

หลักสูตรสถาปนิกคลาวด์ (The Cloud Architect Course)

จากความรู้ศูนย์สู่การออกแบบระบบคลาวด์ — หลักสูตรในชุด B4LCILC

13 สิงหาคม 2569

หลักสูตรสถาปนิกคลาวด์ (The Cloud Architect Course)

จากความรู้ศูนย์ สู่การออกแบบระบบคลาวด์ที่บริษัทกล้าเอาธุรกิจมาเดิมพัน

หลักสูตรนี้ทำงานอย่างไร

คุณไม่ได้กำลังฝึกเพื่อ *ปฏิบัติ* าระบบคลาวด์ คุณกำลังฝึกเพื่อ *ออกแบบ* มัน: รับโจทย์ธุรกิจที่ยุ่งเหยิง (“เราต้องขายให้ลูกค้าหนึ่งล้านคนโดยไม่ทำอเดอร์ด์หายแม้แต่รายการเดียว”) แปลงมันเป็นแผนทางเทคนิคที่วาดออกมาแล้ว คิดต้นทุนแล้ว และปกป้องได้ด้วยเหตุผล จากนั้นโน้มน้าวทั้งวิศวกรที่จะสร้างมันและผู้บริหารที่จะจ่ายเงิน นั่นคืองานของ **cloud architect** (สถาปนิกคลาวด์ — ในประกาศงานยังใช้ชื่อ *solutions architect*, *cloud solutions architect* หรือ *cloud domain architect*) และมันเป็นงานฝีมือที่เรียนรู้ได้ โดยมีหลักสูตรที่ชัดเจน — เล่มนี้แหละ

หลักสูตรแบ่งเป็นห้าช่วง **ช่วงที่ 1 (สัปดาห์ 1–6): องค์ประกอบพื้นฐาน ในฐานะการตัดสินใจ** ทุกส่วนประกอบของคลาวด์ ถูกสอนใหม่ไม่ใช่ในฐานะข้อเท็จจริง แต่ในฐานะ *ทางเลือกที่มี trade-off* (การแลกเปลี่ยนได้อย่างเสียอย่าง) — เพราะหน่วยของงานของสถาปนิกคือการตัดสินใจ **ช่วงที่ 2 (สัปดาห์ 7–14): อาณาเขตของการออกแบบ** หกเสาหลักของ Well-Architected Framework — reliability, security, performance, cost, operations, sustainability — สอนลึกทีละเสา พร้อมแพตเทิร์นมาตรฐานและงานออกแบบตัวอย่าง **ช่วงที่ 3 (สัปดาห์ 15–20): แคล์ตตาล็อกแพตเทิร์น** สถาปัตยกรรมราวหนึ่งโหลที่แก้ปัญหาจริงได้ 95% และ — สำคัญกว่านั้น — เมื่อไหร่ที่ *ไม่ควร* ใช้แต่ละตัว **ช่วงที่ 4 (สัปดาห์ 21–26): การย้ายระบบและโลกจริง** การพาบริษัทที่มีอยู่แล้วขึ้นคลาวด์ และข้อจำกัดต่าง ๆ (กฎหมาย ระบบเก่า การผูกติดผู้ให้บริการ) ที่ทำให้สถาปัตยกรรมจริงยากกว่าสถาปัตยกรรมบนไวต์บอร์ด **ช่วงที่ 5 (เดือน 7–12):**

งานฝีมือ เอกสาร แผนภาพ การรีวิว การนำเสนอ และใบรับรองที่ทำให้คุณมี**น**ายจ้างอยากได้ในฐานะสถาปนิก ปิดท้ายด้วยโปรเจกต์ออกแบบระดับพอร์ตโฟลิโอสามชิ้น

หลักสูตรนี้เหมาะกับใคร มันตั้งสมมติฐานว่าคุณไม่มีความรู้มาก่อน: ทุกสิ่งที่มีนพิงพาถูกสอนซ้ำสั้น ๆ ก่อนถูกใช้งาน ถ้าคุณเรียนจบ **The Cloud Engineer Course** มาแล้ว (หรือทำงานคลาวด์ภาคปฏิบัติอยู่ทุกวันนี้) ช่วงที่ 1 จะคุ้นเคย — *อ่านผ่าน ๆ ได้ แต่ทำแบบฝึกหัดของมันเสมอ* เพราะมันจัดกรอบใหม่ให้สิ่งที่คุณรู้ในฐานะข้อเท็จจริง กลายเป็นสิ่งที่你必须ซึ่งน้ำหนักในฐานะการตัดสินใจ และการจัดกรอบใหม่นั้นแหละคือการเปลี่ยนผ่านทั้งหมดจากวิศวกรสู่สถาปนิก

กติกาตลอดหลักสูตร: เรียน**วันละหนึ่งชั่วโมง หกวันต่อสัปดาห์** — ความสม่ำเสมอชนะความหักโหม ทุกโมดูลจบด้วย **ตารางคำศัพท์** (คำที่คุณต้องเป็นเจ้าของให้ได้), **แบบฝึกหัด** (งานออกแบบ ไม่ใช่การอ่าน — สถาปัตยกรรมเรียนรู้ด้วยการวาดและการตัดสินใจ) และ **Milestone** (จุดหมาย — หลักฐานว่าคุณพร้อมไปต่อ; อย่าข้าม milestone ที่ยังไม่ผ่าน) และตั้งแต่สัปดาห์ที่ 1 ให้จด **Design Journal** (สมุดบันทึกการออกแบบ): ทุกสถาปัตยกรรมที่คุณวาด ทุก trade-off ที่คุณชี้ ทุกคำทำนายเรื่องต้นทุนหรือความล้มเหลวที่คุณทำ อีกสิบเดือนจากนี้มันจะกลายเป็นพอร์ตโฟลิโอสัมภาษณ์ของคุณ; ไม่มีอะไรทำให้กรรมการจ้างงานประทับใจเท่าสมุดบันทึกลงวันที่ที่มีสี่สิบงานออกแบบพร้อมโน้ตข้อสงสัยว่าคุณพลาดตรงไหน

สถาปนิกคลาวด์ทำอะไรกันเน

นึกถึงสถาปนิกอาคาร เธอไม่ได้ก่ออิฐเอง — แต่เธอต้องรู้เป้าว่าอิฐทำอะไรได้และไม่ได้ ราคาเท่าไร และอาคารพังอย่างไร เธอฟังลูกค้า (“บ้านครอบครัว แดดดี ภายใต้งบนี้ บนที่ดินรูปทรงลำบากแปลงนี้”) แล้วแปลงความอยากและข้อจำกัดให้เป็น**แบบและสเปก**ที่แม่นยำพอให้ช่างก่อสร้างสร้างตามได้ และลูกค้าเซ็นอนุมัติได้ จากนั้นเธออยู่ต่อตลอดการก่อสร้าง คอยตอบคำถามและปรับแผนเมื่อพื้นดินกลายเป็นนุ่มกว่าที่ผลสำรวจบอก

สลัอิฐเป็น server แล้วนั่นคืองานนี้ สถาปนิกคลาวด์ใช้สัปดาห์ไปกับส่วนผสมของ: **การฟัง**ผู้มีส่วนได้ส่วนเสียฝั่งธุรกิจและสกัดความต้องการจริงจากความปรารถนาคลุมเครือ; **การออกแบบ** — เลือกส่วนประกอบ วาดแผนภาพ จัดการตัดสินใจและเหตุผล; **การรีวิวงานออกแบบ**ของคนอื่นเทียบกับหกเสาหลัก; **การประมาณ**ว่างานออกแบบหนึ่งจะมีค่าใช้จ่ายต่อเดือนเท่าไร และปกป้องตัวเลขนั้นต่อหน้าฝ่ายการเงิน; **การวางแผนย้าย**ระบบเก่าขึ้นคลาวด์; **การนำเสนอ** — งานออกแบบเดียวกันที่อธิบายแบบหนึ่งให้วิศวกร และอีกแบบที่ต่างสิ้นเชิงให้ผู้บริหาร; และ**การเป็นพี่เลี้ยง**วิศวกร เพื่อให้มาตรฐานอยู่รอดเมื่อปะทะกับเดดไลน์ ในหลายทีมยังมี **pre-sales**: นั่งข้างพนักงานขายต่อหน้าลูกค้ามุงหวัง แล้วสเก็ตช์โซลูชันที่ปิดดีลได้

สิ่งที่สถาปนิก**ไม่ใช่**: คนเขียนโค้ดเก่งสุดในห้อง (มักไม่ใช่), คนพิมพ์เยอะสุด (พิมพ์น้อยสุด) หรืออัจฉริยะสันโดษที่ยื่นพิมพ์เขียวลงมา (วิธีเร็วสุดที่จะถูกเมิน) ผลผลิตจริงของสถาปนิกคือ **การตัดสินใจที่ดี ที่ถูกเขียนบันทึกไว้ และคนอื่นเต็มใจสร้างตาม**

ประกาศรับสมัครงานที่คุณกำลังฝึกเพื่อให้ตรงตาม

ในเดือนสิงหาคม 2569 เราได้รวบรวมประกาศงานและเทมเพลตตำแหน่ง Cloud/Solutions Architect จริง ๆ — เทมเพลต cloud architect ของ recruiter (KORE1), job description ของบริษัทจัดหางาน (4 Corner Resources), นิยามบทบาทสถาปนิกของ Microsoft เองใน Azure Well-Architected Framework, ประกาศ Solutions Architect สาย pre-sales (Arpio, AWS disaster-recovery), ประกาศ Solution Architect ระดับองค์กร (Intel ผ่าน Built In) และประกาศ Cloud Domain Architect ระดับองค์กร (Halliburton, เน้น Azure) เมื่อลอกกลืนอายุบริษัทออก ข้อกำหนดสิบสองข้อชุดเดิมาก็ปรากฏซ้ำแล้วซ้ำเล่า หลักสูตรนี้สร้างขึ้นตามรายการนั้น — นี่คือจุดที่แต่ละข้อถูกสอนปะะ ๆ:

สิ่งที่ประกาศรับสมัครงานขอ (ถ้อยคำของพวกเขา ถอดความ)	หลักสูตรนี้สอนตรงไหน
“Design scalable and secure cloud architectures tailored to business and technical requirements” (ออกแบบสถาปัตยกรรมคลาวด์ที่ขยายได้และปลอดภัย ตอบโจทย์ธุรกิจและเทคโนโลยี) — การออกแบบโซลูชันครบวงจร	ช่วงที่ 1–3, งาน capstone ในช่วงที่ 5
“Deep platform expertise” (ความเชี่ยวชาญแพลตฟอร์มเชิงลึก) ใน AWS และ/หรือ Azure — compute, storage, networking, IAM, โครงสร้างบัญชี	ช่วงที่ 1 + เส้นทางใบรับรองในช่วงที่ 5
“Run Well-Architected reviews” (ดำเนินการรีวิวด้าน Well-Architected) เทียบหกเสาหลัก	ช่วงที่ 2 (เสาหลัก), ช่วงที่ 5 โมดูล 12 (การรันรีวิว)
“Lead migration/modernization initiatives — which workloads move as-is, get rearchitected, or retire” (นำโครงการย้าย/ปรับปรุงระบบ — workload ไหนย้ายตามสภาพ ออกแบบใหม่ หรือปลดระวาง)	ช่วงที่ 4 โมดูล 10 (7 Rs, การวางแผนเป็นระลอก)
“Design landing zones, account structure, guardrails, and reference patterns teams deploy within” (ออกแบบ landing zone, โครงสร้างบัญชี, guardrail และแพตเทิร์นอ้างอิงที่ทีมใช้ deploy ภายใน)	ช่วงที่ 4 โมดูล 10
“Integrate security and compliance into the design rather than bolting it on” (ผสานความ	ช่วงที่ 2 โมดูล 5, ช่วงที่ 4 โมดูล 11

สิ่งที่ประกาศรับสมัครงานขอ (ถ้อยคำของพวกเขา ถอดความ)

หลักสูตรนี้สอนตรงไหน

ปลอดภัยและการปฏิบัติตามข้อกำหนดเข้าไปในงาน
ออกแบบ ไม่ใช่มาแปะที่หลัง) — zero trust, identity

High availability และ disaster recovery —
“RTO/RPO gaps, downtime costs, ransomware
exposure” (ช่องว่าง RTO/RPO, ต้นทุนดาวน์ไทม์,
ความเสี่ยง ransomware)

ช่วงที่ 2 โมดูล 4, ช่วงที่ 3 โมดูล 9 (multi-region)

“Own the cloud cost model — tagging, showback,
reserved capacity, rightsizing” (เป็นเจ้าของโมเดล
ต้นทุนคลาวด์ — การติด tag, showback, reserved
capacity, rightsizing); ประมาณต้นทุนโซลูชัน

ช่วงที่ 2 โมดูล 6, ช่วงที่ 5 โมดูล 13 (การคิดต้นทุน
ข้อเสนอ)

การออกแบบสถาปัตยกรรมเครือข่าย — VPC, hub-
and-spoke, การเชื่อมต่อ hybrid

ช่วงที่ 1 โมดูล 2, ช่วงที่ 3 โมดูล 9, ช่วงที่ 4

“Create architectural documentation, diagrams,
and standards”; “maintain Architecture Decision
Records” (จัดทำเอกสารสถาปัตยกรรม แผนภาพ
และมาตรฐาน; ดูแล Architecture Decision
Records)

ช่วงที่ 1 โมดูล 3 (แผนภาพ), ช่วงที่ 5 โมดูล 12
(ADR, ชุดแผนภาพ)

การสื่อสารกับผู้มีส่วนได้ส่วนเสีย — “present
technical concepts to C-level and technical
audiences”; “defend architectural decisions to
security, finance, and engineering” (นำเสนอ
แนวคิดเทคนิคต่อผู้บริหารระดับ C และผู้ฟังสาย
เทคนิค; ปกป้องการตัดสินใจเชิงสถาปัตยกรรมต่อ
ฝ่ายความปลอดภัย การเงิน และวิศวกรรม)

ช่วงที่ 5 โมดูล 13

“Mentor the cloud and platform engineers who
build against your standards” (เป็นพี่เลี้ยงวิศวกร
คลาวด์และแพลตฟอร์มที่สร้างตามมาตรฐานของ
คุณ); งานสนับสนุน pre-sales — เดโม, POC, RFP

ช่วงที่ 5 โมดูล 13

ใบรับรอง ตรวจสอบกับตลาดเดียวกัน: บันไดมาตรฐานคือ **AWS Certified Solutions Architect – Associate (SAA-C03)** — ใบรับรองเดียวที่ถูกขอมากที่สุด เตรียมตัวโดยทั่วไป **2–3 เดือน** — ตามด้วย **AWS Certified Solutions Architect – Professional (SAP-C02)** ซึ่งใช้เพิ่มอีกราว **4–8 เดือน**;

ช่วงที่ 1 — องค์ประกอบพื้นฐาน ในฐานะการตัดสินใจ (สัปดาห์ 1–6)

ผู้จบหลักสูตร Cloud Engineer: อ่านคำอธิบายผ่าน ๆ ได้ แต่ทำแบบฝึกหัดทุกข้อ ข้อเท็จจริงชุดเดิม แต่คำถามชุดใหม่

โมดูล 1 (สัปดาห์ 1–2): คลาวด์คืออะไร และสถาปนิกคิดอย่างไร

คลาวด์ในหนึ่งย่อหน้า (บทบทวนสำหรับผู้เริ่มจากศูนย์) คลาวด์คือ คอมพิวเตอร์ของคนอื่น เข้าเป็นรายชั่วโมง บริหารจัดการด้วยซอฟต์แวร์ AWS, Microsoft Azure และ Google Cloud รันโกดิ่งเต็มไปด้วย server (**data center** — ศูนย์ข้อมูล) จัดกลุ่มเป็น **Region** (รีเจียน) ตามภูมิศาสตร์ (เช่น “Asia Pacific (Bangkok)”); แต่ละ region มี **Availability Zone (AZ)** ที่แยกอิสระหลายแห่ง — อาคารแยกกันที่มีไฟฟ้าของตัวเอง ใกล้เคียงให้เชื่อมต่อเร็ว ใกล้เคียงที่น้ำท่วมหรือไฟไหม้ครั้งเดียวจะไม่กวาดไปสองแห่ง คุณเช่าส่วนแบ่งของทั้งหมดนี้เป็นรายวินาที: เครื่องเปล่า (**IaaS** — คุณดูแลทุกอย่างบนนั้น), แพลตฟอร์มที่มีคนดูแลให้ (**PaaS** — คุณนำมาแค่แอปพลิเคชัน) หรือซอฟต์แวร์สำเร็จรูป (**SaaS** — คุณแค่ใช้) นั่นคือพื้นฐานรองรับทั้งหมด ทุกสิ่งที่สถาปนิกออกแบบถูกจัดวางอยู่บนมัน

คราวนี้ทำไมตายของสถาปนิก: เปลี่ยนทุกข้อเท็จจริงให้เป็นคำถาม วิศวกรเรียนรู้ว่า “AZ คือ data center ที่แยกอิสระ” สถาปนิกถามทันทีว่า: “*workload ตัวนี้สมควรได้กี่ AZ?*” — เพราะสอง AZ แพงกว่าหนึ่ง สามแพงกว่าสอง และเว็บไซต์โบรชัวร์การตลาดไม่สมควรได้สิ่งที่ระบบชำระเงินสมควรได้ นี่คือแนวคิดแรกและสำคัญที่สุดของหลักสูตร:

สถาปัตยกรรมคือวินัยของการทำให้ trade-off ปรากฏชัดแจ้ง แทบไม่มีส่วนประกอบที่ผิด — มีแต่ส่วนประกอบที่ผิดสำหรับ *workload* ตัวนี้ งบนี่ ทีมนี้ เดดไลน์นี้

สามเหลี่ยม trade-off ทุกงานออกแบบต้องระหว่งสามมุม: **เร็ว** (ประสิทธิภาพ — เร็วสำหรับผู้ใช้ เร็วในการสร้าง), **ถูก** (บิลรายเดือนต่ำ แรงวิศวกรรมต่ำ) และ **ทนทาน** (รอดจากความล้มเหลว ขยายได้ปลอดภัยอยู่เสมอ) คุณดันเข้าหามุมไหนก็ได้สองมุม; มุมที่สามเป็นผู้จ่าย ต้นแบบของสตาร์ทอัปควรเร็วและถูก — ความทนทานรอได้ บัญชีแยกประเภทหลักของธนาคารต้องทนทานและเร็ว — มันจะไม่ถูก

เมื่อผู้มีส่วนได้ส่วนเสียบอกว่า “เราอยากได้ทั้งสามอย่าง” งานของคุณคือเยี่ยมแล้วถามว่าเขาอยากได้อันไหนที่สุด เพราะงานออกแบบเริ่มไม่ได้จนกว่าเขาจะตอบ วาดสามเหลี่ยมนี้ไว้บนหัวของทุกงานออกแบบที่คุณทำในหลักสูตรนี้ และทำเครื่องหมายว่า workload อยู่ตรงไหน มันจะช่วยคุณประหยัดการถกเถียงนับพันครั้ง

Requirements: วัตถุประสงค์ของการออกแบบ สถาปนิกแยก **functional requirement** (ความต้องการเชิงหน้าที่ — ระบบทำอะไร — “ลูกค้าสั่งอาหารกลางวันได้”) ออกจาก **non-functional requirement (NFR)** (ความต้องการที่ไม่ใช่หน้าที่ — มันต้องทำได้ดีแค่ไหน — “ภายใน 2 วินาที รองรับนักเรียนพร้อมกัน 10,000 คน 99.9% ของเวลา ภายใต้ PDPA”) มือใหม่หมกมุ่นกับรายการแรก; สถาปนิกหาเงินเดือนจากรายการที่สอง เพราะ NFR คือสิ่งที่กำหนดสถาปัตยกรรมจริง ๆ คำถามพิเศษสองข้อที่สกัด NFR ออกจากผู้มีส่วนได้ส่วนเสียที่ไม่รู้ตัวว่าตัวเองมีมัน: “What happens to the business if this is down for an hour?” (ถ้าระบบนี้ล่มหนึ่งชั่วโมง ธุรกิจจะเป็นอย่างไร?) และ “What does success look like at ten times today’s size?” (ความสำเร็จหน้าตาเป็นอย่างไรที่สเกลสิบเท่าของวันนี้?)

คำศัพท์:

คำศัพท์	ความหมาย
Region / Availability Zone (AZ)	Region: กลุ่ม data center ตามภูมิศาสตร์ที่คุณเลือกรัน AZ: data center ที่แยกอิสระ (หรือกลุ่มเล็ก) ข้างในนั้น — หน่วยของ “หนึ่งอาคารพังได้”
IaaS / PaaS / SaaS	เช่าเครื่องเปล่า / แพลตฟอร์มที่ดูแลให้ / ซอฟต์แวร์สำเร็จรูป แยกแยะจากควบคุมมากที่สุดสู่ดูแลน้อยสุด
Workload	แอปพลิเคชันหรือระบบใดก็ตาม มองเป็นหนึ่งหน่วยของการออกแบบ — “workload ระบบชำระเงิน”
Functional requirement	สิ่งที่ระบบต้องทำ (“ลูกค้าสั่งซื้อได้”)
Non-functional requirement (NFR)	มันต้องทำได้ดีแค่ไหน — ความเร็ว สเกล uptime ความปลอดภัย การปฏิบัติตามข้อกำหนด ต้นทุน NFR คือตัวขับเคลื่อนสถาปัตยกรรม
Trade-off	สิ่งที่คุณสละไปเพื่อได้สิ่งที่เลือก ทุกการตัดสินใจออกแบบมีมัน; งานของสถาปนิกคือเอ่ยชื่อมันออกมาดัง ๆ
Constraint	ขอบเขตที่ต่อรองไม่ได้: งบ เดดไลน์ กฎหมาย ระบบที่มีอยู่ ทักษะของทีม ข้อจำกัดไม่ใช่อุปสรรคของการออกแบบ — มันคือโจทย์ของการออกแบบ

คำศัพท์	ความหมาย
Stakeholder	ใครก็ตามที่มีส่วนได้ส่วนเสียกับระบบ: ผู้ใช้ วิศวกร การเงิน ความปลอดภัย ผู้บริหาร ผู้กำกับดูแล คณะกลุ่ม คณะภาษา — คุณต้องพูดได้ทุกภาษา
Greenfield / brownfield	ระบบใหม่เอี่ยมไร้ประวัติ (greenfield) vs ระบบที่พันเกี่ยวกับของเดิม (brownfield — งานจริงส่วนใหญ่)
Managed service	ส่วนประกอบที่ผู้ให้บริการดูแลแทนคุณ (รวม backup, การแพตช์, failover) ค่าเริ่มต้นของสถาปนิก เว้นแต่มีเหตุผลเป็นลายลักษณ์อักษร

วิดีโอประจำโมดูลนี้ (ตรวจสอบลิงก์แล้ว):

วิดีโอ	ช่อง	ความยาว	ลิงก์
Top 50+ AWS Services Explained in 10 Minutes	Fireship	~10 นาที	https://www.youtube.com/watch?v=JlbiYCM48to

ดูทัวร์ของ Fireship สองรอบ: รอบแรกตอนนี้เพื่อเห็นแผนที่ รอบสองตอนจบช่วงที่ 1 — คุณจะแปลกใจว่าตอนนี้วางตำแหน่งบริการต่าง ๆ ลงในงานออกแบบได้ก็ตัวแล้ว

แบบฝึกหัด: (1) เลือกแอปที่คุณใช้ทุกวันสามตัว (แอปธนาคาร แอปสั่งอาหาร แอปวิดีโอ) แล้วเขียน NFR สามอันดับแรกของแต่ละตัว พร้อมทำเครื่องหมายบนสามเหลี่ยม trade-off (2) สัมภาษณ์เพื่อนเรื่องไอเดียธุรกิจสิบนาที แล้วสกัด functional requirement ห้าข้อกับ non-functional ห้าข้อ — สังเกตว่า NFR โพล่มาก็ต่อเมื่อคุณถามคำถามพิเศษสองข้อ (3) เริ่ม Design Journal ด้วยรายการที่ 1: สามเหลี่ยม และหนึ่งย่อหน้าว่าทำไม “เราจะสละมุมไหน?” เป็นคำถามธุรกิจ ไม่ใช่คำถามเทคนิค

Milestone: เมื่อได้รับคำบรรยายระบบหนึ่งประโยคใด ๆ คุณสร้าง NFR ที่น่าจะเป็นและตำแหน่งบนสามเหลี่ยมได้ในห้านาที ออกเสียง โดยไม่เปิดโน้ต

โมดูล 2 (สัปดาห์ 3–4): Compute, storage, database, network — สี่การตัดสินใจ

ทุกงานออกแบบที่คุณจะวาดตลอดชีวิตคือสี่ทางเลือกนี้ ตัดสินใจอย่างจริงจัง นี่คือการตัดสินใจที่ถูกสอนในฐานะการตัดสินใจ

การตัดสินใจที่ 1 — Compute: VM, container หรือ serverless? virtual machine (VM) (เครื่องเสมือน) คือส่วนแบ่งของ server ที่เข้ามาและทำตัวเหมือนคอมพิวเตอร์เต็มเครื่อง — ควบคุมได้มากที่สุด ดูแลมากที่สุด (คุณแพตช์เอง ขยายเอง และจ่ายเงินแม้มันว่างงาน) **container** (คอนเทนเนอร์) บรรจุแอปหนึ่งตัวพร้อมทุกสิ่งที่มีมันต้องใช้ให้รันเหมือนกันทุกที; ตัวจัดวงดนตรี (**Kubernetes**) รันและซ่อมแซมพวกมันเป็นฝูง — ความหนาแน่นและความพกพาได้เยี่ยม แต่คุณได้รับแพลตฟอร์มซับซ้อนที่ต้องการคนมีทักษะมาด้วย **Serverless** (เซิร์ฟเวอร์เลส — AWS Lambda, Azure Functions) รันโค้ดของคุณเฉพาะเมื่อถูก trigger และคิดเงินต่อการเรียก — การดูแลใกล้ศูนย์และเหมาะสมกับกราฟฟิกพุ่งเป็นช่วง ๆ แต่มีข้อจำกัด (เพดานเวลารัน, cold start และการแต่งงานลึกขึ้นกับผู้ให้บริการรายเดียว) ตารางการตัดสินใจที่คุณควรเขียนซ้ำได้จากความจำ:

เลือก	เมื่อ	ระวัง
VM	ซอฟต์แวร์ legacy, ไลเซนส์พิเศษ, ต้องควบคุม OS เต็มรูปแบบ, โหลดคงที่คาดการณ์ได้	คุณเป็นเจ้าของการแพตช์ การขยาย และเหตุตอนตีสาม; เวลาว่างงานคิดเงินเต็ม
Containers + Kubernetes	หลาย service, ทีมมีทักษะอยู่แล้ว, ความพกพาสําคัญ	ความซับซ้อนของแพลตฟอร์ม — K8s คืองานเต็มเวลา; เกินจำเป็นสำหรับทีมเล็ก
Serverless	กราฟฟิกพุ่งหรือคาดเดาไม่ได้, กาวเชื่อมแบบ event-driven, ทีมเล็ก, ต้องออกตลาดเร็ว	ข้อจำกัด runtime, cold start, ค่าการณต้นทุนยากขึ้นที่สเกล คงทีมหาศาล, lock-in

ข้อคิดระดับ senior: **นี่คือทางเลือกราย workload ไม่ใช่ศาสนาประจำบริษัท** ระบบจริง ๆ รันทั้งสามแบบเคียงข้างกัน อย่างถูกต้อง

การตัดสินใจที่ 2 — Storage: object, block หรือ file — และร้อนแค่ไหน? Object storage (Amazon S3) คือถังไรกันบั้งสำหรับไฟล์ — ถูก ทนทานเหลือเชื่อ (“สิบเอ็ดเลขเก้า”) คำตอบมาตรฐานของ “ไฟล์ไปอยู่ไหน?” **Block storage** (EBS) คือดิสก์เสมือนที่ยึดติดกับ VM **File storage** (EFS) คือไดรฟ์แชร์ที่หลายเครื่อง mount พร้อมกัน มิติพิเศษของสถาปนิกคือ **อุณหภูมิ**: ข้อมูล hot (ถูกเข้าถึงตลอด ตั้งราคาเพื่อความเร็ว) กับ tier แบบ cold/archive (Glacier — ราคาหลักสตางค์ แต่ใช้เวลาถึงเป็นนาที่ถึงชั่วโมง) การออกแบบ lifecycle rule ให้ข้อมูลเก่าไหลไป tier เย็นคือชัยชนะด้านต้นทุนที่ถูกที่สุดในคลาวด์; การลืมทำคือความผิดพลาดที่พบบ่อยที่สุด

การตัดสินใจที่ 3 — Database: SQL หรือ NoSQL (และรสน managed ตัวไหน)? database แบบ **Relational/SQL** (PostgreSQL, MySQL; แบบ managed คือ RDS/Aurora) เก็บข้อมูลในตารางเข้มงวดพร้อม consistency ที่รับประกัน — ค่าเริ่มต้นสำหรับทุกอย่างที่ความถูกต้องศักดิ์สิทธิ์: เงิน คำสั่งซื้อ สินค้าคงคลัง ผู้ใช้ database แบบ **NoSQL** (DynamoDB, MongoDB) แลกโครงสร้างเข้มงวดกับความ

ยืดหยุ่นและการขยายแนวนอนแทบไร้ขีดจำกัด — ค่าเริ่มต้นสำหรับ session, แคล์ตาล็อก, ฟีด, telemetry ฮิวริสติกการตัดสินใจ: *เริ่มด้วย SQL เว้นแต่คุณเฝ้าชื่อเหตุผลเจาะจงได้ว่าทำไมมันไม่เวิร์ก* (สเกลสุดขีด, schema ยืดหยุ่น, การอ่านทั่วโลกระดับมิลลิวินาทีหลักเดียว) และในคลาวด์ “database” ควรหมายถึง “**managed database**” แทบเสมอ — ผู้ให้บริการจัดการ backup แพตช์ และ failover; ทีมที่รัน database เองบน VM ควรมีเหตุผลเป็นลายลักษณ์อักษร เพิ่มตัวผู้เชี่ยวชาญเฉพาะทางลงคลังศัพท์ของคุณ: **cache** (Redis — ข้อมูลร้อนในหน่วยความจำ อ่านระดับไมโครวินาที), **warehouse** (วิเคราะห์ข้อมูลที่สเกล — ช่วงที่ 3), **queue** (ไม่ใช่ database แต่บ่อยครั้งคือขั้นที่ขาด — ช่วงที่ 3)

การตัดสินใจที่ 4 — Network: รูปทรงของโลกส่วนตัว VPC (Virtual Private Cloud) คือส่วนที่ล้อมรั้วไว้ของคุณในเครือข่ายผู้ให้บริการ ข้างในนั้น **public subnet** ถือกิ่งที่อินเทอร์เน็ตเข้าถึงได้ (load balancer) และ **private subnet** ถือกุญแจที่เหลือ — app server และ, เสมอ, database “*database อยู่ใน private subnet*” คือประโยคที่ถูกพูดซ้ำมากที่สุดในการรีวิวด้านความปลอดภัย; เหตุผล — ไม่มีอะไรโจมตีสิ่งที่ไม่ได้มีเส้นทางจากอินเทอร์เน็ตได้ — คือครึ่งหนึ่งของความปลอดภัยเครือข่าย รอบ ๆ VPC: **load balancer** กระจายทราฟฟิกข้าม server และเลี้ยงเครื่องป่วย; **DNS** (Route 53) แปลงชื่อเป็นที่อยู่; **CDN** (CloudFront) แคชเนื้อหาในเมืองหลายร้อยแห่งให้เร็วทุกที; **API gateway** คือเคาน์เตอร์ประชาสัมพันธ์แบบ managed สำหรับ API ของคุณ (การยืนยันตัวตน, rate limit, การเก็บ log) การเชื่อมกับโลกเก่า: **VPN** (อุโมงค์เข้ารหัสบนอินเทอร์เน็ต) หรือ **Direct Connect** (สายกายภาพส่วนตัว) — สายสะดือของทุกงานออกแบบ hybrid ในช่วงที่ 4

คำศัพท์:

คำศัพท์	ความหมาย
Instance / instance type	VM ที่เช่าหนึ่งเครื่อง / ขนาดของมัน (CPU + RAM) ซึ่งกำหนดราคาต่อชั่วโมง
Container / Docker / Kubernetes (K8s)	แพ็คเกจพกพาสำหรับหนึ่งแอป / เครื่องมือที่สร้างและรันมัน / ตัวจัดวางดนตรีที่รันและซ่อมแซมมันเป็นฝูง
Serverless / Lambda	โค้ดที่รันเฉพาะเมื่อถูก trigger คิดเงินต่อการรัน ไม่มี server ให้ดูแล Lambda คือเวอร์ชันของ AWS
Cold start	ดีเลย์พิเศษเมื่อฟังก์ชัน serverless ถูกรันหลังจากว่างงาน — trade-off คลาสสิกของ serverless
S3 / bucket / durability	object storage ของ AWS / ภาชนะไฟล์ติดชื่อหนึ่งใบ / โอกาสที่ข้อมูลอยู่รอด — 99.99999999% ของ S3 แปลว่าการสูญหายแทบไม่มีวันเกิด

คำศัพท์	ความหมาย
Storage tier / lifecycle policy	ชั้นราคา-ความเร็ว (hot → cold → archive) / กฎอัตโนมัติที่ย้ายข้อมูลแก็ตว์ไป tier ถูกลง
RDS / Aurora / DynamoDB	บริการ SQL แบบ managed ของ AWS / SQL สมรรถนะสูงแบบ cloud-native ของมัน / NoSQL แบบ managed เรืองของมัน
Consistency	การรับประกันว่าทุกคนที่อ่านข้อมูลเห็นความจริงเดียวกันในเวลาเดียวกัน — พลังพิเศษของ SQL และสิ่งที่ NoSQL ผ่อนคลายเพื่อแลกสเกล
Cache / Redis	สำเนาความเร็วหน่วยความจำของข้อมูลร้อนที่วางหน้า database / เครื่องมือมาตรฐานของมัน
VPC / subnet (public, private)	ส่วนเครือข่ายส่วนตัวของคุณ / ส่วนย่อยของมัน — public หันหน้าเข้าอินเทอร์เน็ต private ไม่ database อยู่ private เสมอ
Load balancer / health check	ผู้กำกับการจราจรข้าม server / การทดสอบชีพจรที่มันใช้เพื่อหยุดส่งทราฟฟิกให้เครื่องที่ตาย
CDN / edge	แคชระดับเมืองของเนื้อหาคุณทั่วโลก / “the edge” = ใกล้ผู้ใช้
API / API gateway	ประตูควบคุมได้ที่ซอฟต์แวร์หนึ่งเปิดให้อีกตัว / เคา์เตอร์ประชาสัมพันธ์แบบ managed ของประตูเหล่านั้น
VPN / Direct Connect	อุโมงค์เข้ารหัสบนอินเทอร์เน็ต / สายกายภาพส่วนตัวเข้าคลาวด์ — สองวิธีที่ on-prem พบกับคลาวด์

วิดีโอประจำโมดูลนี้ (ตรวจสอบลิงก์แล้ว):

วิดีโอ	ช่อง	ความยาว	ลิงก์
Kubernetes explained in 15 mins	TechWorld with Nana	~16 นาที	https://www.youtube.com/watch?v=VnvRFRk_51k

วิดีโอ	ช่อง	ความยาว	ลิงก์
Serverless Computing in 100 Seconds	Fireship	~2 นาที	https://www.youtube.com/watch?v=W_VV2Fx32_Y
AWS Networking Basics (VPC & Subnets)	KodeKloud	~30 นาที	https://www.youtube.com/watch?v=QM63dyA_4Pc

แบบฝึกหัด: (1) สำหรับ workload ห้าตัวนี้ เลือก compute, database และ storage แล้วเขียนเหตุผลอย่างละหนึ่งประโยค: เว็บไซต์อาหารกลางวันของโรงเรียน; บัญชีแยกประเภทธุรกรรมของธนาคาร; แอปแชร์รูปภาพ; ตัวสร้างรายงานรายคืนที่รัน 20 นาที; แอปแชตสำหรับผู้ใช้ 5 ล้านคน (2) หยิบทางเลือกหนึ่งของคุณมาแล้วแก้ต่างให้ทางเลือกตรงข้ามอย่างน่าเชื่อที่สุดเท่าที่ทำได้ — สถาปนิกที่ steel-man ทางเลือกอื่นไม่ได้ แปลว่ายังไม่เข้าใจ trade-off (3) ลงสมุด: ตารางการตัดสินใจ compute ของคุณ จากความจำ

Milestone: เมื่อได้ workload ใดก็ตามในหนึ่งประโยค คุณเอย่ซื้อสื่อการตัดสินใจของมันพร้อมเหตุผลได้ภายในสามนาที — และสำหรับอย่างน้อยหนึ่งการตัดสินใจ เอย่ได้ว่าจะอะไรจะช่วยให้คุณเปลี่ยนใจ

โมดูล 3 (สัปดาห์ 5–6): การอ่านและวาดแผนภาพสถาปัตยกรรม

สถาปนิกที่วาดไม่เป็นคือที่ปรึกษาที่ได้แต่พูด แผนภาพคือภาษาทำงานของคุณ: โมดูลนี้ทำให้คุณอ่านมันคล่องและวาดมันได้

แผนภาพต้นแบบ — เรียนตัวนี้ก่อน three-tier architecture (สถาปัตยกรรมสามชั้น) คือ “โครงสร้างประโยค” ของแผนภาพคลาวด์; งานออกแบบส่วนใหญ่คือรูปแปดเหลี่ยมของมัน:

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

→ [Cache]

→ [Database: primary +

standby, private subnets]

Tier 1 (**presentation**): สิ่งที่ใช้สัมผัส — เนื้อหา static จาก CDN, คำขอผ่าน load balancer Tier 2

(**application**): ผูก server ที่สับเปลี่ยนแทนกันได้ซึ่งรันตรรกะของคุณ อยู่ใน private subnet ขยาย

อัตโนมัติ Tier 3 (**data**): database ลึกลับและถูกปกป้องมากที่สุด พร้อม standby ใน AZ ที่สอง ทราฟฟิก

ไหลทางเดียว: ผู้ใช้ไม่มีวันแตะ app tier โดยตรง และ app tier เท่านั้นที่คุยกับ data tier ฝึกวาดจนมือตัวเองโดยสมองไม่ต้องสั่ง — มันคือบทเปิดของ whiteboard interview ทุกที่บนโลก

ธรรมเนียมสัญญาที่ทำให้คุณดูเป็นมืออาชีพ (เพราะมันบังคับให้คุณคิดแบบมืออาชีพ): กล่องคือส่วนประกอบ — ติดป้ายแต่ละกล่องว่าคืออะไร และ ใช้บริการไหน (“App servers — EC2, auto-scaling group”) ลูกศรแสดงทิศทางที่คำขอไหล ติดป้ายโปรโตคอลเมื่อสำคัญ กรอบเส้นประแสดงขอบเขต — VPC, แต่ละ subnet, แต่ละ AZ (วาดขอบเขต AZ แล้ว multi-AZ จะมองเห็นได้แทนที่จะเป็นแค่คำอ้าง) ผู้ใช้/actor ยื่นนอกระบบ ตัวเลขบนลูกศร (1, 2, 3...) ให้คุณเล่าเส้นทางการเดินทางของคำขอ และทุกแผนภาพมีชื่อเรื่อง วันที่ และคำอธิบายสัญลักษณ์ กฎที่ลึกกว่า: **หนึ่งแผนภาพ หนึ่งผู้ชม หนึ่งคำถาม** แผนภาพที่โชว์ทุกอย่างเท่ากับไม่โชว์อะไรเลย; คุณจะเรียนรู้ระดับซุมมาตรฐาน (context → container → deployment) ในช่วงที่ 5

การอ่านแผนภาพของคนอื่น — เอกซเรย์ของสถาปนิก เมื่อได้รับแผนภาพ ให้รันสแกนนี้ออกเสียง: อินเทอร์เน็ตแต่ละระบบนี้ตรงไหน (ทุกจุดสัมผัสคือ attack surface)? ข้อมูลอยู่ไหน แล้วอยู่ใน private subnet หรือเปล่า? อะไรมีตัวสำรอง (หนทาง) และอะไรคือ **single point of failure** — กล่องที่ไม่มีแฝด? ที่กราฟฟิก 10x จะเจ็บตรงไหน? แต่ละกล่องราคาเดือนละเท่าไร? ห้าคำถาม สามสิบวินาที แล้วคุณก็อ่านแผนภาพแบบเดียวกับที่หมออ่านฟิล์มเอกซเรย์ เข้าไปดู AWS Architecture Center (<https://aws.amazon.com/architecture/>) แล้วรันสแกนนี้กับ reference architecture ที่เผยแพร่ไว้สามชิ้น

คำศัพท์:

คำศัพท์	ความหมาย
Three-tier architecture	การแยกชั้นคลาสสิก: presentation (ทางเข้าของผู้ใช้) → application (ตรรกะ) → data (database) รูปทรงเริ่มต้นของระบบเว็บ
Tier / layer	ชั้นแนวนอนของระบบที่มีหนึ่งความรับผิดชอบ โดยเฉพาะกับเพื่อนบ้าน
Single point of failure (SPOF)	ส่วนประกอบใดที่ตายแล้วพาระบบทั้งหมดล่ม สิ่งแรกที่ต้องล่าในทุกแผนภาพ
Multi-AZ	รันตัวสำรองข้ามอย่างน้อยสอง Availability Zone เพื่อให้ data center หนึ่งพังได้แบบไร้ร่องรอย
Auto-scaling (group)	เครื่องถูกเพิ่มและถอดอัตโนมัติตามความต้องการ — ความจุที่หายใจได้

คำศัพท์	ความหมาย
Stateless / stateful	server ที่ไม่ถือข้อมูลเฉพาะตัว (แผดตัวไหนก็แทนได้ จึงขยายอิสระ) vs ตัวที่ถือข้อมูลที่ห้ามหาย เป้าหมาย ออกแบบ: app tier แบบ stateless ดัน state ลงไป database และ cache
Reference architecture	แบบตัวอย่างที่ผู้ให้บริการเผยแพร่และรับรองสำหรับ ปัญหาอดนิยม — สถาปนิกประกอบจากสิ่งเหล่านี้ ก่อนคิดค้นเอง
Context diagram	ระดับชุมสูงสุด: ระบบของคุณเป็นกล่องเดียว บวกผู้ใช้และระบบภายนอกที่มันสัมผัส
Attack surface	ทุกจุดที่โลกภายนอกแตะระบบได้ ยิ่งเล็กยิ่งปลอดภัย
North–south / east–west traffic	ทราฟฟิกเข้า/ออกระบบ vs ทราฟฟิกระหว่างส่วน ประกอบข้างใน

แบบฝึกหัด: (1) วาดแผนภาพสามชั้นจากความจำ หัววันติด จนใช้เวลาไม่ถึงสัปดาห์พร้อมขอบเขตครบ (VPC, subnet, สอง AZ) (2) หยิบ workload หัวตัวจากโมดูล 2 มาวาดทีละตัว — ยี่สิบนาทีต่อแผนภาพ บังคับใช้กฎสัญญาณ (3) หาแผนภาพสถาปัตยกรรมจริงที่ไหนก็ได้ในอินเทอร์เน็ต (AWS Architecture Center มีเป็นร้อย) แล้วเขียนสแกนเอกซเรย์หาคำถามลงสมุด (4) แบบฝึกเล่าเรื่อง: วางแผนภาพตรง หน้า เล่าการคลิกของผู้ใช้หนึ่งคนจากเบราว์เซอร์ถึง database และย้อนกลับ ออกเสียง พร้อมไต่ หมายเลขลูกศร

Milestone — จบช่วงที่ 1: บนไวต์บอร์ด (หรือกระดาน ถ่ายรูปเก็บลงสมุด) คุณวาดงานออกแบบสามชั้น ที่ถูกต้อง สัญกรณ์เรียบร้อย สำหรับ workload หนึ่งประโยคที่ไม่เคยเจอ ได้ในไม่เกินสิบห้านาที เล่า คำขอวิ่งผ่านมันได้ และตอบ “ถ้ากล่องนี้ตายจะพังอะไร?” ได้ทุกกล่อง การวาด-บวก-การซักถามนี้คือ ครั้งแรกของการสัมภาษณ์สถาปนิกจริงปะะ ๆ — จากนั้นไป ทุกอย่างคือความลึก

ช่วงที่ 2 — อาณาเขตของการออกแบบ (สัปดาห์ 7–14)

แผนที่ของช่วงนี้คือ **AWS Well-Architected Framework** — เช็กลิสต์ร่วมของอุตสาหกรรมว่า “ออกแบบอย่างถูกต้อง” แปลว่าอะไร จัดเป็นหกเสาหลัก: **operational excellence, security, reliability, performance efficiency, cost optimization, sustainability** (Azure มี framework ที่

เกือบเหมือนกันเป๊ะ; เรียนอันหนึ่งให้ลึกเท่ากับได้ทั้งคู่) ประกาศงานขอสถาปนิกที่ “run Well-Architected reviews” โดยเรียกชื่อตรง ๆ เราจึงไล่เสาหลักทีละตัว และแต่ละตัวคุณเรียนสามสิ่ง: **คำถามหลัก**ของเสา, **แพตเทิร์นมาตรฐาน**ของมัน (คำตอบน่าเบื่อที่พิสูจน์แล้ว — สถาปนิกประกอบก่อนคิดค้น) และ **ตัวอย่างลงมือทำ**บนโจทย์ต่อเนื่อง

โจทย์ต่อเนื่องของช่วงที่ 2 ทั้งหมด: *ThaiTicket* แพลตฟอร์มขายบัตรอีเวนต์สมมติในกรุงเทพฯ โหลดปกติ: ผู้เข้าชม 2,000 คน/ชั่วโมง แต่เมื่อบัตรศิลปินดังเปิดขายตอน 10:00 น. มันรับผู้เข้าชม 400,000 คนในสิบนาที การชำระเงินห้ามขายที่นั่งซ้ำ และข้อมูลส่วนบุคคลของลูกค้าไทยอยู่ใต้ PDPA เร็ว ถูก ทนทาน — ThaiTicket ต้องการทั้งสามและมีไม่ได้ นั่นแหละที่ทำให้มันเป็นคนไข้ฝึกหัดที่สมบูรณ์แบบ

ดูก่อนเข้าโมดูล 4 (ตรวจสอบลิงก์แล้ว):

วิดีโอ	ช่อง	ความยาว	ลิงก์
The Five Pillars of the AWS Well-Architected Framework	Amazon Web Services (ช่องทางการ; เสาที่หก Sustainability ถูกเพิ่มภายหลัง)	~4 นาที	https://www.youtube.com/watch?v=KvEDbPmha6o
What is the AWS Well-Architected Framework?	Tech With Lucy	~10 นาที	https://www.youtube.com/watch?v=MpDJ6TCWKjk

และนี่ก็มารีควิเรกตัว framework เอง — <https://aws.amazon.com/architecture/well-architected/> และเอกสารฉบับเต็มที่ <https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html> — คุณจะใช้ชีวิตอยู่ในนั้นแปดสัปดาห์

โมดูล 4 (สัปดาห์ 7–8): Reliability — ออกแบบเพื่อวันที่ของพัง

คติของเสา: ทุกอย่างพังได้ ตลอดเวลา ดิสก์ตาย AZ น้ำท่วม deploy ผิดพลาด และ certificate ที่ไหนสักแห่งใกล้หมดอายุเสมอ reliability (ความน่าเชื่อถือ) ไม่ใช่การไร้ซึ่งความล้มเหลว — แต่คือการทำให้ความล้มเหลว**ไม่มีความหมาย**: ออกแบบให้เมื่อ (ไม่ใช่ถ้า) ส่วนประกอบหนึ่งตาย ผู้ใช้ไม่มีวันสังเกต

คำถามหลัก (ถามกับทุกงานออกแบบ ตลอดไป): เมื่อแต่ละส่วนประกอบพังจะเกิดอะไรขึ้น — มี “แล้วยังไงต่อ” เสมอไหม? ระบบรับโหลด 10× อย่างไร? เรารู้ได้อย่างไรว่ามีอะไรพังก่อนลูกค้าบอก? **RTO** และ **RPO** ของเราคือเท่าไร — และใครในฝั่งธุรกิจเป็นคนเซ็นรับ? เราทดสอบการกู้คืนครั้งสุดท้ายเมื่อไหร่?

RTO และ RPO — สองตัวเลขที่คือบทสนทนาเรื่องภัยพิบัติทั้งหมด **Recovery Time Objective**: เราล้มได้นานแค่ไหน? **Recovery Point Objective**: เรายอมเสียข้อมูลได้มากแค่ไหน (คือ backup ดีล่าสุดแก่ได้แค่ไหน)? RTO 4 ชั่วโมงกับ RPO 1 ชั่วโมงแปลว่า: กลับมาภายในสี่ชั่วโมง โดยเสียข้อมูลอย่างมากหนึ่งชั่วโมง สองตัวนี้เป็นการตัดสินใจทางธุรกิจที่ป้ายราคาโตะแบบทวีคูณ — RPO 24 ชั่วโมงคือ backup รายคืน (ถูก); RPO ~ศูนย์คือ replication ต่อเนื่อง (แพง); RTO หลักนาทีแปลว่ามีโครงสร้างพื้นฐานอุ่นเครื่องรอเปล่า ๆ (แพงมาก) งานของสถาปนิกคือทำให้ธุรกิจเลือกตัวเลขอย่างรู้ตัว — และออกแบบให้ตรงกับตัวเลขนั้นเป๊ะ ไม่ใช่เกินเลยแบบโรแมนติก

แพตเทิร์นมาตรฐาน: redundancy (สองชุดของทุกอย่างที่สำคัญ — N+1) · **multi-AZ** (ตัวสำรองข้าม data center; load balancer และ failover ของ managed database ทำให้อัตโนมัติ) · **auto-scaling** (ความจุตามความต้องการ) · **health check + self-healing** (instance ที่ตายถูกตรวจพบและแทนที่โดยเครื่องจักร ไม่ใช่มนุษย์) · **backup ที่ทดสอบแล้ว** (backup ที่ไม่ทดสอบคือความหวัง ไม่ใช่แผน — จัดตารางซ้อมกู้คืน) · **graceful degradation** (เมื่อโหลดล้น ให้ปลดพีเจอร์สำคัญน้อยสุดก่อน: ThaiTicket ยอมทิ้งฟรีวิวฝั่งที่นั่งแต่เก็บหน้าชำระเงินไว้) · **queue เป็นโซ่คัพ** (ช่วงที่ 3) · **การเสี่ยง cascading failure** (timeout, retry พร้อม backoff, circuit breaker — เพื่อไม่ให้ dependency ช้าตัวเดียวลากทั้งฝูงจมน้ำ)

ตัวอย่างลงมือทำ — งานออกแบบ reliability ของ ThaiTicket สอง AZ ใน region กรุงเทพฯ app tier แบบ stateless ใน auto-scaling group หลัง load balancer อุ่นเครื่องล่วงหน้าตามตารางก่อนเวลาเปิดขายที่ประกาศ (auto-scaling ตอบสนองในหลักนาที; การเปิดขาย 10:00 ต้องมีความจุตั้งแต่ 09:45) database เป็น Aurora, primary ใน AZ-a, standby แบบ synchronous ใน AZ-b, failover อัตโนมัติ ≈ ไม่ถึงนาที มี queue คั่นระหว่างคลิก “ซื้อ” กับการประมวลผลชำระเงิน เพื่อให้เมื่อฝั่งผู้ให้บริการชำระเงินอัตโนมัติเข้าคิวแทนที่เว็บจะล้ม Backup: ต่อเนื่อง กู้คืนแบบ point-in-time ซ้อมกู้คืนรายเดือน ตัวเลขที่ตกลงกับธุรกิจ: RTO 15 นาที, RPO ~ศูนย์สำหรับออเดอร์ (มันคือเงิน), RPO 24 ชั่วโมงสำหรับข้อมูลวิเคราะห์ (มันไม่ใช่) แบบฝึกลงสมุด: งานออกแบบนี้ทำให้เราเสียอะไรไปจากมุม “ถูก” ของสามเหลี่ยม?

คำศัพท์:

คำศัพท์	ความหมาย
RTO / RPO	Recovery Time Objective: คุณล้มได้นานแค่ไหน Recovery Point Objective: คุณเสียข้อมูลได้มากแค่ไหน สองตัวเลขที่นิยามทุกบทสนทนา DR — และเป็นการตัดสินใจทางธุรกิจ
High availability (HA)	การออกแบบให้ความล้มเหลวประจำวัน (instance หนึ่ง, AZ หนึ่ง) ไม่ก่อเหตุล้มที่ผู้ใช้มองเห็น
Disaster recovery (DR)	แผนสำหรับความล้มเหลวใหญ่ — ทั้ง region, เหตุ ransomware — พร้อมเป้า RTO/RPO และ runbook

คำศัพท์	ความหมาย
	ที่ทดสอบแล้ว
Redundancy / N+1	ความจุสำรองให้ส่วนประกอบใดตายได้หนึ่งตัว: ต้องใช้ N รัน N+1
Failover	การสลับอัตโนมัติไปยัง standby เมื่อ primary ตาย
Health check	ซีพอร์อัตโนมัติที่ตรวจสอบส่วนประกอบที่ตาย เพื่อให้เครื่องจักรวิ่งอ้อมมันได้
Graceful degradation	การปลดพีเจอร์สำคัญน้อยกว่าภายใต้ความเครียด เพื่อให้แกนกลางอยู่รอด
Timeout / retry with backoff	การเลิกรอสายที่ซ้ำหลังถึงขีดจำกัด / การลองใหม่ด้วยช่วงพักที่ยาวขึ้นเรื่อย ๆ — มารยาทที่ป้องกันความล้มเหลวลูกโซ่
Circuit breaker	ส่วนประกอบที่หยุดเรียก dependency ที่กำลังพังชั่วคราวเพื่อให้มันฟื้น — เหมือนฟิวส์ในบ้าน
Chaos engineering	การจงใจฉีดความล้มเหลว (สไตล์ Netflix) เพื่อพิสูจน์คำอ้างเรื่องความทนทานก่อนที่ความเป็นจริงจะทดสอบแทนคุณ

แบบฝึกหัด: (1) หยิบงานออกแบบเว็บสั่งอาหารกลางวันจากช่วงที่ 1 มาอัปเดตให้รอดจาก: instance ตาย, เสีย AZ, database พัง และมี้อกลางวันพีค 10x — วาดก่อนและหลัง (2) เขียนบทสนทนา RTO/RPO สำหรับสามธุรกิจ (บล็อก, เว็บ e-commerce, ระบบเวชระเบียนโรงพยาบาล) เป็นบทพูดสั้น ๆ ระหว่างสถาปนิกกับเจ้าของ — ฝึกทำให้ต้นทุนมองเห็นได้ในบทสนทนา (3) ลงสมุด: ใส่รายการ SPOF ทุกจุดในงานออกแบบ ThaiTicket ข้างบน มีอย่างน้อยหนึ่งที่จะใจเหลือไว้ (คำใบ้: ก็ region?)

Milestone: คุณรันการซักถาม “ถ้าอันนี้พังจะเกิดอะไรขึ้น?” กับแผนภาพใดก็ได้ต่อเนื่องสิบนาทีโดยไม่มุกแห้ง และคุณอธิบาย RTO กับ RPO ให้เจ้าของกิจการที่ไม่ใช่สายเทคนิคฟังด้วยการเปรียบเทียบร้านค้าถูกน้ำท่วม ได้ในเก้าสิบวินาที

โมดูล 5 (สัปดาห์ 9–10): Security — ออกแบบระบบที่ทำร้ายได้ยาก

กรอบคิด: shared responsibility model (โมเดลความรับผิดชอบร่วม) ผู้ให้บริการดูแลความปลอดภัยของตัวคลาวด์เอง — อาคาร ฮาร์ดแวร์ hypervisor คุณดูแลทุกอย่างที่คุณใส่เข้าไป — ข้อมูล ตัวตน การตั้งค่า โค้ด เหตุข้อมูลรั่วบนคลาวด์ที่โด่งดังแทบทุกกรณีคือการตั้งค่าผิดฝั่งลูกค้า (bucket สาธารณะ,

key หลุด, role ที่สิทธิ์เกิน) นี่คือเหตุผลที่ความปลอดภัยเป็นปัญหา**สถาปัตยกรรม**ก่อนจะเป็นปัญหาเครื่องมือ: ประกาศงานบอกว่า “integrate security into the design rather than bolting it on” และโมดูลนี้คือวิธีทำ

คำถามหลัก: ใครและอะไรเข้าถึงแต่ละส่วนประกอบได้ และทุกสิทธิ์เป็น**ขั้นต่ำ**ที่จำเป็นหรือไม่? ข้อมูลถูกเข้ารหัสที่ไหน — at rest, in transit และใครถือกุญแจ? ขอบเขตเครือข่ายอยู่ตรงไหน และอะไรข้ามมันบ้าง? เราจะ**ตรวจ**จัดการเจาะระบบได้อย่างไร — และเล่าเรื่องย้อนหลังจาก log ได้ไหม? กฎหมายอะไรใช้กับข้อมูลนี้ และข้อมูลอาศัยอยู่ที่ไหนทางกายภาพ?

แพตเทิร์นมาตรฐาน: **least privilege** (ทุกคนและทุกโปรแกรมได้สิทธิ์ขั้นต่ำที่งานต้องการ — กฎทองคำที่ทุก IAM policy ถูกตัดสินเทียบ) · **MFA** ทุกที่ **root** ถูกเก็บล็อกไว้ · การเข้ารหัส **at rest** และ **in transit** **เปิดเสมอ** (ในคลาวด์มันคือช่องตึก; ไม่มีข้อแก้ตัว) · **network segmentation** (การแบ่งส่วนเครือข่าย — public/private subnet, security group เป็น firewall รายเครื่อง; รัศมีความเสียหาย (blast radius) ของการเจาะถูกนิยามโดยกำแพงที่คุณวาดไว้ล่วงหน้า) · **secret อยู่ใน vault ไม่อยู่ในโค้ดเด็ดขาด** · **zero trust** (ตรวจสอบทุกคำขออย่างชัดเจน — ตัวตน อุปกรณ์ บริบท — ไม่เชื่อใจสิ่งใดเพียงเพราะมันอยู่ “ข้างในเครือข่าย”; JD เอ้ยเชื่อมั่น คุณก็ต้องเอ้ยได้) · **defense in depth** (หลายชั้น เพื่อให้การควบคุมล้มเหลวหนึ่งตัวไม่ใช่จบเกม) · **audit logging** (บันทึกสไตร์ CloudTrail ว่าใครทำอะไร — แก้ไขไม่ได้ ถูกเฝ้าดู) · **ความปลอดภัยเป็น guardrail ไม่ใช่ gate** (เข้ารหัสกฎเป็นนโยบายอัตโนมัติที่ทำให้เส้นทางปลอดภัยเป็นเส้นทางง่าย แทนที่จะเป็นประชุมรีวิวที่ทำให้ความปลอดภัยกลายเป็นศัตรูของการส่งมอบ)

ตัวอย่างลงมือทำ — งานออกแบบ security ของ ThaiTicket ข้อมูลลูกค้า (ชื่อ อีเมล ข้อมูลอ้างอิงการชำระเงิน) จัดชั้นเป็นข้อมูลส่วนบุคคลตาม PDPA → เก็บแบบเข้ารหัสใน Aurora ที่ region กรุงเทพฯ (data residency — ถิ่นพำนักของข้อมูล) กุญแจอยู่ใน KMS เครือข่าย: มีแค่ load balancer ที่ public; app tier เป็น private; subnet ของ database รับการเชื่อมต่อ**เฉพาะ**จาก security group ของ app tier มนุษย์: SSO + MFA; วิศวกรได้สิทธิ์อ่านอย่างเดียวใน production เป็นค่าเริ่มต้น และขอสิทธิ์ยกระดับแบบจำกัดเวลาได้ (least privilege พร้อม audit trail) การจัดการข้อมูลบัตรจ่ายเงินมอบให้ผู้ให้บริการชำระเงินที่ผ่านการรับรอง เพื่อให้เลขบัตรดิบไม่มีวันแตะระบบเรา — การตัดสินใจเชิง**ออกแบบที่**ปลดภาระ compliance ทั้งก้อน ซึ่งคือ**สถาปัตยกรรมความปลอดภัย**ในรูปที่ดีที่สุด CloudTrail เปิด แจ้งเตือนเมื่อมีการเข้าถึงผิดปกติ คำถามเหตุร้ายถูกตอบล่วงหน้า: PDPA มีหน้าที่แจ้งเหตุภายใน 72 ชั่วโมง — log และ runbook ต้องทำให้เราเล่าเรื่องได้เร็วกว่านั้น

คำศัพท์:

คำศัพท์	ความหมาย
Shared responsibility model	ผู้ให้บริการดูแลคลาวด์; คุณดูแลสิ่งที่อยู่ข้างใน เหตุร้ายส่วนใหญ่อยู่ฝั่งลูกค้าของเส้นแบ่ง

คำศัพท์	ความหมาย
IAM / role / policy	Identity and Access Management — ใครทำอะไร ได้ policy คือรายการสิทธิ์; role คือมัดสิทธิ์ที่สวมใส่ ได้
Least privilege	กฎทองคำ: สิทธิ์ขั้นต่ำที่จำเป็น ไม่มากไปกว่านั้น รีวิวม้าเสมอ
MFA	หลักฐานยืนยันตัวตนชั้นที่สองนอกเหนือจากรหัสผ่าน ไม่มีข้อต่อรองรับมนุษย์
Encryption at rest / in transit / KMS	ข้อมูลถูกเข้ารหัสบนดิสก์ / บนสาย / บริการ managed ที่ถือกุญแจ
Security group	รายการกฎ firewall รายเครื่อง — “เว็บทราฟฟิกเข้า ได้จาก load balancer เท่านั้น”
Network segmentation / blast radius	การแบ่งเครือข่ายเป็นโซนมีกำแพง / ผู้โจมตีไปได้ ไกลแค่ไหนหลังเจาะหนึ่งครั้ง กำแพงที่วาดลวงหน้า คือตัวนิยาม
Zero trust	ตรวจสอบทุกคำขออย่างชัดเจน; ไม่เชื่อใจสิ่งใดเพียง เพราะอยู่ “ข้างใน” ทำที่เริ่มต้นยุคใหม่
Defense in depth	การควบคุมหลายชั้นซ้อนทับ เพื่อให้หนึ่งความล้ม เหลวไม่ถึงตาย
Secrets management	รหัสผ่าน กุญแจ และ token อาศัยในบริการ vault — ไม่อยู่ในโค้ด ไม่อยู่ในสเปรดชีต
Audit log / CloudTrail	บันทึกแก้ไขไม่ได้ว่าใครทำอะไร เมื่อไหร่ — วิธี ตรวจจับปัญหาและประกอบเรื่องย้อนหลัง
Data classification	การติดป้ายข้อมูลตามความอ่อนไหว (public / internal / personal / regulated) เพื่อให้การควบคุม ตรงกับป้าย ไม่ใช้การเดา
Data residency	การเก็บข้อมูลไว้ภายในพรมแดนประเทศทาง กายภาพ — ข้อกำหนดทางกฎหมายในหลาย อุตสาหกรรม และเป็นข้อมูลป้อนเข้าของ สถาปัตยกรรมเสมอ

คำศัพท์	ความหมาย
PDPA / GDPR	กฎหมายข้อมูลส่วนบุคคลของไทยและยุโรป: ความยินยอม การแจ้งเหตุร้าย (72 ชั่วโมง) กฎการโอนข้ามพรมแดน ช่วงที่ 4 ลงลึกกว่านี้

วิดีโอประจำโมดูลนี้ (ตรวจสอบลิงก์แล้ว): The AWS Shared Responsibility Model — Digital Cloud Training — ~4 นาที — <https://www.youtube.com/watch?v=ESPBBEK-cvo>

แบบฝึกหัด: (1) วาดแผนภาพ ThaiTicket แล้ววางเลย์เออร์ความปลอดภัยทับ: ทำเครื่องหมายทุก trust boundary ทุกจุดเข้ารหัส ทุกที่ที่ credential อาศัยอยู่ นิยามการวางเลย์เออร์นี้ — แผนภาพเดิม เล่นสื่ความปลอดภัย — คือสิ่งที่การรีวิวขอจากคุณเป๊ะ ๆ (2) อ่าน post-mortem สาธารณะของเหตุร้ายจากการตั้งค่าคลาวด์ผิดพลาดหนึ่งกรณี (Capital One ปี 2562 (ค.ศ. 2019) คือกรณีคลาสสิกสำหรับสอน) แล้วเขียนลงสมุดว่าแพตเทิร์นข้อไหนข้างบนที่จะป้องกันมันได้ (3) เล่นบทบาทสมมติกับ AI: มันรับบทผู้ก่อตั้งสตาร์ทอัปที่พูดว่า “เดี๋ยวค่อยเพิ่มความปลอดภัยทีหลัง” — โน้มน้าวเขาออกจากความคิดนั้นด้วยเลขคณิตต้นทุนของการถูกเจาะ อย่างสุภาพ

Milestone: คุณหยิบแผนภาพช่วงที่ 1 ของคุณอันไหนก็ได้มาทำเลย์เออร์ความปลอดภัยของมันได้ในยี่สิบนาที และอธิบาย least privilege, zero trust และ shared responsibility model ให้ผู้มีส่วนได้ส่วนเสียที่ไม่ใช่สายเทคนิคฟังได้โดยไม่มีศัพท์เทคนิค

โมดูล 6 (สัปดาห์ 11–12): Performance และ cost — ความเร็วกับเงิน สองเสาที่ควบคู่กัน

สองเสานี้ถูกสอนด้วยกันเพราะมันคือคั่นโยกเดียวกันที่ถูกดันคนละทิศ และสถาปนิกคือคนที่มีวางอยู่บนคั่นโยก

Performance efficiency — คำถามหลัก: ผู้ใช้จะรู้สึกถึงความช้าที่ไหนก่อน? อะไรถูกคำนวณซ้ำ ๆ ที่ควรคำนวณครั้งเดียวแล้ว cache ไว้? แต่ละส่วนประกอบเป็น*เครื่องมือที่ถูกต้องไหม* (SQL ที่ทำงานของ search engine คือบีกประสิทธิภาพของสถาปัตยกรรม ไม่ใช่ของโค้ด)? ประสิทธิภาพเปลี่ยนอย่างไรที่ 10× — และคอขวดแรกอยู่ไหน?

แพตเทิร์นด้าน performance: ชั้นของ caching — แพตเทิร์นประสิทธิภาพที่ให้ผลตอบแทนสูงสุดหนึ่งเดียว: cache ของเบราว์เซอร์ → **CDN** ที่ edge (เนื้อหา static เสิร์ฟจากเมืองใกล้ผู้ใช้; ดูดซับกราฟิกอ่านส่วนมหาดศาลได้) → cache ระดับแอปพลิเคชัน (**Redis** หน้า database สำหรับการอ่านร้อน: ผังที่นั่ง, session, หน้าสินค้า) → query cache ของ database แต่ละชั้นตอบคำขอก่อนที่มันจะไปถึงแกนกลางราคาแพง; คำถามออกแบบคือเสมอ *อะไร cache ได้ นานแค่ไหน และ invalidate อย่างไรเมื่อความจริงเปลี่ยน* จากนั้น: **read replica** (สำเนา database ที่รับงานอ่าน เพื่อให้ primary เกือบแรงไว้

เขียน) · **asynchronous processing** (อย่าให้ผู้ใช้งานที่ทำได้ — อีเมลใบเสร็จ, thumbnail, รายงาน) · **เครื่องมือถูกงาน** (ค้นหา → search engine; วิเคราะห์ → warehouse; lookup ร้อน → key-value store) · และ **วัดก่อนเสมอ** (งาน performance ที่ไม่มีการวัดคือโศกนาฏกรรม)

Cost optimization — คำถามหลัก: งานออกแบบนี้ราคาเดือนละเท่าไรที่โหดวันนี้ — และต่อหน่วย (ต่อออเดอร์ ต่อลูกค้า)? อะไรร้อนตอนตีสามทั้งที่ไม่จำเป็น? การใช้งานคงที่ส่วนไหนผู้กมได้รับส่วนลดได้? ฝ่ายการเงินจะพูดว่าแนวโน้มเป็นอย่างไร?

แพตเทิร์นด้าน cost: rightsizing (ฝูงเครื่องส่วนใหญ่สเปกเกินอย่างเจียบ ๆ; หดให้พอดีกับความต้องการที่วัดได้คือเงินฟรี) · **กลยุทธ์การจอง** — เมนูราคา: **on-demand** (ราคาเต็ม อีสระเต็ม) สำหรับโหดพุ่ง/ไม่รู้จัก, **reserved instances / savings plans** (ผูกมัด 1–3 ปี ลด 30–70%) สำหรับฐานที่คงที่, **spot** (ลดถึง 90% ถูกเรียกคืนได้ในเวลาสั้น) สำหรับงาน batch ที่ขัดจังหวะได้; ทำของสถาปนิกคือวางแผนชั้น: จองพื้นที่, auto-scale ตรงกลางด้วย on-demand, spot งาน batch · **storage lifecycle** (การจัด tier ของโมดูล 2 แบบอัตโนมัติ) · **ปิดของ** (environment ที่ไม่ใช่ production หลับกลางคืนและสุดสัปดาห์ ลดต้นทุนส่วนนั้นได้สองในสาม) · **จับตา egress** (ข้อมูลออกจากคลาวด์คิดเงิน; งานออกแบบที่คุยข้าม region พร่ำเพรื่อและการดาวน์โหลดสาธารณะก่อนใหญ่เซอร์ไพรส์ทุกคนมาแล้วหนึ่งครั้ง) · **tagging และ showback** (ทุกทรัพยากรติดป้าย owner/project เพื่อให้ทุกบาทมีเจ้าภาพ) · และเมตริกมกฏ **unit economics** (เศรษฐศาสตร์ต่อหน่วย): ไม่ใช่ “บิลคือ ฿800k/เดือน” แต่ “ต้นทุนต่อบัตรที่ขายคือ ฿1.90 และกำลังลดลง” บิลที่โตไม่เป็นไร; **ต้นทุนต่อหน่วย**ที่โตคือกลิ่นไม่ดีของสถาปัตยกรรม วินัยนี้มีชื่อ — **FinOps** — และสถาปนิกนั่งอยู่ตรงกลางของมัน

ตัวอย่างลงมือทำ — ThaiTicket ผ่านทั้งสองเลนส์ Performance: CloudFront เซิร์ฟหน้าศิลปินและรูปผังที่นั่ง (คนมาง 400,000 คนส่วนใหญ่ไม่เคยแตะ server เลย); Redis cache สถานะที่นั่งว่างด้วย TTL 2 วินาที — เก้าพอให้ถูก สดพอที่หน้าชำระเงิน (ซึ่งเช็กซ้ำกับ Aurora แหล่งความจริง) ป้องกันการขายซ้ำ; ใบเสร็จและตั๋วถูกสร้างแบบ asynchronous หลังชำระเงิน Cost: ฐานคงที่ 2,000 คน/ชั่วโมงรันบน savings plan (≈ลด 40%); ช่วงพุ่งตอนเปิดขายรัน on-demand เฉพาะชั่วโมงที่มันมืออยู่; งานวิเคราะห์รันบน spot ตอนกลางคืน; staging หลับนอกเวลางาน; ทุกทรัพยากรติด tag project:thaiticket ข้อเสนอถึง CFO อ่านว่า: “฿62,000/เดือนเป็นฐาน, ≈฿9,000 ต่ออีเวนต์เปิดขายใหญ่, ต้นทุนต่อบัตร ≈฿1.90 ลดลงตามปริมาณ” — และ *ประโยคนั้นคือเสียงของสถาปนิกที่คล่องเรื่องต้นทุน*

คำศัพท์:

คำศัพท์	ความหมาย
Latency / throughput	หนึ่งคำขอใช้เวลานานแค่ไหน / ระบบรับได้กี่คำขอต่อวินาที เกี่ยวข้องกัน แต่ไม่ใช่ตัวเดียวกัน
Bottleneck	จุดแคบสุดที่กำหนดจังหวะของทั้งระบบ การไปปรับที่อื่นคือการตกแต่ง

คำศัพท์	ความหมาย
Cache / TTL / invalidation	สำเนาเร็วของข้อมูลที่แพงต่อการไปเอา / เชื้อถื้อมันได้นานแค่ไหน / ปัญหาของการรีเฟรชเมื่อความจริงเปลี่ยน
CDN	cache ชั้นนอกสุด — เนื้อหาของคุณในเมืองหลายร้อยแห่ง คำตอบแรกของ “ทำให้เร็วทั่วโลก”
Read replica	สำเนา database ที่รับงานอ่าน เพื่อให้ primary เกือบกำลังไว้เขียน
Asynchronous processing	ทำงานไม่เร่งด่วนหลังตอบผู้ใช้ไปแล้ว มักผ่าน queue
On-demand / reserved / savings plan / spot	ยืดหยุ่นราคาเต็ม / ผูกมัด 1–3 ปีลด 30–70% / แนวคิดเดียวกันแต่ยืดหยุ่นกว่า / ลดถึง 90% แต่ถูกเรียกคืนได้ สถาปนิกวางทั้งสี่เป็นชั้น
Rightsizing	หดทรัพยากรสเปกเกินให้พอดีความต้องการที่วัดได้ เงินฟรีในแทบทุกบัญชี
Egress	ข้อมูลออกจากคลาวด์ — คิดเงินต่อ GB เซอร์ไพรส์ บิลชื่อดัง; ออกแบบการไหลของข้อมูลโดยคำนึงถึงมัน
Tagging / showback	ติดป้ายทรัพยากรด้วย owner และ project / โฉว์บิลของแต่ละทีมให้ทีมนั้นเห็น ความมีเจ้าภาพเปลี่ยนพฤติกรรม
Unit economics	ต้นทุนต่อหน่วยธุรกิจ (ต่อออเดอร์ ต่อผู้ใช้) เมตริกที่ทำให้บิลมีความหมาย — และประโยคโปรดของสถาปนิกต่อหน้า CFO
TCO	Total cost of ownership — ราคาหน้าป้ายบวกคนไลเซนส์ การย้าย และต้นทุนขาออก ตลอดชีวิตของทางเลือกหนึ่ง
FinOps	วินัยของการทำให้ค่าใช้จ่ายคลาวด์มองเห็นได้ จัดสรรได้ และถูกปรับให้ดีขึ้นต่อเนื่อง

วิดีโอประจำโมดูลนี้ (ตรวจสอบลิงก์แล้ว): What is FinOps? — FinOps Foundation (ช่องทางการ) — ~2 นาที — https://www.youtube.com/watch?v=Y-c_xw9bHFw

แบบฝึกหัด: (1) ตั้งราคาฐานของ ThaiTicket ใน AWS pricing calculator (ค้น “AWS pricing calculator” — การนั่งแจกแจงรายการงานออกแบบจริงสักครั้ง จะถอดมนตร์ทุกบทสนทนาต้นทุนในอนาคต); เปรียบยอดรวมของคุณกับ ฿62,000 ข้างบนแล้วอธิบายส่วนต่าง (2) เพิ่มเลเยอร์ caching ให้แผนภาพช่วงที่ 1 ของคุณสองอัน: ทำเครื่องหมายทุก cache, TTL ของมัน และเรื่องราวการ invalidate (3) ให้ AI รับบทวิศวกรที่อยากให้ทุกอย่างเป็น on-demand “เพื่อความสะดวก” — เจรจากกลยุทธ์การจอง พร้อมตัวเลข

Milestone: สำหรับงานออกแบบใดก็ตามที่คุณวาดไว้ คุณระบุรายการต้นทุนใหญ่สุดและคอขวดแรกที่น่าจะเป็นได้ — และเสนอหนึ่งการเปลี่ยนแปลงที่ปรับปรุงทั้งคู่พร้อมกัน (แทบมีเสมอ; ปกติคือ caching)

โมดูล 7 (สัปดาห์ 13–14): Operational excellence, sustainability — และการรันทั้งหกเสาพร้อมกัน

Operational excellence คือเสาที่ถามว่า: *มนุษย์รันเจ้าสิ่งนี้ได้จริง ๆ อย่างใจเย็นไหม?* คำถามหลัก: การเปลี่ยนแปลงไปถึง production อย่างไร — ผ่าน pipeline อัตโนมัติที่ทดสอบแล้ว หรือผ่านมนุษย์ผู้กล้า? เรารู้ได้อย่างไรว่าระบบแข็งแรงตอนนี้ได้ยวบยาบ? เมื่อมันพังตอนดีสาม วิศวกร on-call ทำอะไรกันแน่? แพตเทิร์น: **infrastructure as code** (environment ถูกนิยามในข้อความที่ผ่านการรีวิวและควบคุมเวอร์ชัน — Terraform/CloudFormation — จึงทำซ้ำได้และตรวจสอบได้; สถาปัตยกรรมที่มีอยู่แค่ในรูปแบบการคลิกคอนโซลคือนิทานพื้นบ้าน ไม่ใช่วิศวกรรม) · **CI/CD pipeline** พร้อมกลยุทธ์ deploy ที่ปลอดภัย (**blue-green**: ตั้งเวอร์ชันใหม่ข้างเวอร์ชันเก่าแล้วสลับ; **canary**: ให้เวอร์ชันใหม่รับทราฟฟิก 5% แล้วเฝ้าดู) · **observability** (log, metric, trace และ alert ที่ผูกกับอาการที่ผู้ใช้มองเห็น ไม่ใช่เรื่องจุกจิกของเครื่อง) · **runbook** (คู่มือลายลักษณ์อักษรว่าทำอะไรสำหรับแต่ละความล้มเหลวที่รู้จัก) · **blameless post-mortem** (หลังเหตุขัดข้อง: อะไรพัง ทำไม อะไรป้องกันการเกิดซ้ำ — ไม่มีผู้ร้าย ไม่จับคนเล็กพูดความจริง) บทบาทของสถาปนิก: *ออกแบบเพื่อความสามารถในการปฏิบัติการ* — ระบบที่ฉลาดขึ้นสองเท่าแต่สังเกตการณ์ได้ครั้งเดียว คือระบบที่แย่ง

Sustainability เสาที่หก ขอให้คุณเปลื้องโลกกายภาพน้อยลง: rightsize (CPU วางงานเผาไฟฟ้าจริง), scale ตามความต้องการแทนการเผื่อพิ๊งไว้ตลอด, ใช้ managed และ serverless (โครงสร้างพื้นฐานที่แชร์คือโครงสร้างพื้นฐานที่เต็มกว่า), จัด tier ให้ storage และลบของที่ตายแล้ว โชคดีที่ทางเลือกยังยืนกับทางเลือกประหยัดมักเป็นทางเลือกเดียวกัน — พูดยังสองอย่างในการรีวิวแล้วคุณจะชนะใจห้องสองต่อ

การรันทั้งหกเสาพร้อมกัน — Well-Architected review ครั้งแรกของคุณ ในการรีวิวจริง เสาหลักขัดแย้งกัน: reliability แบบ multi-region ดีกับ cost; การรีวิวความปลอดภัยเข้มงวดกับความเร็วปฏิบัติการ; caching เพื่อ performance ดีกับความถูกต้องสดใหม่ของข้อมูล framework ไม่แก้ความขัดแย้งให้คุณ —

มันบังคับให้ความขัดแย้งออกมาอยู่กลางแจ้ง ที่ซึ่งธุรกิจเลือกได้ นั่นคืออัจฉริยภาพทั้งหมดของมัน และเป็นของคนที่จะหยิบใช้ ตัววิธีการรีวิว (ชุดคำถาม การจัดลำดับ findings รายงาน) สอนเต็มในช่วงที่ 5 โมดูล 12; สองสัปดาห์นี้คุณซ้อมทักษะดิบ: การซักถามงานออกแบบเดียวจากหกทิศในหนึ่งการนั่ง

คำศัพท์:

คำศัพท์	ความหมาย
Infrastructure as Code (IaC) / Terraform	โครงสร้างพื้นฐานประกาศในไฟล์ข้อความควบคุมเวอร์ชันที่เครื่องมือแปลงเป็นความจริง / เครื่องมือยอดนิยมสุดของแนวนี้
CI/CD pipeline	สายพานอัตโนมัติที่ build ทดสอบ และ deploy ทุกการเปลี่ยนแปลง
Blue-green / canary deployment	สองแพตเทิร์นปล่อยของปลอดภัย: สลับกราฟฟิกระหว่างสำเนาเก่า-ใหม่ / หยดกราฟฟิกให้เวอร์ชันใหม่แล้วเฝ้าดู
Observability (logs, metrics, traces)	ความสามารถของระบบในการอธิบายตัวเอง: บันทึกเหตุการณ์ ตัวเลขตามเวลา และแผนที่การเดินทางรายคำขอ
Alert / on-call / runbook	การเพจอัตโนมัติเมื่อเกณฑ์แตก / เวรมนุษย์ที่รับสาย / สคริปต์ลายลักษณ์อักษรที่พวกเขาทำตาม
Blameless post-mortem	การทบทวนข้อผิดพลาดกับผู้ร้ายหลังเหตุขัดข้อง ที่ผลิตการป้องกัน ไม่ใช้การลงโทษ
Drift	ความเป็นจริงเบี่ยงจากนิยาม IaC เพราะมีคนไปคลิกอะไรบางอย่าง ศัตรูของการทำซ้ำได้
Toil	งานปฏิบัติการทำมือซ้ำซากที่ระบบอัตโนมัติควรดูดซับไปแล้ว สถาปนิกออกแบบให้มันหายไป
Well-Architected review	การซักถามอย่างมีโครงสร้างของ workload เทียบทั้งหกเสา ผลิต findings ที่จัดลำดับความสำคัญ ช่วงที่ 5 สอนคุณรันมัน

แบบฝึกหัด: (1) แบบฝึกหกเลนส์: หยิบแผนภาพที่ดีที่สุดของคุณมาใช้เวลาสิบนาทีต่อเสาเขียน findings — หกสิบนาที หนึ่งงานออกแบบ หกเลนส์; แบบฝึกนี้คือช่วงที่ 2 ฉบับย่อส่วน ทำซ้ำรายสัปดาห์จากนี้ไป (2) อ่านรายงานหลังเหตุขัดข้องของ AWS สองฉบับ (เผยแพร่บนหน้า status ของเขาหลังเหตุล่มใหญ่)

แล้วชี้ว่าแพตเทิร์นของเสาไหนล้มเหลว (3) ลงสมุด: สำหรับ ThaiTicket เขียนสามความขัดแย้งระหว่างเสาที่คุณจะยกให้ธุรกิจเห็น แต่ละอันเป็นทางเลือกหนึ่งประโยค (“เรามี X หรือ Y ได้ทั้งนี้ — เอาอันไหน?”)

Milestone — จบช่วงที่ 2: การรีวิวจำลอง: AI สร้างสถาปัตยกรรมมีตำหนิของสตาร์ทอัปสินเชื่ออนไลน์ไทย; คุณต้องผลิต findings ลายลักษณ์อักษรครบหกเสา รวมอย่างน้อย: ช่องโหว่ reliability (database แบบ single-AZ ของเขา), ช่องโหว่ security (IAM กว้างเกิน), ช่องโหว่ cost (ทุกอย่าง on-demand), ช่องโหว่ operations (ไม่มี IaC) และบทสนทนา RTO/RPO ที่เขาไม่เคยมี — แต่ละ finding เขียนในรูปแบบคำถามที่เพื่อนร่วมงานฟังได้โดยไม่สะดุ้ง เมื่อรายการ findings ของคุณอ่านแล้วเหมือนรุ่นพี่ใจดีมากกว่าผู้ตรวจสอบบัญชี ช่วงที่ 2 ก็เสร็จสิ้น

ช่วงที่ 3 — แฉตาล็อกแพตเทิร์น (สัปดาห์ 15–20)

สถาปนิกไม่คิดค้น; พวกเขา**คัดเลือก** ช่วงนี้คือแฉตาล็อกสถาปัตยกรรมมาตรฐานของคุณ — แต่ละตัวมี: มันคืออะไร ใช้เมื่อไหร่ *ไม่ควร*ใช้เมื่อไหร่ และโปรไฟล์ต้นทุน/ความซับซ้อน คอลัมน์ “เมื่อไม่ควร” คือสมการจริงของช่วงนี้: คอร์สไหนกับบอกคุณได้ว่า microservices คืออะไร; แต่การรู้ว่าเมื่อไหร่มันจะพังบริษัท คือสิ่งที่เขาจ้างคุณ ระหว่างเรียน ให้เข้าไปดู reference architecture จริงที่ <https://aws.amazon.com/architecture/> เรื่อย ๆ — การส่องหาแพตเทิร์นในป่าจริงคือแบบฝึกที่ทำให้แฉตาล็อกติดหนึบ

โมดูล 8 (สัปดาห์ 15–17): แพตเทิร์นระดับแอปพลิเคชัน

Monolith กับ microservices — วิวทեսภัยสุดของวงการ ยุติอย่างใจเย็น monolith (โมโนลิธ) คือแอปพลิเคชัน deploy ก้อนเดียวที่บรรจุทุกฟีเจอร์; **microservices** (ไมโครเซอร์วิส) แยกระบบเป็น service เล็ก ๆ จำนวนมากที่ deploy อิสระ แต่ละตัวเป็นเจ้าของข้อมูลตัวเอง คุยกันผ่าน API และ event ตารางฉบับช้อตลัด:

แพตเทิร์น	คืออะไร	ใช้เมื่อ	ห้ามใช้เมื่อ	ต้นทุน/ความซับซ้อน
Monolith	หนึ่งแอป หนึ่ง deployment หนึ่ง database	ทีมเล็ก ผลิตภัณฑ์อายุน้อย ขอบเขต domain ยังไม่ชัด — คือ ระบบใหม่ส่วนใหญ่	หลายทีมบล็อกกันเอง; แต่ละส่วนต้อง scale ต่างกันมาก	ซับซ้อนต่ำ ต้นทุนต่ำ; ขยายได้ไกลกว่าที่แพชชั่นยอมรับ — “น่าเบื่อ” คือพีเจอร์
Microservices	service เล็กจำนวนมาก สร้าง deploy และ scale อิสระ	หลายทีมต้องการจังหวัดปล่อยของอิสระ; แต่ละส่วน scale ต่างกันสุดขีด; ขอบเขต domain พิสูจน์แล้วว่านิ่ง	ทีมเล็ก (“distributed monolith” ก็คือ monolith ที่แถมความล้มเหลวของเครือข่ายมาด้วย”); domain ยังขยับอยู่	ซับซ้อนสูง: ทุก function call กลายเป็น network call ที่ฟังได้; ต้องมี CI/CD, observability, on-call ที่โตแล้ว

คำตัดสินของสถาปนิก: เริ่มแบบ monolith ที่แบ่งโมดูลภายในให้ดี; สกัด service ออกเมื่อ — และเมื่อเท่านั้น — ความเจ็บปวดด้านการขยายทีมหรือขยายโหนดมาถึงจริง พุดแบบนี้ในห้องสัมภาษณ์ พร้อมเหตุผล ทำเครื่องหมายคุณว่าเป็น senior; อุดมการณ์ไปทางไหนก็ตาม ทำเครื่องหมายคุณว่าเป็น junior

Event-driven architecture — queue กับ topic ใช้คัพของระบบ แทนที่ส่วนประกอบจะเรียกกันแล้วรอ (synchronous) ส่วนประกอบ publish event (“OrderPlaced”) เข้า queue (SQS — ผู้บริโภคหนึ่งรายรับแต่ละข้อความ ตามจังหวัดของตัวเอง) หรือ topic (SNS — ผู้ subscribe ทุกรายได้สำเนา; fan-out) ผู้ผลิตไม่รู้และไม่สนว่าใครฟังอยู่ ใช้เมื่อ: ส่วนประกอบควรรอดพ้นเหตุลุ่มของกันและกัน (queue ถือข้อความไว้ขณะผู้บริโภคดับ — ความทนทานและ decoupling ในการซื้อครั้งเดียว); โหลดพุ่งเป็นช่วง (queue ดูดซับ spike; worker ระบายมันอย่างสม่ำเสมอ); เหตุการณ์เดียวจุดชนวนหลายปฏิกิริยา (สั่งซื้อแล้ว → ตัดเงิน, อีเมล, สต็อก, วิเคราะห์ — สี่ subscriber ศูนย์ coupling) ห้ามใช้เมื่อ: ผู้ใช้ต้องการคำตอบเดียวนี้ (หน้าชำระเงิน confirm แบบ “เดี๋ยวค่อย” ไม่ได้); หรือทีมเล็กและการเรียก synchronous ง่าย ๆ ก็พอ — ทุก queue เพิ่ม delivery semantics (at-least-once แปลว่าผู้บริโภคต้อง idempotent — รับซ้ำสองครั้งได้อย่างปลอดภัย), การจัดการ dead-letter และภาระ monitoring ต้นทุน/ความซับซ้อน: ส่วนประกอบถูก แต่การดีบั๊กแพงขึ้น — เรื่องราวของคำขอตอนนี้กระจัดกระจายข้าม service และ queue ซึ่งเป็นเหตุที่ observability (โมดูล 7) เลิกเป็นทางเลือกตรงนี้

Three-tier: เป็นของคุณอยู่แล้ว (โมดูล 3) ใช้กับ: เว็บแอปพลิเคชันส่วนกลางอันกว้างใหญ่ — มันคือแพตเทิร์นที่ตัวอื่นถูกวัดเทียบ ไม่เหมาะกับ: workload พุ่งจัดที่มีช่วงเียบว่างงาน (serverless ถูกกว่า) หรือผลิตภัณฑ์หลายทีมขนาดมหึมาจริง ๆ (ดูข้างบน) โพรไฟล์: เป็นที่เข้าใจทุกหนแห่ง ง่ายคนง่าย ต้นทุนปานกลาง

Serverless-first: ประกอบชิ้นส่วน managed — API Gateway → Lambda function → DynamoDB, S3 และ queue — โดยไม่เป็นเจ้าของ server เลย ใช้เมื่อ: ทราฟฟิกพุ่งหรือต่ำ (scale-to-zero แปลว่า ต้นทุนตอนว่าง ~ศูนย์), ทีมเล็ก, ต้องออกตลาดเร็ว, กาวเชื่อม event-driven ห้ามใช้เมื่อ: งานประมวลผล ยาวหรือเฉพาะทาง (ลิมิต runtime), เส้นทางที่ไวต่อ latency สุดขีด (cold start), โหลดคงที่มหาศาล (ราคาต่อการเรียกอาจแพงกว่า server ที่จองไว้ — ทำเลขนิดที่สเกล) หรือเมื่อความพกพาข้ามคลาวด์ เป็นข้อกำหนดจริง (นี่คือ lock-in ลึกสุดของทุกแพลตฟอร์ม — มักคุ้มค่า แต่ต้องพูดออกมาดัง ๆ) โปรไฟล์: ภาวะปฏิบัติการต่ำสุดในเค็ดตาล็อก; ต้นทุนเยี่ยมยอดที่สเกลต่ำ/พุ่ง ต้องตรวจสอบที่สเกลคงที่สูง

คำศัพท์:

คำศัพท์	ความหมาย
Monolith / modular monolith	แอป deploy ก้อนเดียวที่มีทุกอย่างข้างใน / ฉบับมี วินัย: deploy ก้อนเดียว ขอบเขตโมดูลภายในสะอาด — ค่าเริ่มต้นที่ดีที่สุดของระบบใหม่
Microservices	service เล็กจำนวนมาก deploy อิสระ แต่ละตัวเป็น เจ้าของข้อมูลตัวเอง เครื่องมือขยายทีมที่คิดภาษี distributed systems
Coupling / decoupling	ส่วนประกอบพึ่งพาความพร้อมและรายละเอียดของ กันแค่ไหน สถาปัตยกรรมโดยมากคือศิลปะการซื้อ decoupling ในปริมาณที่ถูกต้อง
Synchronous / asynchronous	เรียกแล้วรอ vs ส่งแล้วไปต่อ ทางเลือกพื้นฐานบนทุก ลูกศรที่คุณวาด
Event / event-driven	ข้อเท็จจริงที่ประกาศให้ใครก็ตามที่ฟัง (“OrderPlaced”) / สถาปัตยกรรมที่สร้างจากประกาศ เหล่านั้น
Queue (SQS)	แถวของข้อความ; แต่ละอันถูกรับโดยผู้บริโภคนึง รายตามจังหวะของตัวเอง ใช้อัฟและตัว decouple
Topic / pub-sub (SNS)	ช่องกระจายเสียง; ผู้ subscribe ทุกรายได้ทุก ข้อความ เครื่องมือ fan-out
Dead-letter queue	ที่ที่ข้อความซึ่งประมวลผลล้มเหลวซ้ำ ๆ ถูกพักไว้ให้ มนุษย์ดู — ตาข่ายนิรภัยของแพลตฟอร์ม
Idempotent	ประมวลผลซ้ำสองครั้งแล้วได้ผลเดิมอย่างปลอดภัย — คุณสมบัติบังคับของผู้บริโภค เพราะ queue อาจ

คำศัพท์	ความหมาย
	ส่งข้อความมากกว่าหนึ่งครั้ง
Serverless-first	การประกอบชิ้นส่วน managed แบบ scale-to-zero (function, managed DB, queue) แทนการรัน server
Lock-in	การพึ่งพาบริการเฉพาะของผู้ให้บริการรายเดียว ดี ราคาเป็นต้นทุนการย้ายออกไม่ใช้บาป — เป็นศัพท์ เศรษฐศาสตร์ที่ต้องใช้อย่างเปิดเผย (ช่วงที่ 4)

แบบฝึกหัด: (1) ออกแบบ flow คำสั่งซื้อของ ThaiTicket สองแบบ — three-tier แบบ synchronous แล้วก็ event-driven ด้วย SQS/SNS — และเขียนหนึ่งย่อหน้าว่าแต่ละเวอร์ชันทำอะไรระหว่างที่ผู้ให้บริการ ชำระเงินล้ม; ย่อนำนั่นคือข้อโต้แย้งทั้งหมดของฝั่ง event (2) สำหรับสี่บริษัท (สตาร์ทอัพ 3 คน, scale-up วิศวกร 50 คน, ธนาคาร, แอปโหวตรายการทีวีที่ใช้ 4 คิน/ปี) เลือกแพตเทิร์นและแก้ต่างให้มัน — แล้วเอ่ยชื่อทริกเกอร์ที่จะทำให้แต่ละบริษัทเปลี่ยนแพตเทิร์น (3) หาเรื่องเล่าจากบล็อกวิศวกรรมจริงหนึ่ง เรื่องแนว “เราย้ายไป microservices แล้วเสียใจ” กับหนึ่งเรื่องสำเร็จ; ลงสมุดว่าต่างกันตรงไหน (แทบทุก ครั้งคือขนาดทีมกับบุคลิกภาวะของ domain)

Milestone: เมื่อได้คำบรรยายบริษัท คุณแนะนำแพตเทิร์นพร้อมทางลงของมันได้ — “เริ่มตรงนี้; เมื่อ X เกิด วิวัฒนาการไป Y” — ในห้านาที เส้นทางวิวัฒนาการ ไม่ใช่คำพิพากษา คือวิธีที่สถาปนิกพูดกันจริง ๆ

โมดูล 9 (สัปดาห์ 18–20): แพตเทิร์นด้านข้อมูล ระดับโลก และ hybrid

สถาปัตยกรรมข้อมูล — lake กับ warehouse database เชิงธุรกรรม (ช่วงที่ 1) รันธุรกิจ; ฝั่งวิเคราะห์ อยากรู้คำถามกับมันโดยไม่ทำให้มันช้าลง **data warehouse** (คลังข้อมูล — Redshift, Snowflake, BigQuery) เก็บข้อมูลที่มีโครงสร้าง ถูกทำความสะอาดแล้ว ปรับแต่งเพื่อการวิเคราะห์ SQL ที่เร็ว — ใช้เมื่อคำถามเป็นที่รู้จักและเดสคอปต์ต้องไว; แพงกว่าต่อ TB ต้องมีวินัยการโมเดลข้อมูลตั้งแต่ต้น **data lake** (ทะเลสาบข้อมูล — S3 + catalog + เอนจิน query อย่าง Athena) เก็บทุกอย่าง ดิบ ๆ ถูก ๆ — log, clickstream, รูปภาพ — และใส่โครงสร้างตอนอ่าน; ใช้เมื่ออยากเก็บทุกอย่างตอนนี้แล้วค่อยตัดสินใจ เรื่องคำถามทีหลัง; แต่ถ้าไร้ธรรมาภิบาลมันจะเลื่อมเป็นมุกตลกของวงการ **data swamp** (บึงข้อมูล) ฉันทามติสมัยใหม่คือใช้ทั้งคู่ วางเป็นชั้น (“lakehouse”): ความจริงดิบใน lake, mart ที่คัดสรรใน warehouse ป้อนด้วย **ETL/ELT pipeline** กฎของสถาปนิก: ฝั่งวิเคราะห์ห้าม query database production เด็ดขาด (ใช้ replica หรือ pipeline ป้อนมัน) และทุกชุดข้อมูลมีเจ้าของ มีรายการใน catalog และมีนโยบาย retention — บรรทัดธรรมาภิบาลนั่นคือหนึ่งประโยคในงานออกแบบ และหนึ่งปีแห่ง ความเจ็บปวดถ้าคุณละไว้

Multi-region: active-passive กับ active-active เมื่อ region เดียวไม่พอ — เพราะธุรกิจต้องการ DR จากภัยพิบัติระดับ region หรือผู้ใช้กระจายข้ามทวีป — คุณเลือก:

แพตเทิร์น	คืออะไร	ใช้เมื่อ	ห้ามใช้เมื่อ	ต้นทุน/ความซับซ้อน
Active-passive	region เดียวให้บริการ; region สำรองถือข้อมูลที่ replicate ไว้ ตั้งแต่ “แค่ backup” (cold) จนถึง “สำเนาบางส่วนที่ร้อนอยู่” (warm) ถูกโปรโมตเมื่อเกิดภัยพิบัติ	RTO/RPO ที่ธุรกิจประกาศไว้คุ้มกับมัน; compliance เรียกร้องเรื่องราว DR	ไม่มีใครเห็น RTO/RPO ที่คุ้มค่าใช้จ่าย (ความต้องการซื้อสัตย์ของบริษัทส่วนใหญ่คือ multi-AZ ดี ๆ + backup ข้าม region)	ต้นทุน 1.1×–1.7× ตามความอุ่นของ standby; ซับซ้อนปานกลาง — ข้อกำหนดมาตรการคือ <i>failover ที่คุณซ้อมจริง</i> ไม่จริง standby คือละครฉากหนึ่ง
Active-active	สอง region ขึ้นไปให้บริการพร้อมกัน; ผู้ใช้ถูกนำทางไปที่ใกล้สุด; ข้อมูล replicate สองทาง	ฐานผู้ใช้ทั่วโลกที่ต้องการ latency ต่ำ; ต้องการ RTO ใกล้ศูนย์จริง ๆ (เครือข่ายชำระเงิน, เทรดดิ้ง, SaaS รายใหญ่)	แทบทุกคนที่เหลือ — <i>write conflict</i> ข้าม region คือหนึ่งในปัญหายากแท้จริงของวงการคอมพิวเตอร์	ต้นทุน 2×+ ซับซ้อนสูงสุดในหลักสูตรนี้; ต้องการ data store ที่แก้ conflict ได้และทีมระดับ senior

ประโยคระดับห้องสัมมนา: “*Multi-AZ is table stakes; multi-region is a business case.*” (*multi-AZ คือเดิมพันขั้นต่ำ; multi-region คือ business case*) ให้ธุรกิจประกาศ RTO/RPO ตีราคาทางเลือก แล้วปล่อยให้ตัวเลขเป็นผู้เลือก

Hybrid cloud ส่วนหนึ่ง on-prem ส่วนหนึ่งคลาวด์ เชื่อมด้วย VPN หรือ Direct Connect — สำหรับองค์กรที่ตั้งตัวแล้วส่วนใหญ่ไม่ใช่แพตเทิร์นแต่เป็น*ความจริงยาวนานบนทศวรรษ*: mainframe ที่ย้ายไม่ได้, ระบบโรงงานที่ติดพันธนาคาร latency, ข้อมูลได้กำกับที่ถูกปึกหมุดไว้ on-premises และการ migrate (ช่วงที่ 4) ที่กำลังเดินผ่าน ใช้ในฐานะ: สะพานที่ใจจ้องสร้าง พร้อมทิศทางการเดินทาง ห้ามยอมรับ: “hybrid” ในฐานะคำสวยของ “เราไม่เคยตัดสินใจ” โน้ตการออกแบบ: identity ต้องรวมเป็นหนึ่งก่อน (ล็อกอินเดียวข้ามสองโลก — สิ่ง JD องค์กรหมายถึงด้วยคำ “hybrid identity integration”); ระบุชื่อว่าเป็นระบบไหนคือ source of truth; จับตาต้นทุน egress ข้ามสาย; และคาดหวังว่าลิงก์เครือข่ายจะเป็น SPOF เว้นแต่มีสองเส้น ความซับซ้อน: ทุกอย่างคุณสองอาณาจักร — เหตุผลซื้อสัตย์ที่สถาปนิกผลักดันให้หัดฝัง on-prem ลงเรื่อย ๆ

คำศัพท์:

คำศัพท์	ความหมาย
OLTP / OLAP	การประมวลผลธุรกรรม (อ่าน/เขียนเล็กเร็วจำนวนมาก — ธุรกิจ) vs การประมวลผลวิเคราะห์ (สแกนโมโหฟาร์ — ศึกษาธุรกิจ) แยกมันออกจากกัน
Data warehouse	ที่เก็บมีโครงสร้าง คัดสรรแล้ว ปรับแต่งเพื่อวิเคราะห์ SQL ใด (Redshift, Snowflake, BigQuery)
Data lake / data swamp	ที่เก็บทุกอย่างของทุกอย่างแบบดิบ ใส่โครงสร้างตอนอ่าน (S3 + Athena) / ฉบับไร้ธรรมาภิบาล ที่ซึ่งข้อมูลไปเพื่อสาบสูญ
ETL / ELT	pipeline ที่ย้ายข้อมูลจากระบบต้นทางเข้า lake/warehouse (Extract, Transform, Load — ลำดับสลับได้)
Data governance / catalog / retention	กฎความเป็นเจ้าของ เอกสาร และอายุขัยของทุกชุดข้อมูล — หนึ่งในประโยชน์ที่ประหยัดหนึ่งปีแห่งความเจ็บปวด
Replication (sync / async)	การคัดลอกข้อมูลต่อเนื่องไปยัง database หรือ region อื่น — สอดคล้องทันทีแต่จำกัดระยะทาง vs ช้ากว่านิดแต่ไปได้ทุกที่ ความหน่วงของ async คือที่มาของ RPO
Active-passive / failover drill	DR แบบ region สำรอง / การซ้อมตามกำหนดที่พิสูจน์ว่ามันทำงาน failover ที่ไม่เคยซ้อมคือความหวัง
Active-active	หลาย region ให้บริการพร้อมกันพร้อม replication สองทาง ทรงพลัง; ยากแท้จริง; ส่วนใหญ่ไม่จำเป็น
Write conflict	สอง region แก้ข้อมูลเดียวกันพร้อมกัน — เหตุผลทางเทคนิคที่ active-active คือแดนผู้เชี่ยวชาญ
Hybrid cloud / hybrid identity	on-prem + คลาวด์เชื่อมเป็นอาณาจักรเดียว / การลงชื่อเข้าใช้เดียวข้ามทั้งคู่ — สิ่งแรกที่ต้องรวมเป็นหนึ่ง

คำศัพท์	ความหมาย
Direct Connect	สายกายภาพส่วนตัวเชื่อม data center เข้าคลาวด์ — สายเสถียร hybrid ต้องมีสองเส้นถ้ามันสำคัญ

แบบฝึกหัด: (1) ThaiTicket ไประดับภูมิภาค: ออกแบบการขยายไปสิงคโปร์สองแบบ — active-passive (กรุงเทพฯ เป็นหลัก) และ active-active — พร้อมตัวคูณต้นทุนและ RTO/RPO ของแต่ละแบบ; เขียนคำแนะนำหนึ่งหน้าและ *พันธ* (2) ผู้ค้าปลีกมีข้อมูลขาย 15 ปีใน SQL database ฝั่ง production และอยากได้ “AI-ready analytics”; สเก็ทงานออกแบบ lake + warehouse และเขียนสามประโยคธรรมชาติ (3) สองแพตเทิร์น: เลือก reference architecture สามชิ้นจาก <https://aws.amazon.com/architecture/> แล้วเอ่ยชื่อทุกแพตเทิร์นในเค้ตตาล็อกที่ปรากฏในแต่ละชิ้น

Milestone — จบช่วงที่ 3: ด้านเค้ตตาล็อก: AI ยิงหกลสถานการณ์รัว ๆ; แต่ละอันคุณเอ่ยชื่อแพตเทิร์นเหตุผล คำเตือนเมื่อไม่ควรใช้ และโปรไฟล์ต้นทุน/ความซับซ้อน — ไม่เกินห้านาทีต่อข้อ บทแลกเปลี่ยนแบบนี้เป๊ะ ๆ ที่ความเร็วนี้เป๊ะ ๆ คือหนึ่งในสามส่วนกลางของการสัมภาษณ์สถาปนิกจริง

ช่วงที่ 4 — การย้ายระบบและโลกจริง (สัปดาห์ 21–26)

งานออกแบบ greenfield คือส่วนน้อยของอาชีพ สถาปัตยกรรมส่วนใหญ่เกิดในบริษัทที่มีอยู่แล้ว — พร้อมห้อง server, ซอฟต์แวร์โบราณที่ธุรกิจพึ่งพา, สัญญา และกฎหมาย ช่วงนี้คือโลกใบนั้น และเป็นที่ที่ “lead migration and modernization initiatives” — หัวข้อในแทบทุก JD ที่เราศึกษา — ถูกสอน

โมดูล 10 (สัปดาห์ 21–23): Migration — พาบริษัทขึ้นคลาวด์

The 7 Rs — คลังศัพท์ร่วมของทุกบทสนทนา migration สำหรับแต่ละ workload ในอาณาจักรระบบ คุณเลือกหนึ่ง:

R	ความหมาย	เมื่อ	แรงที่ใช้ / ผลตอบแทน
Retire	ปิดมันทิ้ง — ไม่มีใครใช้จริง	ทุกอาณาจักรมีแบบนี้ 10–20%; หามันให้เจอก่อน	จ๊ับจ๊อย / ประหยัดทันที — R ที่ดีที่สุด
Retain	คงไว้ on-prem ไปก่อน	ระบบที่ติดพันธนาคาร latency, ถูกปักหมุดด้วย	ไม่มี / เลื่อนต้นทุน — คำว่า “ยังไม่ถึงเวลา”

R	ความหมาย	เมื่อ	แรงที่ใช้ / ผลตอบแทน
		compliance หรือใกล้ หมดอายุขัย	อย่างซื้อสตัย
Rehost (“lift and shift”)	ย้ายขึ้น VM คลาวด์ตาม สภาพ	ความเร็วสำคัญ แอป เสถียร ทักษะยังบาง	ต่ำ / ออกจาก data center ได้เร็ว แต่ยังได้ ประโยชน์คลาวด์น้อย — ก้าวแรก ไม่ใช่จุด หมาย
Relocate	ย้ายที่ระดับ hypervisor (เช่น ผุง VMware ไป VMware-on-cloud)	อาณาจักร virtualized ใหญ่ที่มีเดดไลน์	ต่ำ / การย้ายก้อนใหญ่ เร็วสุด; จุดพักระหว่าง ทาง
Repurchase (“drop and shop”)	แทนที่ด้วย SaaS	แอปที่ไม่สร้างความต่าง — อีเมล, HR, CRM	ต่ำ-กลาง / ทั้งหมดหาย ไปจากอาณาจักรของคุณ
Replatform (“lift, tinker, shift”)	อัปเดตเล็ก ๆ ระหว่าง ทาง — DB ดูแลเอง → RDS, แอป → container	ตรงกลางแบบปฏิบัติ นิยม: ประโยชน์จริง ความเสี่ยงจำกัด	กลาง / R ม้างานหลัก ของ migration ส่วน ใหญ่
Refactor / re- architect	เขียนใหม่แบบ cloud- native (แพตเทิร์นช่วงที่ 3)	ระบบหลักที่สร้างความ ต่าง ซึ่งขีดจำกัดของมัน ทำร้ายธุรกิจอยู่	สูง / ผลตอบแทนสูงสุด — ใช้กับ workload ไม่ก็ ตัวที่สมควรได้

วิธีการ migration: assess → mobilize → migrate **Assess (ประเมิน):** ทำ inventory ทุกอย่าง (เครื่องมือ discovery บวกโบราณคดีของการไปถามคน) และให้คะแนนแต่ละ workload ตามคุณค่า ธุรกิจกับความยากในการย้าย — ผลลัพธ์: application inventory ที่มี R กำกับทุกแถว และ **business case** (TCO ของการอยู่ต่อ vs การย้าย; ข้อสตัยว่าบิลจะสูงขึ้นช่วงคาบเกี่ยวที่ทั้งสองอาณาจักรรันพร้อมกัน — ผู้นำที่ไม่ถูกเตือนเรื่อง migration bubble คือผู้นำที่ยกเลิก migration กลางคัน) **Mobilize (เตรียมพร้อม):** สร้าง **landing zone** — รากฐานคลาวด์ที่สร้างไว้ล่วงหน้าและมีธรรมาภิบาล *ก่อน* workload ตัวแรกจะลงจอด: โครงสร้างหลายบัญชี (แยกบัญชีต่อ environment และต่อทีม เพื่อให้รัศมีความเสียหายเล็ก), identity และ logging รวมศูนย์, ฮับเครือข่าย (VPC แบบ hub-and-spoke, Direct Connect กลับไป on-prem) และ **guardrail** — นโยบายอัตโนมัติที่ทำให้เส้นทางปลอดภัย ติด tag และ compliant เป็นค่าเริ่มต้น (AWS Control Tower คือชุดเริ่มต้นแบบ managed) JD บอกว่า “design the landing zone every workload inherits” — นี่แหละคือสิ่งนั้น การข้ามมันเพื่อ “เริ่มย้ายเลย” คือการสร้าง data center รกรุงรัง ใบเดิมขึ้นใหม่บนคลาวด์ในราคาแพงกว่า; มันคือลายเซ็นคลาสสิกของ migration ที่ล้มเหลว **Migrate เป็นระลอก:** จัดกลุ่ม workload เป็นระลอกละไม่กี่ตัว เรียงง่ายก่อน: ระลอกที่ 1 จงใจเลือกเดิมพันต่ำ

(เรียนรู้เครื่องจักรตรงที่ความผิดพลาดราคาถูก) ระลอกหลัง ๆ พาอัณณานิยมกันไปพร้อม cutover ที่ซ่อมแล้ว แต่ละระลอกมีแผน rollback และช่วง **hypercare** ของการเฝ้าดูเข้มข้น สำหรับการย้าย database แต่ละตัว สองคำถามที่สำคัญคือตัวเลขของโมดูล 4 ในกรอบล้อมตัว: cutover ยอมให้ downtime เท่าไหร่ (RTO) — และ replication ต่อเนื่องพร้อม switchover สั้น ๆ มีอยู่จริงเมื่อคำตอบคือ “แทบเป็นศูนย์”

คำศัพท์:

คำศัพท์	ความหมาย
7 Rs	Retire, retain, rehost, relocate, repurchase, replatform, refactor — เมนู migration ราย workload (คุณจะได้ยิน “6 Rs” ด้วย — รายการรุ่นเก่าที่ไม่มี relocate)
Discovery / application inventory	การค้นหาวะอะไรรันอยู่จริง (เครื่องมือ + สัมภาษณ์) / รายการผลลัพธ์ หนึ่งแถวต่อ workload พร้อมเจ้าของ dependency และ R ของมัน
Dependency mapping	การทำผังว่าอะไรคุยกับอะไร — เหตุผลที่ workload ย้ายเป็นกลุ่ม และ migration ที่ไม่มีมันล้มเหลวตั้งแต่วันแรก
Business case / migration bubble	ข้อโต้แย้ง TCO อยู่-หรือ-ย้าย / โหนดต้นทุนชั่วคราว ขณะสองอาณาจักรรันพร้อมกัน เตือนเรื่องมัน ไม่จัน ถูกมันชุ่มโจมตี
Landing zone	รากฐานคลาวด์มีธรรมาภิบาลที่สร้างไว้ล่วงหน้า — บัญชี identity เครือข่าย logging guardrail — ที่ทุก workload สืบทอด สร้างก่อน migration
Multi-account strategy	แยกบัญชีคลาวด์ต่อ environment/ทีม เพื่อให้บิลมีเจ้าภาพและรัศมีความเสียหายถูกจำกัด
Guardrail	นโยบายอัตโนมัติที่ป้องกันหรือชี้แจงการกระทำที่ไม่ compliant — ธรรมาภิบาลในรูปเครื่องจักร ไม่ใช่บันทึกเวียน
Control Tower	บริการ managed ของ AWS สำหรับตั้ง landing zone หลายบัญชีพร้อม guardrail
Wave plan	ตาราง migration เป็นกลุ่มเล็ก ง่ายก่อน dependency ไปด้วยกัน

คำศัพท์	ความหมาย
Cutover / rollback / hypercare	วินาทีแห่งการสลับไปใช้สำเนาบนคลาวด์ / การย้อนกลับที่ซ้อมไว้ / หน้าต่างซัพพอร์ตเข้มข้นหลังจากนั้น

แบบฝึกหัด: (1) migration บนกระดาน: AI สร้าง inventory บริษัทสมมติ 40 server (คุณจะไม่เจอมันอีกครั้งในฐานะ Capstone 2); กำหนด R ให้ทุกแถวและแก้ต่างสืบทอดตัดสินใจที่ยากที่สุด (2) สเก็ทซ์ landing zone: แผนภาพบัญชี, การไหลของ identity, เครือข่าย hub-and-spoke, guardrail ห้าข้อที่คุณจะบังคับใช้ตั้งแต่วันแรก (3) เขียนคำเตือน “migration bubble” สองย่อหน้าถึง CFO — การซ้อมส่งข่าวร้ายแต่เนิ่น ๆ คือคาร์ดิโอของสถาปนิก

Milestone: เมื่อได้ inventory ~20 workload พร้อมคำบรรยาย คุณผลิตตาราง R-ต่อ-workload ที่แตกต่างกันได้, แผนสามระลอกพร้อมเหตุผล และสเก็ทซ์ landing zone — ในการนั่งครั้งเดียว

โมดูล 11 (สัปดาห์ 24–26): ข้อจำกัด — กฎหมาย ระบบเก่า และ lock-in

Data residency และกฎหมายความเป็นส่วนตัว ในฐานะข้อมูลป้อนเข้าของสถาปัตยกรรม กฎหมายข้อมูลส่วนบุคคล — PDPA ของไทย, GDPR ของยุโรป และญาติ ๆ ทั่วโลก — มีรูปทรงร่วมที่สถาปนิกต้องรู้แม่นยำ: ข้อมูลส่วนบุคคลต้องมีฐานทางกฎหมาย (บ่อยครั้งคือความยินยอม); ปังเจกมีสิทธิ (เข้าถึง แก้ไข ลบ — งานออกแบบของคุณต้องหาและลบข้อมูลของคนหนึ่งคนได้ ซึ่งยากถ้าคุณโปรยมันกระจายในสำเนาไร้ธรรมาภิบาล); เหตุรั้วมีหน้าที่แจ้งภายใน 72 ชั่วโมง (ระบบ logging ของคุณต้องทำให้เล่าเรื่องได้เร็วกว่านั้น); และ**กฎการโอนข้ามพรมแดน**จำกัดว่าข้อมูลอาศัยอยู่ที่ไหนได้ทางกายภาพ — ซึ่งเป็นข้อจำกัดของการเลือก region, ข้อจำกัดของการออกแบบ replication (สำเนาวิเคราะห์ที่สิงคโปร์อาจเป็นเหตุการณ์ทางกฎหมาย) และเหตุผลที่ region ในประเทศมีอยู่ วิธีการของสถาปนิก ตามลำดับ: จัดชั้นข้อมูล (อะไรคือข้อมูลส่วนบุคคล?), ทำแผนที่การเดินทางของมัน (ทุกที่เก็บ ทุกสำเนา ทุกพรมแดนที่ข้าม — การไหลที่ไม่มีใครวาดคือที่อยู่ของการละเมิด) แล้วออกแบบการควบคุม: region ที่ตรงตาม residency, การเข้ารหัส, ตาราง retention และเส้นทางการลบที่ไปถึงการหมดอายุของ backup จริง ๆ พูดคำว่า “data protection by design” — มันคือวลีที่กฎหมายทั้งสองใช้ และมันคือชื่อตำแหน่งงานของคุณในหนึ่งประโยคแบบตรงตัวอักษร

Legacy integration (การเชื่อมระบบเก่า) ระบบบิลลิ่งบน mainframe ไม่ย้ายปีไหน ๆ และแอปคลาวด์เฝ้ามองต้องคุยกับมัน แพตเทิร์น: **anti-corruption layer** — บริการแปลภาษาระหว่างใหม่กับเก่า เพื่อให้รูปทรงข้อมูลฟิลิกของระบบ legacy ไม่รั่วเข้ามาทำเสียงงานออกแบบใหม่ของคุณ; **strangler fig** — นำทราฟฟิกผ่าน façade แล้วลอกฟังก์ชันออกจากระบบ legacy ทีละชิ้น จนหลายปีให้หลังมันถูกปิดสวิตช์ได้ (ตั้งชื่อตามต้นไม้ที่ค่อย ๆ โอบรัดต้นไม้เจ้าบ้าน; มันชนะการเขียนใหม่แบบ big-bang ซึ่งล้มเหลวในอัตราระดับตำนาน); สะพาน batch และจุดแตะ event สำหรับข้อมูลที่ต้องไหลข้ามโลก; และความเคารพต่อฟิลิกส์ latency ของการเรียก cloud-ถึง-on-prem แบบพรวดพราด (Direct Connect ช่วยได้;

ดีกว่านั้นคือออกแบบให้ความพราะเพรื่อหายไป) กฎ: **จงกักเก็บ legacy อย่าไปติดมัน** — ทุกส่วนประกอบใหม่ควรถูกสร้างรากับระบบ legacy หายไปแล้ว

เศรษฐศาสตร์ vendor lock-in ทุก managed service แสนสะดวกทำให้การแต่งงานกับผู้ให้บริการรายเดียวลึกขึ้น; ความพกพา (abstraction ข้ามคลาวด์, ดูแลเองทุกอย่าง) เป็นทางเลือกจริงที่มีราคาจริงในรูปความซับซ้อนและความเร็วที่สละไป ทั้งคู่ไม่ใช่บาป — โหมดลั้มเหลวไม่ใช่การเลือก lock-in แต่คือการไม่สังเกต เครื่องมือของสถาปนิกคือการประเมินต้นทุนขาออกเขียนลงในงานออกแบบ: “การใช้ DynamoDB ประหยัด ≈2 ปีวิศวกรตอนนี้; การย้ายทีหลัง ≈ การเขียน data layer ใหม่ 6 เดือน — เรายอมรับสิ่งนี้ และนี่คือขอบเขต interface ที่จะทำให้การเขียนใหม่ลดลง” สามประโยค ดีราคาซื้อสัตย์ การตัดสินใจถูกบันทึก (ADR ของช่วงที่ 5) — นั่นคือการจัดการ lock-in แบบผู้ใหญ multi-cloud ในฐานะกลยุทธ์ (รัน workload เดียวกันแบบพกพบบนสองคลาวด์) มักเป็นคำตอบที่แพงที่สุดเท่าที่เป็นไปได้ และถูกซื้อโดยองค์กรที่สเกลหรือผู้กำกับดูแลเรียกร้อมันเท่านั้น; multi-cloud ในฐานะข้อเท็จจริง (ต่าง workload อยู่ต่างคลาวด์ตามประวัติศาสตร์หรือความเหมาะสม) คือชีวิตปกติ

คำศัพท์:

คำศัพท์	ความหมาย
PDPA / GDPR	กฎหมายคุ้มครองข้อมูลส่วนบุคคลของไทยและยุโรป — คู่แม่แบบของข้อจำกัดกฎหมายความเป็นส่วนตัวทั่วโลก
Data protection by design	การสร้างการควบคุมความเป็นส่วนตัวเข้าไปในสถาปัตยกรรมตั้งแต่สเก็ทแรก — วลีทางกฎหมายและหน้าที่ของสถาปนิก
Lawful basis / consent	เหตุอันชอบด้วยกฎหมายที่จำเป็นต่อการประมวลผลข้อมูลส่วนบุคคล
Right to erasure	สิทธิของปัจเจกที่จะให้ข้อมูลของตนถูกลบ — ซึ่งงานออกแบบของคุณต้องทำให้เป็นไปได้
Data flow mapping	การทำผังทุกที่เก็บ ทุกสำเนา และทุกการข้ามพรมแดนของข้อมูลหมวดหนึ่ง ที่ที่การไหลไม่ถูกวาดการละเมิดอาศัยอยู่
Cross-border transfer	ข้อมูลส่วนบุคคลออกนอกประเทศ — ถูกกำกับ; ทำให้การออกแบบ replication เป็นคำถามทางกฎหมาย

คำศัพท์	ความหมาย
Anti-corruption layer	ส่วนประกอบแปลภาษาที่กันไม่ให้ความรู้สึกของระบบ legacy รวบรวมงานออกแบบใหม่
Strangler fig	การ modernize โดยนำทราฟฟิกผ่าน façade แล้วแทนที่ระบบ legacy ทีละขั้นจนปิดสวิตช์ได้
Big-bang rewrite	การแทนที่ระบบทั้งหมดในคราวเดียว สลับในวันเดียว อัตราล้มเหลวระดับตำนาน; strangler fig มีอยู่ก็เพราะมัน
Exit cost / switching cost	ต้นทุนที่ราคาตามจริงของการออกจากผู้ให้บริการหรือบริการ — ตัวเลขที่เปลี่ยน lock-in จากความกลัวเป็นเศรษฐศาสตร์
Multi-cloud (strategy vs. fact)	การจงใจรูปแบบพกพาบนหลายคลาวด์ (แพง แทบไม่เคยคุ้ม) vs การมี workload อยู่หลายคลาวด์เฉย ๆ (ปกติ)

แบบฝึกหัด: (1) ทำ data-flow map ของข้อมูลลูกค้า ThaiTicket: ทุกที่เก็บ ทุกสำเนา (อย่าลืม log, backup, pipeline วิเคราะห์, ไฟล์ export ของทีมซัพพอร์ต), ทุกฟอร์มแดน; แล้วเขียนเส้นทางการลบสัมผัสนว่าแผนที่หาปัญหาที่แผนภาพซ่อนไว้ได้อย่างไร (2) ออกแบบแผน strangler-fig ให้ระบบสินค้าคงคลัง on-prem อายุ 20 ปี พร้อมเอเย่ชื่อสามชิ้นแรกที่จะลอก (3) เขียนย่อหน้า lock-in สไลด์ DynamoDB (ประโยชน์ตอนนี้, ต้นทุนขาออกทีหลัง, ยอมรับหรือบรรเทา) สำหรับสามบริการที่คุณจะใช้จริง

Milestone — จบช่วงที่ 4: ด้านข้อจำกัด: โจทย์ออกแบบหนึ่งข้อที่คลุกทั้งสามเรื่อง (“บริษัทประกันไทย, ข้อมูลลูกค้า, ระบบกรรมธรรม์บน mainframe, บอร์ดกังวลเรื่องการพึ่งพา AWS”) — คุณผลิตแนวทางหนึ่งหน้าที่แตะ residency, integration และ lock-in แต่ละเรื่องพร้อมแพดเทิร์นที่เอเย่ชื่อได้และต้นทุนที่ซื้อสัตย์ เมื่อข้อจำกัดทำให้คุณตื่นเต้นกว่าที่ greenfield จะทำได้ — เพราะข้อจำกัดคือที่ที่สถาปนิกทำเงินได้เหนือกว่านักวาดแผนภาพ — ช่วงที่ 4 ก็เสร็จสิ้น

ช่วงที่ 5 — งานฝีมือของสถาปนิก (เดือน 7–12)

ทุกอย่างจนถึงตอนนี้ทำให้คุณ**ออกแบบ**ได้ ช่วงนี้ทำให้คุณ**ทำงานเป็นสถาปนิก**ได้ — เอกสาร การรีวิว การนำเสนอ และหลักฐานต่าง ๆ ที่บทบาทนี้ประกอบขึ้นจากจริง ๆ บวกใบรับรองที่พาคุณผ่าน HR และ capstone สามชั้นที่กลายเป็นพอร์ตโฟลิโอของคุณ

โมดูล 12 (เดือน 7–8): งานส่งมอบ — ADR, ชุดแผนภาพ และการรันรีวิว

Architecture Decision Records (ADRs) ADR คือเอกสารหนึ่งหน้าที่จับหนึ่งการตัดสินใจสำคัญ: เราเลือกอะไร ไม่เลือกอะไร และทำไม — เขียน**ตอนที่ตัดสินใจ** ใส่หมายเลข และเก็บใน repository ของโปรเจกต์ตลอดไป ทำไม JD เอ่ยชื่อมัน: อีกสองปี จะมีคนถามว่า “ทำไมถึงเป็น DynamoDB กันแน่?” และ ADR ตอบได้ในสามสิบวินาที — พร้อมบริบท ข้อจำกัด และทางเลือกที่ถูกพิจารณาอย่างซื่อสัตย์ ทีมที่มี ADR ไม่ต้องรีอคติอะไรเลย; ทีมที่ไม่มีเสียวนเป็นวงกลมทุกปี เทมเพลต — ท่องให้ขึ้นใจ:

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.

เขียน ADR หนึ่งฉบับต่อหนึ่งการตัดสินใจสำคัญในทุก capstone พอร์ตโฟลิโอสัมภาษณ์ที่มี ADR จริง นั้นหายากและทรงพลังอย่างร้ายกาจ

ชุดแผนภาพ — หนึ่งระบบ สามระดับZoom เอกสารสถาปัตยกรรมจริงคือชุด (โมเดล C4 ทำให้วินัยนี้แพร่หลาย): **context diagram** — ระบบเป็นกล่องเดียว พร้อมผู้ใช้และระบบภายนอกที่มันสัมผัส; มุมมองสำหรับผู้บริหาร/พนักงานใหม่ และเป็นอันที่คุณจะนำเสนอต่อผู้นำ **container diagram** — Zoom เข้า: ชิ้นส่วนหลักที่รันอยู่ (web app, API, database, queue, cache) และวิธีที่มันคุยกัน; มุมมองวิศวกรรมประจำวัน ประมาณภาพวาดช่วงที่ 1–3 ของคุณ **deployment diagram** — ทั้งหมดรันอยู่ที่ไหนทางกายภาพ: region, AZ, VPC, subnet, scaling group; มุมมองสำหรับการรีวิว ความปลอดภัย และปฏิบัติการ หนึ่งระบบ สามผู้ชม สามแผนภาพ — อัปเดตเสมอ ลงวันที่ อยู่ใน version control ข้าง ๆ ADR แผนภาพที่หาไม่เจอหรือเชื่อไม่ได้ คือแผนภาพที่ไม่มีอยู่จริง

การรัน Well-Architected review คุณรู้จักหกเสาแล้ว (ช่วงที่ 2); นี่คือการประชุม **ก่อน**: เลือก workload และขอบเขต ทำชุดแผนภาพให้เป็นปัจจุบัน เชิญคนที่**ปฏิบัติการ**กำลังนั้น (ไม่ใช่แค่คนออกแบบ) และตั้งโทน — นี่คือการตรวจสอบสุขภาพเพื่อประโยชน์ของทีม ไม่ใช่การตรวจสอบเพื่อแฟ้มของใคร **ระหว่าง (ครึ่งวัน)**: เดินไล่เสาหลักด้วยชุดคำถามของ framework (AWS เผยแพร่ไว้ พร้อม Well-

Architected Tool ฟรีในคอนโซลที่จัดโครงสร้างให้ทั้งกิจกรรม —

<https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html>); แต่ละคำตอบ จัด finding และจัดลำดับความสำคัญไปพร้อมกัน — ประเด็นความเสี่ยงสูงหยิบมือเดียว ไม่ใช่ข้อจุกจิกร้อยข้อ โทนการถามของคุณคือตัวการรีวิว: “help me understand what happens when the payment provider times out” (ช่วยเล่าให้ฟังหน่อยว่าเกิดอะไรขึ้นเมื่อผู้ให้บริการชำระเงิน timeout) เปิดประตูที่ “you don’t handle timeouts?” (คุณไม่ handle timeout หรือ?) กระแทกปิด **หลัง:** รายงานสั้น — findings เด่น แต่ละอันมีความเสี่ยง แรงที่ใช้ และคำแนะนำ — และรายการปรับปรุงลงสู่ backlog จริงของทีมนำทีมเจ้าภาพ ไม่จู้จี้การรีวิวคือละคร ฝึก: รันรีวิวเต็มรูปแบบกับงานออกแบบ Capstone 1 ของคุณเอง; การหาตำแหน่งของตัวเองอย่างมีโครงสร้างคือทักษะที่พบต้นเร็วที่สุดในหลักสูตรนี้

แบบฝึกหัด: (1) เขียน ADR ย้อนหลังให้หาคำตัดสินใจใหญ่ในงานออกแบบช่วงที่ 2–3 ในสมุดของคุณ — คุณจะพบอย่างน้อยหนึ่งอันที่แตกต่างไม่ได้อีกต่อไป ซึ่งนั่นแหละคือบทเรียน (2) ผลิตชุดแผนภาพสามระดับเต็มรูปแบบของ ThaiTicket (3) รันรีวิวจำลองข้างบน ผลิตรายงาน findings หนึ่งหน้า

Milestone: คนแปลหน้า (หรือ AI ที่รับบทเป็น) หยิบแฟ้ม ThaiTicket ของคุณ — สามแผนภาพ + หัว ADR — แล้วตอบ “นี่คืออะไร รันอย่างไร และทำไมถึงถูกสร้างแบบนี้?” ได้ถูกต้องโดยไม่มีคุณอยู่ในห้อง แฟ้มนั้นคืองานส่งมอบของอาชีพสถาปนิก

โมดูล 13 (เดือน 8–9): งานฝีมือด้านมนุษย์ — การนำเสนอ การคิดต้นทุน การรับแรงต้าน และการเป็นพี่เลี้ยง

การนำเสนอต่อผู้บริหาร vs วิศวกร — งานออกแบบเดียว สองภาษา ผู้บริหารซื้อผลลัพธ์; วิศวกรซื้อกลไก
ต่อผู้บริหาร: นำด้วยการตัดสินใจและตัวเลข (“งานออกแบบนี้รองรับเป้าลูกค้าหนึ่งล้านคนที่ ฿1.90 ต่อออเดอร์ และนี่คือสองการตัดสินใจที่ผมต้องการจากคุณ”); สไลด์เดียว ใช้แผนภาพ context ความเสี่ยงถูกวางกรอบเป็นความเสี่ยงธุรกิจ (รายได้, compliance, ชื่อเสียง) และอย่าพูดคำว่า “Kubernetes” เมื่อคำว่า “แพลตฟอร์ม” ใช้แทนได้ เตรียมพร้อมสำหรับสามคำถามที่ผู้บริหารถามเสมอ: *ราคาเท่าไร อะไรพังได้ ทำไมไม่เอาตัวเลือกที่ถูกกว่า?* ต่อวิศวกร: นำด้วยปัญหาและข้อจำกัดก่อนโซลูชัน (วิศวกรที่รู้สึกถึงปัญหาจะยอมรับโซลูชัน; วิศวกรที่ถูกยื่นคำพิพากษาจะล่าหาดำเนินโดยหลักการ), โซว์แผนภาพ container และ deployment, เหยื่อ trade-off และตัวเลือกที่ถูกปิดกั้นด้วยตัวเอง (ความน่าเชื่อถือมาจากสิ่งที่คุณยอมรับ) และเหลือพื้นที่จริงให้เปลี่ยนงานออกแบบ — คำถามในห้องคือการรีวิวฟรี นิสัยสถาปนิกที่แย่งสุดคือเด็ดเดี่ยวสำหรับทั้งสองผู้ชม; นิสัยที่ดีสุดคือเขียน executive summary ก่อน เพราะถ้างานออกแบบรอดการบีบอัดเหลือห้าประโยคไม่ได้ มันยังไม่เสร็จ

การประมาณต้นทุนสำหรับข้อเสนอ วิธีการ: แยกงานออกแบบเป็นส่วนประกอบที่มีต้นทุน ~10 ตัว; ตั้งราคาแต่ละตัวที่โหดคาดการณ์ใน pricing calculator; **เขียนสมมติฐานเป็นลายลักษณ์อักษร** (ค่าขอ/วัน, การเติบโตของข้อมูล, egress — สมมติฐานคือตัวประมาณการ; เมื่อมันเปลี่ยน ตัวเลขก็ขยับตามด้วยมโนธรรมใส); ใช้กลยุทธ์การจอบกับส่วนคงที่; เพิ่มฉากทัศน์การเติบโต (วันนี้, 2x, 10x — ผู้บริหารจำ

เลข 10x ได้); และนำเสนอเป็น *ช่วงพร้อมตัวขับ* (“B55–70k/เดือน ขับเคลื่อนหลัก ๆ โดย egress — นี่คือ คันโยก”) ห้ามเป็นเลขเดี่ยวที่แม่นยำจอมปลอม ใส่ migration bubble ถ้ามี การประมาณต่ำเพื่อให้ผ่าน อนุมัติคือบาปคลาสสิกของสถาปนิกมือใหม่; โพรเจกต์จืดจาง

การรับแรงต้าน คุณจะถูกดันเรื่องต้นทุน (“แพงไป” → กลับไปที่สามเหลี่ยม: “เราตัด B30k ได้โดยยอมรับ single-AZ — นี่คือเลขคณิตเหตุผล; คุณตัดสินใจ และผมจะจัดบันทึกไว้” — การทำให้การแลกเปลี่ยนชัดเจนและ *ถูกบันทึก* แปลงแรงต้านส่วนใหญ่ให้เป็นความเห็นพ้องหรือความเสี่ยงที่ยอมรับโดยรู้ตัว ซึ่งดีทั้งคู่); เรื่องรสนิยม (“วิศวกรชอบ stack อื่น” → steel-man มันออกเสียงดัง ๆ แล้วพาไปที่เกณฑ์ ไม่ใช่ความชอบ: ความตรง NFR, ทักษะทีม, ระบบนิเวศ, ต้นทุนขาออก — และเมื่อรู้สึกจริง ให้ความชอบของวิศวกร ผู้ลงมือชนะ ชื่อความมุ่งมั่นได้ในราคาถูก); และเรื่องอำนาจ (“เพื่อน CTO บอกให้ใช้ X” → อย่าสูญเสียความเห็นด้วยความเห็น; ขอเกณฑ์ รัน X ผ่านการประเมินสาธารณะแบบเดียวกับทุกอย่าง ปลอ่ยให้เมทริกซ์ตอบอย่างสุภาพ) และเมื่อคุณผิด — คุณจะออกแบบบางอย่างที่ล้มเหลว; สถาปนิกทุกคนเจอ — ทำที่ถูกต้องคือ blameless post-mortem ที่ใช้กับตัวเอง ในที่สาธารณะ พร้อมอัปเดต ADR ไม่มีอะไรสร้างความน่าเชื่อถือระดับทศวรรษเร็วกว่านี้; ไม่มีอะไรทำลายมันเร็วกว่าการปกป้องศพ

การเป็นพี่เลี้ยงวิศวกร วลีใน JD คือ “mentor the engineers who build against your standards” และรูปแบบที่ใช้งานจริงคือ: **design review office hours** (ประตูที่เปิดประจำ เพื่อให้คำแนะนำเกิดก่อนโค้ด ไม่ใช่หลัง); **การรีวิงานออกแบบของพวกเขาอย่างเมตตา** — คำถามก่อนคำพิพากษา และมี *เหตุผลเบื้องต้น* หลัมาตรฐานเสมอ (guardrail ที่ถูกอธิบายได้พันธุมิตร; guardrail ที่ถูกยึดเยียดได้ workaround); **มอบหมายการตัดสินใจจริงพร้อมดาข่ายนิรภัย** (“คุณเป็นเจ้าของงานออกแบบ cache; นี่คือข้อจำกัด; ผมจะรีวิ และผมจะหนุนการตัดสินใจของคุณ”); และ **การสอนในที่สาธารณะ** — ทุก ADR ทุกบทสรุปรีวิ ทุกเซสชัน brown-bag ขยายตัวคุณเกินชั่วโมงของตัวเอง ความจริงอาชีพแบบตรงไปตรงมา: สถาปนิกอัจฉริยะสันโดษมีพาดาน; สถาปนิกที่ปั่นวิศวกรห้าคนให้กลายเป็นเพื่อนร่วมงานที่ออกแบบเป็น คือคนที่ได้นำทีมงานสถาปัตยกรรม

แบบฝึกหัด: (1) นำเสนอ ThaiTicket สองครั้ง — เวอร์ชันผู้บริหาร 5 นาที กับเวอร์ชันวิศวกรรม 20 นาที — อัดเสียงทั้งคู่ แล้วพินาศศัพท์เทคนิคที่วิ่งเข้าไปในอันแรก (2) ผลิตข้อเสนอต้นทุนฉบับเต็ม (สมมติฐาน, ช่วง, ตัวขับ, ฉากทัศน์ 2x/10x) (3) ละครแรงต้านกับ AI: สามยก — การโจมตีต้นทุนจาก CFO, ความชอบ stack จากวิศวกรอาวุโส, เกมอำนาจแบบเพื่อน-CTO — จดลงสมุดว่าอะไรได้ผล (4) รีวิงานออกแบบของ junior (ให้ AI สร้างฉบับมีตำหนิ) เป็นลายลักษณ์อักษร คำถามนำก่อน แล้วให้ AI ให้คะแนนโทนของคุณ

Milestone: คีทสองรายการ: นำเสนอ pitch ผู้บริหารและรูดยี่สิบนาทีของ Q&A ผสมแบบดูแต่แพร์ (แฉง AI: หนึ่ง CFO หนึ่งวิศวกรขี้สงสัย) โดยไม่มีศัพท์ข้ามสาย ไม่มีท่าตั้งรับ และไม่มี trade-off ลักอันที่ไม่ถูกจัดบันทึก

โมดูล 14 (เดือน 9–12): ใบริบรองและ capstone สามชั้น

เส้นทางใบรับรอง พร้อมไทม์ไลน์ข้อสัต์ย์ ใบรับรองไม่ได้ทำให้คุณเป็นสถาปนิก — สิบสามโมดูลก่อนหน้าต่างหาก — แต่มันพาคุณไปถึงห้องสัมภาษณ์ และการเตรียมสอบเดินสายไฟบริการของผู้ให้บริการเข้าไปในนิ้วมือคุณ บันได ที่จังหวะวันละหนึ่งชั่วโมง: **AWS Certified Solutions Architect – Associate (SAA-C03)** — 2–3 เดือน; ใบรับรองสายสถาปนิกที่ถูกขอมากที่สุดในประกาศงาน และหลังช่วงที่ 1–3 เนื้อหาส่วนใหญ่จะรู้สึกเหมือนการทบทวนที่มีค่าบริการแปะกำกับ; สอบตัวนี้ก่อน **AWS Certified Solutions Architect – Professional (SAP-C02)** — 4–8 เดือนหลัง Associate; อิงสถานการณ์ ยากจริง และเป็นบรรทัดเรซูเม่ที่แข็งแกร่งที่สุดหนึ่งเดียวในสายนี้; เนื้อหา migration และ multi-account ของช่วงที่ 4 คือครึ่งหนึ่งของหลักสูตรสอบ **Azure AZ-305 (Azure Solutions Architect Expert)** — เพิ่มถ้าตลาดของคุณหนักฝั่ง Microsoft (ตลาดองค์กรส่วนใหญ่พูดสองภาษาเป็นอย่างน้อย); คาดไว้ 2–3 เดือน โดยความรู้ AWS โอนย้ายได้ในส่วนลดชั้นมาก (เริ่มที่ <https://learn.microsoft.com/en-us/credentials/certifications/azure-solutions-architect/>) **TOGAF Foundation** — เลือกได้ 2–4 สัปดาห์; องค์กรใหญ่บางแห่งขอ; มันรับรองระเบียบวิธีและคลังศัพท์ enterprise architecture มากกว่าทักษะคลาวด์ ลำดับสำหรับหลักสูตรนี้: อ่าน SAA คู่ขนานเดือน 7–8 สอบราวเดือน 8; จากนั้นเลือก SA Pro (เส้นทางเทคนิคลึก) หรือ AZ-305 (เส้นทางความกว้าง) จนถึงเดือน 12 โดย SA Pro เสร็จภายในเดือน 14–16 ที่จังหวะข้อสัต์ย์

Capstone สามชั้น แต่ละชั้นคือแพ็คเกจระดับพอร์ตโฟลิโอที่สมบูรณ์: ชุดแผนภาพสามระดับ, ADR 5+ ฉบับ, ประเมินการต้นทุนพร้อมสมมติฐาน และ Well-Architected review ที่รันเองพร้อม findings ใช้เวลา 3–4 สัปดาห์ต่อชั้น สามชั้นงานนี้ บวกสมุดบันทึกของคุณ คือหลักฐานสัมภาษณ์ของ “end-to-end solution design”

Capstone 1 — greenfield: แพลตฟอร์ม e-commerce ไทยสำหรับผู้ไ้ 1 ล้านคน โจทย์: marketplace, ผู้ไ้ลงทะเบียน 1 ล้านคน, 50k พร้อมกันที่ฟีก flash-sale, mobile-first, เชื่อมชำระเงินแบบ PromptPay, ตรงตาม PDPA, availability 99.9%, ประหยัดงบแบบ seed-stage ต้องมี: การเลือกแพลตฟอร์มพร้อมเส้นทางวิวัฒน์, กลยุทธ์ caching, บทสนทนา RTO/RPO เขียนเป็นบทพูด, ประเมินการ unit economics, data-flow map ของข้อมูลส่วนบุคคล

Capstone 2 — brownfield: การย้ายบริษัท on-prem 40 server โจทย์: บริษัทโลจิสติกส์ไทย; 40 server (ERP บน Oracle, แอปจัดการคลังสินค้าอายุ 12 ปี, file share, AD, แอปแผนกต่าง ๆ — ให้ AI สร้าง inventory ฉบับเต็มแล้วจริงไว้); สัญญาเช่า data center หมดใน 14 เดือน; บอร์ดอยากออก ต้องมี: ตาราง 7-R ครบทุกแถว, แผนสามระลอกพร้อมเหตุผล dependency, งานออกแบบ landing zone, ไทม์ไลน์ต้นทุน migration bubble และบันทึกถึง CFO

Capstone 3 — โจทย์หิน: multi-region DR สำหรับ fintech โจทย์: บริษัทชำระเงินที่ถูกกำกับให้ RTO ≤ 15 นาที / RPO ≈ 0 สำหรับ ledger พร้อมข้อจำกัด data residency ของข้อมูลลูกค้า และการทดสอบ failover ต่อหน้าผู้กำกับดูแลปีละครั้ง ต้องมี: บทวิเคราะห์ active-passive vs active-active พร้อมต้นทุน, งานออกแบบ replication พร้อมแผนที่ residency, failover runbook และแผนการซ้อม

เกณฑ์ประเมินตนเอง (ใช้กับแต่ละ capstone อย่างข้อสัต์ย์ ในสมุดบันทึก):

มิติ	1 — ยังไม่ถึง	3 — แน่น	5 — จ้างคนนี้เลย
Requirements	กระโดดสู่โซลูชัน	NFR ถูกระบุและโยงถึงทางเลือกออกแบบ	สามเหลี่ยม trade-off ถูกวางตำแหน่ง, ความขัดแย้งถูกยกให้ “ธุรกิจ” เห็น, การตัดสินใจถูกสกัดออกมา
คุณภาพการออกแบบ	SPOF ยังเหลือ; แพตเทิร์นไม่เข้ารูป	แพตเทิร์นถูกต้อง, multi-AZ, ความล้มเหลวถูกรองรับ	มีเหตุผลเมื่อไม่ควรใช้; เส้นทางวิวัฒน์; งานออกแบบเรียบง่ายสุดที่ตอบโจทย์
หกเสาหลัก	เสาถูกمين	ทุกเสาถูกจัดการอย่างมองเห็นได้	รีวิวที่รันเองพบตำหนิจริง — และงานออกแบบถูกแก้ตามนั้น
ต้นทุน	ไม่มีตัวเลข	ประมาณการแจกแจงสมมติฐานระบุไว้	ช่วงพร้อมตัวขับ, unit economics, ฉากทัศน์ 10x, กลยุทธ์การจูง
เอกสาร	มีแค่แผนภาพ	ชุดสามระดับ + ADR สัญกรณ์ถูกต้อง	คนแปลหน้าตอบ “อะไร/อย่างไร/ทำไม” ได้จากแพ็คเกจเดียว
การสื่อสาร	ชั้นเดียวสำหรับผู้ชม	มีเวอร์ชันผู้บริหารและวิศวกร	บทสรุปห้าประโยคที่ยืนหยัดได้; แรงต้านถูกตอบล่วงหน้าเป็นลายลักษณ์อักษร

ได้ 4+ ทุกมิติในทั้งสาม capstone — แก่จนกว่าจะได้ — แล้วคุณก็เรียนจบหลักสูตรนี้

สิบคำถามสัมภาษณ์สถาปนิก (พร้อมคำตอบที่แข็งแกร่ง)

ฝึกออกเสียง; ความแข็งแรงอยู่ที่โครงสร้างของแต่ละคำตอบ ซึ่งตอนนี้是你的แล้ว

1. “Design a URL shortener / ticketing site / photo app for a million users.” (ออกแบบ URL shortener / เว็บไซต์บัตร / แอปรูปภาพ สำหรับผู้ใช้หนึ่งล้านคน) โครงที่แข็งแรง: requirements ก่อน ออกเสียง (“อ่านมากกว่าเขียนไหม? เป้า availability? งบ?”) → วางตำแหน่งบนสามเหลี่ยม → โครง three-tier หรือ serverless-first พร้อม multi-AZ, cache, CDN → แอชชี่ trade-off โดยไม่ต้องถูกถาม → จบด้วยต้นทุนหลักหน่วยประมาณและเส้นทางวิวัฒน์ 10× ผู้สัมภาษณ์ให้ผ่านที่คำถามที่คุณถามไม่ใช่ กล้องที่คุณวาด

2. “When would you choose NoSQL over a relational database?” (เมื่อไหร่คุณจะเลือก NoSQL แทน relational database) “ผมตั้งต้นที่ SQL — การรับประกันความถูกต้องและทักษะที่หาได้ทั่วไป — จนกว่าจะมีเหตุผลที่แอชชี่ได้มาลบล้าง: สเกลแนวนอนสุดขีด, schema ยืดหยุ่น หรือการเข้าถึง key-value ระดับมิลลิวินาทีหลักเดียว จากนั้นผมเลือก NoSQL ให้เข้ากับ access pattern และผมเขียน ADR ที่รวมสิ่งที่เราสละไป: join และ consistency semantics บางส่วน”

3. “Explain RTO and RPO, and how they drive design.” (อธิบาย RTO กับ RPO และวิธีที่มันขับเคลื่อนการออกแบบ) นิยามทั้งคู่ให้คม → “มันคือการตัดสินใจธุรกิจที่ป้ายราคาโททวีคูณ” → บันทึ: backup รายคืน → replication ต่อเนื่อง → warm standby → active-active โดยต้นทุนได้ขึ้นทุกชั้น → “งานของผมคือทำให้ธุรกิจเลือกอย่างรู้ตัว แล้วออกแบบตรงตามตัวเลขเป๊ะ — และซ่อมมัน เพราะ failover ที่ไม่ทดสอบคือความหวัง”

4. “Monolith or microservices?” (monolith หรือ microservices) คำตัดสินของโมดูล 8 คำต่อคำ: เริ่ม modular-monolith; สกัดเมื่อความเจ็บปวดการขยายทีมหรือขยายโหนดมาถึงจริง; microservices คือเครื่องมือขยายทีมที่คิดภาชี distributed systems ได้คะแนนโบนัส์ถ้าพูดว่า “I’d rather run a good monolith than a bad distributed system” (ผมยอมรับ monolith ที่ดี มากกว่า distributed system ที่แย)

5. “How do you handle a large cloud bill / cost optimization?” (คุณจัดการบิลคลาวด์ก้อนใหญ่ / cost optimization อย่างไร) “ความมองเห็นได้ก่อน — tagging และ showback; จากนั้นเก็บเกี่ยวตามลำดับแรงที่ใช้: ฆ่าทรัพยากรที่ตายแล้ว, rightsize จากการวัด, จัด tier ให้ storage, ตั้งเวลาหลับให้ non-prod, แล้วจ้องฐานที่คงที่ และผมรายงาน unit economics ไม่ใช่ยอดรวม — บิลที่โตพร้อมต้นทุนต่อออเดอร์ที่ลดลงคือความสำเร็จ ไม่ใช่ปัญหา”

6. “How would you migrate a legacy on-prem application?” (คุณจะย้ายแอป legacy จาก on-prem อย่างไร) “ประเมินก่อนย้าย — inventory, dependency และ R ต่อ workload จาก 7 Rs; สร้าง landing zone ก่อนระลอกแรก; ระลอกเรียงง่ายก่อนพร้อม cutover ที่ซ่อมแล้วและแผน rollback; และสำหรับ database อัญมณีมงกฏ ใช้ cutover แบบ replication ที่ปรับขนาดตาม downtime ที่ธุรกิจเซ็นรับอีกอย่าง: คำเตือน migration-bubble อย่างชั่สตัยตั้งแต่ต้น”

7. “How do you secure a cloud architecture?” (คุณทำให้สถาปัตยกรรมคลาวด์ปลอดภัยอย่างไร) เดินไล่ชั้น: identity (least privilege, MFA, ไม่มี root รายวัน) → เครือข่าย (private subnet, segmentation, attack surface น้อยสุด) → ข้อมูล (เข้ารหัส at rest/in transit, classification, residency)

→ การตรวจจับ (audit log, alert) → “และโดยการออกแบบ ไม่ใช่แปะทีหลัง — การควบคุมความปลอดภัยที่ถูกที่สุดคือการตัดสินใจสถาปัตยกรรมที่กำลังความเสี่ยงทั้งก้อน เช่นการไม่แตะข้อมูลบัตรเครดิตเลย”

8. “Tell me about a design decision you got wrong.” (เล่าการตัดสินใจออกแบบที่คุณเคยพลาดให้ฟังหน่อย) เขากำลังทดสอบอีโก้ ไม่ใช่ประวัติศาสตร์ โคร่ง: ตัวอย่างจริง (จาก capstone) → สิ่งที่คุณเคยเชื่อ → สิ่งที่คุณจริงบอก → post-mortem, ADR ที่ถูกอัปเดต, แพตเทิร์นที่ตอนนี้นักออกแบบคอยตรวจสอบสถาปนิกที่ผลิตคำตอบข้อนี้ไม่ได้ คือสถาปนิกที่ไม่เคยถูกรีวิว

9. “How do you explain a complex technical decision to a non-technical executive?” (คุณอธิบายการตัดสินใจเทคนิคซับซ้อนให้ผู้บริหารที่ไม่ใช่สายเทคนิคอย่างไร) “การตัดสินใจและตัวเลขธุรกิจก่อน กลไกเฉพาะเมื่อถูกขอ; context diagram ไม่ใช่ container; ความเสี่ยงในภาษารายได้-compliance-ชื่อเสียง; และผมพกสองการตัดสินใจที่ต้องการจากเขา เขียนเป็นทางเลือกพร้อมป้ายราคา — ผู้บริหารตัดสินใจระหว่างตัวเลือก เขาไม่อนุมัติปริศนา”

10. “An engineer strongly disagrees with your design. What do you do?” (วิศวกรไม่เห็นด้วยกับงานออกแบบของคุณอย่างแรง คุณทำอย่างไร) “ก่อนอื่นผม steel-man เขาออกเสียงดัง ๆ — เขาอาจถูก และการรีวิวที่เปลี่ยนงานออกแบบของผมคือการรีวิวที่กำลังทำงาน ถ้ามันสุลี้จริง เกณฑ์เป็นผู้ตัดสินใจ (ความตรง NFR, ทักษะทีม, ต้นทุนขาออก) ไม่ใช่ความอาวุโส — และผมให้ความชอบของผู้ลงมือขณะเมื่อเสมอกัน เพราะความมุ่งมั่นราคาถูกที่ราคาร้าน ไม่ว่าทางไหน การตัดสินใจและตัวเลือกที่ถูกปิดตกลง ADR เราจึงไม่มีวันเถียงเรื่องเดิมสองครั้ง”

หลักสูตรย่อหน้า

เมื่อไหร่	โมดูล	จุดเน้น	หลักฐานภายนอก
สัปดาห์ 1–2	1	ทบทวนคลาสส์; สามเหลี่ยม trade-off; NFR	เริ่ม Design Journal
สัปดาห์ 3–4	2	Compute/storage/database/network ในฐานะการตัดสินใจ	—
สัปดาห์ 5–6	3	การอ่านและวาดแผนภาพ; ความคล่อง 3-tier	milestone ไวต์บอร์ด 15 นาที

เมื่อไหร่	โมดูล	จุดเน้น	หลักฐานภายนอก
สัปดาห์ 7–8	4	Reliability: multi-AZ, auto-scaling, RTO/RPO	—
สัปดาห์ 9–10	5	Security: least privilege, zero trust, segmentation	—
สัปดาห์ 11–12	6	Performance และ cost: caching, CDN, การจอง, unit economics	งานออกแบบตั้งราคาแล้วใน calculator
สัปดาห์ 13–14	7	Ops excellence และ sustainability; แบบฝึกหทเสา	รีวิวกเสาจำลอง
สัปดาห์ 15–17	8	แพตเทิร์น: monolith/microservices, event-driven, serverless	—
สัปดาห์ 18–20	9	แพตเทิร์น: data lake/warehouse, multi-region, hybrid	ด้านเค็ตตลลอก
สัปดาห์ 21–23	10	Migration: 7 Rs, ระลอก, landing zone	migration บนกระดษ
สัปดาห์ 24–26	11	ข้อจ้ก้ด: PDPA/GDPR, legacy, เศรษฐศษตร lock-in	ด้านข้อจ้ก้ด
เดือน 7–8	12	ADR, ชุดแผนภพ, การรัน Well-Architected review	แพ็ก ThaiTicket; สอบ SAA-C03 (เตรียม 2–3 เดือน)
เดือน 8–9	13	การนำเสนอ, การคิตตันทุนข้อเสนอ, แรงดัน, การเป็นพีเลียง	คิกสองรยการที่อ้ดไว้
เดือน 9–12	14	Capstone 1–3	พอร์ตโฟลิโอครบ; SA Pro (4–8 เดือน) หรือ AZ-305 ก้ล้งดำเนิน

คำสงท้ยจกครุขงคุณ ยีลบกสัปดาห์เข้มข คุณออกแบบได้; ลีบสองเดือนเข้มข คุณประกอบวิษยชีฟได้ — ม้นม้ความต้ง และค้ความต้งที่ล้กสูตรนี้ถูกสร้งมเพื่อปิด นิสยเหล่าน้ตอนน้ค้ อชัพ: วาดกอนเลียง เขียน ADR ในวันท้ต้ดลนใจ ต้งราคาล้งท้เสนอ เอ้ยช้ trade-off กอนครถม และ

ป็นวิศวกรรอบตัวจนมาตรฐานของคุณอายุยืนกว่าการปรากฏตัวของคุณในห้อง สถาปัตยกรรมคืองานเทคนิคหายากที่ดีขึ้นเมื่อคุณแก้ตัวเข้าไปในมัน เพราะวัตถุดิบของมันคือวิจารณ์ญาณ และวิจารณ์ญาณทบต้น จดสมุดบันทึกต่อไป อีกสี่สิบงานออกแบบจากนี้ คุณจะไม่ต้องการหลักสูตรนี้อีก — คุณจะเป็นคนแก้ไขมันเอง

Sources

ประกาศงานและนิยามบทบาทที่ใช้ในตารางจับคู่ข้อกำหนด (สืบค้นเมื่อสิงหาคม 2569): [KORE1 Cloud Architect job description template](#) · [4 Corner Resources Cloud Architect job description](#) · [Solution Architect responsibilities in the Azure Well-Architected Framework](#) ของ Microsoft · [Pre-Sales Solutions Architect – AWS posting](#) ของ Arpio · [Solution Architect – Enterprise posting \(via Built In\)](#) ของ Intel · [Cloud Domain Architect posting](#) ของ Halliburton ค่าประมาณเวลาเตรียมสอบใบรับรอง: [CBT Nuggets' SAA-C03 study-time survey](#) และ [Whizlabs' SAP-C02 preparation guidance](#); นิยามใบรับรองจาก [Microsoft Learn \(AZ-305\)](#) และ [AWS Well-Architected Framework](#) เล่มคู่มือของ *The Cloud Leader Course* ในชุด B4LCILC

บทเรียนใหม่ — ฉบับเดือนสิงหาคม 2026

ในเดือนสิงหาคม 2026 มีการเพิ่มบทเรียนใหม่เข้าไปในหลักสูตรแบบอินเทอร์แอคทีฟทั้งสามที่ [b4lcilc.org](#) — หลักสูตรผู้นำด้านคลาวด์ (Cloud Leader) หลักสูตรวิศวกรคลาวด์ (Cloud Engineer) และหลักสูตรสถาปนิกคลาวด์ (Cloud Architect) ภาคผนวกนี้แนะนำสำหรับอ่านให้ก้าวหน้าเนื้อหาใหม่นั้นด้วยฉบับย่อที่เก็บความอย่างชัดเจน ในน้ำเสียงเดียวกับส่วนอื่น ๆ ของหนังสือเล่มนี้

ขอพูดถึงสิ่งที่คุณถืออยู่ในมือสักนิด หน้ากระดาษเหล่านี้คือฉบับสำหรับอ่าน: แนวคิด อุปมา และประโยคที่ควรค่าแก่การพูดออกเสียง ฉบับอินเทอร์แอคทีฟเต็มรูปแบบบนเว็บมีมากกว่านี้มากทีเดียว — บทเรียนใหม่แต่ละบทมาพร้อมแบบทดสอบที่ตรวจสอบความเข้าใจก่อนให้คุณไปต่อ ห้องปฏิบัติการลงมือทำจริง (รวมถึง terminal สำหรับฝึกที่ทำงานเหมือนของจริงแต่ไม่มีทางทำให้เสียเงิน) วิดีโอ แบบฝึกคำศัพท์ และ — สำหรับเนื้อหา capstone — แบบฝึกออกแบบพร้อมรายการตรวจสอบข้อกำหนดและเฉลยตัวอย่างที่ทำให้ดูเป็นขั้นเป็นตอน อ่านฉบับย่อเหล่านี้เพื่อให้โครงร่างของแต่ละแนวคิดเข้าไปอยู่ในหัวก่อน แล้วเมื่อมีโอกาส จงกลับไปนั่งเรียนฉบับอินเทอร์แอคทีฟเพื่อให้ความรู้ฝังแน่น

เนื้อหาที่เพิ่มเข้ามาแบ่งเป็นสามกลุ่ม สำหรับหลักสูตร**ผู้นำด้านคลาวด์**: บทแนะนำ terminal อย่างนุ่มนวล กฎหมายความเป็นส่วนตัวของไทยเราเอง คู่มือผู้นำสำหรับข้อเสนอโครงการ AI และบิลค่าใช้จ่าย AI และแบบฝึก capstone ที่ประกอบทั้งหลักสูตรเข้าเป็นหนึ่งหน้ากระดาษ สำหรับหลักสูตร**วิศวกรคลาวด์**: serverless computing — แพตเทิร์นที่เปลี่ยนสคริปต์งานปฏิบัติการเล็ก ๆ ให้กลายเป็นฟังก์ชันที่รันโดยไม่มีค่าใช้จ่าย สำหรับหลักสูตร**สถาปนิกคลาวด์**: เลขคณิตของคำสัญญาเรื่องความพร้อมใช้งาน แพตเทิร์น Retrieval-Augmented Generation ที่เดี๋ยวนี้ถูกคำถามถึงกันเป็นรายสัปดาห์ และแบบฝึกประเมินต้นทุนเต็มรูปแบบ — ตีราคางานออกแบบก่อนที่จะใครจะลงมือสร้าง

สำหรับหลักสูตรสถาปนิกคลาวด์ (Cloud Architect)

เลขคณิตของเลขเก้า — SLA, SLO และ error budget

ความพร้อมใช้งาน (availability) ถูกบอกกันเป็นจำนวนเลขเก้า และสิ่งแรกที่สถาปนิกทำกับเลขเก้าคือแปลงมันให้เป็นนาฬิกา — เพราะนาฬิกาเป็นสิ่งทีธุรกิจสัมผัสได้จริง เดือนที่มี 30 วันมีทั้งหมด 43,200 นาที:

ความพร้อมใช้งาน	เวลาหยุดทำงานที่ยอมให้ได้ต่อเดือน
99.9% (“สามเก้า”)	ประมาณ 43 นาที
99.99% (“สี่เก้า”)	ประมาณ 4.3 นาที
99.999% (“ห้าเก้า”)	ประมาณ 26 วินาที

ห้าเก้า — ตัวเลขที่ผู้บริหารชอบเอ่ยออกมาดัง ๆ — อยู่พ้นขีดความสามารถของการกู้ระบบด้วยมือมนุษย์ทุกรูปแบบ; failover ต้องเป็นอัตโนมัติ ไม่อย่างนั้นคำสัญญาก็เป็นเพียงนิยาย เลขเก้าที่เพิ่มขึ้นแต่ละตัวมีต้นทุนทางวิศวกรรมสูงขึ้นราวสิบเท่า; การแปลงเป็นนาฬิกาคือวิธีทำให้ผู้มีส่วนได้ส่วนเสียเลือกอย่างรู้ว่าตัวเองกำลังเลือกอะไร

อักษรย่อสามคำแบบทสนทนานี้เอาไว้ **Service-Level Indicator (SLI)** คือค่าที่วัดได้ **Service-Level Objective (SLO)** คือเป้าหมายภายในบนค่าที่วัดนั้น **Service-Level Agreement (SLA)** คือสัญญากับภายนอก ที่มีเงินผูกอยู่กับการทำพลาด วินัยสำคัญ: SLO ต้องเข้มกว่า SLA เสมอ — คุณเล็งเข้าไปข้างในเส้นที่คุณถูกจ้างให้รักษา — และ SLI ต้องวัดสิ่งที่ผู้ใช้สัมผัสจริง: server ที่เปิดอยู่แต่ตอบกลับเป็น error ก็คือ server ที่ล้ม ในทุกความหมายที่มีความหมาย

error budget (งบความผิดพลาด) คือสิ่งที่ทำให้ **SLO** ใช้งานได้จริง: ช่องว่างระหว่างความสมบูรณ์แบบกับเป้าหมาย ถูกปฏิบัติเสมือนทรัพยากรที่เบิกจ่ายได้ SLO ที่ 99.9% ให้งบความล้มเหลว 43 นาทีต่อเดือน; การ deploy และการทดลองต่าง ๆ เบิกจ่ายจากงบนี้; เมื่อจบหมด ทีมหยุดปล่อยของแล้วหันไป

ทำงานด้านความน่าเชื่อถือ — เป็นการปิดฉากข้อถกเถียงที่เก่าแก่ที่สุดในวงการซอฟต์แวร์ ระหว่างการส่งของกับความเสถียร ด้วยการแทนที่ความเห็นด้วยเลขคณิต

ที่นี้มาถึงกับดัก: availability คุณทบกันไปตลอดสายโซ่ บริการสามตัวต่ออนุกรมกัน แต่ละตัวได้ 99.9% อย่างซื่อสัตย์ รวมกันเป็น $0.999 \times 0.999 \times 0.999 \approx 99.7\%$ — ราว 130 นาทีต่อเดือน สามเท่าของที่บริการตัวใดตัวหนึ่งยอมรับ ส่วน redundancy ทำงานในทางกลับกัน: instance ระดับ 99.9% ที่เป็นอิสระต่อกันสองตัววางขนานกัน จะล้มพร้อมกันเพียง $0.1\% \times 0.1\%$ ของเวลา — จึงเป็นเหตุผลว่าทำไม multi-zone ถึงคุ้มค่าขนาดนั้น ประโยชน์ระดับสัมผัสงาน: *คุณสัญญาความพร้อมใช้งานได้ไม่มากไปกว่าผลคูณของ dependency แบบ synchronous ทั้งหมดของคุณ* ดังนั้นถ้าไม่ย่นสายโซ่ให้สั้นลง ก็ต้องลดค่าสัญญาลง

สถาปัตยกรรมในยุค AI — RAG และญาติพี่น้อง

เมื่อถึงปี 2026 แคล์ตาล็อกแพตเทิร์นมีสมาชิกใหม่ที่ลูกค้าถามถึงเป็นรายสัปดาห์: มันคืออะไร เมื่อไรมันคุ้มค่าตัว และเมื่อไรมันจะทำให้คุณขายหน้า API ของ **Large Language Model (LLM)** — Amazon Bedrock, Azure OpenAI Service — ในเชิงสถาปัตยกรรมคือ *dependency* ที่แพง ซ้ำ และไร้ความจำ (*stateless*) มันถูกคิดเงินตาม **token** — เศษเสี้ยวของคำ ยาวราวสี่ตัวอักษร มันตอบในหลักวินาที ไม่ใช่ มิลลิวินาที จำอะไรไม่ได้เลยระหว่างการเรียกแต่ละครั้ง และบางครั้งก็ผิดอย่างมั่นอกมั่นใจ — เป็นส่วนประกอบที่มีโหมดความล้มเหลวแปลกไม่เหมือนใคร สมควรได้รับ timeout, retry, circuit breaker และ สัญญาณเตือนงบประมาณเช่นเดียวกับ dependency อื่นใด เพราะรูปการเรียกที่หลุดการควบคุมเผาเงินจริงเป็นรายชั่วโมง

โมเดลไม่รู้จักเอกสารของคุณ; คำตอบมาตรฐานคือ **RAG — Retrieval-Augmented Generation:** แทนที่จะสอนข้อมูลของคุณให้โมเดล คุณค้นหาข้อเท็จจริงที่เกี่ยวข้อง ณ เวลาถามคำถาม เอกสารถูกหั่นเป็นชั้นย่อย (chunk) แต่ละชั้นถูกแปลงเป็น **embedding** — รายการตัวเลขที่ระบุตำแหน่งความหมายของมันในปริภูมิ — แล้วเก็บไว้ใน **vector database** (pgvector บน AWS, AI Search บน Azure) ซึ่งคืนชั้นที่ใกล้เคียงที่สุดกลับมา ชั้นเหล่านั้นถูกแปลงใน prompt: โมเดลไม่ได้กำลังเปิดค้นข้อมูลของคุณ มันกำลังสรุปข้อความตัดตอนที่คุณไปหยิบมาให้มัน

บนกระดานไวต์บอร์ด RAG คือสองท่อลำเลียง (pipeline): เส้นทาง **ingestion** ที่รับล่วงหน้า (เอกสารถูกหั่น แปลงเป็น embedding และจัดเก็บ) กับเส้นทาง **query** ที่รับต่อทุกคำขอ — สองแพตเทิร์นที่มีอยู่แล้วในแคล์ตาล็อก จึงเป็นเหตุให้สถาปนิกเก็บเรื่องนี้ได้ในป้ายเดียว

trade-off ก็คือสามข้อมาตรฐานเดิม แต่คมขึ้น: ต้นทุน (ทุก query มีราคาเป็นเงินจริง — จง cache คำถามซ้ำ และส่งคำถามง่าย ๆ ไปหาโมเดลที่ถูกกว่า) latency (หลักวินาที ไม่ใช่ มิลลิวินาที — จง stream คำตอบออกไป) และความเป็นส่วนตัว (เอกสารของคุณออกนอกขอบเขตของคุณ; ถ้า vector database เพิกเฉยต่อการควบคุมสิทธิ์เข้าถึงระดับเอกสาร แชตบอตจะยินดีสรุปสเปรดชีตเงินเดือนให้ทุกคนที่ถาม) และจงรู้ด้วยว่าเมื่อไร *ไม่ควร* สร้างมัน: คำขอ “เราต้องการแชตบอต” จำนวนมากคือกล่องค้นหาที่แต่งตัวตามแฟชั่น และคำตอบที่ต้องเป๊ะและตรวจสอบย้อนกลับได้ — ยอดรวม ยอดคงเหลือ

บัญชีแยกประเภท — เป็นงานที่ผิดพลาดผิดตัวสำหรับระบบที่ตอบถูก 97% ของเวลา ประโยชน์ระดับ
สัมภาษณงาน: RAG คือ โจทย์ search ที่ถูกเย็บติดกับ API สำหรับสรุปความ

ตีราคางานออกแบบก่อนที่จะใครจะลงมือสร้าง

บัดนี้โมดูล Pillars of Performance and Cost ของหลักสูตรปิดท้ายด้วยแบบฝึกออกแบบเต็มรูปแบบ:
การประมาณการภายใต้แรงกดดัน โจทย์: ฝ่ายบริหารต้องการประมาณการค่าใช้จ่ายรายเดือนของ
ผลิตภัณฑ์จัดการเอกสาร ก่อนจะอนุมัติให้ลงมือสร้าง ภาระงาน — ผู้ใช้ลงทะเบียน 200,000 คน ใช้งาน
จริงวันละ 40,000 คน แต่ละคนยิงราว 25 API request ต่อวัน และอัปโหลดเอกสารเฉลี่ยเดือนละ 2
เมกะไบต์ เก็บรักษาไว้เจ็ดปี งานออกแบบ — API tier ที่มี load balancer กระจายอยู่สองโซน ฐานข้อมูล
PostgreSQL แบบ managed หลายโซน object storage การส่งออกข้อมูล 50 กิกะไบต์ให้พาร์ตเนอร์ทุก
คืน และสภาพแวดล้อม development กับ staging แยกต่างหาก ภารกิจ: ประมาณการรายเดือนหนึ่งหน้า
กระดาษเป็นตัวเลขกลม ๆ โดยเขียนสมมติฐานเป็นลายลักษณ์อักษรก่อนที่จะตัวเลขราคาใด ๆ จะปรากฏ
แสดงเลขคณิตของ capacity (ผู้ใช้ → request ต่อวินาที → จำนวน instance) และฉากทัศน์วันนี้ / 2 เท่า
/ 10 เท่า

รายการตรวจสอบของแบบฝึกสอนวินัยเหล่านี้: ตีราคา compute จากโหนดช่วงพัก ไม่ใช่ค่าเฉลี่ย; นับ
standby ของฐานข้อมูลด้วย; จำลอง storage เป็นการเติบโตสะสมตลอดเจ็ดปีพร้อม lifecycle tier;
คำนวณ egress จริง ๆ แทนที่จะเดา — บรรทัดที่ประมาณการส่วนใหญ่ลืมน และเป็นบรรทัดที่กลายเป็น
ก้อนใหญ่ที่สุดบ่อยครั้งที่สุด; อย่าลืม NAT (Network Address Translation) gateway สภาพแวดล้อมที่
ไม่ใช่ production งาน backup งาน monitoring และแผน support; ใช้ส่วนลดแบบผูกพันระยะยาวกับ
ฐานการใช้งานคงที่เท่านั้น; และปิดท้ายด้วยช่วงตัวเลขพร้อมระบุตัวขับเคลื่อนหลัก บวกการตรวจความ
สมเหตุสมผลเชิง unit economics — ต้นทุนต่อผู้ใช้ที่ใช้งานจริงต่อเดือน ตัวเลขเป๊ะ ๆ เพียงตัวเดียวนั้น
ผิดตั้งแต่วันที่มันถูกเอ่ยออกมา