistas y las imágenes de mapas de asientos (los 400,000 curiosos en su mayoría nunca tocan los servidores); Redis guarda en caché la disponibilidad de asientos con un TTL de 2 segundos — lo bastante desactualizado para ser barato, lo bastante fresco para que el checkout (que verifica de nuevo contra Aurora, la fuente de la verdad) impida la doble venta; los recibos y boletos se generan asincrónicamente después del pago. Costos: la línea base estable de 2,000 visitantes/hora corre con un savings plan ($\approx 40\%$ de descuento); los picos de venta corren en on-demand durante la hora que existen; los trabajos de analítica corren en spot de noche; staging duerme fuera del horario laboral; cada recurso etiquetado `project:thaiticket`. La propuesta al CFO dice: “\$62,000/mes de línea base, $\approx$ \$9,000 por cada gran evento de venta, costo por boleto $\approx$ \$1.90 y decreciente con el volumen” — y *esa frase* es cómo suena un arquitecto fluido en costos.

Vocabulario:

Término	Definición
Latencia / throughput	Cuánto tarda una solicitud / cuántas solicitudes por segundo sostiene el sistema. Relacionados, no lo mismo.
Cuello de botella	El punto más estrecho que fija el ritmo de todo el sistema. Optimizar en cualquier otro lugar es decoración.
Caché / TTL / invalidación	Una copia rápida de datos costosos de obtener / cuánto tiempo puede confiarse en ella / el problema difícil de refrescarla cuando la verdad cambia.
CDN	La caché más externa — su contenido en cientos de ciudades. Primera respuesta a “hazlo rápido globalmente”.
Réplica de lectura	Una copia de la base de datos que sirve lecturas, para que la primaria conserve su

Término	Definición
Procesamiento asíncrono	fuerza para las escrituras. Hacer el trabajo no urgente después de responder al usuario, normalmente mediante una cola.
On-demand / reservada / savings plan / spot	Flexible a precio completo / compromiso de 1–3 años con 30–70% de descuento / la misma idea, más flexible / hasta 90% de descuento pero reclamable. El arquitecto apila las cuatro.
Rightsizing	Encoger recursos sobredimensionados a la necesidad medida. Dinero gratis en casi toda cuenta.
Egress	Datos que salen de la nube — facturados por GB. La sorpresa famosa de la factura; diseñe los flujos de datos con esto en mente.
Etiquetado / showback	Etiquetar cada recurso con dueño y proyecto / mostrar a cada equipo su propia factura. La rendición de cuentas cambia el comportamiento.
Unit economics	Costo por unidad de negocio (por pedido, por usuario). La métrica que hace significativa una factura — y la frase favorita del arquitecto frente a un CFO.
TCO	Costo total de propiedad — el precio de etiqueta más personas, licencias, migración y costos de salida durante toda la vida de una decisión.
FinOps	La disciplina de hacer el gasto cloud visible, asignado y continuamente optimizado.

Video de este módulo (enlace verificado): What is FinOps? — FinOps Foundation (oficial) — ~2 min — https://www.youtube.com/watch?v=Y-c_xw9bHFw

Ejercicios: (1) Cotice la línea base de ThaiTicket en la calculadora de precios de AWS (busque “AWS pricing calculator” — el ejercicio de detallar un diseño real una vez desmitifica toda conversación futura de costos); compare su total con los \$62,000 de arriba y explique cualquier brecha. (2) Añada la superposición de caché a dos de sus diagramas de la Etapa 1: marque cada caché, su TTL y su historia de invalidación. (3) Una IA interpreta a un ingeniero que quiere todo en on-demand “para mantenerlo simple” — negocie la estrategia de reservas, con números.

Hito: para cualquier diseño que haya dibujado, usted puede enunciar sus tres mayores líneas de costo y su probable primer cuello de botella — y proponer un cambio que mejore ambos a la vez (casi siempre hay uno; la caché suele serlo).

Módulo 7 (Semanas 13–14): Excelencia operativa, sostenibilidad — y ejecutar los seis pilares juntos

La excelencia operativa es el pilar que pregunta: *¿pueden los humanos realmente operar esta cosa, con calma?* Preguntas clave: ¿Cómo llega un cambio a producción — a través de un pipeline automatizado y probado, o a través de un humano heroico? ¿Cómo sabemos que el sistema está sano *ahora mismo*? Cuando se rompe a las 3 a.m., ¿qué hace realmente el ingeniero de guardia? Patrones: **infraestructura como código** (el entorno definido en texto revisado y bajo control de versiones — Terraform/CloudFormation — para que sea reproducible y auditable; una arquitectura que existe solo como clics de consola es folclore, no ingeniería) · **pipelines de CI/CD** con estrategias de despliegue seguras (**blue-green**: levante la versión nueva junto a la vieja y cambie; **canary**: dele a la versión nueva el 5% del tráfico y observe) · **observabilidad** (logs, métricas, trazas y alertas ligadas a síntomas *visibles para el usuario*, no a trivialidades de máquina) · **runbooks** (el qué-hacer escrito para cada falla conocida) · **post-mortems sin culpas** (tras los incidentes: qué falló, por qué, qué previene una repetición — sin villanos, o la gente deja de decir la verdad). La parte del arquitecto: *diseñar* para la operabilidad — un sistema el doble de ingenioso y la mitad de observable es un sistema peor.

La sostenibilidad, el sexto pilar, le pide desperdiciar menos del mundo físico: haga rightsizing (las CPUs ociosas queman electricidad real), escale según la demanda en lugar de aprovisionar para el pico, use servicios gestionados y serverless (la infraestructura compartida es infraestructura más llena), use niveles de almacenamiento y borre lo que está muerto. Convenientemente, la elección sostenible y la elección barata suelen ser la misma elección — diga ambas en las revisiones y se ganará la sala dos veces.

Ejecutar los pilares juntos — su primera revisión Well-Architected. En las revisiones reales los pilares *chocan*: la confiabilidad multi-región pelea con el costo; la revisión de seguridad estricta pelea con la velocidad operativa; la caché de rendimiento pelea con la corrección de frescura de los datos. El marco no resuelve los conflictos por usted — los fuerza a salir a la luz, donde el negocio puede elegir. Ese es todo su genio, y el suyo para empuñarlo. El método de la revisión en sí (los conjuntos de preguntas, priorizar hallazgos, el informe) se enseña completo en la Etapa 5, Módulo 12; esta quincena usted ensaya la habilidad cruda: interrogar un diseño desde seis direcciones en una sola sesión.

Vocabulario:

Término	Definición
Infrastructure as Code (IaC) / Terraform	Infraestructura declarada en archivos de texto bajo control de versiones que las herramientas convierten en realidad / la herramienta más popular de este tipo.
Pipeline de CI/CD	La cinta automatizada que construye, prueba y despliega cada cambio.
Despliegue blue-green / canary	Dos patrones de lanzamiento seguro: cambiar el tráfico entre la copia vieja y la nueva / gotear tráfico a la versión nueva y observar.

Término	Definición
Observabilidad (logs, métricas, trazas)	La capacidad del sistema de explicarse: registros de eventos, números en el tiempo y mapas del viaje de cada solicitud.
Alerta / on-call / runbook	El aviso automático cuando se rompen umbrales / la rotación humana que lo responde / el guion escrito que siguen.
Post-mortem sin culpas	La revisión honesta y sin villanos después de un incidente, que produce prevención, no castigo.
Drift	La realidad divergiendo de la definición de IaC porque alguien hizo clic en algo. El enemigo de la reproducibilidad.
Toil	Trabajo operativo manual y repetitivo que la automatización debió haber absorbido. Los arquitectos lo diseñan fuera.
Revisión Well-Architected	Un interrogatorio estructurado de un workload contra los seis pilares, que produce hallazgos priorizados. La Etapa 5 le enseña a dirigir una.

Ejercicios: (1) El drill de las seis lentes: tome su mejor diagrama y dedique diez minutos por pilar a escribir hallazgos — sesenta minutos, un diseño, seis lentes; este drill *es* la Etapa 2 en miniatura, repítalo semanalmente de ahora en adelante. (2) Lea dos informes post-incidente de AWS (publicados en sus páginas de estado tras caídas mayores) e identifique cuál patrón de cuál pilar falló. (3) Diario: para ThaiTicket, escriba los tres conflictos entre pilares que usted le presentaría al negocio, cada uno como una elección de una frase (“Podemos tener X o Y con este presupuesto — ¿cuál?”).

Hito — fin de la Etapa 2: la revisión simulada: una IA genera una arquitectura defectuosa para una startup tailandesa de préstamos en línea; usted debe producir hallazgos escritos en los seis pilares, incluyendo al menos: una brecha de confiabilidad (su base de datos en una sola AZ), una brecha de seguridad (IAM demasiado amplio), una brecha de costos (todo en on-demand), una brecha operativa (sin IaC) y la conversación de RTO/RPO que nunca tuvieron — cada hallazgo formulado como una pregunta que un colega podría escuchar sin estremecerse. Cuando su lista de hallazgos se lea como la de un colega senior servicial y no la de un auditor, la Etapa 2 está terminada.

ETAPA 3 — EL CATÁLOGO DE PATRONES (Semanas 15–20)

Los arquitectos no inventan; **seleccionan**. Esta etapa es su catálogo de las arquitecturas estándar — para cada una: qué es, cuándo usarla, cuándo *no*, y su perfil de costo/complejidad. Las columnas de “cuándo no” son la verdadera carga de esta etapa: cualquier curso puede

decirle qué son los microservicios; saber cuándo destrozarán a una empresa es aquello por lo que le pagan. Mientras estudia, siga navegando arquitecturas de referencia reales en <https://aws.amazon.com/architecture/> — detectar patrones en la naturaleza es el ejercicio que hace que el catálogo se fije.

Módulo 8 (Semanas 15–17): Patrones de aplicación

Monolito vs. microservicios — la discusión más ruidosa de la industria, resuelta con calma. Un **monolito** es una aplicación desplegable que contiene todas las funciones; los **microservicios** dividen el sistema en muchos servicios pequeños, desplegados independientemente, cada uno dueño de sus datos, hablando por APIs y eventos. La tabla honesta:

Patrón	Qué es	Úselo cuando	NO lo use cuando	Costo/ complejidad
Monolito	Una app, un despliegue, una base de datos	Equipo pequeño, producto joven, fronteras de dominio poco claras — es decir, la mayoría de los sistemas nuevos	Múltiples equipos bloqueados entre sí; partes que necesitan escalar de manera muy distinta	Baja complejidad, bajo costo; escala más lejos de lo que la moda admite — lo “aburrido” es una virtud
Microservicios	Muchos servicios pequeños, contruidos, desplegados y escalados de forma independiente	Muchos equipos que necesitan cadencia de lanzamiento independiente; escalado radicalmente distinto por parte; fronteras de dominio probadas y estables	Equipo pequeño (“un monolito distribuido es un monolito con fallas de red añadidas”); dominio todavía cambiante	Alta complejidad: cada llamada de función se vuelve una llamada de red que puede fallar; necesita CI/CD maduro, observabilidad, on-call

El fallo del arquitecto: **empiece monolítico, modular por dentro; extraiga servicios cuando — y solo cuando — llegue de verdad un dolor de escalado de equipo o de carga.** Decir esto en las entrevistas, con razones, lo marca como senior; la ideología en cualquiera de las dos direcciones lo marca como junior.

Arquitectura orientada a eventos — colas y topics, los amortiguadores. En lugar de componentes que se llaman entre sí y *esperan* (síncrono), los componentes publican **eventos** (“OrderPlaced”) a una **cola** (SQS — un consumidor toma cada mensaje, a su propio ritmo) o a un **topic** (SNS — cada suscriptor recibe una copia; fan-out). El productor no sabe ni le importa quién escucha. Úsela cuando: los componentes deben sobrevivir las caídas de los otros (la cola

retiene los mensajes mientras un consumidor está caído — resiliencia y desacoplamiento en una sola compra); la carga tiene picos (la cola absorbe el pico; los workers la drenan con constancia); una ocurrencia dispara muchas reacciones (pedido realizado — cobro, correo, inventario, analítica — cuatro suscriptores, cero acoplamiento). NO la use cuando: el usuario necesita la respuesta *ahora* (el checkout no puede confirmar “eventualmente”); o el equipo es pequeño y una simple llamada síncrona bastaría — cada cola añade semánticas de entrega (at-least-once significa que los consumidores deben ser **idempotentes** — seguros de ejecutar dos veces), manejo de dead letters y una carga de monitoreo. Costo/complejidad: componentes baratos, depuración más cara — la historia de una solicitud está ahora dispersa entre servicios y colas, y por eso la observabilidad (Módulo 7) deja de ser opcional aquí.

Tres capas: ya es suya (Módulo 3). Úsela para: el amplio centro de las aplicaciones web — es el patrón contra el que se miden los demás. No para: workloads con picos muy marcados y valles ociosos (serverless es más barato) o productos genuinamente enormes de múltiples equipos (vea arriba). Perfil: bien entendida en todas partes, se contrata fácil, costo medio.

Serverless-first: componga piezas gestionadas — API Gateway — funciones Lambda — DynamoDB, S3 y colas — sin poseer ningún servidor. Úsela cuando: tráfico con picos o bajo (escalar a cero significa que lo ocioso cuesta ~nada), equipos pequeños, rapidez de salida al mercado, pegamento orientado a eventos. NO la use cuando: cómputo de larga duración o especializado (límites de runtime), rutas ultrasensibles a la latencia (cold starts), carga *estable* masiva (el precio por invocación puede superar a los servidores reservados — haga la aritmética a escala), o cuando la portabilidad entre nubes es un requisito genuino (este es el lock-in más profundo de todos los patrones — a menudo vale la pena, pero dígalos en voz alta). Perfil: la carga de operaciones más baja del catálogo; costo excelente a escala baja/con picos, requiere verificación a gran escala estable.

Vocabulario:

Término	Definición
Monolito / monolito modular	Una app desplegable con todo adentro / la versión disciplinada: un desplegable, fronteras internas de módulos limpias — el mejor punto de partida para sistemas nuevos.
Microservicios	Muchos servicios pequeños desplegados independientemente, cada uno dueño de sus propios datos. Una herramienta de escalado de equipos que cobra un impuesto de sistemas distribuidos.
Acoplamiento / desacoplamiento	Cuánto dependen los componentes de la disponibilidad y los detalles de los otros. La arquitectura es en gran parte el arte de comprar la cantidad correcta de desacoplamiento.
Síncrono / asíncrono	Llamar-y-esperar vs. enviar-y-continuar. La elección fundamental en cada flecha que

Término	Definición
Evento / orientado a eventos	usted dibuja. Un hecho anunciado a quien escuche (“OrderPlaced”) / una arquitectura construida a partir de tales anuncios.
Cola (SQS)	Una fila de mensajes; cada uno tomado por un consumidor a su propio ritmo. Amortiguador y desacoplador.
Topic / pub-sub (SNS)	Un canal de difusión; cada suscriptor recibe cada mensaje. La herramienta de fan-out.
Dead-letter queue	Donde los mensajes que fallan repetidamente al procesarse se apartan para los humanos — la red de seguridad del patrón.
Idempotente	Seguro de procesar dos veces con el mismo resultado — requerido de los consumidores, porque las colas pueden entregar un mensaje más de una vez.
Serverless-first	Componer piezas gestionadas que escalan a cero (funciones, bases de datos gestionadas, colas) en lugar de operar servidores.
Lock-in	Dependencia de los servicios propietarios de un proveedor, cotizada como costo de cambio. No es un pecado — un término económico a sopesar a la vista de todos (Etapa 4).

Ejercicios: (1) Diseñe el flujo de pedidos de ThaiTicket dos veces — tres capas síncrono, y luego orientado a eventos con SQS/SNS — y escriba un párrafo sobre qué hace cada versión durante una caída del proveedor de pagos; ese párrafo es todo el argumento a favor de los eventos. (2) Para cuatro empresas (startup de 3 personas, scale-up de 50 ingenieros, un banco, una app de votación de TV usada 4 noches al año), elija un patrón y defiéndalo — y luego nombre el disparador que haría que cada empresa cambiara de patrón. (3) Encuentre una historia real de blog de ingeniería de “migramos a microservicios y nos arrepentimos” y una historia de éxito; anote en el diario la diferencia (casi siempre es el tamaño del equipo y la madurez del dominio).

Hito: dada la descripción de una empresa, usted puede recomendar un patrón *con su rampa de salida* — “empiece aquí; cuando pase X, evolucione hacia Y” — en cinco minutos. Rutas de evolución, no veredictos, son la manera en que los arquitectos hablan de verdad.

Módulo 9 (Semanas 18–20): Patrones de datos, globales e híbridos

Arquitecturas de datos — lake vs. warehouse. Las bases de datos transaccionales (Etapa 1) hacen funcionar el negocio; la analítica quiere *hacerle preguntas* sin frenarlo. Un **data warehouse** (Redshift, Snowflake, BigQuery) almacena datos *estructurados y limpios* optimizados para analítica SQL rápida — úselo cuando las preguntas son conocidas y los tableros deben ser rápidos; cuesta más por TB, necesita disciplina de modelado por adelantado.

Un **data lake** (S3 + catálogo + motores de consulta como Athena) almacena *todo, en bruto, barato* — logs, clickstreams, imágenes — y aplica la estructura al momento de leer; úselo cuando quiere conservarlo todo ahora y decidir las preguntas después; pero sin gobernanza degenera en el chiste de la industria, el **data swamp** (pantano de datos). El consenso moderno es ambos, en capas (“lakehouse”): la verdad en bruto en el lake, los marts curados en el warehouse, alimentados por **pipelines de ETL/ELT**. Las reglas del arquitecto: la analítica nunca consulta la base de datos de producción (la alimentan réplicas o pipelines), y cada dataset tiene un dueño, una entrada de catálogo y una política de retención — esa línea de gobernanza es una frase en su diseño y un año de dolor si la omite.

Multi-región: activo-pasivo vs. activo-activo. Cuando una región no basta — porque el negocio exige DR ante desastres a escala de región, o los usuarios abarcan continentes — usted elige:

Patrón	Qué es	Úselo cuando	NO lo use cuando	Costo/ complejidad
Activo-pasivo	Una región sirve; una región en espera guarda datos replicados, desde “solo respaldos” (fría) hasta “copia reducida corriendo” (tibia), promovida ante el desastre	RTO/RPO declarados por el negocio lo justifican; el cumplimiento exige una historia de DR	Nadie ha firmado el RTO/RPO que justifica el gasto (la necesidad honesta de la mayoría de las empresas es un buen multi-AZ + respaldos entre regiones)	1.1×–1.7× de costo según la tibieza del standby; complejidad moderada — el requisito asesino es un <i>failover que usted realmente ensaya</i> , o el standby es teatro
Activo-activo	Dos o más regiones sirven simultáneamente; los usuarios enrutados a la más cercana; datos replicados en ambos sentidos	Base de usuarios global que quiere latencia local; RTO cercano a cero genuinamente requerido (redes de pagos, trading, grandes SaaS)	Casi todos los demás — los <i>conflictos de escritura</i> entre regiones son uno de los problemas genuinamente difíciles de la computación	2×+ de costo, la complejidad más alta de este curso; necesita almacenes de datos que resuelvan conflictos y equipos senior

La frase de nivel de entrevista: “*Multi-AZ is table stakes; multi-region is a business case.*” (Multi-AZ es lo mínimo esperable; multi-región es un caso de negocio.) Haga que el negocio declare el RTO/RPO, coteje las opciones y deje que los números elijan.

Nube híbrida. Parte on-prem, parte cloud, unidas por VPN o Direct Connect — para la mayoría de las empresas establecidas no es un patrón sino una *realidad de una década*: mainframes que no pueden moverse, sistemas de fábrica atados a la latencia, datos regulados fijados en las

instalaciones, y una migración (Etapa 4) de paso. Úsela como: un puente deliberado con una dirección de viaje. NO acepte: “híbrido” como eufemismo de “nunca decidimos”. Notas de diseño: la identidad debe unificarse primero (un solo inicio de sesión entre ambos mundos — lo que las descripciones de puesto empresariales quieren decir con “hybrid identity integration”); nombre qué sistemas son la fuente de la verdad; vigile los costos de egress a través del enlace; y espere que el enlace de red *sea* un SPOF a menos que se duplique. Complejidad: todo por partida doble, un parque por cada mundo — la razón honesta por la que los arquitectos empujan a encoger el lado on-prem con constancia.

Vocabulario:

Término	Definición
OLTP / OLAP	Procesamiento de transacciones (muchas lecturas/escrituras pequeñas y rápidas — hace funcionar el negocio) vs. procesamiento analítico (escaneos enormes — estudia el negocio). Sepárelos.
Data warehouse	Almacenamiento estructurado y curado, optimizado para analítica SQL rápida (Redshift, Snowflake, BigQuery).
Data lake / data swamp	Almacenamiento barato de todo, en bruto, estructurado al leer (S3 + Athena) / la versión sin gobernanza, donde los datos van a perderse.
ETL / ELT	Los pipelines que mueven datos desde los sistemas de origen hacia el lake/warehouse (Extract, Transform, Load — el orden varía).
Gobernanza de datos / catálogo / retención	Reglas de propiedad, documentación y vida útil para cada dataset — una frase de diseño que ahorra un año de dolor.
Replicación (síncrona / asíncrona)	Copiar datos continuamente a otra base de datos o región — instantáneamente consistente pero limitada por la distancia vs. levemente atrasada pero a cualquier lugar. El rezago asíncrono es de donde viene el RPO.
Activo-pasivo / simulacro de failover	DR con región en espera / el ensayo agendado que demuestra que funciona. Un failover sin ensayar es una esperanza.
Activo-activo	Múltiples regiones sirviendo a la vez con replicación bidireccional. Poderoso; genuinamente difícil; normalmente innecesario.
Conflicto de escritura	Dos regiones cambiando los mismos datos a la vez — la razón técnica por la que activo-

Término	Definición
Nube híbrida / identidad híbrida	activo es territorio de expertos. On-prem + cloud unidos como un solo parque / un solo inicio de sesión entre ambos — lo primero que hay que unificar.
Direct Connect	La línea física privada que une el centro de datos con la nube — el umbilical híbrido, duplicado si importa.

Ejercicios: (1) ThaiTicket se vuelve regional: diseñe la expansión a Singapur dos veces — activo-pasivo (Bangkok primaria) y activo-activo — con múltiplos de costo y RTO/RPO para cada una; escriba la recomendación de una página y *tome una decisión*. (2) Un minorista tiene 15 años de ventas en una base de datos SQL de producción y quiere “analítica lista para IA”; esboce el diseño de lake + warehouse y escriba las tres frases de gobernanza. (3) Detección de patrones: elija tres arquitecturas de referencia de <https://aws.amazon.com/architecture/> y nombre cada patrón del catálogo presente en cada una.

Hito — fin de la Etapa 3: el desafío del catálogo: una IA le da seis escenarios a ritmo rápido; para cada uno usted nombra el patrón, la razón, la advertencia de cuándo-NO y el perfil de costo/complejidad — en menos de cinco minutos cada uno. Este intercambio exacto, a esta velocidad exacta, es el tercio medio de una entrevista real de arquitecto.

ETAPA 4 — MIGRACIÓN Y EL MUNDO REAL (Semanas 21–26)

El diseño greenfield es la minoría del trabajo. La mayor parte de la arquitectura ocurre en empresas que ya existen — con salas de servidores, software antiguo del que depende el negocio, contratos y leyes. Esta etapa es ese mundo, y es donde se enseña “liderar iniciativas de migración y modernización” — un punto en casi todas las descripciones de puesto que estudiamos.

Módulo 10 (Semanas 21–23): Migración — mover una empresa a la nube

Las 7 R — el vocabulario compartido de toda conversación de migración. Para cada workload del parque, usted elige una:

R	Significado	Cuándo	Esfuerzo / recompensa
Retire (retirar)	Apagarlo — nadie lo usa realmente	Todo parque tiene un 10–20% de estos; encuéntrelos primero	Trivial / ahorro instantáneo — la mejor R
Retain (retener)	Dejarlo on-prem, por ahora	Sistemas atados a la latencia, fijados por cumplimiento o próximos al fin de su	Ninguno / difiere el costo — un “todavía no” honesto

R	Significado	Cuándo	Esfuerzo / recompensa
		vida	
Rehost (“lift and shift”)	Moverlo a VMs cloud tal cual	La velocidad importa, las apps son estables, las habilidades escasean	Bajo / salida rápida del centro de datos, pero poco beneficio cloud aún — un primer paso, no un destino
Relocate (reubicar)	Moverlo a nivel de hipervisor (p. ej., flota VMware a VMware-en-cloud)	Grandes parques virtualizados con fecha límite	Bajo / el movimiento masivo más rápido; una escala intermedia
Repurchase (“drop and shop”)	Reemplazarlo con SaaS	Apps no diferenciadas — correo, RR. HH., CRM	Bajo-medio / categorías enteras salen de su parque
Replatform (“lift, tinker, shift”)	Pequeñas mejoras en el camino — BD autogestionada — RDS, app — contenedores	El punto medio pragmático: beneficio real, riesgo acotado	Medio / la R caballo de batalla de la mayoría de las migraciones
Refactor / re-architect (rearquitectar)	Reescribir para cloud-native (patrones de la Etapa 3)	Sistemas centrales diferenciadores cuyos límites lastiman al negocio	Alto / la mayor recompensa — gástela en los pocos workloads que la merecen

El método de migración: evaluar — movilizar — migrar. Evaluar: inventaríe todo (herramientas de descubrimiento más la arqueología de preguntarle a la gente), y califique cada workload por valor de negocio y dificultad de migración — productos: un inventario de aplicaciones con una R por fila, y un **caso de negocio** (TCO de quedarse vs. moverse; sea honesto en que la factura *sube* durante el periodo de traslape en que ambos parques corren — los líderes a quienes no se les advierte sobre la burbuja de migración se convierten en líderes que cancelan migraciones a la mitad). **Movilizar:** construya la **landing zone** (zona de aterrizaje) — la fundación cloud pre-construida y gobernada *antes* de que aterrice el primer workload: estructura multi-cuenta (cuentas separadas por entorno y equipo, para que los radios de daño se mantengan pequeños), identidad y registro centralizados, el hub de red (VPCs hub-and-spoke, el Direct Connect de vuelta a on-prem) y **guardrails** (barandillas) — políticas automatizadas que hacen del camino seguro, etiquetado y conforme el camino por defecto (AWS Control Tower es el kit de inicio gestionado). Las descripciones de puesto dicen “diseñar la landing zone que cada workload hereda” — esto es eso. Saltársela para “simplemente empezar a migrar” recrea el centro de datos desordenado en la nube a mayor costo; es la firma clásica de la migración fallida. **Migrar por olas:** agrupe los workloads en olas de unos pocos, ordenadas de lo fácil primero: la Ola 1 es deliberadamente de bajo riesgo (aprenda la maquinaria donde los errores son baratos), las olas posteriores llevan las joyas de la corona con

cutovers ensayados, cada una con un plan de rollback y un período de **hypercare** de vigilancia intensificada. Para cada movimiento de base de datos, las dos preguntas que importan son los números del Módulo 4 disfrazados: cuánta caída puede tomar el cutover (RTO) — y la replicación continua con un cambio corto existe para cuando la respuesta es “casi ninguna”.

Vocabulario:

Término	Definición
7 Rs	Retire, retain, rehost, relocate, repurchase, replatform, refactor — el menú de migración por workload. (También oírás “6 Rs” — la lista más vieja sin relocate.)
Descubrimiento / inventario de aplicaciones	Averiguar qué corre realmente (herramientas + entrevistas) / la lista resultante, una fila por workload, con dueño, dependencias y su R.
Mapeo de dependencias	Trazar qué habla con qué — la razón por la que los workloads migran en grupos, y las migraciones sin él fallan el primer día.
Caso de negocio / burbuja de migración	El argumento de TCO quedarse-vs-moverse / la joroba temporal de costos mientras ambos parques corren. Advierta sobre ella o será emboscado por ella.
Landing zone	La fundación cloud gobernada y pre-construida — cuentas, identidad, red, registro, guardrails — que cada workload hereda. Construida <i>antes</i> de la migración.
Estrategia multi-cuenta	Cuentas cloud separadas por entorno/equipo, para que la facturación sea atribuible y el radio de daño quede contenido.
Guardrail	Una política automatizada que previene o señala acciones no conformes — gobernanza como maquinaria, no como memorandos.
Control Tower	El servicio gestionado de AWS para levantar una landing zone multi-cuenta con guardrails.
Plan de olas	El calendario de migración en grupos pequeños, lo fácil primero, las dependencias juntas.
Cutover / rollback / hypercare	El momento de cambiar a la copia cloud / el deshacer ensayado / la ventana de soporte intensificado después.

Ejercicios: (1) La migración en papel: una IA genera el inventario de una empresa ficticia de 40 servidores (la volverá a encontrar como Capstone 2); asigne una R a cada fila y defienda las diez

decisiones más difíciles. (2) Esboce una landing zone: diagrama de cuentas, flujo de identidad, hub-and-spoke de red, cinco guardrails que impondría desde el primer día. (3) Escriba la advertencia de dos párrafos sobre la “burbuja de migración” a un CFO — practicar las malas noticias temprano es el cardio del arquitecto.

Hito: dado un inventario de ~20 workloads con descripciones, usted produce una tabla defendible de R por workload, un plan de tres olas con razonamiento y un esbozo de landing zone — en una sola sesión.

Módulo 11 (Semanas 24-26): Restricciones — ley, sistemas heredados y lock-in

La residencia de datos y las leyes de privacidad como insumos de arquitectura. Las leyes de datos personales — la **PDPA** de Tailandia, el **GDPR** de Europa y sus primas en todo el mundo — comparten una forma que un arquitecto debe conocer al dedillo: los datos personales necesitan una base legal (a menudo el consentimiento); los individuos tienen derechos (acceso, corrección, borrado — su diseño debe poder *encontrar y borrar los datos de una persona*, lo cual es difícil si los ha dispersado en copias sin gobernanza); las brechas conllevan un deber de **notificación en 72 horas** (su registro debe permitirle contar la historia en menos); y las **reglas de transferencia transfronteriza** restringen dónde pueden vivir físicamente los datos — lo cual es una restricción de *selección de región*, una restricción de *diseño de replicación* (esa copia de analítica en Singapur puede ser un evento legal) y una razón por la que existen las regiones dentro del país. El método del arquitecto, en orden: clasifique los datos (¿qué es personal?), mapee sus viajes (cada almacén, cada copia, cada frontera cruzada — los flujos que nadie dibujó son donde viven las violaciones), y luego diseñe los controles: regiones conformes con la residencia, cifrado, calendarios de retención y una ruta de borrado que realmente alcance la expiración de los respaldos. Diga “data protection by design” (protección de datos desde el diseño) — es la frase que usan ambas leyes, y es literalmente su título de trabajo en una frase.

Integración con sistemas heredados. El sistema de facturación en mainframe no se mueve este año, y la flamante app cloud debe hablar con él. Los patrones: una **anti-corruption layer** (capa anticorrupción) — un servicio de traducción entre lo nuevo y lo viejo, para que las formas de datos extrañas del sistema heredado no se filtren y corrompan su diseño nuevo; el **strangler fig** (higuera estranguladora) — enrute el tráfico a través de una fachada, y luego desprenda funciones del sistema heredado una por una hasta que, años después, pueda apagarse (nombrado por la higuera que envuelve lentamente a un árbol huésped; vence a las reescrituras big-bang, que fallan a una tasa legendaria); puentes por lotes y tomas de eventos para los datos que deben fluir entre mundos; y respeto por la física de latencia de las llamadas parlanchinas cloud-a-on-prem (Direct Connect ayuda; mejor aún, diseñe la charla fuera). La regla: **contenga lo heredado, no se contagie de ello** — cada componente nuevo debe construirse como si el sistema heredado ya no existiera.

La economía del vendor lock-in. Cada servicio gestionado conveniente profundiza su matrimonio con un proveedor; la portabilidad (abstracciones multi-cloud, todo autogestionado) es una opción real que cuesta dinero real en complejidad y velocidad sacrificada. Ninguno es un pecado — el modo de falla no es *elegir* el lock-in, es no *notarlo*. La herramienta del arquitecto es una estimación de costo de salida escrita en el diseño: “Usar DynamoDB ahorra ≈2 años-ingeniero ahora; cambiar después ≈ una reescritura de 6 meses de la capa de datos — lo

aceptamos, y aquí está la frontera de interfaz que encogería la reescritura.” Tres frases, cotizadas con honestidad, decisión registrada (los ADRs de la Etapa 5) — eso es gestión madura del lock-in. El multi-cloud *como estrategia* (correr el mismo workload de forma portátil en dos nubes) suele ser la respuesta más cara posible y la compran exactamente las organizaciones cuya escala o regulador lo exige; el multi-cloud *como hecho* (distintos workloads en distintas nubes por historia o por conveniencia) es la vida normal.

Vocabulario:

Término	Definición
PDPA / GDPR	Las leyes de protección de datos personales de Tailandia y Europa — el par plantilla de las restricciones de leyes de privacidad en todo el mundo.
Data protection by design	Construir los controles de privacidad en la arquitectura desde el primer boceto — la frase legal y el deber del arquitecto.
Base legal / consentimiento	La justificación legal requerida para procesar datos personales.
Derecho al borrado	El derecho de un individuo a que sus datos sean eliminados — algo que su diseño debe hacer <i>posible</i> .
Mapeo de flujos de datos	Trazar cada almacén, copia y cruce de frontera de una categoría de datos. Donde están los flujos no dibujados, viven las violaciones.
Transferencia transfronteriza	Datos personales saliendo del país — regulada; convierte el diseño de replicación en una cuestión legal.
Anti-corruption layer	Un componente de traducción que impide que las rarezas de un sistema heredado se filtren en un diseño nuevo.
Strangler fig	Modernizar enrutando a través de una fachada y reemplazando el sistema heredado pieza por pieza hasta poder apagarlo.
Reescritura big-bang	Reemplazar un sistema de una sola vez, con cutover en un solo día. Tasa de fracaso legendaria; el strangler fig existe por su culpa.
Costo de salida / costo de cambio	El costo cotizado con realismo de dejar un proveedor o servicio — el número que convierte el lock-in de miedo en economía.
Multi-cloud (estrategia vs. hecho)	Correr deliberadamente de forma portátil en múltiples nubes (caro, rara vez justificado)

Término	Definición
	vs. simplemente tener workloads en varias (normal).

Ejercicios: (1) Mapee los flujos de los datos de clientes de ThaiTicket: cada almacén, cada copia (no olvide los logs, los respaldos, el pipeline de analítica, las exportaciones del equipo de soporte), cada frontera; luego escriba la ruta de borrado. Sienta cómo el mapa encuentra problemas que el diagrama escondía. (2) Diseñe el plan de strangler fig para un sistema de inventario on-prem de 20 años, con los tres primeros desprendimientos nombrados. (3) Escriba el párrafo de lock-in al estilo DynamoDB (beneficio ahora, costo de salida después, aceptado o mitigado) para tres servicios que usted realmente usaría.

Hito — fin de la Etapa 4: el desafío de las restricciones: un encargo de diseño cargado con las tres (“aseguradora tailandesa, datos de clientes, sistema de pólizas en mainframe, junta nerviosa por la dependencia de AWS”) — usted produce un enfoque de una página que toca residencia, integración y lock-in, cada uno con un patrón nombrado y un costo honesto. Cuando las restricciones lo emocionan más de lo que lo haría el greenfield — porque las restricciones son donde los arquitectos ganan más que los dibujantes de diagramas — la Etapa 4 está terminada.

ETAPA 5 — EL OFICIO DEL ARQUITECTO (Meses 7–12)

Todo lo anterior lo hizo capaz de *diseñar*. Esta etapa lo hace capaz de *trabajar como arquitecto* — los documentos, revisiones, presentaciones y pruebas de las que el rol está hecho realmente, más las certificaciones que lo hacen pasar de RR. HH., y tres capstones que se convierten en su portafolio.

Módulo 12 (Meses 7–8): Los entregables — ADRs, conjuntos de diagramas y dirigir revisiones

Architecture Decision Records (ADRs, registros de decisiones de arquitectura). Un ADR es un documento de una página que captura una decisión significativa: qué elegimos, qué no elegimos y por qué — escrito *cuando se toma la decisión*, numerado y conservado en el repositorio del proyecto para siempre. Por qué las descripciones de puesto los nombran: dentro de dos años, alguien preguntará “¿por qué demonios esto es DynamoDB?” y el ADR responde en treinta segundos — con el contexto, las restricciones y las alternativas honestamente consideradas. Los equipos con ADRs no relitigen nada; los equipos sin ellos discuten en círculos cada año. La plantilla — memorícela:

```
# 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.

Escriba un ADR por cada decisión significativa en cada capstone. Un portafolio de entrevistas que contiene ADRs reales es raro y devastadoramente efectivo.

El conjunto de diagramas — un sistema, tres niveles de zoom. La documentación de arquitectura real es un *conjunto* (el modelo C4 popularizó esta disciplina): el **diagrama de contexto** — el sistema como una sola caja, con sus usuarios y los sistemas externos que toca; la vista para ejecutivos/recién llegados, y la que presentará al liderazgo. El **diagrama de contenedores** — acerque el zoom: las piezas principales en ejecución (app web, API, base de datos, cola, caché) y cómo hablan; la vista cotidiana de ingeniería, aproximadamente sus dibujos de las Etapas 1–3. El **diagrama de despliegue** — dónde corre todo físicamente: región, AZs, VPC, subredes, grupos de escalado; la vista para revisiones, seguridad y operaciones. Un sistema, tres audiencias, tres diagramas — mantenidos al día, fechados, bajo control de versiones junto a los ADRs. Un diagrama que no puede encontrarse o en el que no se puede confiar es un diagrama que no existe.

Dirigir una revisión Well-Architected. Usted conoce los seis pilares (Etapa 2); aquí está la reunión en sí. **Antes:** elija el workload y el alcance, ponga al día el conjunto de diagramas, invite a las personas que *operan* la cosa (no solo a sus diseñadores) y fije el tono — esto es un chequeo de salud para beneficio del equipo, no una auditoría para el expediente de nadie.

Durante (medio día): recorra los pilares con los conjuntos de preguntas del marco (AWS los publica, con una herramienta Well-Architected gratuita en la consola que estructura todo el ejercicio — <https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html>); para cada respuesta, anote el hallazgo y *priorice sobre la marcha* — un puñado de asuntos de alto riesgo, no cien detallitos. Su tono de preguntas es la revisión: “help me understand what happens when the payment provider times out” (ayúdeme a entender qué pasa cuando el proveedor de pagos expira) abre puertas que “you don’t handle timeouts?” (¿no manejan timeouts?) cierra de golpe. **Después:** un informe corto — los hallazgos principales, cada uno con riesgo, esfuerzo y una recomendación — y los elementos de mejora aterrizan en el backlog real del equipo con dueños, o la revisión fue teatro. Práctica: dirija una revisión completa contra su propio diseño del Capstone 1; encontrar sus propias fallas de forma estructurada es la habilidad de este curso que más rápido se capitaliza.

Ejercicios: (1) Escriba retroactivamente ADRs para cinco decisiones grandes de sus diseños de diario de las Etapas 2–3 — encontrará al menos una que ya no puede justificar, y esa es la lección. (2) Produzca el conjunto completo de tres diagramas para ThaiTicket. (3) Dirija la revisión simulada de arriba y produzca el informe de hallazgos de una página.

Hito: un extraño (o una IA interpretándolo) puede tomar su paquete de ThaiTicket — tres diagramas + cinco ADRs — y responder correctamente “¿qué es esto, cómo corre y por qué está construido así?” sin usted en la sala. Ese paquete es el entregable del trabajo del arquitecto.

Módulo 13 (Meses 8–9): El oficio humano — presentar, costear, manejar la resistencia y mentorar

Presentar a ejecutivos vs. a ingenieros — un diseño, dos idiomas. Los ejecutivos compran resultados; los ingenieros compran mecanismos. A los ejecutivos: empiece con la decisión y el número (“Este diseño soporta la meta del millón de clientes a \$1.90 por pedido, y aquí están las dos decisiones que necesito de ustedes”); una diapositiva, el diagrama de *contexto*, los riesgos formulados como riesgos de negocio (ingresos, cumplimiento, reputación), y nunca diga “Kubernetes” cuando “la plataforma” bastará. Prepárese para las tres preguntas que los ejecutivos siempre hacen: *¿cuánto cuesta, qué podría salir mal, por qué no la opción más barata?* A los ingenieros: empiece con el problema y las restricciones *antes* de la solución (los ingenieros que sienten el problema aceptan la solución; los ingenieros a quienes se les entrega un veredicto cazan fallas por principio), muestre los diagramas de contenedores y despliegue, nombre usted mismo los trade-offs y las opciones rechazadas (la credibilidad viene de lo que usted admite), y deje espacio real para cambiar el diseño — las preguntas en la sala son revisión gratuita. El peor hábito de arquitecto es una sola presentación para ambas audiencias; el mejor hábito es escribir el resumen ejecutivo *primero*, porque si el diseño no sobrevive a la compresión en cinco frases, no está terminado.

Estimar costos para una propuesta. El método: descomponga el diseño en sus ~10 componentes con costo; cotice cada uno a la carga esperada en la calculadora de precios; **declare sus supuestos por escrito** (solicitudes/día, crecimiento de datos, egress — los supuestos son la estimación; cuando cambian, el número se mueve con la conciencia tranquila); aplique la estrategia de reservas a las partes estables; añada escenarios de crecimiento (hoy, 2×, 10× — los ejecutivos recuerdan el número de 10×); y presente como un *rango con impulsores* (“\$55–70k/mes, impulsado principalmente por el egress — aquí está la palanca”), nunca como un número único de falsa precisión. Incluya la burbuja de migración si la hay. Subestimar para ganar la aprobación es el pecado clásico del arquitecto junior; el proyecto lo recuerda.

Manejar la resistencia. Lo presionarán por el costo (“demasiado caro” — regrese al triángulo: “podemos recortar \$30k aceptando una sola AZ — aquí está la matemática de la caída; usted decide, y yo lo dejo por escrito” — hacer el intercambio explícito y *registrado* convierte la mayor parte de la resistencia en acuerdo o en riesgo aceptado informado, ambos aceptables); por gusto (“un ingeniero prefiere otro stack” — defienda con fuerza esa alternativa en voz alta, y luego llévela a criterios, no a preferencias: ajuste a los NFRs, habilidades del equipo, ecosistema, costo de salida — y cuando esté genuinamente parejo, deje ganar la preferencia del ingeniero que implementa, comprando compromiso a bajo precio); y por autoridad (“el amigo del CTO dice que usen X” — nunca pelee opinión contra opinión; pida los criterios, pase X por la misma evaluación pública que todo lo demás, deje que la matriz responda con cortesía). Y cuando usted se equivoque — diseñará algo que falla; a todo arquitecto le pasa — el movimiento es el post-mortem sin culpas aplicado a usted mismo, en público, con el ADR actualizado. Nada construye credibilidad de décadas más rápido; nada la destruye más rápido que defender un cadáver.

Mentorar ingenieros. La frase de las descripciones de puesto es “mentorar a los ingenieros que construyen según sus estándares”, y las formas de trabajo son: **horas de oficina de revisión de diseño** (una puerta permanente, para que la guía llegue antes del código, no

después); **revisar sus diseños con amabilidad** — preguntas antes que veredictos, y siempre el *porqué* detrás del estándar (un guardrail explicado recluta un aliado; un guardrail impuesto recluta una evasión); **delegar decisiones reales** con una red de seguridad (“usted es dueño del diseño de la caché; aquí están las restricciones; yo reviso, y respaldaré su decisión”); y **enseñar en público** — cada ADR, cada informe de revisión, cada sesión informal lo escala más allá de sus propias horas. La verdad cruda de la carrera: un arquitecto genio solitario toca techo; un arquitecto que ha convertido a cinco ingenieros en colegas capaces de diseñar dirige la práctica.

Ejercicios: (1) Presente ThaiTicket dos veces — una versión ejecutiva de 5 minutos y una versión de ingeniería de 20 minutos — grabe ambas y escuche la jerga filtrándose en la primera. (2) Produzca la propuesta de costos completa por escrito (supuestos, rango, impulsores, escenarios de 2×/10×). (3) Teatro de resistencia con una IA: tres rondas — ataque de costos de un CFO, preferencia de stack de un ingeniero senior, jugada de autoridad del amigo del CTO — anote en el diario qué funcionó. (4) Revise por escrito el diseño de un junior (una IA puede generar uno defectuoso), con preguntas primero, y haga que la IA califique su tono.

Hito: el doble reto: entregue el pitch ejecutivo y sobreviva veinte minutos de preguntas y respuestas mixtas hostiles-pero-justas (panel de IA: un CFO, un ingeniero escéptico) sin cruces de jerga, sin actitud defensiva y sin un solo trade-off no escrito.

Módulo 14 (Meses 9–12): Certificaciones y los tres capstones

La ruta de certificaciones, con líneas de tiempo honestas. Las certificaciones no lo hacen arquitecto — los trece módulos anteriores lo hacen — pero le consiguen entrevistas, y prepararse para ellas le cablea los servicios del proveedor en los dedos. La escalera, al ritmo de una hora al día: **AWS Certified Solutions Architect – Associate (SAA-C03)** — 2–3 meses; la credencial de arquitecto más solicitada en los anuncios de empleo, y después de las Etapas 1–3 gran parte le parecerá repaso con nombres de servicios adjuntos; tómela primero. **AWS Certified Solutions Architect – Professional (SAP-C02)** — 4–8 meses después de la Associate; basada en escenarios, genuinamente difícil, y la línea de currículum más fuerte de este campo; el material de migración y multi-cuenta de la Etapa 4 es la mitad de su temario. **Azure AZ-305 (Azure Solutions Architect Expert)** — añádala si su mercado es muy de Microsoft (la mayoría de los mercados empresariales son al menos bilingües); espere 2–3 meses con el conocimiento de AWS transfiriéndose con gran descuento (empiece en <https://learn.microsoft.com/en-us/credentials/certifications/azure-solutions-architect/>). **TOGAF Foundation** — opcional, 2–4 semanas; algunas grandes empresas la piden; certifica *método* y vocabulario de arquitectura empresarial más que habilidad cloud. Secuencia para este curso: estudio de la SAA junto a los Meses 7–8, examen alrededor del Mes 8; luego o la SA Pro (ruta técnica profunda) o la AZ-305 (ruta de amplitud) hasta el Mes 12, con la SA Pro completada para el Mes 14–16 al ritmo honesto.

Los tres capstones. Cada uno es un paquete completo de calidad de portafolio: conjunto de diagramas de tres niveles, 5+ ADRs, estimación de costos con supuestos, y una revisión Well-Architected autoejecutada con hallazgos. Dedique 3–4 semanas a cada uno. Estos tres artefactos, más su diario, *son* su evidencia de entrevista de “diseño de soluciones de extremo a extremo”.

Capstone 1 — el greenfield: una plataforma tailandesa de e-commerce para 1M de usuarios. Encargo: marketplace, 1M de usuarios registrados, 50k concurrentes en el pico de una venta flash, mobile-first, integración de pagos estilo PromptPay, conforme con la PDPA, 99.9% de disponibilidad, consciente del presupuesto de etapa semilla. Debe incluir: elección de patrón con ruta de evolución, estrategia de caché, conversación de RTO/RPO escrita como diálogo, estimación de unit economics, mapa de flujos de datos personales.

Capstone 2 — el brownfield: migrar una empresa on-prem de 40 servidores. Encargo: una firma tailandesa de logística; 40 servidores (ERP sobre Oracle, app de gestión de almacén de 12 años, archivos compartidos, AD, varias apps departamentales — genere el inventario completo con una IA y congélelo); el contrato del centro de datos expira en 14 meses; la junta quiere salir. Debe incluir: tabla completa de inventario con las 7 R, plan de tres olas con razonamiento de dependencias, diseño de landing zone, línea de tiempo de costos de la burbuja de migración, y el memorando al CFO.

Capstone 3 — el difícil: DR multi-región para una fintech. Encargo: una empresa de pagos regulada a $RTO \leq 15$ minutos / $RPO \approx 0$ para el libro mayor, con restricciones de residencia de datos sobre los datos de clientes y una prueba anual de failover presenciada por el regulador. Debe incluir: análisis activo-pasivo vs. activo-activo con costos, diseño de replicación con el mapa de residencia, el runbook de failover y el plan de simulacros.

La rúbrica de autoevaluación (aplíquela a cada capstone, con honestidad, en su diario):

Dimensión	1 — Todavía no	3 — Sólido	5 — Contraten a esta persona
Requisitos	Saltó a la solución	NFRs declarados y trazados hasta las decisiones de diseño	Triángulo de trade-offs posicionado, conflictos presentados a “el negocio”, decisiones extraídas
Calidad del diseño	Quedan SPOFs; patrón mal ajustado	Patrones correctos, multi-AZ, fallas manejadas	Razonamiento de cuándo-NO mostrado; ruta de evolución; el diseño más simple que cumple los requisitos
Seis pilares	Pilares ignorados	Cada pilar visiblemente abordado	La revisión autoejecutada encontró fallas reales — y el diseño fue revisado en respuesta
Costos	Sin números	Estimación detallada, supuestos declarados	Rango con impulsores, unit economics, escenario de 10x, estrategia de

Dimensión	1 — Todavía no	3 — Sólido	5 — Contraten a esta persona
Documentación	Solo un diagrama	Conjunto de tres niveles + ADRs, notación correcta	reservas Un extraño podría responder “qué/cómo/por qué” solo con el paquete
Comunicación	Un artefacto para todas las audiencias	Existen versiones para ejecutivos y para ingenieros	Resumen de cinco frases que sobrevive; resistencia pre-respondida por escrito

Califique cada dimensión con 4+ en los tres capstones — revisando hasta lograrlo — y habrá completado el curso.

Diez preguntas de entrevista de arquitecto (con respuestas sólidas)

Practíquelas en voz alta; la fuerza está en la *estructura* de cada respuesta, que ahora es suya.

1. “Design a URL shortener / ticketing site / photo app for a million users.” (Diseñe un acortador de URLs / un sitio de boletos / una app de fotos para un millón de usuarios.)

Forma sólida: los requisitos primero, en voz alta (“¿Las lecturas dominan a las escrituras? ¿Meta de disponibilidad? ¿Presupuesto?”) — posición en el triángulo — esqueleto de tres capas o serverless-first con multi-AZ, caché, CDN — nombre los trade-offs sin que se lo pidan — termine con el orden de magnitud del costo y la ruta de evolución a 10×. Los entrevistadores aprueban por las *preguntas que usted hace*, no por las cajas que dibuja.

2. “When would you choose NoSQL over a relational database?” (¿Cuándo elegiría NoSQL sobre una base de datos relacional?) “Mi predeterminado es SQL — garantías de corrección y habilidades universales — hasta que una razón con nombre lo anule: escala horizontal extrema, esquema flexible o acceso clave-valor de un dígito de milisegundos. Entonces elijo la variante NoSQL que se ajuste al patrón de acceso, y escribo el ADR incluyendo lo que cedemos: los joins, y algo de semántica de consistencia.”

3. “Explain RTO and RPO, and how they drive design.” (Explique RTO y RPO, y cómo dirigen el diseño.) Defina ambos con precisión — “son decisiones de negocio con etiquetas de precio exponenciales” — la escalera: respaldos nocturnos — replicación continua — standby tibio — activo-activo, con el costo subiendo en cada peldaño — “mi trabajo es hacer que el negocio elija a sabiendas, y luego diseñar exactamente para el número — y ensayarlo, porque un failover no probado es una esperanza.”

4. “Monolith or microservices?” (¿Monolito o microservicios?) El fallo del Módulo 8, textual: empiece con un monolito modular; extraiga cuando llegue de verdad el dolor de escalado de equipo o de carga; los microservicios son una herramienta de escalado de equipos

que cobra un impuesto de sistemas distribuidos. Puntos extra por “prefiero operar un buen monolito que un mal sistema distribuido.”

5. “How do you handle a large cloud bill / cost optimization?” (¿Cómo maneja una factura cloud grande / la optimización de costos?) “Visibilidad primero — etiquetado y showback; luego la cosecha en orden de esfuerzo: matar los recursos muertos, rightsizing a partir de mediciones, niveles de almacenamiento, dormir los entornos no productivos, y luego reservar la línea base estable. Y reporto unit economics, no totales — una factura creciente con costo por pedido decreciente es un éxito, no un problema.”

6. “How would you migrate a legacy on-prem application?” (¿Cómo migraría una aplicación heredada on-prem?) “Evaluar antes de mover — inventario, dependencias y una R por workload de las 7 R; construir la landing zone antes de la primera ola; olas de lo fácil primero con cutovers ensayados y planes de rollback; y para la base de datos joya de la corona, cutover basado en replicación dimensionado a la caída que el negocio firmó. Además: la advertencia honesta de la burbuja de migración por adelantado.”

7. “How do you secure a cloud architecture?” (¿Cómo asegura una arquitectura cloud?) Recorra las capas: identidad (mínimo privilegio, MFA, sin root diario) → red (subredes privadas, segmentación, superficie de ataque mínima) → datos (cifrado en reposo/en tránsito, clasificación, residencia) → detección (logs de auditoría, alertas) → “y desde el diseño, no atornillada después — el control de seguridad más barato es la decisión de arquitectura que elimina el riesgo por completo, como nunca tocar datos de tarjetas en bruto.”

8. “Tell me about a design decision you got wrong.” (Cuénteme sobre una decisión de diseño en la que se equivocó.) Están probando el ego, no la historia. Forma: ejemplo real (de capstone) → lo que usted creía → lo que dijo la realidad → el post-mortem, el ADR actualizado, el patrón que ahora revisa. Un arquitecto que no puede producir esta respuesta es un arquitecto que no ha sido revisado.

9. “How do you explain a complex technical decision to a non-technical executive?” (¿Cómo explica una decisión técnica compleja a un ejecutivo no técnico?) “La decisión y el número de negocio primero, el mecanismo solo si lo piden; diagrama de contexto, no de contenedores; los riesgos en términos de ingresos-cumplimiento-reputación; y llevo las dos decisiones que necesito de ellos, escritas como opciones con etiquetas de precio — los ejecutivos deciden entre opciones; no aprueban misterios.”

10. “An engineer strongly disagrees with your design. What do you do?” (Un ingeniero está en fuerte desacuerdo con su diseño. ¿Qué hace usted?) “Primero definiendo su postura con fuerza en voz alta — puede tener razón, y la revisión que cambia mi diseño es la revisión funcionando. Si está genuinamente parejo, deciden los criterios (ajuste a los NFRs, habilidades del equipo, costo de salida), no la antigüedad — y dejo que la preferencia del implementador gane los empates, porque el compromiso es barato a ese precio. En cualquier caso, la decisión y la opción rechazada van al ADR, para que nunca lo discutamos dos veces.”

El plan de estudios en una página

Cuándo	Módulo	Enfoque	Prueba externa
Semanas 1–2	1	Repaso cloud; triángulo de trade-offs; NFRs	Diario de Diseño iniciado
Semanas 3–4	2	Cómputo/almacenamiento/base de datos/red como decisiones	—
Semanas 5–6	3	Leer y dibujar diagramas; fluidez en 3 capas	Hito de pizarra en 15 minutos
Semanas 7–8	4	Confiabilidad: multi-AZ, auto-scaling, RTO/RPO	—
Semanas 9–10	5	Seguridad: mínimo privilegio, zero trust, segmentación	—
Semanas 11–12	6	Rendimiento y costos: caché, CDN, reservas, unit economics	Diseño cotizado en la calculadora
Semanas 13–14	7	Excelencia operativa y sostenibilidad; drill de los seis pilares	Revisión simulada de los seis pilares
Semanas 15–17	8	Patrones: monolito/microservicios, orientado a eventos, serverless	—
Semanas 18–20	9	Patrones: data lake/warehouse, multi-región, híbrido	Desafío del catálogo
Semanas 21–23	10	Migración: 7 Rs, olas, landing zones	Migración en papel
Semanas 24–26	11	Restricciones: PDPA/GDPR, sistemas heredados, economía del lock-in	Desafío de restricciones
Meses 7–8	12	ADRs, conjuntos de diagramas, dirigir revisiones Well-Architected	Paquete ThaiTicket; examen SAA-C03 (2–3 meses de preparación)

Cuándo	Módulo	Enfoque	Prueba externa
Meses 8–9	13	Presentar, costear propuestas, resistencia, mentoría	Doble reto grabado
Meses 9–12	14	Capstones 1–3	Portafolio completo; SA Pro (4–8 meses) o AZ-305 en marcha

Unas palabras de cierre de su profesor. A las veintiséis semanas, usted podía diseñar; a los doce meses, puede *ejercer* — hay una diferencia, y es la diferencia que este curso fue construido para cerrar. Los hábitos son ahora la carrera: dibuje antes de discutir, escriba el ADR el día que decida, cotice lo que proponga, nombre el trade-off antes de que alguien pregunte, y haga crecer a los ingenieros a su alrededor hasta que sus estándares sobrevivan a su presencia en la sala. La arquitectura es el raro trabajo técnico que *mejora* a medida que usted envejece en él, porque su materia prima es el juicio, y el juicio se capitaliza. Siga con el diario. Dentro de cuarenta diseños, usted no necesitará este curso — lo estará corrigiendo.

Fuentes

Descripciones de puesto y definiciones de rol usadas para el mapeo de requisitos (recuperadas en agosto de 2026): la [plantilla de descripción de puesto de Cloud Architect de KORE1](#) · la [descripción de puesto de Cloud Architect de 4 Corner Resources](#) · las [responsabilidades del Solution Architect de Microsoft en el Azure Well-Architected Framework](#) · la [publicación de Pre-Sales Solutions Architect – AWS de Arpio](#) · la [publicación de Solution Architect – Enterprise de Intel \(vía Built In\)](#) · la [publicación de Cloud Domain Architect de Halliburton](#). Estimaciones de tiempo de preparación de certificaciones: la [encuesta de tiempo de estudio de la SAA-C03 de CBT Nuggets](#) y la [guía de preparación de la SAP-C02 de Whizlabs](#); definiciones de certificaciones de [Microsoft Learn \(AZ-305\)](#) y el [AWS Well-Architected Framework](#). Un volumen complementario de *El Curso del Líder Cloud (The Cloud Leader Course)* en la serie B4LCILC.

LECCIONES NUEVAS — EDICIÓN DE AGOSTO DE 2026

En agosto de 2026 se añadieron lecciones nuevas a los tres cursos interactivos de b4lcilc.org — los cursos de Líder Cloud, Ingeniero Cloud y Arquitecto Cloud. Este apéndice pone al día la edición de lectura con condensaciones fieles de ese material nuevo, en la misma voz que el resto de este libro.

Unas palabras sobre lo que usted tiene en las manos. Estas páginas son la edición de lectura: las ideas, las metáforas y las frases que vale la pena decir en voz alta. Las versiones interactivas completas en línea traen bastante más — cada lección nueva incluye quizzes que comprueban su comprensión antes de dejarlo avanzar, laboratorios prácticos (incluida una terminal de práctica que se comporta como la real pero no puede costar dinero), videos, drills de vocabulario y — para el material del capstone — ejercicios de diseño guiados con listas de requisitos y

respuestas modelo resueltas. Lea estas condensaciones para meterse en la cabeza la forma de cada idea; después, cuando pueda, haga las versiones interactivas para que el conocimiento se quede.

Las adiciones caen en tres grupos. Para el curso de **Líder Cloud**: una introducción amable a la terminal, la ley de privacidad propia de Tailandia, una guía del líder para propuestas de IA y facturas de IA, y un ejercicio capstone que ensambla el curso entero en una sola página. Para el curso de **Ingeniero Cloud**: la computación serverless — el patrón que convierte los pequeños scripts operativos en funciones que no cuestan nada ejecutar. Para el curso de **Arquitecto Cloud**: la aritmética de las promesas de disponibilidad, el patrón de Generación Aumentada por Recuperación (Retrieval-Augmented Generation) por el que los clientes ahora preguntan cada semana, y un ejercicio completo de costeo — ponerle precio a un diseño antes de que alguien lo construya.

Para el curso de Arquitecto Cloud

La aritmética de los nueves — SLAs, SLOs y presupuestos de error

La disponibilidad se cotiza en nueves, y lo primero que un arquitecto hace con un nueve es convertirlo en minutos — porque los minutos son lo que el negocio puede sentir. Un mes de 30 días contiene 43,200 minutos:

Disponibilidad	Caída permitida por mes
99.9% (“tres nueves”)	unos 43 minutos
99.99% (“cuatro nueves”)	unos 4.3 minutos
99.999% (“cinco nueves”)	unos 26 segundos

Cinco nueves — el número que a los ejecutivos les gusta decir en voz alta — está más allá de cualquier recuperación operada por humanos; el failover debe ser automático o la promesa es ficción. Cada nueve adicional cuesta unas diez veces más de ingeniería; convertir a minutos es como usted logra que un interesado elija a sabiendas.

Tres siglas cargan la conversación. Un **Service-Level Indicator (SLI, indicador de nivel de servicio)** es una medición. Un **Service-Level Objective (SLO, objetivo de nivel de servicio)** es la meta interna sobre esa medición. Un **Service-Level Agreement (SLA, acuerdo de nivel de servicio)** es el contrato externo, con dinero atado a un incumplimiento. La disciplina: el SLO siempre es más estricto que el SLA — usted apunta por dentro de la línea que le pagan por sostener — y el SLI mide lo que los usuarios experimentan: un servidor que está encendido mientras devuelve errores está caído en todos los sentidos que importan.

El presupuesto de error hace operativos a los SLO: la brecha entre la perfección y el objetivo, tratada como un recurso que se gasta. Un SLO de 99.9% otorga 43 minutos de falla al mes; los despliegues y los experimentos gastan de ahí; cuando se agota, el equipo deja de lanzar y trabaja en confiabilidad — poniendo fin a la discusión más vieja del software, lanzar contra estabilidad, al reemplazar la opinión con aritmética.

Ahora la trampa: la disponibilidad se multiplica a lo largo de una cadena. Tres servicios en serie, cada uno honestamente al 99.9%, se componen en $0.999 \times 0.999 \times 0.999 \approx 99.7\%$ — unos 130 minutos al mes, tres veces lo que cualquier servicio individual admite. La redundancia funciona al revés: dos instancias independientes al 99.9% en paralelo fallan juntas solo el $0.1\% \times 0.1\%$ del tiempo — por eso el multi-zona es tan buena inversión. La frase de nivel entrevista: *usted no puede prometer más disponibilidad que el producto de sus dependencias sincronas, así que o acorta la cadena o baja la promesa.*

Arquitectura en la era de la IA — RAG y sus parientes

Para 2026 el catálogo de patrones tiene una entrada nueva por la que los clientes preguntan cada semana: qué es, cuándo se gana su lugar y cuándo lo dejará en ridículo. Una API de **Large Language Model (LLM, modelo de lenguaje grande)** — Amazon Bedrock, Azure OpenAI Service — es, arquitectónicamente, *una dependencia cara, lenta y sin estado*. Se factura por **token** — un fragmento de palabra de unos cuatro caracteres. Responde en segundos, no en milisegundos, no recuerda nada entre llamadas, y de vez en cuando se equivoca con toda confianza — un componente con modos de falla inusuales, que merece los mismos timeouts, reintentos, circuit breakers y alarmas de presupuesto que cualquier dependencia, porque un bucle desbocado de llamadas quema dinero real por hora.

El modelo no conoce sus documentos; la respuesta estándar es la **Generación Aumentada por Recuperación (RAG, Retrieval-Augmented Generation)**: en lugar de enseñarle al modelo sus datos, usted *busca* los hechos relevantes en el momento de la pregunta. Los documentos se cortan en fragmentos, cada uno se convierte en un **embedding** — una lista de números que posiciona su significado en el espacio — y se guarda en una **base de datos vectorial** (pgvector en AWS, AI Search en Azure), que devuelve los fragmentos más cercanos. Esos fragmentos se pegan dentro del prompt: el modelo no está consultando sus datos; está resumiendo los extractos que usted fue a buscarle.

En la pizarra, RAG son dos tuberías: una **ruta de ingesta** que corre por adelantado (documentos fragmentados, convertidos a embeddings, almacenados) y una **ruta de consulta** que corre por cada petición — dos patrones que ya existen en el catálogo, y por eso los arquitectos lo aprenden en una tarde.

Los trade-offs son los tres de siempre, afilados: costo (cada consulta cuesta dinero real — cachee las repetidas, enrute las preguntas fáciles a un modelo más barato), latencia (segundos, no milisegundos — transmita la respuesta en streaming) y privacidad (sus documentos salen de su frontera; si la base de datos vectorial ignora el control de acceso a nivel de documento, el chatbot le resumirá alegremente la hoja de cálculo de salarios a cualquiera que pregunte). Sepa cuándo *no* construirlo: muchas peticiones de “necesitamos un chatbot” son una caja de búsqueda vestida a la moda, y las respuestas exactas y auditables — sumas, saldos, libros contables — son el trabajo equivocado para un sistema que acierta el 97% de las veces. La frase de nivel entrevista: *RAG es un problema de búsqueda engrapado a una API de resumen.*

Costear un diseño antes de que alguien lo construya

El módulo de los Pilares de Rendimiento y Costo del curso ahora cierra con un ejercicio de diseño completo: estimación bajo fuego. El encargo: la dirección quiere una estimación de costo mensual para un producto de gestión documental antes de aprobar la construcción. La

carga de trabajo — 200,000 usuarios registrados, 40,000 activos al día, cada uno haciendo unas 25 peticiones a la API al día y subiendo en promedio 2 megabytes de documentos al mes, conservados durante siete años. El diseño: una capa de API con balanceo de carga en dos zonas, una base de datos PostgreSQL gestionada multi-zona, almacenamiento de objetos, una exportación nocturna de 50 gigabytes a un socio, y entornos separados de desarrollo y staging. La tarea: una estimación mensual de una página en números redondos, con los supuestos puestos por escrito antes de que aparezca cualquier precio, la aritmética de capacidad mostrada (usuarios $\rightarrow$ peticiones por segundo $\rightarrow$ número de instancias), y escenarios de hoy / $2\times$ / $10\times$.

La lista de verificación del ejercicio enseña la disciplina: cotice el cómputo desde la carga pico, no la promedio; cuente el standby de la base de datos; modele el almacenamiento como crecimiento a siete años con niveles de ciclo de vida; calcule el egress en lugar de adivinarlo — la línea que la mayoría de las estimaciones olvida y la que más a menudo domina; recuerde los gateways NAT (Network Address Translation, traducción de direcciones de red), los entornos no productivos, los respaldos, el monitoreo y el plan de soporte; aplique los descuentos por compromiso solo a la línea base estable; y termine con un rango con impulsores nombrados más una comprobación de cordura de economía unitaria — el costo por usuario activo por mes. Un único número preciso está equivocado el mismo día en que se pronuncia.