

ຫຼັກສູດສະຖາປະນິກ Cloud (The Cloud Architect Course)

ຈາກຄວາມຮູ້ສູນ ສູ່ການອອກແບບລະບົບ cloud — ຫຼັກສູດ B4LCILC

13 ສິງຫາ 2026

ຫຼັກສູດສະຖາປະນິກ Cloud (The Cloud Architect Course)

ຈາກຄວາມຮູ້ສູນ ສູ່ການອອກແບບລະບົບ cloud ທີ່ບໍລິສັດກ້າເອົາທຸລະກິດຂອງຕົນມາວາງເດີມພັນ.

ຫຼັກສູດນິເຮັດວຽກແນວໃດ

ທ່ານບໍ່ໄດ້ຝຶກເພື່ອ *ປະຕິບັດງານ* ລະບົບ cloud. ທ່ານກຳລັງຝຶກເພື່ອ **ອອກແບບ** ພວກມັນ: ເອົາບັນຫາທຸລະກິດທີ່ຍຸ່ງເຫຍື່ງ (“ພວກເຮົາຕ້ອງຂາຍໃຫ້ລູກຄ້າໜຶ່ງລ້ານຄົນ ແລະ ຫ້າມເສຍຄ່າສິ່ງຊື້ຈັກອັນ”) ມາປ່ຽນເປັນແຜນເຕັກນິກທີ່ແຕ້ມແລ້ວ, ຄິດໄລ່ຕົ້ນທຶນແລ້ວ, ປົກປ້ອງໄດ້, ແລ້ວຊັກຊວນທັງວິສະວະກອນທີ່ຈະສ້າງມັນ ແລະ ຜູ້ບໍລິຫານທີ່ຈະຈ່າຍເງິນໃຫ້ມັນ. ນັ້ນຄືວຽກຂອງ **ສະຖາປະນິກ cloud** (ປະກາດຮັບໃນຊື່ *solutions architect*, *cloud solutions architect*, ຫຼື *cloud domain architect* ນຳ), ແລະ ມັນຄືຝຶກຊາຊີບທີ່ຮຽນຮູ້ໄດ້ ພ້ອມຫຼັກສູດທີ່ຈະແຈ້ງ — ອັນນີ້ເອງ.

ຫຼັກສູດນີ້ມີຫ້າໄລຍະ. **ໄລຍະທີ 1 (ອາທິດທີ 1–6):** ອົງປະກອບພື້ນຖານ, ໃນຖານະການຕັດສິນໃຈ. ທຸກສ່ວນປະກອບຂອງ cloud, ສອນຄືນໃໝ່ ບໍ່ແມ່ນເປັນຂໍ້ເທັດຈິງ ແຕ່ເປັນ *ທາງເລືອກທີ່ມີການແລກປ່ຽນ (trade-off)* — ເພາະຫົວໜ່ວຍວຽກຂອງສະຖາປະນິກ ຄືການຕັດສິນໃຈ. **ໄລຍະທີ 2 (ອາທິດທີ 7–14):** ຂອບເຂດການອອກແບບ. ຫຼັກສູດຂອງ Well-Architected Framework — reliability, security, performance, cost, operations, sustainability — ແຕ່ລະອັນສອນຢ່າງເລິກ ພ້ອມຮູບແບບມາດຕະຖານຂອງມັນ ແລະ ການອອກແບບຕົວຢ່າງ. **ໄລຍະທີ 3 (ອາທິດທີ 15–20):** ຄັງລວມ pattern. ສະຖາປະນິກຕະຍະກຳສືບກວ່າແບບ ທີ່ແກ້ 95% ຂອງບັນຫາຕົວຈິງ, ແລະ — ສຳຄັນກວ່ານັ້ນ — ເມື່ອໃດ *ບໍ່ຄວນ* ໃຊ້ແຕ່ລະອັນ. **ໄລຍະທີ 4 (ອາທິດທີ 21–26):** ການ migrate ແລະ ໂລກຕົວຈິງ. ການຍ້າຍບໍລິສັດທີ່ມີຢູ່ແລ້ວຂຶ້ນ cloud, ແລະ ຂໍ້ຈຳກັດ (ກົດໝາຍ, ລະບົບເກົ່າ, lock-in) ທີ່ເຮັດໃຫ້ສະຖາປະນິກຕະຍະກຳຕົວຈິງ ຍາກກວ່າສະຖາປະນິກທີ່ເບິ່ງກະດານຂາວ. **ໄລຍະທີ 5 (ເດືອນ**

ທີ 7–12): ຝຶນວິຊາຊີບ. ເອກະສານ, ແຜນວາດ, ການທົບທວນ, ການນຳສະເໜີ, ແລະ certification ທີ່ເຮັດໃຫ້ທ່ານ ຖືກຈ້າງໄດ້ ໃນຖານະສະຖາປະນິກ, ປິດທ້າຍດ້ວຍສາມໂຄງການອອກແບບລະດັບ portfolio.

ຫຼັກສູດນີ້ສຳລັບໃຜ. ມັນສົມມຸດວ່າບໍ່ມີຄວາມຮູ້ມາກ່ອນເລີຍ: ທຸກສິ່ງທີ່ມັນອາໄສ ຖືກສອນຄືນສັ້ນໆກ່ອນຖືກໃຊ້. ຖ້າທ່ານຮຽນຈົບ **The Cloud Engineer Course** ແລ້ວ (ຫຼືທ່ານເຮັດວຽກ cloud ແບບລົງມືຢູ່ທຸກມື້ນີ້), ໄລຍະທີ 1 ຈະຄຸ້ນເຄີຍ — ອ່ານກວາດຜ່ານໄດ້, ແຕ່ເຮັດບົດຝຶກຫັດຂອງມັນຢູ່ດີ, ເພາະມັນປັບກອບສິ່ງທີ່ທ່ານຮູ້ເປັນຂໍ້ເທັດຈິງໃຫ້ກາຍເປັນສິ່ງທີ່ທ່ານຕ້ອງຊຶ້ງນຳໜັກເປັນການຕັດສິນໃຈ, ແລະ ການປັບກອບນັ້ນເອງ ຄືການປ່ຽນຜ່ານທັງໝົດຈາກວິສະວະກອນສູ່ສະຖາປະນິກ.

ກົດລະບຽບຕະຫຼອດຫຼັກສູດ: ຮຽນ **ມືລະໝັ້ງຊົ່ວໂມງ, ຫົກມືຕ່ອາທິດ** — ຄວາມສະໝ່ຳສະເໝີຊະນະຄວາມໜັກໜ່ວງ. ທຸກ module ຈົບລົງດ້ວຍ **ຕາຕະລາງຄຳສັບ** (ຄຳທີ່ທ່ານຕ້ອງເປັນເຈົ້າຂອງ), **ບົດຝຶກຫັດ** (ວຽກອອກແບບ, ບໍ່ແມ່ນການອ່ານ — ສະຖາປັດຕະຍະກຳຮຽນຮູ້ດ້ວຍການແຕ້ມ ແລະ ການຕັດສິນໃຈ), ແລະ **Milestone** (ຫຼັກໝາຍ — ຫຼັກຖານວ່າທ່ານຮ້ອນກ້າວຕໍ່; ຢ່າຂ້າມ milestone ທີ່ຍັງບໍ່ບັນລຸ). ແລະ ຕັ້ງແຕ່ອາທິດທີ 1, ໃຫ້ຮັກສາ **Design Journal** (ປຶ້ມບັນທຶກການອອກແບບ): ທຸກສະຖາປັດຕະຍະກຳທີ່ທ່ານແຕ້ມ, ທຸກ trade-off ທີ່ທ່ານຊື່, ທຸກຄຳທຳນາຍທີ່ທ່ານເຮັດກ່ຽວກັບຕົນທຶນ ຫຼື ຄວາມລົ້ມເຫຼວ. ອີກສິບເດືອນຈາກນີ້ ມັນຈະກາຍເປັນ portfolio ສຳພາດຂອງທ່ານ; ບໍ່ມີຫຍັງປະທັບໃຈຄະນະກຳມະການຈ້າງງານ ເທົ່າກັບປຶ້ມບັນທຶກມືວັນທີ ຂອງສິລິບການອອກແບບ ພ້ອມບັນທຶກຊື່ສັດວ່າທ່ານຜິດຢູ່ໃສ.

ສະຖາປະນິກ cloud ເຮັດຫຍັງແທ້ໆ

ນຶກເຖິງສະຖາປະນິກອາຄານ. ນາງບໍ່ໄດ້ກໍ່ອິດ — ແຕ່ນາງຕ້ອງຮູ້ຢ່າງແມ່ນຢ່າວ່າອິດເຮັດຫຍັງໄດ້ ແລະ ເຮັດຫຍັງບໍ່ໄດ້, ມັນມີລາຄາເທົ່າໃດ, ແລະ ອາຄານພັງແນວໃດ. ນາງຟັງລູກຄ້າ (“ບ້ານຄອບຄົວ, ແດດດີ, ຢູ່ໃນງົບນີ້, ເທິງດິນຕອນທີ່ຮູບຮ່າງແປກນີ້”) ແລ້ວປ່ຽນຄວາມຢາກ ແລະ ຂໍ້ຈຳກັດ ໃຫ້ເປັນ *ແບບແຕ້ມ ແລະ ຂໍ້ກຳນົດ* ທີ່ແມ່ນຢ່າພໍໃຫ້ຊ່າງກໍ່ສ້າງສ້າງຕາມໄດ້ ແລະ ລູກຄ້າອະນຸມັດໄດ້. ຈາກນັ້ນນາງຢູ່ຕໍ່ຕະຫຼອດການກໍ່ສ້າງ, ຕອບຄຳຖາມ ແລະ ປັບແຜນ ເມື່ອພົ້ນດິນປາກົດວ່າອ່ອນກວ່າທີ່ການສຳຫຼວດບອກ.

ປ່ຽນອິດເປັນ server ແລ້ວນັ້ນຄືວຽກນີ້. ອາທິດຂອງສະຖາປະນິກ cloud ປະກອບດ້ວຍສ່ວນປະສົມຂອງ: **ການຟັງ** ຜູ້ມີສ່ວນໄດ້ເສຍທາງທຸລະກິດ ແລະ ສະກັດຄວາມຕ້ອງການຕົວຈິງ ອອກຈາກຄວາມປາຖະໜາທີ່ມີວ; **ການອອກແບບ** — ເລືອກສ່ວນປະກອບ, ແຕ້ມແຜນວາດ, ຂຽນບັນທຶກການຕັດສິນໃຈ ແລະ ເຫດຜົນ; **ການທົບທວນ** ການອອກແບບຂອງຄົນອື່ນ ຕໍ່ຫົກເສົາຫຼັກ; **ການປະເມີນ** ວ່າການອອກແບບໜຶ່ງຈະມີຄ່າໃຊ້ຈ່າຍເດືອນລະເທົ່າໃດ ແລະ ປົກປ້ອງຕົວເລກນັ້ນຕໍ່ຝ່າຍການເງິນ; **ການວາງແຜນ migrate** ລະບົບເກົ່າຂຶ້ນ cloud; **ການນຳສະເໜີ** — ການອອກແບບອັນດຽວກັນ ອະທິບາຍທາງໜຶ່ງໃຫ້ວິສະວະກອນ ແລະ ອີກທາງທີ່ຕ່າງກັນສື່ນເຊິ່ງໃຫ້ຜູ້ບໍລິຫານ; ແລະ **ການເປັນພິລ່ຽງ** ວິສະວະກອນ ເພື່ອໃຫ້ມາດຕະຖານຢູ່ລອດເມື່ອປະທະກັບເສັ້ນຕາຍ. ໃນຫຼາຍທີມຍັງມີ **pre-sales**: ນັ່ງຂ້າງນັກຂາຍ ຕໍ່ໜ້າລູກຄ້າມຸ່ງຫວັງ, ແຕ້ມຮ່າງ solution ທີ່ຊະນະດີລ.

ສິ່ງທີ່ສະຖາປະນິກ *ບໍ່ແມ່ນ*: ນັກຂຽນໂຄດເກັງທີ່ສຸດໃນຫ້ອງ (ປົກກະຕິບໍ່ແມ່ນ), ຄົນທີ່ພິມຫຼາຍທີ່ສຸດ (ພິມໜ້ອຍທີ່ສຸດ), ຫຼື ອັດສະລິຍະໂດດດ່ຽວທີ່ຢືນແບບແຜນລົງມາໃຫ້ (ວິທີໄວທີ່ສຸດທີ່ຈະຖືກເມີນເສີຍ). ຜົນຜະລິດຕົວຈິງຂອງສະຖາປະນິກຄື **ການຕັດສິນໃຈທີ່ດີ, ຂຽນບັນທຶກໄວ້, ທີ່ຄົນອື່ນເຕັມໃຈສ້າງຕາມ.**

ປະກາດຮັບສະໝັກວຽກ ທີ່ທ່ານກຳລັງຝຶກໄປຫາ

ໃນເດືອນສິງຫາ 2026 ພວກເຮົາໄດ້ດຶງປະກາດ ແລະ ແມ່ແບບວຽກ Cloud/Solutions Architect ຕົວຈິງ — ແມ່ແບບ cloud-architect ຂອງ recruiter (KORE1), ປະກາດວຽກຂອງບໍລິສັດຈັດຫາງານ (4 Corner Resources), ນິຍາມບົດບາດສະຖາປະນິກຂອງ Microsoft ເອງໃນ Azure Well-Architected Framework, ປະກາດ Solutions Architect ສາຍ pre-sales (Arpio, AWS disaster-recovery), ປະກາດ Solution Architect ລະດັບວິສາຫະກິດ (Intel, ຜ່ານ Built In), ແລະ ປະກາດ Cloud Domain Architect ລະດັບວິສາຫະກິດ (Halliburton, Azure-first). ເມື່ອປອກລົດຊາດຂອງແຕ່ລະບໍລິສັດອອກ ຂໍ້ກຳນົດສິບສອງຢ່າງດຽວກັນກໍປາກົດເຊົ້າແລ້ວເຊົ້າອີກ. ຫຼັກສູດນີ້ຖືກສ້າງຕາມລາຍການນັ້ນ — ນິຕິບ່ອນທີ່ແຕ່ລະຂໍ້ຖືກສອນແທ້ໆ:

ສິ່ງທີ່ປະກາດຮັບສະໝັກວຽກຮ້ອງ (ຄຳຂອງເຂົາ, ຖອດຄວາມ)	ຫຼັກສູດນີ້ສອນມັນຢູ່ໃສ
“Design scalable and secure cloud architectures tailored to business and technical requirements” (ອອກແບບສະຖາປັດຕະຍະກຳ cloud ທີ່ຂະຫຍາຍໄດ້ ແລະ ປອດໄພ ຕາມຄວາມຕ້ອງການທຸລະກິດ ແລະ ເຕັກນິກ) — ການອອກແບບ solution ແບບຄົບວົງຈອນ	ໄລຍະທີ 1–3, capstone ໃນໄລຍະທີ 5
“Deep platform expertise” (ຄວາມຊ່ຽວຊານ ແພລດຟອມຂັ້ນເລິກ) ໃນ AWS ແລະ/ຫຼື Azure — compute, storage, networking, IAM, ໂຄງສ້າງບັນຊີ	ໄລຍະທີ 1 + ເສັ້ນທາງ certification ໃນໄລຍະທີ 5
“Run Well-Architected reviews” (ດຳເນີນການທົບທວນ Well-Architected) ຕໍ່ຫຼັກເສົາຫຼັກ	ໄລຍະທີ 2 (ເສົາຫຼັກ), ໄລຍະທີ 5 Module 12 (ການດຳເນີນການທົບທວນ)
“Lead migration/modernization initiatives — which workloads move as-is, get rearchitected, or retire” (ນຳພາການ migrate/modernize — workload ໃດ ຍ້າຍຕາມສະພາບ, ຖືກອອກແບບໃໝ່, ຫຼື ປົດລະວາງ)	ໄລຍະທີ 4 Module 10 (7 Rs, ການວາງແຜນ wave)
“Design landing zones, account structure, guardrails, and reference patterns teams deploy within” (ອອກແບບ landing zone, ໂຄງສ້າງບັນຊີ, guardrail, ແລະ reference pattern ທີ່ທີມ deploy ພາຍໃນ)	ໄລຍະທີ 4 Module 10
“Integrate security and compliance into the design rather than bolting it on” (ບູລະນາການຄວາມປອດໄພ ແລະ compliance ເຂົ້າໃນການອອກແບບ ແທນທີ່ຈະຕິດເສີມພາຍຫຼັງ) — zero trust, identity	ໄລຍະທີ 2 Module 5, ໄລຍະທີ 4 Module 11

ສິ່ງທີ່ປະກາດຮັບສະໝັກຮຽກຮ້ອງ (ຄໍາຂອງເຂົາ, ຖອດຄວາມ)

ຫຼັກສູດນິສອນມັນຢູ່ໃສ

High availability ແລະ disaster recovery — “RTO/RPO gaps, downtime costs, ransomware exposure” (ຊ່ອງວ່າງ RTO/RPO, ຕົ້ນທຶນ downtime, ຄວາມສ່ຽງ ransomware)

ໄລຍະທີ 2 Module 4, ໄລຍະທີ 3 Module 9 (multi-region)

“Own the cloud cost model — tagging, showback, reserved capacity, rightsizing” (ເປັນເຈົ້າຂອງແບບຈໍາລອງຕົ້ນທຶນ cloud — tagging, showback, reserved capacity, rightsizing); ປະເມີນຕົ້ນທຶນ solution

ໄລຍະທີ 2 Module 6, ໄລຍະທີ 5 Module 13 (ການຄິດໄລ່ຕົ້ນທຶນຂັ້ນສະເໝີ)

ການອອກແບບສະຖາປັດຕະຍະກຳເຄືອຂ່າຍ — VPC, hub-and-spoke, ການເຊື່ອມຕໍ່ hybrid

ໄລຍະທີ 1 Module 2, ໄລຍະທີ 3 Module 9, ໄລຍະທີ 4

“Create architectural documentation, diagrams, and standards”; “maintain Architecture Decision Records” (ສ້າງເອກະສານ, ແຜນວາດ, ແລະ ມາດຕະຖານສະຖາປັດຕະຍະກຳ; ຮັກສາ Architecture Decision Records)

ໄລຍະທີ 1 Module 3 (ແຜນວາດ), ໄລຍະທີ 5 Module 12 (ADR, ຊຸດແຜນວາດ)

ການສື່ສານກັບຜູ້ມີສ່ວນໄດ້ເສຍ — “present technical concepts to C-level and technical audiences”; “defend architectural decisions to security, finance, and engineering” (ນຳສະເໜີແນວຄິດເຕັກນິກຕໍ່ຜູ້ບໍລິຫານ C-level ແລະ ຜູ້ຟັງສາຍເຕັກນິກ; ປຶກສາຫາລືການຕັດສິນໃຈສະຖາປັດຕະຍະກຳ ຕໍ່ຝ່າຍຄວາມປອດໄພ, ການເງິນ, ແລະ ວິສະວະກຳ)

ໄລຍະທີ 5 Module 13

“Mentor the cloud and platform engineers who build against your standards” (ເປັນພີລັດຖະວິສະວະກອນ cloud ແລະ platform ທີ່ສ້າງຕາມມາດຕະຖານຂອງທ່ານ); ການສະໜັບສະໜູນ pre-sales — demo, POC, RFP

ໄລຍະທີ 5 Module 13

Certification, ກວດສອບຕໍ່ຕະຫຼາດດຽວກັນ: ບັນໄດມາດຕະຖານຄື **AWS Certified Solutions Architect – Associate (SAA-C03)** — ໃບຢັ້ງຢືນດ່ຽວທີ່ຖືກຮຽກຫາຫຼາຍທີ່ສຸດ, ປົກກະຕິກຽມ 2–3 ເດືອນ — ຕາມດ້ວຍ **AWS Certified Solutions Architect – Professional (SAP-C02)**, ປົກກະຕິອີກ 4–8 ເດືອນ; ບໍລິສັດສາຍ

ໄລຍະທີ 1 — ອົງປະກອບພື້ນຖານ, ໃນຖານະການຕັດສິນໃຈ (ອາທິດທີ 1–6)

ຜູ້ຈົບ Cloud Engineer Course: ອ່ານກວາດຄໍາອະທິບາຍໄດ້, ແຕ່ເຮັດທຸກບົດຝຶກຫັດ. ຂໍເທັດຈິງຄືເກົ່າ; ຄໍາຖາມ ແມ່ນໃໝ່.

Module 1 (ອາທິດທີ 1–2): Cloud ແມ່ນຫຍັງ, ແລະ ສະຖາປະນິກຄິດແນວໃດ

Cloud ໃນໜຶ່ງວັກ (ບົດທວນສໍາລັບຜູ້ເລີ່ມຈາກສູນ). Cloud (ຄລາວ) ຄື ຄອມພິວເຕີຂອງຄົນອື່ນ, ເຊົ່າເປັນຊົ່ວໂມງ, ຄຸ້ມຄອງດ້ວຍ software. AWS, Microsoft Azure, ແລະ Google Cloud ດໍາເນີນສາງເຕັມໄປດ້ວຍ server (data center — ສູນຂໍ້ມູນ) ຈັດກຸ່ມເປັນ **Region** (ພາກພື້ນ) ຕາມພູມສາດ (ເຊັ່ນ “Asia Pacific (Bangkok)”); ແຕ່ລະ region ບັນຈຸ **Availability Zone (AZ)** ທີ່ແຍກອິດສະຫຼະຫຼາຍແຫ່ງ — ອາຄານແຍກກັນ ມີໄຟຟ້າເປັນ ເອກະລາດ, ໃກ້ພໍໃຫ້ເຊື່ອມຕໍ່ໄວ, ໂກພ໌ທີ່ນ້ຳຖ້ວມ ຫຼື ໄຟໄໝ້ຄັງດຽວ ຈະເອົາສອງແຫ່ງໄປພ້ອມກັນບໍ່ໄດ້. ທ່ານເຊົ່າ ສ່ວນແບ່ງຂອງທັງໝົດນີ້ຕໍ່ວິນາທີ: ເຄື່ອງດິບ (IaaS — ທ່ານຄຸ້ມຄອງທຸກຢ່າງເທິງມັນ), ແພລດຟອມ managed (PaaS — ທ່ານເອົາມາແຕ່ແອັບພລິເຄຊັນຂອງທ່ານ), ຫຼື software ສໍາເລັດຮູບ (SaaS — ທ່ານພຽງໃຊ້ມັນ). ນັ້ນ ຄືພື້ນຖານທັງໝົດ. ທຸກສິ່ງທີ່ສະຖາປະນິກອອກແບບ ຖືກຈັດວາງເທິງມັນ.

ຕໍ່ໄປຄືທ່າຂອງສະຖາປະນິກ: ປ່ຽນທຸກຂໍ້ເທັດຈິງເປັນຄໍາຖາມ. ວິສະວະກອນຮຽນວ່າ “AZ ຄື data center ທີ່ແຍກ ອິດສະຫຼະ.” ສະຖາປະນິກຖາມຫັນທີ: “workload ນີ້ສົມຄວນໄດ້ຈັກ AZ?” — ເພາະສອງ AZ ແພງກວ່າໜຶ່ງ, ສາມ ແພງກວ່າສອງ, ແລະ ເວັບໂປຼຊິວການຕະຫຼາດ ບໍ່ສົມຄວນໄດ້ຮັບສິ່ງທີ່ລະບົບຊໍາລະເງິນສົມຄວນໄດ້ຮັບ. ນີ້ຄືແນວ ຄິດທໍາອິດ ແລະ ສໍາຄັນທີ່ສຸດຂອງຫຼັກສູດ:

ສະຖາປັດຕະຍະກໍາ ຄືວິໄນຂອງການເຮັດໃຫ້ trade-off ຈະແຈ້ງອອກມາ. ເກືອບບໍ່ມີສ່ວນປະກອບທີ່ຜິດ — ມີແຕ່ສ່ວນປະກອບທີ່ຜິດ ສໍາລັບ workload ນີ້, ງົບນີ້, ທີ່ມີນີ້, ເສັ້ນຕາຍນີ້.

ສາມຫຼ່ຽມ trade-off. ທຸກການອອກແບບຈະລະຈາລະຫວ່າງສາມແຈ: **ໄວ** (performance — ໄວສໍາລັບຜູ້ໃຊ້, ໄວ ໃນການສ້າງ), **ຖືກ** (ໃບປິ່ນລາຍເດືອນຕໍ່າ, ແຮງງານວິສະວະກໍາຕໍ່າ), ແລະ **ທົນທານ** (ຢູ່ລອດຄວາມລົ້ມເຫຼວ, ຂະຫຍາຍໄດ້, ປອດໄພຢູ່ສະເໝີ). ທ່ານຕົ້ນໄປຫາສອງແຈໃດກໍໄດ້; ແຈທີສາມຈໍາຍາລາຄາໃຫ້. Prototype ຂອງ startup ຄວນໄວ ແລະ ຖືກ — ຄວາມທົນທານລໍໄດ້. ບັນຊີແຍກປະເພດຫຼັກຂອງທະນາຄານຕ້ອງທົນທານ ແລະ ໄວ

— ມັນຈະບໍ່ຖືກ. ເມື່ອຜູ້ມີສ່ວນໄດ້ເສຍເວົ້າວ່າ “ພວກເຮົາຢາກໄດ້ທັງສາມ,” ວຽກຂອງທ່ານຄືຍົມ ແລ້ວຖາມວ່າເຂົາຢາກໄດ້ອັນໃດ *ທີ່ສຸດ*, ເພາະການອອກແບບເລີມບໍ່ໄດ້ຈົນກວ່າເຂົາຈະຕອບ. ແຕ່ມາສາມຫຼ່ຽມນີ້ໄວ້ເທິງສຸດຂອງທຸກການອອກແບບທີ່ທ່ານເຮັດໃນຫຼັກສູດນີ້, ແລະ ໝາຍວ່າ workload ນັ້ນນັ່ງຢູ່ໃສ. ມັນຈະຊ່ວຍທ່ານຝຶນຈາກການຖຽງກັນພັນຄັ້ງ.

Requirements: ວັດຖຸດິບຂອງການອອກແບບ. ສະຖາປະນິກແຍກ **functional requirements** (ຄວາມຕ້ອງການດ້ານໜ້າທີ່ — ລະບົບເຮັດຫຍັງ: “ລູກຄ້າສັງອາຫານທ່ຽງໄດ້”) ອອກຈາກ **non-functional requirements (NFRs)** (ຄວາມຕ້ອງການທີ່ບໍ່ແມ່ນໜ້າທີ່ — ມັນຕ້ອງເຮັດໄດ້ດີປານໃດ: “ພາຍໃນ 2 ວິນາທີ, ສໍາລັບນັກຮຽນ 10,000 ຄົນພ້ອມກັນ, 99.9% ຂອງເວລາ, ພາຍໃນ PDPA”). ມີໃຈໃຫ້ໝັ້ນຄົງກັບລາຍການທໍາອິດ; ສະຖາປະນິກຫາເງິນເດືອນຈາກລາຍການທີ່ສອງ, ເພາະ NFR ຄືສິ່ງທີ່ກໍານົດສະຖາປັດຕະຍະກໍາແທ້ໆ. ສອງຄໍາຖາມວິເສດ ສະກັດ NFR ອອກຈາກຜູ້ມີສ່ວນໄດ້ເສຍ ທີ່ບໍ່ຮູ້ວ່າຕົນມີມັນ: “ຈະເກີດຫຍັງຂຶ້ນກັບທຸລະກິດ ຖ້າອັນນີ້ລົ້ມໄປໜຶ່ງຊົ່ວໂມງ?” ແລະ “ຄວາມສໍາເລັດມີໜ້າຕາແນວໃດ ທີ່ຂະໜາດສືບເທົ່າຂອງມື້ນີ້?”

ຄໍາສັບ:

ຄໍາສັບ	ຄວາມໝາຍ
Region / Availability Zone (AZ)	Region: ກຸ່ມ data center ຕາມພູມສາດທີ່ທ່ານເລືອກແລ່ນຢູ່ໃນ. AZ: data center ທີ່ແຍກອິດສະຫຼະ (ຫຼືກຸ່ມນ້ອຍ) ພາຍໃນມັນ — ຫົວໜ່ວຍຂອງ “ອາຄານດຽວລົ້ມເຫຼວໄດ້.”
IaaS / PaaS / SaaS	ເຊົາເຄື່ອງດິບ / ແບບລາຍການ managed / software ສໍາເລັດຮູບ. ແຖບເລືອນຈາກຄວບຄຸມຫຼາຍທີ່ສຸດ ໄປສູ່ບໍາລຸງຮັກສາໜ້ອຍທີ່ສຸດ.
Workload	ແອັບພລິເຄຊັນ ຫຼື ລະບົບໃດໜຶ່ງ, ຖືເປັນຫົວໜ່ວຍການອອກແບບ — “workload ຊໍາລະເງິນ.”
Functional requirement	ສິ່ງທີ່ລະບົບຕ້ອງເຮັດ (“ລູກຄ້າສັງຊື້ໄດ້”).
Non-functional requirement (NFR)	ມັນຕ້ອງເຮັດໄດ້ດີປານໃດ — ຄວາມໄວ, ຂະໜາດ, uptime, ຄວາມປອດໄພ, compliance, ຕົ້ນທຶນ. NFR ຂັບເຄື່ອນສະຖາປັດຕະຍະກໍາ.
Trade-off	ສິ່ງທີ່ທ່ານຍອມເສຍ ເພື່ອໄດ້ສິ່ງທີ່ທ່ານເລືອກ. ທຸກການຕັດສິນໃຈອອກແບບມີມັນ; ວຽກຂອງສະຖາປະນິກຄືການເອີ້ນຊື່ມັນອອກສຽງ.
Constraint	ຂອບເຂດທີ່ຕໍ່ລອງບໍ່ໄດ້: ງົບ, ເສັ້ນຕາຍ, ກົດໝາຍ, ລະບົບເດີມ, ທັກສະທຶມ. ຂໍ້ຈໍາກັດບໍ່ແມ່ນອຸປະສັກຂອງການອອກ

ຄຳສັບ	ຄວາມໝາຍ
	ແບບ — ພວກມັນ ຄື ໂຈດການອອກແບບ.
Stakeholder	ໃຜກໍ່ຕາມທີ່ມີສ່ວນໄດ້ເສຍໃນລະບົບ: ຜູ້ໃຊ້, ວິສະວະກອນ, ການເງິນ, ຄວາມປອດໄພ, ຜູ້ບໍລິຫານ, ຜູ້ຄຸມກິດ. ຄົນລະກຸ່ມ, ຄົນລະພາສາ — ທ່ານເວົ້າໄດ້ໝົດ.
Greenfield / brownfield	ລະບົບໃໝ່ຫຼ້າໆບໍ່ມີປະຫວັດ (greenfield) ຫຼື ລະບົບທີ່ມີພື້ນຖານຢູ່ກັບລະບົບເດີມ (brownfield — ວຽກຕົວຈິງສ່ວນໃຫຍ່).
Managed service	ສ່ວນປະກອບທີ່ຜູ້ໃຫ້ບໍລິການດຳເນີນການໃຫ້ (ລວມ backup, patch, failover). ຄຳມາດຕະຖານຂອງສະຖາປະນິກ, ເວັ້ນແຕ່ມີເຫດຜົນເປັນລາຍລັກອັກສອນ.

ວິດີໂອສຳລັບ module ນີ້ (ລົງກວດສອບແລ້ວ):

ວິດີໂອ	ຊ່ອງ	ຄວາມຍາວ	ລິງ
Top 50+ AWS Services Explained in 10 Minutes	Fireship	~10 ນາທີ	https://www.youtube.com/watch?v=JlbIYCM48to

ເບິ່ງຫົວຂອງ Fireship ສອງເທື່ອ: ເທື່ອໜຶ່ງຕອນນີ້ເພື່ອແຜນທີ່, ເທື່ອໜຶ່ງທ້າຍໄລຍະທີ 1 — ທ່ານຈະແປກໃຈວ່າຕອນນີ້ທ່ານ ວາງຕຳແໜ່ງ ບໍລິການໄດ້ຈັກອັນໃນການອອກແບບ.

ບົດຝຶກຫັດ: (1) ເລືອກສາມແອັບທີ່ທ່ານໃຊ້ທຸກມື້ (ແອັບທະນາຄານ, ແອັບສົງຄາຫານ, ແອັບວິດີໂອ) ແລ້ວ, ສຳລັບແຕ່ລະອັນ, ຂຽນສາມ NFR ອັນດັບຕົ້ນຂອງມັນ ແລະ ໝາຍມັນລົງເທິງສາມຫຼ່ຽມ trade-off. (2) ສຳພາດໝູ່ກ່ຽວກັບໄອເດຍທຸລະກິດສິບນາທີ ແລ້ວສະກັດ requirement ຫ້າອັນແບບ functional ແລະ ຫ້າອັນແບບ non-functional — ສັງເກດວ່າ NFR ອອກມາກໍຕໍ່ເນື່ອທ່ານຖາມສອງຄຳຖາມວິເສດ. (3) ເລີ່ມ Design Journal ຂອງທ່ານດ້ວຍບັນທຶກ #1: ສາມຫຼ່ຽມ, ແລະ ໜຶ່ງວັກວ່າເປັນຫຍັງ “ເຮົາຈະເສຍສະລະແຈໃດ?” ຈຶ່ງເປັນຄຳຖາມທຸລະກິດ, ບໍ່ແມ່ນຄຳຖາມເຕັກນິກ.

Milestone: ເມື່ອໄດ້ຄຳບັນຍາຍລະບົບໃດກໍຕາມໜຶ່ງປະໂຫຍກ, ທ່ານສາມາດອອກ NFR ທີ່ເປັນໄປໄດ້ຂອງມັນ ແລະ ຕຳແໜ່ງຂອງມັນເທິງສາມຫຼ່ຽມ ພາຍໃນຫ້ານາທີ, ອອກສຽງ, ໂດຍບໍ່ເບິ່ງບັນທຶກ.

Module 2 (ອາທິດທີ 3–4): Compute, storage, database, network — ສຶກສາຕັດສິນໃຈ

ທຸກການອອກແບບທີ່ທ່ານຈະແຕ້ມຕະຫຼອດຊີວິດ ຄືສື່ທາງເລືອກນີ້, ເລືອກຢ່າງມີເຈດຕະນາ. ນີ້ຄືແຕ່ລະອັນ ສອນໃນ ຖານະການຕັດສິນໃຈ.

ການຕັດສິນໃຈທີ 1 — Compute: VM, container, ຫຼື serverless? Virtual machine (VM) (ເຄື່ອງຈັກເສມືອນ) ຄືສ່ວນແບ່ງເຊົ່າຂອງ server ທີ່ປະພຶດຄືຄອມພິວເຕີແຕ່ມໝວຍ — ຄວບຄຸມສູງສຸດ, ບຳລຸງຮັກສາສູງສຸດ (ທ່ານ patch ມັນ, ທ່ານ scale ມັນ, ທ່ານຈ່າຍຂະນະມັນຫວ່າງ). **Container** ຫຸ້ມຫໍ່ແອັບພລິເຄຊັນໜຶ່ງອັນຜ່ອມທຸກສິ່ງທີ່ມັນຕ້ອງການ ເພື່ອໃຫ້ມັນແລ່ນຄືກັນທຸກປະການໃນທຸກບ່ອນ; orchestrator (**Kubernetes**) ແລ່ນ ແລະ ຮັກສາຝູງຂອງພວກມັນ — ຄວາມໝາແໜ້ນ ແລະ ຄວາມເຄື່ອນຍ້າຍໄດ້ດີເລີດ, ແຕ່ທ່ານໄດ້ຮັບເອົາແພລດຟອມທີ່ຊັບຊ້ອນ ທີ່ຕ້ອງການຄົ້ນພົບກັບສະ. **Serverless** (AWS Lambda, Azure Functions) ແລ່ນໂຄດຂອງທ່ານສະເພາະເມື່ອຖືກກະຕຸ້ນ ແລະ ຄິດເງິນຕໍ່ການເອີ້ນ — ບຳລຸງຮັກສາເກືອບສູນ ແລະ ສົມບູນແບບສຳລັບ traffic ແບບຝຸ່ງເປັນຊ່ວງ, ແຕ່ມີຂໍ້ຈຳກັດ (ເພດານເວລາປະມວນຜົນ, cold start, ແລະ ການແຕ່ງງານເລິກກວ່າກັບຜູ້ໃຫ້ບໍລິການດຽວ). ຕາຕະລາງການຕັດສິນໃຈ ທີ່ທ່ານຄວນສ້າງຄືນຈາກຄວາມຈຳໄດ້:

ເລືອກ	ເມື່ອໃດ	ລະດັບຫຍັງ
VM	Software ເກົ່າ, ໃບອະນຸຍາດພິເສດ, ຕ້ອງການຄວບຄຸມ OS ເຕັມທີ, ໂຫຼດຄົງທີ່ຄາດເດົາໄດ້	ທ່ານເປັນເຈົ້າຂອງ patching, scaling, ແລະ ຕີສາມ; ເວລາຫວ່າງຄິດເງິນເຕັມ
Containers + Kubernetes	ຫຼາຍ service, ທີມມີທັກສະຢູ່ແລ້ວ, ຄວາມເຄື່ອນຍ້າຍໄດ້ສຳຄັນ	ຄວາມຊັບຊ້ອນຂອງແພລດຟອມ — K8s ຄົວຽກເຕັມເວລາ; ເກີນຄວາມຈຳເປັນສຳລັບທີມນ້ອຍ
Serverless	Traffic ຝຸ່ງເປັນຊ່ວງ ຫຼື ຄາດເດົາບໍ່ໄດ້, ກາວເຊື່ອມແບບ event-driven, ທີມນ້ອຍ, ຕ້ອງອອກຕະຫຼາດໄວ	ຂໍ້ຈຳກັດ runtime, cold start, ຄາດຕົ້ນທຶນຍາກຂຶ້ນເມື່ອໂຫຼດຄົງທີ່ຂະໜາດໃຫຍ່, lock-in

ຄວາມເຂົ້າໃຈຂັ້ນອາວຸໂສ: **ນີ້ຄືທາງເລືອກຕໍ່ workload, ບໍ່ແມ່ນສາສະໜາຂອງບໍລິສັດ.** ລະບົບຕົວຈິງແລ່ນທັງສາມແບບຄຽງຄູ່ກັນ, ຢ່າງຖືກຕ້ອງ.

ການຕັດສິນໃຈທີ 2 — Storage: object, block, ຫຼື file — ແລະ ຮ້ອນປານໃດ? Object storage (Amazon S3) ຄືຖັງບໍ່ມີກິນສຳລັບໄຟລີ້ — ຖືກ, ທຶນທານແບບເຫຼືອເຊື້ອ (“eleven nines”), ຄຳຕອບມາດຕະຖານຂອງ “ໄຟລີ້ໄປໄວ້ໃສ່?” **Block storage** (EBS) ຄືດິສເສມືອນທີ່ຕິດກັບ VM. **File storage** (EFS) ຄືໂດຣຟ໌ແບ່ງປັນ ທີ່ຫຼາຍເຄື່ອງ mount ຜ່ອມກັນ. ມີຕິເສີມຂອງສະຖາປະນິກຄື **ອຸນຫະພູມ:** ຂໍ້ມູນ hot (ເຂົ້າເຖິງຕະຫຼອດ, ຕັ້ງລາຄາເພື່ອຄວາມໄວ) ຫຼຽບກັບຂັ້ນ cold/archive (Glacier — ຫຼຽນເສດ, ແຕ່ໃຊ້ນາທີຫາຊົ່ວໂມງໃນການດຶງຄືນ). ການອອກແບບ lifecycle rule ທີ່ຄ່ອຍໆເລື່ອນຂໍ້ມູນເກົ່າໄປຂັ້ນ cold ຄືໄຊຊະນະຕົ້ນທຶນທີ່ຖືກທີ່ສຸດໃນ cloud; ການລືມເຮັດຄືສິ່ງທີ່ພົບເລື້ອຍທີ່ສຸດ.

ການຕັດສິນໃຈທີ 3 — Database: SQL ຫຼື NoSQL (ແລະ ລົດຊາດ managed ອັນໃດ)? Database ແບບ relational/SQL (PostgreSQL, MySQL; ແບບ managed ຄື RDS/Aurora) ເກັບຂໍ້ມູນໃນຕາຕະລາງເຄັງຄັດ

ພ້ອມ consistency ທີ່ຮັບປະກັນ — ຄຳມາດຕະຖານສຳລັບທຸກຢ່າງທີ່ຄວາມຖືກຕ້ອງເປັນສິ່ງສັກສິດ: ເງິນ, ຄຳສັ່ງ ຊື້, ສາງສິນຄ້າ, ຜູ້ໃຊ້. Database ແບບ **NoSQL** (DynamoDB, MongoDB) ແລກໂຄງສ້າງເຄັ່ງຄັດ ກັບຄວາມ ຍືດຍຸ່ນ ແລະ ການຂະຫຍາຍແນວນອນເກືອບບໍ່ຈຳກັດ — ຄຳມາດຕະຖານສຳລັບ session, catalog, feed, telemetry. ຫຼັກຕັດສິນ: *ເລີ່ມດ້ວຍ SQL ເວັ້ນແຕ່ທ່ານບອກຊື່ເຫດຜົນສະເພາະທີ່ມັນຈະໃຊ້ບໍ່ໄດ້* (ຂະໜາດສຸດຂີດ, schema ຍືດຍຸ່ນ, ການອ່ານຫົວໂລກລະດັບ millisecond ຫຼັກດຽວ). ແລະ ໃນ cloud, “database” ເກືອບສະເໝີ ຄວນໝາຍເຖິງ “**managed database**” — ຜູ້ໃຫ້ບໍລິການຈັດການ backup, patch, ແລະ failover; ທີ່ມີແຜ່ນ database ຂອງຕົນເອງເທິງ VM ຄວນມີເຫດຜົນເປັນລາຍລັກອັກສອນ. ເພີ່ມຜູ້ຊ່ຽວຊານສະເພາະໃສ່ຄຳສັບຂອງ ທ່ານ: **cache** (Redis — ຂໍ້ມູນ hot ໃນ memory, ອ່ານລະດັບ microsecond), **warehouse** (ການວິເຄາະ ຂະໜາດໃຫຍ່ — ໄລຍະທີ 3), **queue** (ບໍ່ແມ່ນ database, ແຕ່ມັກເປັນຊິ້ນສ່ວນທີຂາດ — ໄລຍະທີ 3).

ການຕັດສິນໃຈທີ 4 — Network: ຮູບຮ່າງຂອງໂລກສ່ວນຕົວ. VPC (Virtual Private Cloud) ຄືສ່ວນທີ່ມີຮູບ ກັນຂອງທ່ານ ໃນເຄືອຂ່າຍຂອງຜູ້ໃຫ້ບໍລິການ. ພາຍໃນມັນ, **public subnet** ຖືສິ່ງທີ່ອິນເຕີເນັດແຕະຕ້ອງໄດ້ (load balancer), ແລະ **private subnet** ຖືທຸກຢ່າງອື່ນ — app server ແລະ, ສະເໝີ, database. “*The database sits in a private subnet*” (database ຢູ່ໃນ private subnet) ຄືປະໂຫຍກທີ່ຖືກເວົ້າຊ້າຫຼາຍທີ່ສຸດ ໃນການທົບ ທວນສະຖາປັດຕະຍະກຳ; ເຫດຜົນ — ບໍ່ມີຫຍັງໂຈມຕີສິ່ງທີ່ບໍ່ມີເສັ້ນທາງຈາກອິນເຕີເນັດໄດ້ — ຄືເຄິ່ງໜຶ່ງຂອງ ຄວາມປອດໄພເຄືອຂ່າຍ. ອ້ອມ VPC: **load balancer** ກະຈາຍ traffic ໃຫ້ຫຼາຍ server ແລະ ອ້ອມຜ່ານເຄື່ອງທີ່ ບໍ່ສະບາຍ; **DNS** (Route 53) ປ່ຽນຊື່ເປັນທີ່ຢູ່; **CDN** (CloudFront) cache ເນື້ອຫາໄວ້ໃນຫຼາຍຮ້ອຍເມືອງ ໃຫ້ໄວທຸກບ່ອນ; **API gateway** ຄືໂຕະຕ້ອນຮັບແບບ managed ຂອງ API ຂອງທ່ານ (ການຢັ້ງຢືນຕົວຕົນ, ຈຳກັດ ອັດຕາ, ບັນທຶກ log). ການເຊື່ອມກັບໂລກເກົ່າ: **VPN** (ອຸໂມງເຂົ້າລະຫັດຜ່ານອິນເຕີເນັດ) ຫຼື **Direct Connect** (ສາຍກາຍະພາບສ່ວນຕົວ) — ສາຍແຮ່ຂອງທຸກການອອກແບບ hybrid ໃນໄລຍະທີ 4.

ຄຳສັບ:

ຄຳສັບ	ຄວາມໝາຍ
Instance / instance type	VM ເຊົ່າໜຶ່ງໜ່ວຍ / ຂະໜາດຂອງມັນ (CPU + RAM), ຊຶ່ງກຳນົດລາຄາຕໍ່ຊົ່ວໂມງ.
Container / Docker / Kubernetes (K8s)	ຫີບຫໍ່ເຄືອນຍ້າຍໄດ້ຂອງແອັບພລິເຄຊັນ / ເຄື່ອງມືສ້າງ ແລະ ແຜ່ນພວກມັນ / orchestrator ທີ່ແຜ່ນ ແລະ ຮັກສາຝູງ ຂອງພວກມັນ.
Serverless / Lambda	ໂຄດທີ່ແຜ່ນສະເພາະເມື່ອຖືກກະຕຸ້ນ, ຄິດເງິນຕໍ່ການແຜ່ນ, ບໍ່ມີ server ໃຫ້ຄຸ້ມຄອງ. Lambda ຄືສະບັບຂອງ AWS.
Cold start	ຄວາມຊັກຊ້າເລີ່ມ ເມື່ອ serverless function ແຜ່ນ ຫຼັງຈາກຫວ່າງມາດົນ — trade-off ຄວາມສະດວກຂອງ serverless.

ຄຳສັບ	ຄວາມໝາຍ
S3 / bucket / durability	Object storage ຂອງ AWS / ພາຊະນະໄຟລີມີຊື່ໜຶ່ງອັນ / ໂອກາດທີ່ຂໍ້ມູນຢູ່ລອດ — 99.999999999% ຂອງ S3 ໝາຍວ່າການສູນເສຍເກືອບບໍ່ເກີດຈັກເທືອ.
Storage tier / lifecycle policy	ຂັ້ນລາຄາ-ຄວາມໄວ (hot → cold → archive) / ກົດອັດຕະໂນມັດທີ່ຍ້າຍຂໍ້ມູນເກົ່າໄປຂັ້ນຖືກກວ່າ.
RDS / Aurora / DynamoDB	ບໍລິການ SQL ແບບ managed ຂອງ AWS / SQL ປະສິດທິພາບສູງແບບ cloud-native ຂອງມັນ / NoSQL managed ເຮືອທຸງຂອງມັນ.
Consistency	ການຮັບປະກັນວ່າທຸກຄົນທີ່ອ່ານຂໍ້ມູນ ເຫັນຄວາມຈິງດຽວກັນໃນເວລາດຽວກັນ — ພະລັງພິເສດຂອງ SQL, ແລະ ສິ່ງທີ່ NoSQL ຜ່ອນຄາຍເພື່ອໄດ້ຂະໜາດ.
Cache / Redis	ສຳເນົາຄວາມໄວ memory ຂອງຂໍ້ມູນ hot ວາງໜ້າ database / ເຄື່ອງມືມາດຕະຖານສຳລັບມັນ.
VPC / subnet (public, private)	ສ່ວນເຄືອຂ່າຍສ່ວນຕົວຂອງທ່ານ / ສ່ວນຍ່ອຍຂອງມັນ — public ຫັນໜ້າຫາອິນເຕີເນັດ, private ບໍ່. Database ຢູ່ private. ສະເໝີ.
Load balancer / health check	ຜູ້ຄວບຄຸມຈະລາຈອນຂ້າມ server / ການທົດສອບຊີບພະຈອນ ທີ່ມັນໃຊ້ຢຸດສົ່ງ traffic ຫາເຄື່ອງທີ່ຕາຍ.
CDN / edge	Cache ລະດັບເມືອງຂອງເນື້ອຫາທ່ານທົ່ວໂລກ / “the edge” = ໃກ້ຜູ້ໃຊ້.
API / API gateway	ປະຕູຄວບຄຸມທີ່ software ອັນໜຶ່ງເປີດໃຫ້ອີກອັນ / ໂຕະຕ້ອນຮັບແບບ managed ຂອງປະຕູເຫຼົ່ານັ້ນ.
VPN / Direct Connect	ອຸໂມງເຂົ້າລະຫັດຜ່ານອິນເຕີເນັດ / ສາຍກາຍະພາບສ່ວນຕົວສູ່ cloud — ສອງວິທີທີ່ on-prem ພົບ cloud.

ວິດີໂອສຳລັບ module ນີ້ (ລິ້ງກວດສອບແລ້ວ):

ວິດີໂອ	ຊ່ອງ	ຄວາມຍາວ	ລິ້ງ
Kubernetes explained in 15	TechWorld with Nana	~16 ນາທີ	https://www.youtube.com/watch?v=VnvRFRk_51k

ວິດີໂອ	ຊ່ອງ	ຄວາມຍາວ	ລິງ
mins			
Serverless Computing in 100 Seconds	Fireship	~2 ນາທີ	https://www.youtube.com/watch?v=W_VV2Fx32_Y
AWS Networking Basics (VPC & Subnets)	KodeKloud	~30 ນາທີ	https://www.youtube.com/watch?v=QM63dyA_4Pc

ບົດຝຶກຫັດ: (1) ສໍາລັບຫ້າ workload ເຫຼົ່ານີ້, ເລືອກ compute, database, ແລະ storage, ພ້ອມຂຽນເຫດຜົນອັນລະຫຼັງປະໂຫຍກ: ເວັບສັງຄານທ່າງຂອງໂຮງຮຽນ; ບັນຊີທຸລະກໍາຂອງທະນາຄານ; ແອັບແປງປັນຮູບ; ຕົວສ້າງລາຍງານກາງຄືນທີ່ແລ່ນ 20 ນາທີ; ແອັບແຊັດສໍາລັບຜູ້ໃຊ້ 5 ລ້ານຄົນ. (2) ເອົາທາງເລືອກໜຶ່ງຂອງທ່ານ ແລ້ວໂຕ້ແຍ້ງທາງເລືອກ *ກົງກັນຂ້າມ* ໃຫ້ໜ້າເຊື່ອທີ່ສຸດເທົ່າທີ່ເຮັດໄດ້ — ສະຖາປະນິກທີ່ steel-man ທາງເລືອກອື່ນບໍ່ໄດ້ຍັງບໍ່ເຂົ້າໃຈ trade-off. (3) ປຶ້ມບັນທຶກ: ຕາຕະລາງການຕັດສິນໃຈ compute ຂອງທ່ານ, ຈາກຄວາມຈໍາ.

Milestone: ເມື່ອໄດ້ workload ໃດກໍຕາມໃນໜຶ່ງປະໂຫຍກ, ທ່ານບອກຊີ້ສືການຕັດສິນໃຈຂອງມັນ ພ້ອມເຫດຜົນພາຍໃນສາມນາທີ — ແລະ ຢ່າງໜ້ອຍໜຶ່ງການຕັດສິນໃຈ, ບອກໄດ້ວ່າຫຍັງຈະເຮັດໃຫ້ທ່ານປ່ຽນໃຈ.

Module 3 (ອາທິດທີ 5–6): ການອ່ານ ແລະ ແຕ້ມແຜນວາດສະຖາປັດຕະຍະກໍາ

ສະຖາປະນິກທີ່ແຕ້ມບໍ່ໄດ້ ຄືທີ່ປຶກສາທີ່ໄດ້ແຕ່ເວົ້າ. ແຜນວາດຄືພາສາເຮັດວຽກຂອງທ່ານ: module ນີ້ເຮັດໃຫ້ທ່ານຄ່ອງໃນການອ່ານພວກມັນ ແລະ ສາມາດແຕ້ມພວກມັນ.

ແຜນວາດຕົ້ນແບບ — ຮຽນອັນນີກອນ. Three-tier architecture (ສະຖາປັດຕະຍະກໍາສາມຊັ້ນ) ຄື “ໂຄງສ້າງປະໂຫຍກ” ຂອງແຜນວາດ cloud; ການອອກແບບສ່ວນໃຫຍ່ຄືການປ່ຽນແປງຂອງມັນ:

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

→ [Cache]

→ [Database: primary +

standby, private subnets]

ຊັ້ນ 1 (presentation): ສິ່ງທີ່ຜູ້ໃຊ້ແຕະຕ້ອງ — ເນື້ອຫາ static ຈາກ CDN, ຄໍາຮ້ອງຂໍຜ່ານ load balancer. **ຊັ້ນ 2 (application):** ຝູງ server ທີ່ແທນກັນໄດ້ ແລ່ນ logic ຂອງທ່ານ, ໃນ private subnet, ຂະຫຍາຍໂດຍອັດຕະໂນມັດ. **ຊັ້ນ 3 (data):** database, ເລິກທີ່ສຸດ ແລະ ຖືກປົກປ້ອງທີ່ສຸດ, ພ້ອມ standby ໃນ AZ ທີ່ສອງ.

Traffic ໄຫຼທາງດຽວ: ຜູ້ໃຊ້ບໍ່ແຕະຊັ້ນ app ໂດຍກົງຈັກເທື່ອ, ຊັ້ນ app ເທົ່ານັ້ນທີ່ເວົ້າກັບຊັ້ນ data. ຝຶກແຕ້ມອັນນີ

ຈົນມີເຮັດເອງໂດຍສະໝອງບໍ່ຕ້ອງສັງ — ມັນຄືບົດເປີດການສໍາພາດກະດານຂາວ ຢູ່ທຸກມຸມໂລກ.

ທ່ານຽມສັນຍາລັກ ທີ່ເຮັດໃຫ້ທ່ານເບິ່ງເປັນມືອາຊີບ (ເພາະມັນເຮັດໃຫ້ທ່ານ ຄິດ ແບບມືອາຊີບ): ກ່ອງຄືສ່ວນປະກອບ — ຕິດປ້າຍແຕ່ລະອັນວ່າມັນແມ່ນຫຍັງ ແລະ ໃຊ້ບໍລິການໃດ (“App servers — EC2, auto-scaling group”). ລູກສອນສະແດງທິດທາງທີ່ ຄໍາຮ້ອງຂໍ ໄຫຼ, ຕິດປ້າຍ protocol ເມື່ອສໍາຄັນ. ຂອບເສັ້ນປະສະແດງຂອບເຂດ — VPC, ແຕ່ລະ subnet, ແຕ່ລະ AZ (ແຕ້ມຂອບ AZ ແລ້ວ multi-AZ ຈະເບິ່ງເຫັນໄດ້ ແທນທີ່ຈະເປັນພຽງຄໍາອ້າງ). ຜູ້ໃຊ້/actor ຢືນຢູ່ນອກລະບົບ. ຕົວເລກເທິງລູກສອນ (1, 2, 3...) ໃຫ້ທ່ານບັນຍາຍການເດີນທາງຂອງຄໍາຮ້ອງຂໍ. ແລະ ທຸກແຜນວາດມີຫົວຂໍ້, ວັນທີ, ແລະ legend. ກົດທີ່ເລິກກວ່າ: **ໝຶ່ງແຜນວາດ, ໝຶ່ງກຸ່ມຜູ້ຟັງ, ໝຶ່ງຄໍາຖາມ.** ແຜນວາດທີ່ສະແດງທຸກຢ່າງ ບໍ່ສະແດງຫຍັງເລີຍ; ທ່ານຈະໄດ້ຮຽນຊຸດມາດຕະຖານຂອງລະດັບ zoom (context → container → deployment) ໃນໄລຍະທີ 5.

ການອ່ານແຜນວາດຂອງຄົນອື່ນ — x-ray ຂອງສະຖາປະນິກ. ເມື່ອໄດ້ຮັບແຜນວາດ, ແລ່ນການສະແດງນິຍົມສຽງ: ອີນເຕີເນັດແຕ່ລະລະບົບນີ້ຢູ່ໃສ (ທຸກຈຸດສໍາຜັດຄື attack surface)? ຂໍ້ມູນຢູ່ໃສ, ແລະ ມັນຢູ່ໃນ private subnet ບໍ່? ຫຍັງມີຄູ່ (ທຶນທານ) ແລະ ຫຍັງເປັນ **single point of failure** — ກ່ອງທີ່ບໍ່ມີແຜດ? ບ່ອນໃດຈະເຈັບເມື່ອ traffic 10×? ແຕ່ລະກ່ອງມີຄ່າໃຊ້ຈ່າຍເດືອນລະເທົ່າໃດ? ຫ້າຄໍາຖາມ, ສາມສິບວິນາທີ, ແລ້ວທ່ານໄດ້ອ່ານແຜນວາດ ຄືກັບທີ່ໝໍອ່ານ x-ray. ທ່ອງ AWS Architecture Center (<https://aws.amazon.com/architecture/>) ແລ້ວແລ່ນການສະແດງໃສ່ reference architecture ທີ່ຕິພົມສາມອັນ.

ຄໍາສັບ:

ຄໍາສັບ	ຄວາມໝາຍ
Three-tier architecture	ການແຍກຄລາສສິກ: presentation (ທາງເຂົ້າຜູ້ໃຊ້) → application (logic) → data (database). ຮູບຮ່າງມາດຕະຖານຂອງລະບົບເວັບ.
Tier / layer	ຊັ້ນແນວນອນຂອງລະບົບທີ່ມີໜ້າທີ່ດຽວ, ເວົ້າກັບພຽງເລື່ອນບ້ານຂອງມັນ.
Single point of failure (SPOF)	ສ່ວນປະກອບໃດກໍຕາມ ທີ່ຄວາມຕາຍໂດດດ່ຽວຂອງມັນດຶງລະບົບລົງ. ສິ່ງທ້າອິດທີ່ຕ້ອງລ່າໃນທຸກແຜນວາດ.
Multi-AZ	ການແລ່ນຕົວສໍາເນົາຂ້າມຢ່າງໜ້ອຍສອງ Availability Zone ເພື່ອໃຫ້ data center ໜຶ່ງລົ້ມເຫຼວໄດ້ໂດຍບໍ່ມີໃຜເຫັນ.
Auto-scaling (group)	ເຄື່ອງຖືກເພີ່ມ ແລະ ຖອດອອກໂດຍອັດຕະໂນມັດຕາມຄວາມຕ້ອງການ — ຄວາມຈຸທີ່ຫາຍໃຈໄດ້.
Stateless / stateful	Server ທີ່ບໍ່ຖືຂໍ້ມູນສະເພາະ (ແຜດໃດກໍແທນໄດ້, ຈຶ່ງຂະຫຍາຍໄດ້ອິດສະຫຼະ) ທຽບກັບອັນທີ່ຖືຂໍ້ມູນທີ່ຫ້າມເສຍ. ເປົ້າໝາຍອອກແບບ: ຂຶ້ນ app ເປັນ stateless, state ຖືກຢູ່ລົງສູ່ database ແລະ cache.

ຄຳສັບ	ຄວາມໝາຍ
Reference architecture	ການອອກແບບຕົວຢ່າງທີ່ຜູ້ໃຫ້ບໍລິການຕິພົມຮັບຮອງສຳລັບບັນຫາທົ່ວໄປ — ສະຖາປະນິກປະກອບຈາກພວກມີກ່ອນຈະປະດິດ.
Context diagram	ລະດັບ zoom ສູງສຸດ: ລະບົບຂອງທ່ານເປັນກ່ອງດຽວ, ບວກຜູ້ໃຊ້ ແລະ ລະບົບພາຍນອກທີ່ມັນສຳຜັດ.
Attack surface	ທຸກຈຸດທີ່ໂລກພາຍນອກແຕະລະບົບໄດ້. ນ້ອຍກວ່າ ປອດໄພກວ່າ.
North–south / east–west traffic	Traffic ເຂົ້າ/ອອກລະບົບ ທຽບກັບ traffic ລະຫວ່າງສ່ວນປະກອບພາຍໃນມັນ.

ບົດຝຶກຫັດ: (1) ແຕ້ມແຜນວາດສາມຊັ້ນຈາກຄວາມຈຳ, ຫ້າມິຕິດຕໍ່ກັນ, ຈົນມັນໃຊ້ເວລາຕໍ່າກວ່າສິນາທິ ພ້ອມຂອບເຂດທັງໝົດ (VPC, subnet, ສອງ AZ) ຖືກແຕ້ມ. (2) ເອົາຫ້າ workload ຂອງ Module 2 ມາແຕ້ມແຕ່ລະອັນ — ຊາວນາທີ່ແຜນວາດ, ບັງຄັບໃຊ້ກົດສັນຍາລັກ. (3) ຊອກຫາແຜນວາດສະຖາປະນາປັດຕະຍະກຳຈິງໃດກໍໄດ້ອອນລາຍ (AWS Architecture Center ມີຫຼາຍຮ້ອຍ) ແລ້ວຂຽນການສະແດງນ x-ray ຫ້າຄຳຖາມຂອງມັນ ໃນປຶ້ມບັນທຶກ. (4) ບົດຝຶກບັນຍາຍ: ແຜນວາດຢູ່ຕໍ່ໜ້າ, ບັນຍາຍການຄລິກຂອງຜູ້ໃຊ້ຄົນໜຶ່ງ ຈາກ browser ຫາ database ແລ້ວກັບຄືນ, ອອກສຽງ, ນັບເລກລູກສອນໄປນຳ.

Milestone — ຈົບໄລຍະທີ 1: ເທິງກະດານຂາວ (ຫຼືເຈ້ຍ, ຖ່າຍຮູບໄວ້ໃນປຶ້ມບັນທຶກ), ທ່ານແຕ້ມການອອກແບບສາມຊັ້ນທີ່ຖືກຕ້ອງ ສັນຍາລັກຄົບ ສຳລັບ workload ໃໝ່ໜຶ່ງປະໂຫຍກ ພາຍໃນສິບຫ້ານາທີ, ບັນຍາຍຄຳຮ້ອງຂໍຜ່ານມັນ, ແລະ ຕອບ “ຫຍັງຝັງຖ້າກ່ອງນິຕາຍ?” ສຳລັບທຸກກ່ອງ. ການແຕ້ມ-ບວກ-ການສອບຖາມນີ້ ຄືເຄິ່ງທຳອິດຂອງການສຳພາດສະຖາປະນິກຕົວຈິງແທ້ໆ — ຈາກນີ້ໄປ, ທຸກຢ່າງຄືຄວາມເລິກ.

ໄລຍະທີ 2 — ຂອບເຂດການອອກແບບ (ອາທິດທີ 7–14)

ແຜນທີ່ຂອງໄລຍະນີ້ຄື AWS Well-Architected Framework — ລາຍການກວດສອບຮ່ວມຂອງວົງການ ວ່າ “ອອກແບບຢ່າງຖືກຕ້ອງ” ໝາຍເຖິງຫຍັງ, ຈັດເປັນຫົກເສົາຫຼັກ: **operational excellence** (ຄວາມເປັນເລີດດ້ານປະຕິບັດການ), **security** (ຄວາມປອດໄພ), **reliability** (ຄວາມໝັ້ນເຊື່ອຖື), **performance efficiency** (ປະສິດທິພາບ), **cost optimization** (ການເພີ່ມປະສິດທິພາບຕົ້ນທຶນ), **sustainability** (ຄວາມຍືນຍົງ). (Azure ມີ framework ທີ່ເກືອບຄືກັນ; ຮຽນອັນໜຶ່ງໃຫ້ເລິກ ແລ້ວທ່ານໄດ້ຮຽນທັງສອງ.) ປະກາດຮັບສະໝັກວຽກຮຽກຫາສະຖາປະນິກທີ່ “run Well-Architected reviews” ໄດ້ໂດຍລະບຸຊື່, ດັ່ງນັ້ນພວກເຮົາຈຶ່ງເອົາເສົາຫຼັກມາທີ່ລະອັນ, ແລະ ສຳລັບແຕ່ລະອັນ ທ່ານຈະຮຽນສາມສິ່ງ: **ຄຳຖາມຫຼັກ** ຂອງເສົານັ້ນ, **pattern** ມາດຕະຖານ ຂອງມັນ (ຄຳ

ຕອບທີ່ໜ້າເບື້ອງ ແຕ່ພິສູດແລ້ວ — ສະຖາປະນິກປະກອບກ່ອນຈະປະດິດ), ແລະ ຕົວຢ່າງທີ່ເຮັດໃຫ້ເບິ່ງ ເທິງ ສະຖານະການທີ່ແລ່ນຕໍ່ເນື່ອງ.

ສະຖານະການທີ່ແລ່ນຕໍ່ເນື່ອງຕະຫຼອດໄລຍະທີ 2: *ThaiTicket*, ແຜນລາຍງານສົມມຸດໃນບາງກອກ. ໂຫຼດ ປົກກະຕິ: ຜູ້ເຂົ້າຊົມ 2,000 ຄົນ/ຊົ່ວໂມງ. ແຕ່ເມື່ອປີຂອງສິນລະປິນດັ່ງເປີດຂາຍ 10:00 ໂມງເຊົ້າ, ມັນຮັບຜູ້ເຂົ້າຊົມ 400,000 ຄົນໃນສິບນາທີ, ການຊຳລະເງິນໜ້າຂາຍບ່ອນນັ້ງຊ້າ, ແລະ ຂໍ້ມູນສ່ວນບຸກຄົນຂອງລູກຄ້າໄຫຼຢູ່ພາຍໃຕ້ PDPA. ໄວ, ຖືກ, ຫົນຫານ — *ThaiTicket* ຕ້ອງການທັງສາມ ແລະ ມີທັງສາມບໍ່ໄດ້, ຊຶ່ງນັ້ນເອງເຮັດໃຫ້ມັນເປັນຄົນ ເຈັບຝຶກຫັດທີ່ສົມບູນແບບ.

ເບິ່ງກ່ອນ Module 4 (ລິ້ງກວດສອບແລ້ວ):

ວິດີໂອ	ຊ່ອງ	ຄວາມຍາວ	ລິ້ງ
The Five Pillars of the AWS Well-Architected Framework	Amazon Web Services (ທາງການ; ເສົາທີ່ຫົກ, Sustainability, ຖືກ ເພີ່ມພາຍຫຼັງ)	~4 ນາທີ	https://www.youtube.com/watch?v=KvEDbPmha6o
What is the AWS Well-Architected Framework?	Tech With Lucy	~10 ນາທີ	https://www.youtube.com/watch?v=MpDJ6TCWKjk

ແລະ bookmark ຕົວ framework ເອງໄວ້ — <https://aws.amazon.com/architecture/well-architected/> ແລະ ເອກະສານສະບັບເຕັມທີ່ <https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html> — ທ່ານຈະໃຊ້ຊີວິດ ຢູ່ໃນນັ້ນແປດອາທິດ.

Module 4 (ອາທິດທີ 7–8): Reliability — ອອກແບບເພື່ອມີທີ່ສົ່ງຕ່າງໆຝັ່ງ

ຄຳສາລະພາບຂອງເສົານີ: ທຸກສິ່ງລົ້ມເຫຼວ, ຕະຫຼອດເວລາ. ດິສຕາຍ, AZ ຖືກນ້ຳຖ້ວມ, ການ deploy ຜິດພາດ, ແລະ ໃບຢັ້ງຢືນສັກໜ່ວຍໃດໜຶ່ງກໍາລັງຈະໝົດອາຍຸຢູ່ສະເໝີ. Reliability ບໍ່ແມ່ນການປາສະຈາກຄວາມລົ້ມເຫຼວ — ມັນຄືການເຮັດໃຫ້ຄວາມລົ້ມເຫຼວ *ບໍ່ມີຄວາມໝາຍ*. ອອກແບບເພື່ອວ່າເມື່ອ (ບໍ່ແມ່ນ ຖ້າ) ສ່ວນປະກອບໜຶ່ງ ຕາຍ, ຜູ້ໃຊ້ບໍ່ຮູ້ສຶກຫຍັງເລີຍ.

ຄຳຖາມຫຼັກ (ຖາມສົງເຫຼົ້າມີຕໍ່ທຸກການອອກແບບ, ຕະຫຼອດໄປ): ຈະເກີດຫຍັງເມື່ອແຕ່ລະສ່ວນປະກອບລົ້ມເຫຼວ — ມີ “ແລ້ວແນວໃດຕໍ່” ຢູ່ສະເໝີບໍ່? ລະບົບຮັບມືໂຫຼດ 10× ແນວໃດ? ພວກເຮົາ ຮູ້ ໄດ້ແນວໃດວ່າມີສິ່ງໜຶ່ງລົ້ມເຫຼວ ກ່ອນລູກຄ້າຈະບອກ? RTO ແລະ RPO ຂອງພວກເຮົາແມ່ນເທົ່າໃດ — ແລະ ໃຜໃນຝ່າຍທຸລະກິດເປັນຄົນເຊັ່ນ ຮັບຮອງມັນ? ພວກເຮົາ *ທົດສອບ* ການກູ້ຄືນຄັ້ງສຸດທ້າຍເມື່ອໃດ?

RTO ແລະ RPO — ສອງຕົວເລກທີ່ຄືບົດສົນທະນາເລື່ອງໄພພິບັດທັງໝົດ. **Recovery Time Objective:** ພວກເຮົາລົ້ມໄດ້ດົນປານໃດ? **Recovery Point Objective:** ພວກເຮົາເສຍຂໍ້ມູນໄດ້ຫຼາຍປານໃດ (ນັ້ນຄື backup ດີອັນສຸດທ້າຍເກົ່າໄດ້ປານໃດ)? RTO 4 ຊົ່ວໂມງ ແລະ RPO 1 ຊົ່ວໂມງ ໝາຍວ່າ: ກັບມາພາຍໃນສີ່ຊົ່ວໂມງ, ໂດຍເສຍຂໍ້ມູນຫຼາຍສຸດໜຶ່ງຊົ່ວໂມງ. ນິຕິການຕັດສິນໃຈ *ທາງທຸລະກິດ* ທີ່ມີປ່າຍລາຄາແບບເລກກຳລັງ — RPO 24 ຊົ່ວໂມງ ຄື backup ກາງຄືນ (ຖືກ); RPO ~ສູນ ຄື replication ຕໍ່ເນື່ອງ (ແພງ); RTO ລະດັບນາທີ ໝາຍເຖິງໂຄງສ້າງພື້ນຖານທີ່ອຸ່ນຢູ່ ຫວ່າງລໍຖ້າເປັນ standby (ແພງຫຼາຍ). ວຽກຂອງສະຖາປະນິກຄືເຮັດໃຫ້ຝ່າຍທຸລະກິດເລືອກຕົວເລກ *ຢ່າງຮູ້ຕົວ* — ແລ້ວອອກແບບໃຫ້ຕົງກັບມັນພໍດີ, ບໍ່ແມ່ນເກີນມັນຢ່າງເພີ່ມ.

Pattern ມາດຕະຖານ: **redundancy** (ມີສອງອັນສໍາລັບທຸກສິ່ງທີ່ສໍາຄັນ — N+1) · **multi-AZ** (ຕົວສໍາເນົາຂ້າມ data center; load balancer ແລະ managed database failover ເຮັດໃຫ້ມັນອັດຕະໂນມັດ) · **auto-scaling** (ຄວາມຈຸຕາມຄວາມຕ້ອງການ) · **health check + self-healing** (instance ທີ່ຕາຍຖືກກວດພົບ ແລະ ປ່ຽນແທນໂດຍເຄື່ອງຈັກ, ບໍ່ແມ່ນຄົນ) · **backup ທີ່ຖືກທົດສອບ** (backup ທີ່ບໍ່ໄດ້ທົດສອບຄືຄວາມຫວັງ, ບໍ່ແມ່ນແຜນ — ຈັດຕາຕະລາງຊ້ອມ restore) · **graceful degradation** (ເມື່ອໂຫຼດເກີນ, ຖິ້ມຄຸນສົມບັດທີ່ສໍາຄັນນ້ອຍທີ່ສຸດກ່ອນ: ThaiTicket ຕັດການສະແດງຕົວຢ່າງແຜນຜັງບ່ອນນັ່ງໄດ້ ແລະ ຮັກສາການຊໍາລະເງິນໄວ້) · **queue ເປັນຕົວດູດແຮງກະແຫກ** (ໄລຍະທີ 3) · **ຫຼີກລ້ຽງ cascading failure** (timeout, retry ພ້ອມ backoff, circuit breaker — ເພື່ອບໍ່ໃຫ້ dependency ຊ້າອັນດຽວ ຈົມທັງຝູງ).

ຕົວຢ່າງທີ່ເຮັດໃຫ້ເບິ່ງ — ການອອກແບບ reliability ຂອງ ThaiTicket. ສອງ AZ ໃນ region ບາງກອກ. ຊັ້ນ app ແບບ stateless ໃນ auto-scaling group ຫຼັງ load balancer, ອຸ່ນເຄື່ອງໄວ້ລ່ວງໜ້າຕາມຕາຕະລາງ ກ່ອນເວລາເປີດຂາຍທີ່ປະກາດໄວ້ (auto-scaling ຕອບສະໜອງເປັນນາທີ; ການເປີດຂາຍ 10:00 ຕ້ອງມີຄວາມຈຸພ້ອມແຕ່ 09:45). Database Aurora, primary ຢູ່ AZ-a, standby ແບບ synchronous ຢູ່ AZ-b, failover ອັດຕະໂນມັດ ≈ ຕໍາກວ່າໜຶ່ງນາທີ. Queue ຢູ່ລະຫວ່າງການຄລິກ “ຊື້” ກັບການປະມວນຜົນຊໍາລະເງິນ ເພື່ອວ່າຄວາມຊ້າຂອງຜູ້ໃຫ້ບໍລິການຊໍາລະເງິນ ຈະເຮັດໃຫ້ຄໍາສັ່ງຊື້ເຂົ້າຄິວ ແທນທີ່ຈະເຮັດໃຫ້ເວັບລົ້ມ. Backup: ຕໍ່ເນື່ອງ, restore ຍ້ອນເວລາໄດ້, ຊ້ອມ restore ທຸກເດືອນ. ຕົວເລກທີ່ຕົກລົງກັບຝ່າຍທຸລະກິດ: RTO 15 ນາທີ, RPO ~ສູນສໍາລັບຄໍາສັ່ງຊື້ (ມັນຄືເງິນ), RPO 24 ຊົ່ວໂມງ ສໍາລັບຂໍ້ມູນວິເຄາະ (ມັນບໍ່ແມ່ນ). ບົດຝຶກໃນປຶ້ມບັນທຶກ: ອັນນີ້ເອົາຫຍັງໄປຈາກແຈ “ຖືກ” ຂອງສາມຫຼ່ຽມ?

ຄໍາສັບ:

ຄໍາສັບ	ຄວາມໝາຍ
RTO / RPO	Recovery Time Objective: ທ່ານລົ້ມໄດ້ດົນປານໃດ. Recovery Point Objective: ທ່ານເສຍຂໍ້ມູນໄດ້ຫຼາຍປານໃດ. ສອງຕົວເລກທີ່ນິຍາມທຸກບົດສົນທະນາ DR — ແລະ ພວກມັນຄືການຕັດສິນໃຈທາງທຸລະກິດ.
High availability (HA)	ການອອກແບບເພື່ອໃຫ້ຄວາມລົ້ມເຫຼວປົກກະຕິ (instance ໜຶ່ງ, AZ ໜຶ່ງ) ບໍ່ເຮັດໃຫ້ຜູ້ໃຊ້ເຫັນການຂາດຕອນ.

ຄໍາສັບ	ຄວາມໝາຍ
Disaster recovery (DR)	ແຜນສໍາລັບຄວາມລົ້ມເຫຼວ ໃຫຍ່ — ທັງ region, ເຫດການ ransomware — ຜ່ອມເປົ້າ RTO/RPO ແລະ runbook ທີ່ຖືກທົດສອບ.
Redundancy / N+1	ຄວາມຈຸສໍາຮອງເພື່ອໃຫ້ສ່ວນປະກອບໃດອັນໜຶ່ງຕາຍໄດ້: ຕ້ອງການ N, ແລ້ວ N+1.
Failover	ການສະຫຼັບອັດຕະໂນມັດໄປຫາ standby ເມື່ອ primary ຕາຍ.
Health check	ຊີບພະຈອນອັດຕະໂນມັດ ທີ່ກວດພົບສ່ວນປະກອບທີ່ຕາຍ ເພື່ອໃຫ້ເຄື່ອງຈັກອ້ອມຜ່ານມັນໄດ້.
Graceful degradation	ການຖິ້ມຄຸນສົມບັດທີ່ສໍາຄັນນ້ອຍກວ່າ ພາຍໃຕ້ຄວາມກົດດັນ ເພື່ອໃຫ້ແກນຫຼັກຢູ່ລອດ.
Timeout / retry ຜ່ອມ backoff	ການຍອມແຜ່ຕໍ່ການເອີ້ນທີ່ຊ້າ ຫຼັງເກີນຂີດຈໍາກັດ / ການລອງໃໝ່ຜ່ອມການພັກຫຼືຍາວຂຶ້ນ — ມາລະຍາດທີ່ປ້ອງກັນຄວາມລົ້ມເຫຼວແບບລູກໂສ້.
Circuit breaker	ສ່ວນປະກອບທີ່ຢຸດເອີ້ນ dependency ທີ່ກໍາລັງລົ້ມເຫຼວໄລຍະໜຶ່ງ ເພື່ອໃຫ້ມັນຟື້ນຕົວ — ຄືຟິວໃນເຮືອນ.
Chaos engineering	ການສິດຄວາມລົ້ມເຫຼວເຂົ້າໄປໂດຍເຈດຕະນາ (ແບບ Netflix) ເພື່ອພິສູດຄໍາອ້າງເລື່ອງຄວາມທົນທານ ກ່ອນທີ່ຄວາມເປັນຈິງຈະທົດສອບແທນທ່ານ.

ບົດຝຶກຫັດ: (1) ເອົາການອອກແບບເວັບສັງຄາຫານທ່ຽງຈາກໄລຍະທີ່ 1 ມາຍົກລະດັບໃຫ້ຢູ່ລອດ: instance ຕາຍ, AZ ຫາຍ, database ລົ້ມເຫຼວ, ແລະ ຊ່ວງອາຫານທ່ຽງຜຸ່ງ 10× — ແຕ່ມຳ່ອນ ແລະ ຫຼັງ. (2) ຂຽນບົດສົນທະນາ RTO/RPO ສໍາລັບສາມທຸລະກິດ (ບູຮອກ, ເວັບອີຄອມເມີຊ, ລະບົບເວດຊະລະບຽນໂຮງໝໍ) ເປັນບົດສົນທະນາສັ້ນລະຫວ່າງສະຖາປະນິກກັບເຈົ້າຂອງ — ຝຶກເຮັດໃຫ້ຕົນທົນເບິ່ງເຫັນໄດ້ໃນບົດສົນທະນາ. (3) ປຶ້ມບັນທຶກ: ລາຍຊື່ SPOF ທຸກອັນໃນການອອກແບບ ThaiTicket ຂ້າງເທິງ. ມີຢ່າງໜ້ອຍໜຶ່ງອັນທີ່ຖືກປະໄວ້ໂດຍເຈດຕະນາ. (ໃບ: ຈັກ region?)

Milestone: ທ່ານດໍາເນີນການສອບຖາມ “ຈະເກີດຫຍັງເມື່ອອັນນີ້ລົ້ມເຫຼວ?” ເທິງແຜນວາດໃດກໍ່ໄດ້ ຕິດຕໍ່ກັນສືບນາທີ່ໂດຍບໍ່ໝົດຄໍາຖາມ, ແລະ ທ່ານອະທິບາຍ RTO ທຽບ RPO ໃຫ້ເຈົ້າຂອງທີ່ບໍ່ແມ່ນສາຍເຕັກນິກ ດ້ວຍການປຽບທຽບຮ້ານຄ້າຖືກນ້ຳຖ້ວມ ພາຍໃນເກົ້າສິບວິນາທີ.

Module 5 (ອາທິດທີ 9–10): Security — ອອກແບບລະບົບທີ່ທຳຮ້າຍໄດ້ຍາກ

ກອບຄິດ: shared responsibility model (ແບບຈຳລອງຄວາມຮັບຜິດຊອບຮ່ວມ). ຜູ້ໃຫ້ບໍລິການຮັກສາຄວາມປອດໄພຂອງ *ຕົວ cloud ເອງ* — ອາຄານ, ຮາດແວ, hypervisor. ທ່ານຮັກສາຄວາມປອດໄພຂອງທຸກສິ່ງທີ່ທ່ານ *ເອົາເຂົ້າໄປໃນມັນ* — ຂໍ້ມູນ, ຕົວຕົນ, ການຕັ້ງຄ່າ, ໂຄດ. ເກືອບທຸກກໍລະນີການລະເມີດ cloud ທີ່ດັງ ຄືການຕັ້ງຄ່າຜິດຢູ່ຝັ່ງລູກຄ້າ (bucket ສາທາລະນະ, key ຮົວ, role ທີ່ໃຫ້ສິດເກີນ), ຊຶ່ງເປັນເຫດຜົນວ່າຄວາມປອດໄພຄືບັນຫາ *ສະຖາປັດຕະຍະກຳ* ກ່ອນທີ່ຈະເປັນບັນຫາເຄື່ອງມື: ປະກາດຮັບສະໝັກວຽກເວົ້າວ່າ “integrate security into the design rather than bolting it on,” ແລະ module ນີ້ຄືວິທີເຮັດ.

ຄຳຖາມຫຼັກ: ໃຜ ແລະ ຫຍັງ ເຂົ້າເຖິງແຕ່ລະສ່ວນປະກອບໄດ້, ແລະ ທຸກສິດອະນຸຍາດເປັນ *ຂັ້ນຕ່ຳ* ທີ່ຈຳເປັນບໍ່? ຂໍ້ມູນຖືກເຂົ້າລະຫັດຢູ່ໃສ — ຕອນເກັບ, ຕອນສົ່ງ, ແລະ ໃຜຖືກະແຈ? ຂອບເຂດເຄືອຂ່າຍຢູ່ໃສ, ແລະ ຫຍັງຂ້າມມັນ? ພວກເຮົາຈະ *ກວດພົບ* ການລະເມີດແນວໃດ — ແລະ ພວກເຮົາເລືອກເລື່ອງຄືນຈາກ log ໄດ້ບໍ່ຫຼັງເກີດເຫດ? ກົດໝາຍໃດໃຊ້ກັບຂໍ້ມູນນີ້, ແລະ ຂໍ້ມູນຢູ່ບ່ອນໃດທາງກາຍະພາບ?

Pattern ມາດຕະຖານ: least privilege (ສິດຂັ້ນຕ່ຳ — ທຸກຄົນ ແລະ ທຸກໂປຣແກຣມ ໄດ້ສິດເຂົ້າເຖິງໜ້ອຍທີ່ສຸດເທົ່າທີ່ວຽກຕ້ອງການ — ກົດທອງທີ່ໃຊ້ຕັດສິນທຸກ IAM policy) · MFA ທຸກບ່ອນ, root ລັອກເກັບໄວ້ · ເຂົ້າລະຫັດຕອນເກັບ ແລະ ຕອນສົ່ງ, ເປີດຢູ່ສະເໝີ (ມັນຄືກ່ອງໝາຍຖືກໃນ cloud; ບໍ່ມີຂໍ້ອ້າງ) · network segmentation (ການແບ່ງສ່ວນເຄືອຂ່າຍ — public/private subnet, security group ເປັນ firewall ຕໍ່ server; blast radius ຂອງການລະເມີດ ຖືກນິຍາມໂດຍກຳແພງທີ່ທ່ານແຕ້ມໄວ້ລ່ວງໜ້າ) · secret ຢູ່ໃນ vault, ບໍ່ເຄີຍຢູ່ໃນໂຄດ · zero trust (ຢືນຢັນທຸກຄຳຮ້ອງຂໍຢ່າງຊັດເຈນ — ຕົວຕົນ, ອຸປະກອນ, ບໍລິບົດ — ບໍ່ໄວ້ໃຈຫຍັງພຽງເພາະມັນ “ຢູ່ໃນເຄືອຂ່າຍ”; ປະກາດວຽກລະບຸຊື່ມັນ, ທ່ານກໍຕ້ອງລະບຸ) · defense in depth (ປ້ອງກັນເປັນຊັ້ນ, ເພື່ອວ່າການຄວບຄຸມທີລື້ມເຫຼວອັນດຽວ ບໍ່ແມ່ນຈົບເກມ) · audit logging (ບັນທຶກແບບ CloudTrail ວ່າໃຜເຮັດຫຍັງ — ແກ້ໄຂບໍ່ໄດ້, ຖືກເຝົ້າຕິດຕາມ) · ຄວາມປອດໄພເປັນ guardrail, ບໍ່ແມ່ນປະຕູກັ້ນ (ເຂົ້າລະຫັດກົດເປັນ policy ອັດຕະໂນມັດ ທີ່ເຮັດໃຫ້ເສັ້ນທາງປອດໄພເປັນເສັ້ນທາງງ່າຍທີ່ສຸດ, ແທນທີ່ຈະເປັນກອງປະຊຸມທົບທວນ ທີ່ເຮັດໃຫ້ຄວາມປອດໄພກາຍເປັນສັດຕູຂອງການສົ່ງມອບ).

ຕົວຢ່າງທີ່ເຮັດໃຫ້ເບິ່ງ — ການອອກແບບ security ຂອງ ThaiTicket. ຂໍ້ມູນລູກຄ້າ (ຊື່, ອີເມວ, ເລກອ້າງອີງການຂໍາລະ) ຖືກຈັດປະເພດເປັນຂໍ້ມູນສ່ວນບຸກຄົນຕາມ PDPA → ເກັບແບບເຂົ້າລະຫັດໃນ Aurora ໃນ region ບາງກອກ (data residency), ກະແຈຢູ່ໃນ KMS. ເຄືອຂ່າຍ: ມີແຕ່ load balancer ທີ່ເປັນສາທາລະນະ; ຊັ້ນ app ເປັນ private; subnet ຂອງ database ຮັບການເຊື່ອມຕໍ່ *ສະເພາະ* ຈາກ security group ຂອງຊັ້ນ app. ຄົນ: SSO + MFA; ວິສະວະກອນໄດ້ສິດເຂົ້າເຖິງ production ແບບອ່ານຢ່າງດຽວເປັນຄຳມາດຕະຖານ ແລະ ໄດ້ສິດຍົກລະດັບແບບຈຳກັດເວລາເມື່ອຮ້ອງຂໍ (least privilege ພ້ອມຮ່ອງຮອຍກວດສອບ). ການຈັດການບັດຂໍາລະຖືກມອບໃຫ້ຜູ້ໃຫ້ບໍລິການຂໍາລະເງິນທີ່ໄດ້ຮັບການຢັ້ງຢືນ ເພື່ອວ່າເລກບັດດິບຈະບໍ່ແຕະລະບົບຂອງພວກເຮົາຈັກເທືອ — ການຕັດສິນໃຈ *ອອກແບບ* ທີ່ຕັດພາລະ compliance ທັງກ້ອນອອກໄປ, ຊຶ່ງນັ້ນຄືສະຖາປັດຕະຍະກຳຄວາມປອດໄພໃນຮູບແບບທີ່ດີທີ່ສຸດ. ເປີດ CloudTrail, ຕັ້ງແຈ້ງເຕືອນເມື່ອມີການເຂົ້າເຖິງຜິດປົກກະຕິ. ຄຳຖາມເລື່ອງການລະເມີດຖືກຕອບໄວ້ລ່ວງໜ້າ: PDPA ມີໜ້າທີ່ແຈ້ງພາຍໃນ 72 ຊົ່ວໂມງ — log ແລະ runbook ຕ້ອງເຮັດໃຫ້ພວກເຮົາເລົ່າເລື່ອງໄດ້ໃນເວລາໜ້ອຍກວ່ານັ້ນ.

ຄໍາສັບ:

ຄໍາສັບ	ຄວາມໝາຍ
Shared responsibility model	ຜູ້ໃຫ້ບໍລິການປົກປ້ອງ cloud; ທ່ານປົກປ້ອງສິ່ງທີ່ຢູ່ໃນມັນ. ການລະເມີດສ່ວນໃຫຍ່ຢູ່ຝັ່ງລູກຄ້າຂອງເສັ້ນ.
IAM / role / policy	Identity and Access Management — ໃຜເຮັດຫຍັງໄດ້. Policy ຄືລາຍການສິດອະນຸຍາດ; role ຄືມັດຂອງພວກມັນທີ່ສວມໃສ່ໄດ້.
Least privilege	ກົດທອງ: ສິດເຂົ້າເຖິງຂັ້ນຕໍ່າທີ່ຈໍາເປັນ, ບໍ່ເກີນມັນ, ທົບທວນເປັນປະຈຳ.
MFA	ຫຼັກຖານຢືນຢັນຕົວຕົນອັນທີສອງ ນອກເໜືອລະຫັດຜ່ານ. ຕໍ່ລອງບໍ່ໄດ້ສໍາລັບຄົນ.
ການເຂົ້າລະຫັດຕອນເກັບ / ຕອນສົ່ງ / KMS	ຂໍ້ມູນຖືກກວນເທິງດິສ / ເທິງສາຍ / ບໍລິການ managed ທີ່ຖືກະແຈ.
Security group	ລາຍການກົດ firewall ຕໍ່ server — “traffic ເວັບເຂົ້າໄດ້ຈາກ load balancer ເທົ່ານັ້ນ.”
Network segmentation / blast radius	ການແບ່ງເຄືອຂ່າຍເປັນເຂດມີກໍາແພງ / ຜູ້ໃຈມຕິເສີມໄປໄດ້ໄກປານໃດຫຼັງລະເມີດຄັ້ງໜຶ່ງ. ກໍາແພງທີ່ແຕ້ມໄວ້ລ່ວງໜ້າເປັນຜູ້ກໍານົດ.
Zero trust	ຢືນຢັນທຸກຄໍາຮ້ອງຂໍຢ່າງຊັດເຈນ; ບໍ່ໄວ້ໃຈຫຍັງເພາະມັນຢູ່ “ຂ້າງໃນ.” ທໍາມາດຕະຖານຂອງຍຸກນີ້.
Defense in depth	ການຄວບຄຸມຫຼາຍຊັ້ນທັບກັນ ເພື່ອວ່າຄວາມລົ້ມເຫຼວອັນດຽວບໍ່ເຖິງຕາຍ.
Secrets management	ລະຫັດຜ່ານ, ກະແຈ, ແລະ token ຢູ່ໃນບໍລິການ vault — ບໍ່ເຄີຍຢູ່ໃນໂຄດ, ບໍ່ເຄີຍຢູ່ໃນສະເປຣດຊິດ.
Audit log / CloudTrail	ບັນທຶກທີ່ແກ້ໄຂບໍ່ໄດ້ວ່າໃຜເຮັດຫຍັງ, ເມື່ອໃດ — ວິທີທີ່ທ່ານກວດພົບບັນຫາ ແລະ ປະກອບເລື່ອງຄືນພາຍຫຼັງ.
Data classification	ການຕິດປ້າຍຂໍ້ມູນຕາມຄວາມອ່ອນໄຫວ (ສາທາລະນະ / ພາຍໃນ / ສ່ວນບຸກຄົນ / ຖືກກໍາກັບ) ເພື່ອໃຫ້ການຄວບຄຸມກົງກັບປ້າຍ, ບໍ່ແມ່ນກົງກັບການເດົາ.

ຄຳສັບ

ຄວາມໝາຍ

Data residency

ການເກັບຂໍ້ມູນໄວ້ພາຍໃນຊາຍແດນຂອງປະເທດໜຶ່ງ — ຂໍ້ກຳນົດທາງກົດໝາຍໃນຫຼາຍອຸດສາຫະກຳ, ແລະ ເປັນຂໍ້ມູນນຳເຂົ້າຂອງສະຖາປັດຕະຍະກຳສະເໝີ.

PDPA / GDPR

ກົດໝາຍຂໍ້ມູນສ່ວນບຸກຄົນຂອງໄທ ແລະ ເອີຣົບ: ການຍິນຍອມ, ການແຈ້ງການລະເມີດ (72 ຊົ່ວໂມງ), ກົດການໂອນຂ້າມຊາຍແດນ. ໄລຍະທີ່ 4 ລົງເລິກກວ່ານີ້.

ວິດີໂອສຳລັບ module ນີ້ (ລົງກວດສອບແລ້ວ): The AWS Shared Responsibility Model — Digital Cloud Training — ~4 ນາທີ — <https://www.youtube.com/watch?v=ESPBBEK-cvo>

ບົດຝຶກຫັດ: (1) ແຕ້ມແຜນວາດ ThaiTicket ແລ້ວວາງຊັ້ນຄວາມປອດໄພທັບ: ໝາຍທຸກຂອບເຂດຄວາມໄວ້ວາງໃຈ, ທຸກຈຸດເຂົ້າລະຫັດ, ທຸກບ່ອນທີ່ຂໍ້ມູນປະຈຳຕົວອາໄສຢູ່. ນີ້ໄວ້ວາງຊັ້ນທັບນີ້ — ແຜນວາດອັນເກົ່າ, ເລນສີຄວາມປອດໄພ — ຄືສິ່ງທີ່ການທົບທວນຮຽກຮ້ອງຈາກທ່ານພໍດີ. (2) ອ່ານບົດ post-mortem ສາທາລະນະໜຶ່ງບົດກ່ຽວກັບການລະເມີດຈາກການຕັ້ງຄ່າ cloud ຜິດ (Capital One 2019 ຄືກໍລະນີສອນຄລາສສິກ) ແລ້ວຂຽນໃນປຶ້ມບັນທຶກວ່າ *pattern* ອັນໃດຂ້າງເທິງ ຈະປ້ອງກັນມັນໄດ້. (3) ສະແດງບົດບາດກັບ AI: ມັນເປັນຜູ້ກໍ່ຕັ້ງ startup ທີ່ເວົ້າວ່າ “ເຮົາຄ່ອຍເພີ່ມຄວາມປອດໄພທີ່ສູງ” — ຊັກຊວນເຂົ້າອອກຈາກຄວາມຄິດນັ້ນດ້ວຍເລກຄະນິດຂອງຕົ້ນທຶນການລະເມີດ, ຢ່າງອ່ອນໂຍນ.

Milestone: ທ່ານເອົາແຜນວາດໃດກໍໄດ້ຈາກໄລຍະທີ່ 1 ມາຜະລິດຊັ້ນຄວາມປອດໄພທັບໄດ້ພາຍໃນຊາວນາທີ, ແລະ ອະທິບາຍ least privilege, zero trust, ແລະ shared responsibility model ໃຫ້ຜູ້ມີສ່ວນໄດ້ເສຍທົບແມ່ນສາຍເຕັກນິກໄດ້ ໂດຍບໍ່ໃຊ້ສັບແສງ.

Module 6 (ອາທິດທີ 11–12): Performance ແລະ cost — ຄວາມໄວ ແລະ ເງິນ, ສອງເສົາທີ່ຄູ່ກັນ

ສອງເສົາທີ່ຖືກສອນຄູ່ກັນ ເພາະພວກມັນຄືຄັນໂຍກອັນດຽວກັນທີ່ຖືກດັນໄປຄົນລະທາງ, ແລະ ສະຖາປະນິກຄືຄັນທີ່ມີຢູ່ເທິງຄັນໂຍກນັ້ນ.

Performance efficiency — ຄຳຖາມຫຼັກ: ຜູ້ໃຊ້ຈະຮູ້ສຶກເຖິງຄວາມຊ້າຢູ່ໃສກ່ອນ? ຫຍັງຖືກຄິດໄລ່ຊ້າໆ ທັງທີ່ຄິດເທືອດຽວແລ້ວ cache ໄວ້ໄດ້? ແຕ່ລະສ່ວນປະກອບເປັນ *ເຄື່ອງມືທີ່ຖືກຕ້ອງ* ບໍ່ (SQL ທີ່ເຮັດວຽກຂອງ search engine ຄື bug ປະສິດທິພາບຂອງສະຖາປັດຕະຍະກຳ, ບໍ່ແມ່ນຂອງໂຄດ)? ປະສິດທິພາບປ່ຽນແນວໃດທີ່ 10× — ແລະ ຄໍາຂວດອັນທຳອິດຢູ່ໃສ?

Pattern ດ້ານ performance: ຊັ້ນ caching — pattern ປະສິດທິພາບທີ່ໃຫ້ຜົນຮຸ້ນທີ່ສຸດອັນດຽວ: browser cache → **CDN** ທີ່ຂອບ (ເນື້ອຫາ static ຖືກສົ່ງຈາກເມືອງໃກ້ຜູ້ໃຊ້; ຄູດຊັບ traffic ອ່ານສ່ວນໃຫຍ່ໄດ້) → application cache (**Redis** ວາງໜ້າ database ສຳລັບການອ່ານ hot: ແຜນຜັງບ່ອນນັ້ງ, session, ໜ້າສື່ນຄ້າ)

→ database query cache. ແຕ່ລະຊັ້ນຕອບຄໍາຮ້ອງຂໍກ່ອນມັນຈະໄປເຖິງແກນທີ່ແພງ; ຄໍາຖາມການອອກແບບຄືສະເໝີວ່າ ຫຍັງ cache ໄດ້, ດົນປານໃດ, ແລະ ຈະ invalidate ແນວໃດເມື່ອຄວາມຈິງປ່ຽນ. ຈາກນັ້ນ: read replica (ສໍາເນົາ database ທີ່ບໍລິການການອ່ານ ເພື່ອໃຫ້ primary ຍັງຂຽນຕໍ່ໄດ້) · asynchronous processing (ຢ່າໃຫ້ຜູ້ໃຊ້ລໍຖ້າທີ່ເຮັດພາຍຫຼັງໄດ້ — ໃບຮັບທາງອີເມວ, ຮູບຫຍໍ້, ລາຍງານ) · ເຄື່ອງມືທີ່ຖືກຕ້ອງຕໍ່ວຽກ (ຄົ້ນຫາ → search engine; ວິເຄາະ → warehouse; ການເບິ່ງຄ່າ hot → key-value store) · ແລະ ວັດແທກກ່ອນ (ວຽກປະສິດທິພາບໂດຍບໍ່ວັດແທກ ຄືຄວາມເຊື່ອງມຸງງາຍ).

Cost optimization — ຄໍາຖາມຫຼັກ: ການອອກແບບນີ້ມີຄ່າໃຊ້ຈ່າຍເດືອນລະເທົ່າໃດທີ່ໂຫຼດປັດຈຸບັນ — ແລະ ຕໍ່ຫົວໜ່ວຍ (ຕໍ່ຄໍາສັ່ງຊື້, ຕໍ່ລູກຄ້າ)? ຫຍັງແລ່ນຢູ່ຕອນຕີສາມທັງທີ່ບໍ່ຈໍາເປັນ? ການໃຊ້ງານຄົງທີ່ອັນໃດຜູກມັດເພື່ອຮັບສ່ວນຫຼຸດໄດ້? ຝ່າຍການເງິນຈະບອກວ່າແນວໃນ້ມເປັນແນວໃດ?

Pattern ດ້ານຕົ້ນທຶນ: rightsizing (ຝຸງເຄື່ອງສ່ວນໃຫຍ່ຖືກຈັດຊັບພະຍາກອນເກີນຢ່າງງຽບໆ; ຫຍໍ້ລົງໃຫ້ພໍດີກັບຄວາມຕ້ອງການທີ່ວັດແທກໄດ້ ຄືເງິນຟຣີ) · **ຍຸດທະສາດການຈອງ** — ເມນູລາຄາ: on-demand (ລາຄາເຕັມ, ອິດສະຫຼະເຕັມ) ສໍາລັບໂຫຼດພຸ່ງ/ຄາດເດົາບໍ່ໄດ້, reserved instance / savings plan (ຜູກມັດ 1–3 ປີ, ຫຼຸດ 30–70%) ສໍາລັບຖານໂຫຼດຄົງທີ່, spot (ຫຼຸດເຖິງ 90%, ຖືກຮຽກຄືນໄດ້ໃນເວລາອັນສັ້ນ) ສໍາລັບວຽກ batch ທີ່ຂັດຈັງຫວະໄດ້; ທ່າຂອງສະຖາປະນິກຄືວາງເປັນຊັ້ນ: ຈອງພື້ນ, auto-scale ຊັ້ນກາງແບບ on-demand, ໃຊ້ spot ກັບ batch · **storage lifecycle** (ການແບ່ງຂັ້ນຂອງ Module 2, ແບບອັດຕະໂນມັດ) · **ປິດສິ່ງທີ່ບໍ່ໃຊ້** (ສະພາບແວດລ້ອມທີ່ບໍ່ແມ່ນ production ຫຼັບກາງຄືນ ແລະ ວັນພັກ ຫຼຸດຄ່າໃຊ້ຈ່າຍໄດ້ສອງສ່ວນສາມ) · **ເຝົ້າ egress** (ຂໍ້ມູນທີ່ ອອກ ຈາກ cloud ຖືກຄິດເງິນ; ການອອກແບບຂ້າມ region ທີ່ຄຸຍກັນຫຼາຍ ແລະ ການດາວໂຫຼດສາທາລະນະຂະໜາດໃຫຍ່ ເຮັດໃຫ້ທຸກຄົນຕົກໃຈຢ່າງໜ້ອຍເທື່ອໜຶ່ງ) · **tagging ແລະ showback** (ທຸກຊັບພະຍາກອນຖືກຕິດປ້າຍເຈົ້າຂອງ/ໂຄງການ ເພື່ອໃຫ້ທຸກບາດລະບຸທີ່ມາໄດ້) · ແລະ ຕົວຊີ້ວັດມົງກຸດ, **unit economics** (ເສດຖະສາດຕໍ່ຫົວໜ່ວຍ): ບໍ່ແມ່ນ “ໃບປິ່ນ \$800k/ເດືອນ” ແຕ່ “ຕົ້ນທຶນຕໍ່ປີທີ່ຂາຍໄດ້ \$1.90 ແລະ ກໍາລັງຫຼຸດລົງ.” ໃບປິ່ນທີ່ໃຫຍ່ຂຶ້ນນັ້ນບໍ່ເປັນຫຍັງ; *ຕົ້ນທຶນຕໍ່ຫົວໜ່ວຍ* ທີ່ໃຫຍ່ຂຶ້ນຕ່າງຫາກຄືກິນຜິດປົກກະຕິຂອງສະຖາປັດຕະຍະກຳ. ວິໄນນີ້ມີຊື່ — **FinOps** — ແລະ ສະຖາປະນິກນັ້ນຢູ່ໃຈກາງຂອງມັນ.

ຕົວຢ່າງທີ່ເຮັດໃຫ້ເບິ່ງ — ThaiTicket, ທັງສອງເລນສ໌. Performance: CloudFront ບໍລິການໜ້າສິນລະປິ່ນ ແລະ ຮູບແຜນຜັງບ່ອນນັ່ງ (ຄົ້ນເບິ່ງ 400,000 ຄົນສ່ວນໃຫຍ່ບໍ່ເຄີຍແຕະ server ເລີຍ); Redis cache ຄວາມພ້ອມຂອງບ່ອນນັ່ງດ້ວຍ TTL 2 ວິນາທີ — ເກົາພໍທີ່ຈະຖືກ, ສົດພໍທີ່ການຊໍາລະເງິນ (ຊຶ່ງກວດຊ້າກັບ Aurora, ແຫຼ່ງຄວາມຈິງ) ຈະປ້ອງກັນການຂາຍຊ້າ; ໃບຮັບ ແລະ ປິຖືກສ້າງແບບ asynchronous ຫຼັງການຊໍາລະ. ຕົ້ນທຶນ: ຖານໂຫຼດຄົງທີ່ 2,000 ຄົນ/ຊົ່ວໂມງ ແລ່ນເທິງ savings plan (~ຫຼຸດ 40%); ຊ່ວງເປີດຂາຍພຸ່ງ ແລ່ນແບບ on-demand ສະເພາະຊົ່ວໂມງທີ່ມັນມີ; ວຽກວິເຄາະແລ່ນເທິງ spot ຕອນກາງຄືນ; staging ຫຼັບນອກເວລາລັດຖະການ; ທຸກຊັບພະຍາກອນຕິດປ້າຍ project:thaiticket. ຂໍສະເໜີເຖິງ CFO ອ່ານວ່າ: “\$62,000/ເດືອນ ເປັນຖານ, ~\$9,000 ຕໍ່ເຫດການເປີດຂາຍໃຫຍ່ໜຶ່ງຄັ້ງ, ຕົ້ນທຶນຕໍ່ປີ ~\$1.90 ຫຼຸດລົງຕາມປະລິມານ” — ແລະ *ປະໂຫຍກນັ້ນເອງ* ຄືສຽງຂອງສະຖາປະນິກທີ່ຄ່ອງເລື່ອງຕົ້ນທຶນ.

ຄໍາສັບ:

ຄໍາສັບ	ຄວາມໝາຍ
Latency / throughput	ຄໍາຮ້ອງຂໍໜຶ່ງໃຊ້ເວລາດົນປານໃດ / ລະບົບຮັບໄດ້ຈັກຄໍາຮ້ອງຂໍຕໍ່ວິນາທີ. ກ່ຽວຂ້ອງກັນ, ແຕ່ບໍ່ຄືກັນ.
Bottleneck	ຈຸດແຄບທີ່ສຸດ ທີ່ກຳນົດຈັງຫວະຂອງທັງລະບົບ. ການປັບປຸງບ່ອນອື່ນຄືການປະດັບປະດາ.
Cache / TTL / invalidation	ສຳເນົາໄວຂອງຂໍ້ມູນທີ່ດຶງມາແພງ / ມັນເຊື່ອຖືໄດ້ດົນປານໃດ / ບັນຫາຍາກຂອງການຟື້ນຟູມັນເມື່ອຄວາມຈິງປ່ຽນ.
CDN	Cache ຂັ້ນນອກສຸດ — ເນື້ອຫາຂອງທ່ານໃນຫຼາຍຮ້ອຍເມືອງ. ຄໍາຕອບທຳອິດຂອງ “ເຮັດໃຫ້ໄວທົ່ວໂລກ.”
Read replica	ສຳເນົາ database ທີ່ບໍ່ລິການການອ່ານ, ເພື່ອໃຫ້ primary ເກັບແຮງໄວ້ຂຽນ.
Asynchronous processing	ການເຮັດວຽກທີ່ບໍ່ຮີບດ່ວນຫຼັງຕອບຜູ້ໃຊ້ແລ້ວ, ປົກກະຕິຜ່ານ queue.
On-demand / reserved / savings plan / spot	ລາຄາເຕັມແບບຍືດຍຸ່ນ / ຜູກມັດ 1–3 ປີ ຫຼຸດ 30–70% / ແນວຄິດດຽວກັນ, ຍືດຍຸ່ນກວ່າ / ຫຼຸດເຖິງ 90% ແຕ່ຖືກຮຽກຄືນໄດ້. ສະຖາປະນິກວາງທັງສີ່ເປັນຊັ້ນ.
Rightsizing	ການຫຍໍ້ຊັບພະຍາກອນທີ່ຈັດເກີນ ລົງໃຫ້ພໍດີກັບຄວາມຕ້ອງການທີ່ຈິດແທກໄດ້. ເງິນຟຣີໃນເກືອບທຸກບັນຊີ.
Egress	ຂໍ້ມູນທີ່ອອກຈາກ cloud — ຄິດເງິນຕໍ່ GB. ຄວາມຕຶກໃຈເລືອງໃບບິນທີ່ດັງທີ່ສຸດ; ອອກແບບການໄຫຼຂອງຂໍ້ມູນໂດຍນຶກເຖິງມັນ.
Tagging / showback	ການຕິດປ້າຍທຸກຊັບພະຍາກອນດ້ວຍເຈົ້າຂອງ ແລະ ໂຄງການ / ການສະແດງໃບບິນຂອງແຕ່ລະທີມໃຫ້ທີມນັ້ນເບິ່ງ. ຄວາມຮັບຜິດຊອບປ່ຽນພຶດຕິກຳ.
Unit economics	ຕົ້ນທຶນຕໍ່ຫົວໜ່ວຍທຸລະກິດ (ຕໍ່ຄໍາສັ່ງຊື້, ຕໍ່ຜູ້ໃຊ້). ຕົວຊີ້ວັດທີ່ເຮັດໃຫ້ໃບບິນມີຄວາມໝາຍ — ແລະ ປະໂຫຍກທີ່ສະຖາປະນິກມັກທີ່ສຸດຕໍ່ໜ້າ CFO.
TCO	Total cost of ownership — ລາຄາປ້າຍ ບວກຄືນ, ໃບອະນຸຍາດ, ການ migrate, ແລະ ຕົ້ນທຶນການອອກຕະຫຼອດອາຍຸຂອງທາງເລືອກໜຶ່ງ.

FinOps

ວິໄນຂອງການເຮັດໃຫ້ການໃຊ້ຈ່າຍ cloud ເບິ່ງເຫັນໄດ້, ຈັດສັນໄດ້, ແລະ ຖືກປັບປຸງຢ່າງຕໍ່ເນື່ອງ.

ວິດີໂອສຳລັບ module ນີ້ (ລິ້ງກວດສອບແລ້ວ): What is FinOps? — FinOps Foundation (ທາງການ) — ~2 ນາທີ — https://www.youtube.com/watch?v=Y-c_xw9bHFw

ບົດຝຶກຫັດ: (1) ຄິດລາຄາຖານຂອງ ThaiTicket ໃນ AWS pricing calculator (ຄົ້ນຫາ “AWS pricing calculator” — ບົດຝຶກແຈກແຈງລາຍການຂອງການອອກແບບຈິງເທື່ອດຽວ ຈະຄາຍຄວາມລຶກລັບຂອງທຸກບົດສິນທະນາເລື່ອງຕົ້ນທຶນໃນອະນາຄົດ); ທຽບຍອດລວມຂອງທ່ານກັບ ฿62,000 ຂ້າງເທິງ ແລ້ວອະທິບາຍຊ່ອງວ່າງ. (2) ເພີ່ມຂັ້ນ caching ທັບໃສ່ສອງແຜນວາດຈາກໄລຍະທີ 1: ໝາຍທຸກ cache, TTL ຂອງມັນ, ແລະ ເລື່ອງລາວການ invalidate ຂອງມັນ. (3) AI ສະແດງເປັນວິສະວະກອນທີ່ຢາກໃຫ້ທຸກຢ່າງເປັນ on-demand “ເພື່ອໃຫ້ງ່າຍ” — ເຈລະຈາຍຸດທະສາດການຈອງ, ຜ່ອມຕົວເລກ.

Milestone: ສຳລັບການອອກແບບໃດກໍ່ໄດ້ທີ່ທ່ານແຕ້ມ, ທ່ານບອກລາຍການຕົ້ນທຶນສາມອັນໃຫຍ່ທີ່ສຸດ ແລະ ຄໍາຂວດອັນທຳອິດທີ່ເປັນໄປໄດ້ຂອງມັນ — ແລະ ສະເໜີການປ່ຽນແປງໜຶ່ງຢ່າງທີ່ປັບປຸງທັງສອງຜ່ອມກັນ (ເກືອບສະເໝີໝຶ່ງ; ປົກກະຕິແມ່ນ caching).

Module 7 (ອາທິດທີ 13–14): Operational excellence, sustainability — ແລະ ການແລ່ນທັງຫົກເສົາຜ່ອມກັນ

Operational excellence ຄືເສົາທີ່ຖາມວ່າ: *ຄົນແລ່ນສິ່ງນີ້ໄດ້ຈິງບໍ່, ຢ່າງສະຫງົບ?* ຄຳຖາມຫຼັກ: ການປ່ຽນແປງໄປເຖິງ production ແນວໃດ — ຜ່ານ pipeline ອັດຕະໂນມັດທີ່ຖືກທົດສອບ, ຫຼື ຜ່ານວິລະບູລຸດທີ່ເປັນຄົນ? ພວກເຮົາຮູ້ໄດ້ແນວໃດວ່າລະບົບແຂງແຮງ *ຢູ່ຕອນນີ້*? ເມື່ອມັນພັງຕອນຕີສາມ, ວິສະວະກອນເວນ on-call ເຮັດຫຍັງແທ້? Pattern: **infrastructure as code** (ສະພາບແວດລ້ອມຖືກນິຍາມເປັນຂໍ້ຄວາມທີ່ຖືກທົດທວນ ແລະ ຄວບຄຸມເວີຊັນ — Terraform/CloudFormation — ເພື່ອໃຫ້ມັນສ້າງຄືນໄດ້ ແລະ ກວດສອບໄດ້; ສະຖາປັດຕະຍະກຳທີ່ມີຢູ່ພຽງໃນຮູບການຄລິກ console ຄືນທາງພື້ນບ້ານ, ບໍ່ແມ່ນວິສະວະກຳ) · **CI/CD pipeline** ຜ່ອມຍຸດທະສາດ deploy ທີ່ປອດໄພ (**blue-green**: ຕັ້ງເວີຊັນໃໝ່ຂຶ້ນຂ້າງເວີຊັນເກົ່າແລ້ວສະຫຼັບ; **canary**: ໃຫ້ເວີຊັນໃໝ່ຮັບ traffic 5% ແລ້ວເຝົ້າເບິ່ງ) · **observability** (log, metric, trace, ແລະ ການແຈ້ງເຕືອນທີ່ຜູກກັບອາການທີ່ *ຜູ້ໃຊ້ເຫັນ*, ບໍ່ແມ່ນເລື່ອງຫຍຸ້ງຫຍຸ້ງຂອງເຄື່ອງ) · **runbook** (ຄູ່ມືລາຍລັກອັກສອນວ່າຕ້ອງເຮັດຫຍັງ ສຳລັບແຕ່ລະຄວາມລົ້ມເຫຼວທີ່ຮູ້ຈັກ) · **blameless post-mortem** (ຫຼັງເຫດການ: ຫຍັງລົ້ມເຫຼວ, ເປັນຫຍັງ, ຫຍັງປ້ອງກັນການເກີດຊ້າ — ບໍ່ມີຜູ້ຮ້າຍ, ບໍ່ດັງນັ້ນຄົນຈະເຊົາເວົ້າຄວາມຈິງ). ສ່ວນຂອງສະຖາປະນິກ: *ອອກແບບ* ເພື່ອຄວາມສາມາດໃນການປະຕິບັດງານ — ລະບົບທີ່ສະຫຼາດເປັນສອງເທົ່າ ແຕ່ສິ່ງເກດເຫັນໄດ້ເຄິ່ງດຽວ ຄືລະບົບທີ່ແຍ່ກວ່າ.

Sustainability, ເສົາທີ່ຫົກ, ຮຽກຮ້ອງໃຫ້ທ່ານສິ້ນເປືອງໂລກກາຍະພາບໜ້ອຍລົງ: rightsize (CPU ທີ່ຫວ່າງເຜົາໄຟຟ້າຈິງ), ຂະຫຍາຍຕາມຄວາມຕ້ອງການແທນການຈັດຊື້ພະຍາກອນຕາມຈຸດສູງສຸດ, ໃຊ້ບໍລິການ managed ແລະ serverless (ໂຄງສ້າງພື້ນຖານທີ່ແບ່ງປັນຄືໂຄງສ້າງພື້ນຖານທີ່ເຕັມກວ່າ), ແບ່ງຂັ້ນ storage,

ແລະ ລົບສິ່ງທີ່ຕາຍແລ້ວ. ສະດວກດີ, ທາງເລືອກທີ່ຍືນຍົງ ແລະ ທາງເລືອກທີ່ຖືກ ປົກກະຕິແມ່ນທາງເລືອກອັນດຽວກັນ — ເວົ້າທັງສອງໃນການທົບທວນ ແລ້ວທ່ານຈະຄອງຫ້ອງໄດ້ສອງເທົ່າ.

ການແລ່ນເສົາທັງໝົດຝ່ອມກັນ — Well-Architected review ຄັ້ງທໍາອິດຂອງທ່ານ. ໃນການທົບທວນຕົວຈິງ ເສົາຫຼັກ *ຂັດແຍ້ງກັນ*: ຄວາມໜ້າເຊື່ອຖືແບບ multi-region ຕີກັບຕົ້ນທຶນ; ການທົບທວນຄວາມປອດໄພທີ່ເຄັ່ງຄັດຕີກັບຄວາມໄວໃນການປະຕິບັດງານ; caching ເພື່ອປະສິດທິພາບຕີກັບຄວາມຖືກຕ້ອງຂອງຄວາມສົດຂອງຂໍ້ມູນ. Framework ບໍ່ໄດ້ແກ້ຂັດແຍ້ງໃຫ້ທ່ານ — ມັນບັງຄັບໃຫ້ພວກມັນອອກມາຢູ່ທີ່ແຈ້ງ, ບ່ອນທີ່ຝ່າຍທຸລະກິດເລືອກໄດ້. ນັ້ນຄືອັດສະລິຍະພາບທັງໝົດຂອງມັນ, ແລະ ເປັນອາວຸດຂອງທ່ານ. ຕົວວິທີການທົບທວນເອງ (ຊຸດຄໍາຖາມ, ການຈັດລຳດັບຄວາມສຳຄັນຂອງສິ່ງທີ່ພົບ, ບົດລາຍງານ) ຖືກສອນເຕັມຮູບແບບໃນໄລຍະທີ 5, Module 12; ສອງອາທິດນີ້ທ່ານຊ້ອມທັກສະດີບ: ການສອບຖາມການອອກແບບອັນດຽວຈາກຫົກທິດທາງ ໃນການນັ່ງເທື່ອດຽວ.

ຄໍາສັບ:

ຄໍາສັບ	ຄວາມໝາຍ
Infrastructure as Code (IaC) / Terraform	ໂຄງສ້າງພື້ນຖານທີ່ຖືກປະກາດໄວ້ໃນໄຟລີ້ຂໍ້ຄວາມທີ່ຄວບຄຸມເວີຊັນ ຊຶ່ງເຄື່ອງມືປ່ຽນໃຫ້ເປັນຈິງ / ເຄື່ອງມືປະເພດນັ້ນທີ່ນິຍົມທີ່ສຸດ.
CI/CD pipeline	ສາຍພານອັດຕະໂນມັດທີ່ build, ທົດສອບ, ແລະ deploy ທຸກການປ່ຽນແປງ.
Blue-green / canary deployment	ສອງ pattern ການປ່ອຍທີ່ປອດໄພ: ສະຫຼັບ traffic ລະຫວ່າງສໍາເນົາເກົ່າ ແລະ ໃໝ່ / ຄ່ອຍໆປ່ອຍ traffic ໄປຫາເວີຊັນໃໝ່ແລ້ວເຝົ້າເບິ່ງ.
Observability (log, metric, trace)	ຄວາມສາມາດຂອງລະບົບໃນການອະທິບາຍຕົນເອງ: ບັນທຶກເຫດການ, ຕົວເລກຕາມເວລາ, ແລະ ແຜນທີ່ການເດີນທາງຕໍ່ຄໍາຮ້ອງຂໍ.
Alert / on-call / runbook	ການແຈ້ງເຕືອນອັດຕະໂນມັດເມື່ອເກີນຂີດ / ເວນຄົນທີ່ຮັບສາຍ / ບົດຄູ່ມືລາຍລັກອັກສອນທີ່ເຂົາເຈົ້າເຮັດຕາມ.
Blameless post-mortem	ການທົບທວນຫຼັງເຫດການແບບຊິສັດ ບໍ່ມີຜູ້ຮ້າຍ ທີ່ຜະລິດການປ້ອງກັນ, ບໍ່ແມ່ນການລົງໂທດ.
Drift	ຄວາມເປັນຈິງທີ່ແຕກຕ່າງອອກຈາກນິຍາມ IaC ເພາະມີຄົນໄປຄລິກຫຍັງບາງຢ່າງ. ສັດຕູຂອງການສ້າງຄືນໄດ້.
Toil	ວຽກປະຕິບັດການດ້ວຍມືທີ່ຊ້າຊາກ ຊຶ່ງລະບົບອັດຕະໂນມັດຄວນດູດຊັບໄປແລ້ວ. ສະຖາປະນິກອອກແບບໃຫ້ມັນຫາຍໄປ.

Well-Architected review

ການສອບຖາມ workload ຢ່າງເປັນລະບົບ ຕໍ່ທັງຫົກເສົາ, ຜະລິດສິ່ງທີ່ພົບແບບຈັດລຳດັບຄວາມສຳຄັນ. ໄລຍະທີ 5 ສອນທ່ານດຳເນີນມັນ.

ບົດຝຶກຫັດ: (1) ບົດຝຶກຫົກເສົາ: ເອົາແຜນວາດທີ່ດີທີ່ສຸດຂອງທ່ານ ແລ້ວໃຊ້ເວລາສືບນາທີ່ຕໍ່ເສົາຂຽນສິ່ງທີ່ພົບ — ຫົກເສົາ, ໜຶ່ງການອອກແບບ, ຫົກເສົາ; ບົດຝຶກທີ 2 ແບບຫຍໍ້, ເຮັດຊ້ຳທຸກອາທິດຈາກນີ້ໄປ. (2) ອ່ານບົດລາຍງານຫຼັກເຫດການຂອງ AWS ສອງບົດ (ຕິພົມເທິງໜ້າ status ຫຼັກເຫດຂັດຂ້ອງໃຫຍ່) ແລ້ວລະບຸວ່າ pattern ຂອງເສົາໃດລົມເຫຼວ. (3) ປຶ້ມບັນທຶກ: ສຳລັບ ThaiTicket, ຂຽນຂໍ້ຂັດແຍ້ງລະຫວ່າງເສົາສາມຂໍ້ທີ່ທ່ານຈະຍົກຂຶ້ນໃຫ້ຝ່າຍທຸລະກິດ, ແຕ່ລະຂໍ້ເປັນທາງເລືອກໜຶ່ງປະໂຫຍກ (“ດ້ວຍງົບນີ້ ເຮົາໄດ້ X ຫຼື Y — ເຮົາອັນໃດ?”).

Milestone — ຈົບໄລຍະທີ 2: ການທົບທວນຈຳລອງ: AI ສ້າງສະຖາປັດຕະຍະກຳທີ່ມີຂໍ້ບົກຜ່ອງໃຫ້ startup ສິນເຊື່ອອອນລາຍຂອງໄທ; ທ່ານຕ້ອງຜະລິດສິ່ງທີ່ພົບເປັນລາຍລັກອັກສອນຄົບທັງຫົກເສົາ, ລວມຢ່າງໜ້ອຍ: ຊ່ອງວ່າງດ້ານ reliability (database ຂອງເຂົາຢູ່ AZ ດຽວ), ຊ່ອງວ່າງດ້ານ security (IAM ກວ້າງເກີນ), ຊ່ອງວ່າງດ້ານຕົ້ນທຶນ (ທຸກຢ່າງເປັນ on-demand), ຊ່ອງວ່າງດ້ານປະຕິບັດການ (ບໍ່ມີ IaC), ແລະ ບົດສິນທະນາ RTO/RPO ທີ່ເຂົາບໍ່ເຄີຍມີ — ແຕ່ລະຂໍ້ຄົ້ນພົບຖືກຮຽບຮຽງເປັນຄຳຖາມທີ່ເພື່ອນຮ່ວມງານຟັງໄດ້ໂດຍບໍ່ຫົດຫູ່. ເມື່ອລາຍການສິ່ງທີ່ພົບຂອງທ່ານອ່ານແລ້ວຄືເພື່ອນຮ່ວມງານອາວຸໂສທີ່ຢາກຊ່ວຍ ແທນທີ່ຈະຄຶຜູ້ກວດສອບ, ໄລຍະທີ 2 ຈົບແລ້ວ.

ໄລຍະທີ 3 — ຄັງລວມ pattern (ອາທິດທີ 15–20)

ສະຖາປະນິກບໍ່ປະດິດ; ພວກເຂົາ **ເລືອກ**. ໄລຍະນີ້ຄືຄັງລວມສະຖາປັດຕະຍະກຳມາດຕະຖານຂອງທ່ານ — ສຳລັບແຕ່ລະອັນ: ມັນແມ່ນຫຍັງ, ໃຊ້ເມື່ອໃດ, ບໍ່ໃຊ້ເມື່ອໃດ, ແລະ ໂປຣໄຟລ໌ຕົ້ນທຶນ/ຄວາມຊັບຊ້ອນຂອງມັນ. ຖ້າ “ເມື່ອໃດບໍ່ຄວນໃຊ້” ຄືເນື້ອແທ້ຂອງໄລຍະນີ້: ຫຼັກສູດໃດກໍບອກທ່ານໄດ້ວ່າ microservices ແມ່ນຫຍັງ; ການຮູ້ວ່າເມື່ອໃດພວກມັນຈະທຳລາຍບໍລິສັດ ຄືສິ່ງທີ່ທ່ານໄດ້ຄຳ້າງ. ຂະນະຮຽນ, ໃຫ້ທ່ານ reference architecture ຈິ່ງຢູ່ <https://aws.amazon.com/architecture/> ຕໍ່ໄປ — ການສັງເກດ pattern ໃນທຳມະຊາດ ຄືບົດຝຶກທີ່ເຮັດໃຫ້ຄັງລວມນິຕິດແໜ້ນ.

Module 8 (ອາທິດທີ 15–17): Pattern ຂອງແອັບພລິເຄຊັນ

Monolith ຫຽບ microservices — ການຖຽງທີ່ດັ່ງທີ່ສຸດຂອງວົງການ, ຕັດສິນຢ່າງສະຫງົບ. Monolith ຄືແອັບພລິເຄຊັນ deploy ໄດ້ອັນດຽວ ທີ່ບັນຈຸທຸກຄຸນສົມບັດ; microservices ແຍກລະບົບເປັນ service ນ້ອຍຫຼາຍ

ອັນ ທີ່ deploy ອິດສະຫຼະ, ແຕ່ລະອັນເປັນເຈົ້າຂອງຂໍ້ມູນຕົນເອງ, ເວົ້າກັນຜ່ານ API ແລະ event. ຕາຕະລາງແບບ ຂຶ້ນ:

Pattern	ມັນແມ່ນຫຍັງ	ໃຊ້ເມື່ອ	ຫ້າມໃຊ້ເມື່ອ	ຕົ້ນທຶນ/ຄວາມຊັບຊ້ອນ
Monolith	ແອັບດຽວ, deploy ດຽວ, database ດຽວ	ທີມນ້ອຍ, ຜະລິດຕະພັນຍັງໜຸ່ມ, ຂອບເຂດ domain ຍັງບໍ່ຈະແຈ້ງ — ນັ້ນຄື ລະບົບໃໝ່ສ່ວນໃຫຍ່	ຫຼາຍທີມຕິດຂັດ ເພາະກັນ ແລະ ກັນ; ບາງສ່ວນຕ້ອງຂະຫຍາຍຕ່າງກັນ ຫຼາຍ	ຄວາມຊັບຊ້ອນຕ່ຳ, ຕົ້ນທຶນຕ່ຳ; ຂະຫຍາຍໄດ້ໄກ ກວ່າທີ່ແຟຊັ້ນ ຍອມຮັບ — “ໜ້າເບື້ອ” ຄືຄຸນສົມບັດ
Microservices	Service ນ້ອຍຫຼາຍອັນ, build, deploy, ແລະ scale ອິດສະຫຼະ	ຫຼາຍທີມທີ່ຕ້ອງການຈັງຫວະປ່ອຍງານ ອິດສະຫຼະ; ແຕ່ລະສ່ວນຂະຫຍາຍຕ່າງກັນສຸດຂີດ; ຂອບເຂດ domain ພິສູດແລ້ວ ແລະ ໝັ້ນຄົງ	ທີມນ້ອຍ (“distributed monolith ຄື monolith ທີ່ເພີ່ມຄວາມລົ້ມເຫຼວຂອງເຄືອຂ່າຍເຂົ້າໄປ”); domain ຍັງປ່ຽນຢູ່	ຄວາມຊັບຊ້ອນສູງ: ທຸກການເອີ້ນ function ກາຍເປັນການເອີ້ນຜ່ານເຄືອຂ່າຍທີ່ລົ້ມເຫຼວໄດ້; ຕ້ອງການ CI/CD, observability, on-call ທີ່ເປັນຜູ້ໃຫຍ່

ຄໍາຕັດສິນຂອງສະຖາປະນິກ: **ເລີ່ມແບບ monolithic**, ແຕ່ເປັນ **module ຢູ່ພາຍໃນ**; ແຍກ **service ອອກເມື່ອ — ແລະ ສະເພາະເມື່ອ — ຄວາມເຈັບປວດຈາກການຂະຫຍາຍທີມ ຫຼື ການຂະຫຍາຍໂຫຼດມາເຖິງແທ້ໆ**. ການເວົ້າສິ່ງນີ້ໃນການສຳພາດ, ພ້ອມເຫດຜົນ, ບົ່ງບອກວ່າທ່ານເປັນຜູ້ອາວຸໂສ; ອຸດົມການໄປທາງໃດທາງໜຶ່ງ ບົ່ງບອກວ່າທ່ານເປັນຜູ້ນ້ອຍ.

Event-driven architecture — queue ແລະ topic, ຕິວດູດແຮງກະແຫກ. ແທນທີ່ສ່ວນປະກອບຈະເອີ້ນຫາກັນແລ້ວ *ລໍຖ້າ* (synchronous), ສ່ວນປະກອບເຜີຍແຜ່ **event** (“OrderPlaced”) ໄປຫາ **queue** (SQS — ຜູ້ບໍລິໂພກຄົນດຽວຮັບແຕ່ລະຂໍ້ຄວາມ, ຕາມຈັງຫວະຂອງຕົນ) ຫຼື **topic** (SNS — ທຸກຜູ້ສະໝັກຮັບໄດ້ສຳເນົາ; fan-out). ຜູ້ຜະລິດບໍ່ຮູ້ ແລະ ບໍ່ສົນວ່າໃຜກຳລັງຟັງ. ໃຊ້ເມື່ອ: ສ່ວນປະກອບຄວນຢູ່ລອດການຂັດຂ້ອງຂອງກັນ ແລະ ກັນ (queue ຖືຂໍ້ຄວາມໄວ້ຂະນະຜູ້ບໍລິໂພກລົ້ມ — ໄດ້ທັງຄວາມທົນທານ ແລະ ການແຍກຂາດຈາກກັນໃນການຊື້ຄັງດຽວ); ໂຫຼດຜຸ່ງເປັນຊ່ວງ (queue ຄູດຊັບການຜຸ່ງ; worker ຄ່ອຍໆລະບາຍມັນ); ເຫດການໜຶ່ງກະຕຸ້ນປະຕິກິລິຍາຫຼາຍຢ່າງ (ສັງຊີແລ້ວ → ຫັກເງິນ, ອີເມວ, ສາງ, ວິເຄາະ — ສື່ຜູ້ສະໝັກຮັບ, ຜູກຕິດເປັນສູນ). ຫ້າມໃຊ້ເມື່ອ: ຜູ້ໃຊ້ຕ້ອງການຄຳຕອບ *ດຽວນີ້* (ການຊຳລະເງິນຈະ “ຄ່ອຍ” ຍືນຍັນບໍ່ໄດ້); ຫຼື ທີມນ້ອຍ ແລະ ການເອີ້ນແບບ synchronous ງ່າຍໆກໍພຽງພໍ — ທຸກ queue ເພີ່ມ semantics ຂອງການສົ່ງມອບ (at-least-once ໝາຍວ່າຜູ້ບໍລິໂພກຕ້ອງ **idempotent** — ແລ່ນສອງເທື່ອກໍປອດໄພ), ການຈັດການ dead-letter, ແລະ ພາລະການເຝົ້າຕິດຕາມ. ຕົ້ນທຶນ/ຄວາມຊັບຊ້ອນ: ສ່ວນປະກອບຖືກ, ການ debug ແພງກວ່າ — ເລືອງລາວຂອງຄຳຮ້ອງຂໍໜຶ່ງອັນ

ຕອນນີ້ກະຈາຍຢູ່ຕາມ service ແລະ queue, ຊຶ່ງເປັນເຫດຜົນວ່າ observability (Module 7) ເຊົາເປັນທາງເລືອກຢູ່ຈຸດນີ້.

Three-tier: ເປັນຂອງທ່ານແລ້ວ (Module 3). ໃຊ້ສໍາລັບ: ໃຈກາງອັນກວ້າງຂອງແອັບພລິເຄຊັນເວັບ — ມັນຄື pattern ທີ່ອັນອີນຖືກວັດທຽບກັບ. ບໍ່ໃຊ້ສໍາລັບ: workload ທີ່ພຸ່ງແຮງຜ່ອມຮ່ອງເວລາຫວ່າງ (serverless ຖືກກວ່າ) ຫຼື ຜະລິດຕະພັນຫຼາຍທີມທີ່ໃຫຍ່ແທ້ (ເບິ່ງຂ້າງເທິງ). ໂປຣໄຟລ໌: ເຂົາໃຈກັນດີທຸກບ່ອນ, ຈ້າງຄົນງ່າຍ, ຕົ້ນທຶນປານກາງ.

Serverless-first: ປະກອບຊິ້ນສ່ວນ managed — API Gateway → Lambda function → DynamoDB, S3, ແລະ queue — ໂດຍບໍ່ເປັນເຈົ້າຂອງ server ເລີຍ. ໃຊ້ເມື່ອ: traffic ພຸ່ງເປັນຊ່ວງ ຫຼື ຕໍ່າ (scale-to-zero ໝາຍວ່າຕົ້ນທຶນຕອນຫວ່າງ ~ບໍ່ມີ), ທີມນ້ອຍ, ຕ້ອງອອກຕະຫຼາດໄວ, ກາວເຊື່ອມແບບ event-driven. ຫ້າມໃຊ້ເມື່ອ: compute ທີ່ແລ່ນດົນ ຫຼື ສະເພາະທາງ (ຂ້ໍຈຳກັດ runtime), ເສັ້ນທາງທີ່ອ່ອນໄຫວຕໍ່ latency ສູງຂຶ້ນ (cold start), ໂຫຼດ ຄົງທີ່ ຂະໜາດມະຫາສານ (ລາຄາຕໍ່ການເອີ້ນ ອາດເກີນ server ທີ່ຈອງໄວ້ — ຄິດເລກທີ່ຂະໜາດຈິງ), ຫຼື ເມື່ອຄວາມເຄື່ອນຍ້າຍໄດ້ລະຫວ່າງ cloud ເປັນຂໍ້ກຳນົດແທ້ (ນີ້ຄື lock-in ທີ່ເລິກທີ່ສຸດໃນບັນດາ pattern — ມັກຄຸ້ມ, ແຕ່ໃຫ້ເວົ້າອອກມາດັ່ງໆ). ໂປຣໄຟລ໌: ພາລະປະຕິບັດການຕໍ່າທີ່ສຸດໃນຄັ້ງລວມ; ຕົ້ນທຶນຍອດຢ້ຽມທີ່ຂະໜາດຕໍ່າ/ພຸ່ງເປັນຊ່ວງ, ຕ້ອງກວດທີ່ຂະໜາດຄົງທີ່ສູງ.

ຄໍາສັບ:

ຄໍາສັບ	ຄວາມໝາຍ
Monolith / modular monolith	ແອັບ deploy ໄດ້ອັນດຽວທີ່ມີທຸກຢ່າງຢູ່ໃນນັ້ນ / ສະບັບມີວິໄນ: deploy ອັນດຽວ, ຂອບເຂດ module ພາຍໃນສະອາດ — ຄໍາມາດຕະຖານທີ່ດີທີ່ສຸດສໍາລັບລະບົບໃໝ່.
Microservices	Service ນ້ອຍຫຼາຍອັນທີ່ deploy ອິດສະຫຼະ, ແຕ່ລະອັນເປັນເຈົ້າຂອງຂໍ້ມູນຕົນ. ເຄື່ອງມືຂະຫຍາຍທີມ ທີ່ເກັບພາສີລະບົບກະຈາຍ.
Coupling / decoupling	ສ່ວນປະກອບເພິ່ງພາຄວາມຜ່ອມ ແລະ ລາຍລະອຽດຂອງກັນ ແລະ ກັນຫຼາຍປານໃດ. ສະຖາປັດຕະຍະກຳສ່ວນໃຫຍ່ຄືສິນລະປະຂອງການຊຶກການແຍກຂາດໃນປະລິມານທີ່ພໍດີ.
Synchronous / asynchronous	ເອີ້ນແລ້ວລໍ ຫຽບກັບ ສົ່ງແລ້ວໄປຕໍ່. ທາງເລືອກພື້ນຖານເທິງທຸກລູກສອນທີ່ທ່ານແຕ້ມ.
Event / event-driven	ຂັ້ນຕົ້ນຈິງທີ່ປະກາດໃຫ້ໃຜກໍ່ໄດ້ທີ່ຟັງ (“OrderPlaced”) / ສະຖາປັດຕະຍະກຳທີ່ສ້າງຈາກການປະກາດແບບນັ້ນ.

ຄຳສັບ	ຄວາມໝາຍ
Queue (SQS)	ແຖວຂໍ້ຄວາມ; ແຕ່ລະອັນຖືກຮັບໂດຍຜູ້ບໍລິໂພກຄົນດຽວຕາມຈັງຫວະຂອງຕົນ. ຕົວຕູດແຮງກະແທກ ແລະ ຕົວແຍກຂາດ.
Topic / pub-sub (SNS)	ຊ່ອງກະຈາຍສຽງ; ທຸກຜູ້ສະໝັກຮັບໄດ້ທຸກຂໍ້ຄວາມ. ເຄື່ອງມື fan-out.
Dead-letter queue	ບ່ອນທີ່ຂໍ້ຄວາມທີ່ປະມວນຜົນລົ້ມເຫຼວຊ້າໆ ຖືກວາງແຍກໄວ້ໃຫ້ຄົນເບິ່ງ — ຕາໜ່າງນິລະໄພຂອງ pattern ນີ້.
Idempotent	ປະມວນຜົນສອງເທື່ອກໍໄດ້ຜົນເທົ່າເກົ່າ — ຈຳເປັນສຳລັບຜູ້ບໍລິໂພກ, ເພາະ queue ອາດສົ່ງຂໍ້ຄວາມຫຼາຍກວ່າໜຶ່ງເທື່ອ.
Serverless-first	ການປະກອບຊິນສ່ວນ managed ທີ່ scale-to-zero (function, DB managed, queue) ແທນການແລ່ນ server.
Lock-in	ການເພິ່ງພາບໍລິການສະເພາະຂອງຜູ້ໃຫ້ບໍລິການລາຍດຽວ, ຕິລາຄາເປັນຕົ້ນທຶນການປ່ຽນ. ບໍ່ແມ່ນບາບ — ເປັນຄຳສັບເສດຖະສາດທີ່ຕ້ອງຊົ່ວນ້ຳໜັກຢູ່ທີ່ແຈ້ງ (ໄລຍະທີ 4).

ບົດຝຶກຫັດ: (1) ອອກແບບກະແສຄຳສັ່ງຊື້ຂອງ ThaiTicket ສອງເທື່ອ — ແບບ synchronous three-tier, ແລ້ວແບບ event-driven ດ້ວຍ SQS/SNS — ແລ້ວຂຽນໜຶ່ງວົກວ່າແຕ່ລະສະບັບເຮັດຫຍັງເມື່ອຜູ້ໃຫ້ບໍລິການຊຳລະເງິນຂັດຂ້ອງ; ວົກນັ້ນຄືເຫດຜົນທັງໝົດຂອງ event. (2) ສຳລັບສື່ບໍລິສັດ (startup 3 ຄົນ, scale-up ວິສະວະກອນ 50 ຄົນ, ທະນາຄານ, ແອັບໂຫວດລາຍການໂທລະທັດທີ່ໃຊ້ 4 ຄົນຕໍ່ປີ), ເລືອກ pattern ແລ້ວປົກປ້ອງມັນ — ຈາກນັ້ນບອກຊື່ຕົວກະຕຸ້ນທີ່ຈະເຮັດໃຫ້ແຕ່ລະບໍລິສັດປ່ຽນ pattern. (3) ຊອກຫາເລື່ອງຈາກປຶ້ມວິສະວະກຳຈິງໜຶ່ງເລື່ອງແບບ “ພວກເຮົາຍ້າຍໄປ microservices ແລ້ວເສຍໃຈ” ແລະ ອີກເລື່ອງທີ່ສຳເລັດ; ບັນທຶກຄວາມຕ່າງ (ເກືອບສະເໝີແມ່ນຂະໜາດທີມ ແລະ ຄວາມເປັນຜູ້ໃຫຍ່ຂອງ domain).

Milestone: ເມື່ອໄດ້ຄຳບັນຍາຍບໍລິສັດ, ທ່ານແນະນຳ pattern *ຜ່ອນທາງອອກຂອງມັນ* ໄດ້ — “ເລີ່ມທີ່ນີ້; ເມື່ອ X ເກີດຂຶ້ນ, ວິວັດໄປສູ່ Y” — ພາຍໃນຫ້ານາທີ. ເສັ້ນທາງວິວັດ, ບໍ່ແມ່ນຄຳຕັດສິນ, ຄືວິທີທີ່ສະຖາປະນິກເວົ້າກັນຈິງ.

Module 9 (ອາທິດທີ 18–20): Pattern ດ້ານຂໍ້ມູນ, ຫົວໂລກ, ແລະ hybrid

ສະຖາປັດຕະຍະກຳຂໍ້ມູນ — lake ທຽບ warehouse. Database ແບບທຸລະກຳ (ໄລຍະທີ 1) ດຳເນີນທຸລະກິດ; ການວິເຄາະຢາກ ຖາມຄຳຖາມ ຕໍ່ມັນ ໂດຍບໍ່ເຮັດໃຫ້ມັນຊ້າ. **Data warehouse** (Redshift, Snowflake,

BigQuery) ເກັບຂໍ້ມູນ *ທີ່ມີໂຄງສ້າງ, ທຳຄວາມສະອາດແລ້ວ* ປັບແຕ່ງມາເພື່ອການວິເຄາະ SQL ໄວ — ໃຊ້ເມື່ອຮູ້ ຄຳຖາມແລ້ວ ແລະ dashboard ຕ້ອງໄວ; ແພງກວ່າຕໍ່ TB, ຕ້ອງມີວິໄນການສ້າງແບບຈຳລອງແຕ່ຕົ້ນ. **Data lake** (S3 + catalog + query engine ເຊັ່ນ Athena) ເກັບ *ທຸກຢ່າງ, ດິບ, ຖືກ* — log, clickstream, ຮູບ — ແລ້ວໃສ່ ໂຄງສ້າງຕອນອ່ານ; ໃຊ້ເມື່ອທ່ານຢາກເກັບທຸກຢ່າງໄວ້ກ່ອນ ແລ້ວຄ່ອຍຕັດສິນຄຳຖາມພາຍຫຼັງ; ແຕ່ຖ້າບໍ່ມີການ ກຳກັບ ມັນຈະເສື່ອມເປັນເລື່ອງຕະຫຼົກຂອງວົງການ, **data swamp** (ໝອງຂໍ້ມູນ). ຂໍສະຫຼຸບຮ່ວມສະໄໝແມ່ນເອົາ ທັງສອງ, ວາງເປັນຊັ້ນ (“lakehouse”): ຄວາມຈິງດິບຢູ່ໃນ lake, mart ທີ່ຄັດສັ່ນຢູ່ໃນ warehouse, ປ້ອນໂດຍ **ETL/ELT pipeline**. ກົດຂອງສະຖາປະນິກ: ການວິເຄາະບໍ່ query database production ຈັກເທື່ອ (replica ຫຼື pipeline ເປັນຜູ້ປ້ອນ), ແລະ ທຸກຊຸດຂໍ້ມູນມີເຈົ້າຂອງ, ມີລາຍການໃນ catalog, ແລະ ມີນະໂຍບາຍການເກັບ ຮັກສາ — ແຖວການກຳກັບນັ້ນຄືໜຶ່ງປະໂຫຍກໃນການອອກແບບຂອງທ່ານ ແລະ ຄືຄວາມເຈັບປວດໜຶ່ງປີຖ້າທ່ານ ລືມມັນ.

Multi-region: active-passive ຫຼືບ active-active. ເມື່ອ region ດຽວບໍ່ພຽງພໍ — ເພາະທຸລະກິດຮຽກຮ້ອງ DR ຈາກໄພພິບັດລະດັບ region, ຫຼື ຜູ້ໃຊ້ກະຈາຍຢູ່ຫຼາຍທະວີບ — ທ່ານເລືອກ:

Pattern	ມັນແມ່ນຫຍັງ	ໃຊ້ເມື່ອ	ຫ້າມໃຊ້ເມື່ອ	ຕົ້ນທຶນ/ຄວາມຊັບຊ້ອນ
Active-passive	Region ໜຶ່ງໃຫ້ ບໍລິການ; region standby ຖືຂໍ້ມູນທີ່ replicate ໄວ້, ຈາກ “backup ຢ່າງດຽວ” (cold) ຫາ “ສຳເນົາ ຂະໜາດຫຍໍ້ແລ້ວ ຢູ່” (warm), ຖືກຍົກ ຂຶ້ນເມື່ອເກີດ ໄພພິບັດ	RTO/RPO ທີ່ຝ່າຍ ທຸລະກິດລະບຸ ເປັນ ເຫດຜົນຮອງຮັບ; compliance ຮຽກຮ້ອງໃຫ້ມີ ເລື່ອງ DR	ບໍ່ມີໃຜເຊັນຮັບຮອງ RTO/RPO ທີ່ຖືກ ກັບລາຍຈ່າຍ (ຄວາມຕ້ອງການທີ່ ຊື່ສັດຂອງບໍລິສັດ ສ່ວນໃຫຍ່ຄື multi-AZ ທີ່ດີ + backup ຂ້າມ region)	ຕົ້ນທຶນ 1.1×–1.7× ຕາມຄວາມອຸ່ນຂອງ standby; ຄວາມ ຊັບຊ້ອນປານກາງ — ຂໍກຳນົດທີ່ຂ້າຄື ການ <i>failover</i> ທີ່ ທ່ານຊ້ອມແທ້, ບໍ່ ດັ່ງນັ້ນ standby ຄື ລະຄອນ
Active-active	ສອງ region ຂຶ້ນໄປ ໃຫ້ບໍລິການ ພ້ອມກັນ; ຜູ້ໃຊ້ຖືກ ນຳໄປຫາອັນໃກ້ ທີ່ສຸດ; ຂໍ້ມູນ replicate ສອງທາງ	ຖານຜູ້ໃຊ້ຫົວໂລກທີ່ ຕ້ອງການ latency ໃນທ້ອງຖິ່ນ; RTO ເກືອບສູນຈຳເປັນແທ້ (ເຄືອຂ່າຍຊຳລະ ເງິນ, ການຊື້ຂາຍ, SaaS ໃຫຍ່)	ເກືອບທຸກຄົນນອກ ເໝືອຈາກນັ້ນ — <i>write conflict</i> ຂ້າມ region ຄືໜຶ່ງໃນ ບັນຫາທີ່ຍາກແທ້ໆ ຂອງວົງການຄອມ ພິວເຕີ	ຕົ້ນທຶນ 2×+, ຊັບ ຊ້ອນທີ່ສຸດໃນ ຫຼັກສູດນີ້; ຕ້ອງມີ data store ທີ່ແກ້ໄຂ ຂັດແຍ້ງໄດ້ ແລະ ທີມອາວຸໂສ

ປະໂຫຍກລະດັບສຳພາດ: “Multi-AZ is table stakes; multi-region is a business case.” (Multi-AZ ຄື ເງື່ອນໄຂພື້ນຖານ; multi-region ຄືກໍລະນີທາງທຸລະກິດ.) ໃຫ້ຝ່າຍທຸລະກິດລະບຸ RTO/RPO, ຕິລາຄາທາງເລືອກ, ແລ້ວປ່ອຍໃຫ້ຕົວເລກເປັນຜູ້ເລືອກ.

Hybrid cloud. ສ່ວນໜຶ່ງ on-prem, ສ່ວນໜຶ່ງ cloud, ເຊື່ອມກັນດ້ວຍ VPN ຫຼື Direct Connect — ສໍາລັບ ວິສາຫະກິດທີ່ຕັ້ງໜ້າແລ້ວສ່ວນໃຫຍ່ ນີ້ບໍ່ແມ່ນ pattern ແຕ່ເປັນ *ຄວາມເປັນຈິງທີ່ຍາວເປັນທົດສະວັດ*: mainframe ທີ່ຍ້າຍບໍ່ໄດ້, ລະບົບໂຮງງານທີ່ຜູກກັບ latency, ຂໍ້ມູນທີ່ຖືກກໍາກັບໃຫ້ປັກຢູ່ on-prem, ແລະ ການ migrate (ໄລຍະ ທີ 4) ທີ່ກໍາລັງຜ່ານໄປ. ໃຊ້ໃນຖານະ: ຂົວທີ່ຈິງໃຈ ຜ່ອມທົດທາງການເດີນທາງ. ຫ້າມຍອມຮັບ: “hybrid” ໃນຄວາມ ໝາຍອ້ອມຄ້ອມວ່າ “ພວກເຮົາບໍ່ເຄີຍຕັດສິນໃຈ.” ບັນທຶກການອອກແບບ: ຕົວຕົນຕ້ອງເປັນອັນດຽວກັນກ່ອນ (login ດຽວຂ້າມທັງສອງໂລກ — ນີ້ຄືສິ່ງທີ່ປະກາດວຽກວິສາຫະກິດໝາຍເຖິງດ້ວຍຄຳວ່າ “hybrid identity integration”); ລະບຸວ່າລະບົບໃດເປັນແຫຼ່ງຄວາມຈິງ; ເປົ້າຕົ້ນທີ່ egress ຂ້າມສາຍ; ແລະ ຄາດໄວ້ວ່າສາຍ ເຊື່ອມຕໍ່ ຄື SPOF ເວັ້ນແຕ່ຈະມີສອງເສັ້ນ. ຄວາມຊັບຊ້ອນ: ທຸກຢ່າງເປັນສອງກອງມູນ — ນັ້ນຄືເຫດຜົນຊີ້ສັດ ວ່າເປັນຫຍັງສະຖາປະນິກຈຶ່ງຢູ່ໃຫ້ຫຍັງຝັ່ງ on-prem ລົງເລື້ອຍໆ.

ຄໍາສັບ:

ຄໍາສັບ	ຄວາມໝາຍ
OLTP / OLAP	ການປະມວນຜົນທຸລະກໍາ (ການອ່ານ/ຂຽນນ້ອຍໆໄວໆ ຫຼາຍອັນ — ດໍາເນີນທຸລະກິດ) ທຽບກັບການປະມວນຜົນ ວິເຄາະ (ການສະແດງຂະໜາດໃຫຍ່ — ສຶກສາ ທຸລະກິດ). ແຍກພວກມັນອອກ.
Data warehouse	ບ່ອນເກັບຂໍ້ມູນທີ່ມີໂຄງສ້າງ, ຄັດສິດແລ້ວ, ປັບແຕ່ງເພື່ອ ການວິເຄາະ SQL ໄວ (Redshift, Snowflake, BigQuery).
Data lake / data swamp	ບ່ອນເກັບທຸກຢ່າງແບບຖືກ, ດິບ, ໃສ່ໂຄງສ້າງຕອນອ່ານ (S3 + Athena) / ສະບັບທີ່ບໍ່ມີການກໍາກັບ, ບ່ອນທີ່ຂໍ້ມູນ ໄປສູ່ນຫາຍ.
ETL / ELT	Pipeline ທີ່ຍ້າຍຂໍ້ມູນຈາກລະບົບຕົ້ນທາງເຂົ້າສູ່ lake/warehouse (Extract, Transform, Load — ລໍາດັບແຕກຕ່າງກັນ).
Data governance / catalog / retention	ຄວາມເປັນເຈົ້າຂອງ, ເອກະສານ, ແລະ ກົດອາຍຸຂອງທຸກ ຊຸດຂໍ້ມູນ — ໜຶ່ງປະໂຫຍກໃນການອອກແບບ ທີ່ປະຢັດ ຄວາມເຈັບປວດໜຶ່ງປີ.
Replication (sync / async)	ການສໍາເນົາຂໍ້ມູນຢ່າງຕໍ່ເນື່ອງໄປຫາ database ຫຼື region ອື່ນ — ສອດຄ່ອງທັນທີແຕ່ຈໍາກັດໄລຍະທາງ ທຽບ ກັບ ຊ້າໜ້ອຍໜຶ່ງແຕ່ໄປໄດ້ທຸກບ່ອນ. ຄວາມຊັກຊ້າຂອງ async ຄືບ່ອນທີ່ RPO ມາຈາກ.

ຄຳສັບ	ຄວາມໝາຍ
Active-passive / failover drill	DR ແບບ region standby / ການຊ້ອມຕາມກຳນົດທີ່ ພິສູດວ່າມັນໃຊ້ໄດ້. Failover ທີ່ບໍ່ໄດ້ຊ້ອມ ຄືຄວາມຫວັງ.
Active-active	ຫຼາຍ region ໃຫ້ບໍລິການພ້ອມກັນ ພ້ອມ replication ສອງທາງ. ຊົງພະລັງ; ຍາກແທ້; ປົກກະຕິບໍ່ຈຳເປັນ.
Write conflict	ສອງ region ປ່ຽນຂໍ້ມູນອັນດຽວກັນພ້ອມກັນ — ເຫດຜົນ ທາງເຕັກນິກທີ່ active-active ເປັນເຂດຂອງຜູ້ຊ່ຽວຊານ.
Hybrid cloud / hybrid identity	On-prem + cloud ເຊື່ອມເປັນກອງມູນດຽວ / ການ sign-on ດຽວຂ້າມທັງສອງ — ສິ່ງທຳອິດທີ່ຕ້ອງລວມ.
Direct Connect	ສາຍກາຍະພາບສ່ວນຕົວທີ່ເຊື່ອມ data center ກັບ cloud — ສາຍແຮ່ຂອງ hybrid, ເຮັດສອງເສັ້ນຖ້າມັນ ສຳຄັນ.

ບົດຝຶກຫັດ: (1) ThaiTicket ຂະຫຍາຍສູ່ພາກພື້ນ: ອອກແບບການຂະຫຍາຍໄປສົ່ງກະໂປສອງເທື່ອ — active-passive (ບາງກອກເປັນ primary) ແລະ active-active — ພ້ອມຕົວຄຸນຕົ້ນທຶນ ແລະ RTO/RPO ຂອງແຕ່ລະອັນ; ຂຽນຄຳແນະນຳໜຶ່ງໜ້າ ແລ້ວ *ຕັດສິນໃຈ*. (2) ຮ້ານຄ້າປົກແຫ່ງໜຶ່ງມີຂໍ້ມູນການຂາຍ 15 ປີໃນ database SQL production ແລະ ຢາກໄດ້ “ການວິເຄາະທີ່ພ້ອມສຳລັບ AI”; ຮ່າງການອອກແບບ lake + warehouse ແລ້ວຂຽນສາມປະໂຫຍກເລື່ອງການກຳກັບ. (3) ສັງເກດ pattern: ເລືອກ reference architecture ສາມອັນຈາກ <https://aws.amazon.com/architecture/> ແລ້ວບອກຊື່ທຸກ pattern ໃນຄັງລວມທີ່ປາກົດໃນແຕ່ລະອັນ.

Milestone — ຈົບໄລຍະທີ 3: ດ່ານທົດສອບຄັ້ງລວມ: AI ໃຫ້ສະຖານະການໄວໆຫົກອັນ; ສຳລັບແຕ່ລະອັນ ທ່ານບອກຊື່ pattern, ເຫດຜົນ, ຄຳເຕືອນ “ເມື່ອໃດບໍ່ຄວນໃຊ້”, ແລະ ໂປຣໄຟລ໌ຕົ້ນທຶນ/ຄວາມຊັບຊ້ອນ — ອັນລະຕຳກວ່າຫ້າພາກສ່ວນ. ການໂຕ້ຕອບແບບນີ້ພໍດີ, ທີ່ຄວາມໄວນີ້ພໍດີ, ຄືໜຶ່ງສ່ວນສາມກາງຂອງການສຳພາດສະຖາປະນິກຕົວຈິງ.

ໄລຍະທີ 4 — ການ migrate ແລະ ໂລກຕົວຈິງ (ອາທິດທີ 21–26)

ການອອກແບບ greenfield ຄືສ່ວນນ້ອຍຂອງວຽກນີ້. ສະຖາປັດຕະຍະກຳສ່ວນໃຫຍ່ເກີດຂຶ້ນໃນບໍລິສັດທີ່ມີຢູ່ແລ້ວ — ພ້ອມທັງ server, software ບູຮານທີ່ທຸລະກິດເພີ່ງພາ, ສັນຍາ, ແລະ ກົດໝາຍ. ໄລຍະນີ້ຄືໂລກນັ້ນ, ແລະ ມັນຄືບ່ອນທີ່ “lead migration and modernization initiatives” — ຂໍ້ໝັ້ງໃນເກືອບທຸກປະກາດວຽກທີ່ພວກເຮົາສຶກສາ — ຖືກສອນ.

Module 10 (ອາທິດທີ 21–23): ການ migrate — ຍ້າຍບໍລິສັດໜຶ່ງຂຶ້ນ cloud

7 Rs — ຄໍາສັບຮ່ວມຂອງທຸກບົດສົນທະນາເລື່ອງ migration. ສໍາລັບແຕ່ລະ workload ໃນກອງມູນ, ທ່ານເລືອກໜຶ່ງອັນ:

R	ຄວາມໝາຍ	ເນື້ອໃນ	ແຮງງານ / ຜົນຕອບແທນ
Retire	ປິດມັນໄປ — ບໍ່ມີໃຜໃຊ້ຈິງ	ທຸກກອງມູນມີແບບນີ້ 10–20%; ຫາພວກມັນກ່ອນ	ງ່າຍໆ / ປະຢັດທັນທີ — R ທີ່ດີທີ່ສຸດ
Retain	ປະໄວ້ on-prem, ຕອນນີ້ກ່ອນ	ລະບົບທີ່ຜູກກັບ latency, ຖືກປົກໂດຍ compliance, ຫຼື ໃກ້ໝົດອາຍຸ	ບໍ່ມີ / ເລືອນຕື່ນຫິນ — ຄໍາວ່າ “ຍັງບໍ່ທັນ” ແບບຊື່ສັດ
Rehost (“lift and shift”)	ຍ້າຍໄປ VM ເທິງ cloud ຕາມສະພາບ	ຄວາມໄວສໍາຄັນ, ແອັບພິເຄຊັນຄົງ, ທັກສະຍັງບາງ	ຕໍ່າ / ອອກຈາກ data center ໄດ້ໄວ, ແຕ່ຍັງໄດ້ປະໂຫຍດ cloud ໜ້ອຍ — ກ້າວທໍາອິດ, ບໍ່ແມ່ນຈຸດໝາຍ
Relocate	ຍ້າຍທີ່ລະດັບ hypervisor (ເຊັ່ນ ຝູງ VMware ໄປ VMware-on-cloud)	ກອງມູນ virtualized ຂະໜາດໃຫຍ່ທີ່ມີເສັ້ນຕາຍ	ຕໍ່າ / ຍ້າຍກ້ອນໃຫຍ່ໄດ້ໄວທີ່ສຸດ; ເປັນຈຸດແວ່ພັກ
Repurchase (“drop and shop”)	ປ່ຽນໄປໃຊ້ SaaS	ແອັບທີ່ບໍ່ສ້າງຄວາມແຕກຕ່າງ — ອີເມວ, HR, CRM	ຕໍ່າ-ປານກາງ / ທັງໝວດອອກຈາກກອງມູນຂອງທ່ານ
Replatform (“lift, tinker, shift”)	ຍົກລະດັບເລັກນ້ອຍລະຫວ່າງທາງ — DB ທີ່ຄຸ້ມຄອງເອງ → RDS, ແອັບ → container	ທາງກາງແບບຈິງຈັງ: ໄດ້ປະໂຫຍດຈິງ, ຄວາມສ່ຽງມີຂອບເຂດ	ປານກາງ / R ຕົວແຮງງານຫຼັກຂອງ migration ສ່ວນໃຫຍ່
Refactor / re-architect	ຂຽນໃໝ່ໃຫ້ເປັນ cloud-native (pattern ຈາກໄລຍະທີ 3)	ລະບົບແກນທີ່ສ້າງຄວາມແຕກຕ່າງ ຊຶ່ງຂໍ້ຈຳກັດຂອງມັນທໍາຮ້າຍທຸລະກິດ	ສູງ / ຜົນຕອບແທນສູງທີ່ສຸດ — ໃຊ້ມັນກັບ workload ຈຳນວນໜ້ອຍທີ່ສົມຄວນ

ວິທີການ migrate: assess → mobilize → migrate. Assess (ປະເມີນ): ສໍາຫຼວດທຸກຢ່າງ (ເຄື່ອງມື discovery ບວກກັບໂບຮານຄະດີຂອງການໄປຖາມຄົນ), ແລ້ວໃຫ້ຄະແນນແຕ່ລະ workload ດ້ານຄຸນຄ່າທຸລະກິດ ແລະ ຄວາມຍາກໃນການ migrate — ຜົນຜະລິດ: ບັນຊີລາຍການແອັບພິເຄຊັນທີ່ມີ R ຕໍ່ແຖວ, ແລະ **business**

case (TCO ຂອງການຢູ່ຕໍ່ ທຽບກັບການຍ້າຍ; ຈົ່ງຊິສັດວ່າໃບປິ່ນຈະ *ສູງຂຶ້ນ* ໃນຊ່ວງທີ່ບໍ່ຊ້ອນທີ່ທັງສອງກອງມູນ ແລ່ນຜ່ອມກັນ — ຜູ້ນຳທີ່ບໍ່ຖືກເຕືອນເລືອງ migration bubble ຈະກາຍເປັນຜູ້ນຳທີ່ຍົກເລີກ migration ກາງທາງ).

Mobilize (ລະດົມ): ສ້າງ **landing zone** — ພື້ນຖານ cloud ທີ່ສ້າງໄວ້ລ່ວງໜ້າ ແລະ ຖືກກຳກັບ *ກ່ອນ* workload ອັນທຳອິດຈະລົງຈອດ: ໂຄງສ້າງຫຼາຍບັນຊີ (ບັນຊີແຍກຕ່າງສະພາບແວດລ້ອມ ແລະ ຕໍ່ທີມ, ເພື່ອໃຫ້ blast radius ນ້ອຍ), ຕິວຕິນ ແລະ logging ແບບລວມສູນ, ຮັບເຄືອຂ່າຍ (VPC ແບບ hub-and-spoke, Direct Connect ກັບໄປ on-prem), ແລະ **guardrail** — policy ອັດຕະໂນມັດທີ່ເຮັດໃຫ້ເສັ້ນທາງທີ່ປອດໄພ, ຕິດປ້າຍ, ແລະ ສອດຄ່ອງ ເປັນຄຳມາດຕະຖານ (AWS Control Tower ຄືຊຸດເລີ່ມຕົ້ນແບບ managed). ປະກາດວຽກເວົ້າວ່າ “design the landing zone every workload inherits” — ນີ້ຄືອັນນັ້ນ. ການຂ້າມມັນເພື່ອ “ເລີ່ມ migrate ໄປກ່ອນ” ຄືການສ້າງ data center ທີ່ອີກເຮືອຂຶ້ນມາໃໝ່ໃນ cloud ດ້ວຍຕົນທຶນສູງກວ່າ; ນັ້ນຄືລາຍເຊັນຄລາສສິກຂອງ migration ທີ່ລົ້ມເຫຼວ. **Migrate ເປັນ wave (ຄືນ):** ຈັດ workload ເປັນຄືນລະສອງສາມອັນ, ຮຽງງ່າຍກ່ອນ: Wave 1 ຖືກເລືອກໃຫ້ເດີມພັນຕ່າງໂດຍເຈດຕະນາ (ຮຽນເຄືອງຈັກໃນບ່ອນທີ່ຄວາມຜິດພາດຖືກ), ຄືນຕໍ່ມາຈົ່ງເອົາເພັດຍອດມົງກຸດ ຜ່ອມການ cutover ທີ່ຊ້ອມມາແລ້ວ, ແຕ່ລະຄືນມີແຜນ rollback ແລະ ຊ່ວງ **hypercare** ຂອງການເປົ່າຕິດຕາມເຂັ້ມຊັ້ນ. ສຳລັບການຍ້າຍ database ແຕ່ລະຄັ້ງ, ສອງຄຳຖາມທີ່ສຳຄັນຄືຕິວເລກຂອງ Module 4 ໃນຄາບແປງໂສມ: ການ cutover ໃຊ້ downtime ໄດ້ຫຼາຍປານໃດ (RTO) — ແລະ replication ຕໍ່ເນື່ອງຜ່ອມການສະຫຼັບສັນຍາ ມີໄວ້ສຳລັບເມື່ອຄຳຕອບຄື “ເກືອບບໍ່ໄດ້ເລີຍ.”

ຄຳສັບ:

ຄຳສັບ	ຄວາມໝາຍ
7 Rs	Retire, retain, rehost, relocate, repurchase, replatform, refactor — ເມນູ migration ຕໍ່ workload. (ທ່ານຈະໄດ້ຍິນ “6 Rs” ນຳ — ລາຍການເກົ່າທີ່ບໍ່ມີ relocate.)
Discovery / application inventory	ການຄົ້ນຫາວ່າຫຍັງແລ່ນຢູ່ແທ້ (ເຄືອງມື + ການສຳພາດ) / ລາຍການທີ່ໄດ້ອອກມາ, ໜຶ່ງແຖວຕໍ່ workload, ຜ່ອມເຈົ້າຂອງ, dependency, ແລະ R ຂອງມັນ.
Dependency mapping	ການແຕ້ມແຜນວ່າອັນໃດເວົ້າກັບອັນໃດ — ເຫດຜົນທີ່ workload ຍ້າຍເປັນກຸ່ມ, ແລະ migration ທີ່ບໍ່ມີມັນລົ້ມເຫຼວຕັ້ງແຕ່ມີທຳອິດ.
Business case / migration bubble	ຂໍ້ໄດ້ແຍ້ງ TCO ຢູ່ຕໍ່-ທຽບ-ຍ້າຍ / ໂນນຕົນທຶນຊົ່ວຄາວຂະນະທັງສອງກອງມູນແລ່ນ. ເຕືອນເລືອງມັນ ຫຼື ຖືກມັນຊຸມໂຈມຕີ.
Landing zone	ພື້ນຖານ cloud ທີ່ຖືກກຳກັບ ແລະ ສ້າງໄວ້ລ່ວງໜ້າ — ບັນຊີ, ຕິວຕິນ, ເຄືອຂ່າຍ, logging, guardrail — ທີ່ທຸກ workload ສືບທອດ. ສ້າງ <i>ກ່ອນ</i> migration.

ຄຳສັບ	ຄວາມໝາຍ
Multi-account strategy	ບັນຊີ cloud ແຍກຕໍ່ສະພາບແວດລ້ອມ/ທີມ, ເພື່ອໃຫ້ການເກັບເງິນລະບຸທີມໄດ້ ແລະ blast radius ຖືກຈຳກັດ.
Guardrail	Policy ອັດຕະໂນມັດທີ່ປ້ອງກັນ ຫຼື ໝາຍທາງການກະທຳທີ່ບໍ່ສອດຄ່ອງ — ການກຳກັບໃນຮູບເຄື່ອງຈັກ, ບໍ່ແມ່ນໃນຮູບບົດບັນທຶກ.
Control Tower	ບໍລິການ managed ຂອງ AWS ສຳລັບຕັ້ງ landing zone ຫຼາຍບັນຊີ ພ້ອມ guardrail.
Wave plan	ຕາຕະລາງ migration ເປັນກຸ່ມນ້ອຍ, ງ່າຍກ່ອນ, dependency ໄປພ້ອມກັນ.
Cutover / rollback / hypercare	ຊ່ວງເວລາທີ່ສະຫຼັບໄປໃຊ້ສຳເນົາເທິງ cloud / ການຍ້ອນຄືນທີ່ຊ້ອມໄວ້ / ໜ້າຕ່າງການສະໜັບສະໜູນເຂັ້ມຊັ້ນ ຫຼັງຈາກນັ້ນ.

ບົດຝຶກຫັດ: (1) Migration ເທິງໄຊ້ຍ: AI ສ້າງບັນຊີລາຍການບໍລິສັດສົມມຸດ 40 server (ທ່ານຈະພົບມັນອີກໃນຖານະ Capstone 2); ກຳນົດ R ໃຫ້ທຸກແຖວ ແລ້ວປົກປ້ອງສິບການຕັດສິນໃຈທີ່ຍາກທີ່ສຸດ. (2) ຮ່າງ landing zone: ແຜນວາດບັນຊີ, ກະແສຕົວຕົນ, ຮັບເຄືອຂ່າຍ hub-and-spoke, ຫ້າ guardrail ທີ່ທ່ານຈະບັງຄັບໃຊ້ຕັ້ງແຕ່ມື້ທຳອິດ. (3) ຂຽນຄຳເຕືອນ “migration bubble” ສອງວັກເຖິງ CFO — ການຝຶກແຈ້ງຂ່າວຮ້າຍແຕ່ຫົວທີ່ ຄືການອອກກຳລັງກາຍຂອງສະຖາປະນິກ.

Milestone: ເມື່ອໄດ້ບັນຊີລາຍການ ~20 workload ພ້ອມຄຳບັນຍາຍ, ທ່ານຜະລິດຕາຕະລາງ R-ຕໍ່-workload ທີ່ປົກປ້ອງໄດ້, ແຜນສາມຄືນພ້ອມເຫດຜົນ, ແລະ ຮ່າງ landing zone — ໃນການນັ່ງເທື່ອດຽວ.

Module 11 (ອາທິດທີ 24–26): ຂໍ້ຈຳກັດ — ກົດໝາຍ, ລະບົບເກົາ, ແລະ lock-in

Data residency ແລະ ກົດໝາຍຄວາມເປັນສ່ວນຕົວ ໃນຖານະຂໍ້ມູນນຳເຂົ້າຂອງສະຖາປັດຕະຍະກຳ. ກົດໝາຍຂໍ້ມູນສ່ວນບຸກຄົນ — PDPA ຂອງໄທ, GDPR ຂອງເອີຣົບ, ແລະ ຍາດພີ່ນ້ອງຂອງພວກມັນທົ່ວໂລກ — ມີຮູບຮ່າງຮ່ວມກັນທີ່ສະຖາປະນິກຕ້ອງແມ່ນຢ່າ: ຂໍ້ມູນສ່ວນບຸກຄົນຕ້ອງມີຖານທາງກົດໝາຍ (ມັກເປັນການຍິນຍອມ); ບຸກຄົນມີສິດ (ເຂົ້າເຖິງ, ແກ້ໄຂ, ລຶບ — ການອອກແບບຂອງທ່ານຕ້ອງ ຫາ ແລະ ລຶບຂໍ້ມູນຂອງຄົນໜຶ່ງຄົນ ໄດ້, ຊຶ່ງຍາກຖ້າທ່ານກະຈາຍມັນໄປຕາມສຳເນົາທີ່ບໍ່ມີການກຳກັບ); ການລະເມີດມີໜ້າທີ່ ແຈ້ງພາຍໃນ 72 ຊົ່ວໂມງ (logging ຂອງທ່ານຕ້ອງໃຫ້ທ່ານເລົ່າເລື່ອງໄດ້ໃນເວລາໜ້ອຍກວ່ານັ້ນ); ແລະ ກົດການໂອນຂ້າມຊາຍແດນ ຈຳກັດວ່າຂໍ້ມູນຢູ່ບ່ອນໃດໄດ້ທາງກາຍະພາບ — ຊຶ່ງເປັນຂໍ້ຈຳກັດຂອງ ການເລືອກ region, ຂໍ້ຈຳກັດຂອງ ການອອກແບບ replication (ສຳເນົາວິເຄາະຢູ່ສິ່ງກະໂປອາດເປັນເຫດການທາງກົດໝາຍ), ແລະ ເປັນເຫດຜົນທີ່ region ພາຍໃນປະເທດມີຢູ່. ວິທີການຂອງສະຖາປະນິກ, ຕາມລຳດັບ: ຈັດປະເພດຂໍ້ມູນ (ອັນໃດເປັນຂໍ້ມູນສ່ວນບຸກຄົນ?), ແຕ້ມ

ແຜນການເດີນທາງຂອງມັນ (ທຸກບ່ອນເກັບ, ທຸກສຳເນົາ, ທຸກຊາຍແດນທີ່ຂ້າມ — ກະແສທີ່ບໍ່ມີໃຜແຕ້ມ ຄືບ່ອນທີ່ການລະເມີດອາໄສຢູ່), ແລ້ວອອກແບບການຄວບຄຸມ: region ທີ່ສອດຄ່ອງກັບ residency, ການເຂົ້າລະຫັດ, ຕາຕະລາງການເກັບຮັກສາ, ແລະ ເສັ້ນທາງການລຶບທີ່ໄປເຖິງວັນໝົດອາຍຸຂອງ backup ແທ້ໆ. ເວົ້າວ່າ “data protection by design” — ມັນຄືວະລິທີ່ກົດໝາຍທັງສອງໃຊ້, ແລະ ມັນຄືຕຳແໜ່ງງານຂອງທ່ານໃນຮູບປະໂຫຍກ.

ການບູລະນາການກັບລະບົບເກົ່າ (legacy). ລະບົບອອກໃບບິນເທິງ mainframe ຈະບໍ່ຍ້າຍໃນປີນີ້, ແລະ ແອັບ cloud ໃໝ່ເອີ້ນຕ້ອງເວົ້າກັບມັນ. Pattern: **anti-corruption layer** — service ແປພາສາລະຫວ່າງໃໝ່ ແລະ ເກົ່າ, ເພື່ອບໍ່ໃຫ້ຮູບຮ່າງຂໍ້ມູນແປກໆຂອງລະບົບເກົ່າຮົ່ວເຂົ້າມາທຳລາຍການອອກແບບໃໝ່ຂອງທ່ານ; **strangler fig** (ຕົ້ນໄຮລັດ) — ນຳ traffic ຜ່ານໜ້າກາກ, ແລ້ວປອກໜ້າທີ່ອອກຈາກລະບົບເກົ່າເທື່ອລະອັນ ຈົນອີກຫຼາຍປີຕໍ່ມາມັນຖືກປິດໄດ້ (ຕັ້ງຊື່ຕາມຕົ້ນໄຮທີ່ຄ່ອຍໆຫຸ້ມຕົ້ນໄມ້ເຈົ້າບ້ານ; ມັນເໝືອນວ່າການຂຽນໃໝ່ແບບ big-bang, ຊຶ່ງລົ້ມເຫຼວໃນອັດຕາທີ່ເປັນຕຳນານ); ຂົວແບບ batch ແລະ ຈຸດແຕະ event ສຳລັບຂໍ້ມູນທີ່ຕ້ອງໄຫຼລະຫວ່າງສອງໂລກ; ແລະ ຄວາມເຄົາລົບຕໍ່ຟີຊິກຂອງ latency ໃນການເອີ້ນ cloud-ຫາ-on-prem ທີ່ຄຸຍກັນຫຼາຍ (Direct Connect ຊ່ວຍ; ດຶກວ່ານັ້ນ, ອອກແບບເອົາການຄຸຍນັ້ນອອກໄປ). ກົດ: **ກັກລະບົບເກົ່າໄວ້, ຢາຕິດເຊືອຈາກມັນ** — ທຸກສ່ວນປະກອບໃໝ່ຄວນຖືກສ້າງຄືກັບວ່າລະບົບເກົ່າຫາຍໄປແລ້ວ.

ເສດຖະສາດຂອງ vendor lock-in. ທຸກບໍລິການ managed ທີ່ສະດວກ ເຮັດໃຫ້ການແຕ່ງງານກັບຜູ້ໃຫ້ບໍລິການລາຍດຽວເລິກຂຶ້ນ; ຄວາມເຄື່ອນຍ້າຍໄດ້ (abstraction ແບບ multi-cloud, ຄຸ້ມຄອງເອງທຸກຢ່າງ) ຄືທາງເລືອກຈິງທີ່ມີຕົ້ນທຶນຈິງໃນຮູບຄວາມຊັບຊ້ອນ ແລະ ຄວາມໄວທີ່ຕ້ອງເສຍໄປ. ບໍ່ມີອັນໃດເປັນບາບ — ຄວາມລົ້ມເຫຼວບໍ່ແມ່ນການ *ເລືອກ* lock-in, ແຕ່ຄືການບໍ່ *ສັງເກດເຫັນ* ມັນ. ເຄື່ອງມືຂອງສະຖາປະນິກຄືການປະເມີນຕົ້ນທຶນການອອກຂຽນໄວ້ໃນການອອກແບບ: “ການໃຊ້ DynamoDB ປະຢັດ ≈2 ປີ-ວິສະວະກອນ ຕອນນີ້; ການປ່ຽນພາຍຫຼັງ ≈ ການຂຽນຊັ້ນຂໍ້ມູນໃໝ່ 6 ເດືອນ — ພວກເຮົາຍອມຮັບສິ່ງນີ້, ແລະ ນີ້ຄືຂອບເຂດ interface ທີ່ຈະຫຍັດການຂຽນໃໝ່ນັ້ນ.” ສາມປະໂຫຍກ, ຕິລາຄາຢ່າງຊື່ສັດ, ບັນທຶກການຕັດສິນໃຈໄວ້ (ADR ໃນໄລຍະທີ 5) — ນັ້ນຄືການຈັດການ lock-in ແບບຜູ້ໃຫຍ່. Multi-cloud *ໃນຖານະຍຸດທະສາດ* (ແລ່ນ workload ອັນດຽວກັນແບບເຄື່ອນຍ້າຍໄດ້ເທິງສອງ cloud) ປົກກະຕິຄືຄຳຕອບທີ່ແພງທີ່ສຸດເທົ່າທີ່ເປັນໄປໄດ້ ແລະ ຖືກຊື້ໂດຍອົງກອນທີ່ຂະໜາດ ຫຼື ຜູ້ຄຸມກົດຮຽກຮ້ອງເທົ່ານັ້ນ; multi-cloud *ໃນຖານະຂັ້ນຕົ້ນຈິງ* (workload ຕ່າງກັນຢູ່ cloud ຕ່າງກັນ ຕາມປະຫວັດສາດ ຫຼື ຄວາມເໝາະສົມ) ຄືຊີວິດປົກກະຕິ.

ຄຳສັບ:

ຄຳສັບ	ຄວາມໝາຍ
PDPA / GDPR	ກົດໝາຍປົກປ້ອງຂໍ້ມູນສ່ວນບຸກຄົນຂອງໄທ ແລະ ເອີຣົບ — ຄູ່ແມ່ແບບຂອງຂໍ້ຈຳກັດດ້ານກົດໝາຍຄວາມເປັນສ່ວນຕົວຫົວໂລກ.
Data protection by design	ການສ້າງການຄວບຄຸມຄວາມເປັນສ່ວນຕົວເຂົ້າໃນສະຖາປັດຕະຍະກຳຕັ້ງແຕ່ຮ່າງທຳອິດ — ວະລິທາງກົດໝາຍ ແລະ ໜ້າທີ່ຂອງສະຖາປະນິກ.

ຄຳສັບ	ຄວາມໝາຍ
Lawful basis / consent	ເຫດຜົນທາງກົດໝາຍທີ່ຈຳເປັນສຳລັບການປະມວນຜົນຂໍ້ມູນສ່ວນບຸກຄົນ.
Right to erasure	ສິດຂອງບຸກຄົນທີ່ຈະໃຫ້ລຶບຂໍ້ມູນຂອງຕົນ — ຊຶ່ງການອອກແບບຂອງທ່ານຕ້ອງເຮັດໃຫ້ <i>ເປັນໄປໄດ້</i> .
Data flow mapping	ການແຕ້ມແຜນທຸກບ່ອນເກັບ, ທຸກສຳເນົາ, ແລະ ທຸກການຂ້າມຊາຍແດນ ຂອງຂໍ້ມູນໝວດໜຶ່ງ. ບ່ອນທີ່ກະແສບໍ່ຖືກແຕ້ມ, ການລະເມີດອາໄສຢູ່.
Cross-border transfer	ຂໍ້ມູນສ່ວນບຸກຄົນທີ່ອອກນອກປະເທດ — ຖືກກຳກັບ; ເຮັດໃຫ້ການອອກແບບ replication ກາຍເປັນຄຳຖາມທາງກົດໝາຍ.
Anti-corruption layer	ສ່ວນປະກອບແປພາສາ ທີ່ກັນຄວາມແປກຂອງລະບົບເກົ່າ ບໍ່ໃຫ້ຮົວເຂົ້າການອອກແບບໃໝ່.
Strangler fig	ການປັບປຸງໃຫ້ທັນສະໄໝ ໂດຍນຳ traffic ຜ່ານໜ້າກາກແລ້ວປ່ຽນລະບົບເກົ່າເທື່ອລະຊິນ ຈົນປົດມັນໄດ້.
Big-bang rewrite	ການປ່ຽນລະບົບທັງໝົດຜ່ອນກັນ, ສະໜັບໃນມື້ດຽວ. ອັດຕາລົ້ມເຫຼວເປັນຕຳນານ; strangler fig ມີຢູ່ກໍເພາະສິ່ງນີ້.
Exit cost / switching cost	ຕົ້ນທຶນທີ່ຕິດຄ່າຢ່າງສົມຈິງຂອງການອອກຈາກຜູ້ໃຫ້ບໍລິການ ຫຼື ບໍລິການໜຶ່ງ — ຕົວເລກທີ່ປ່ຽນ lock-in ຈາກຄວາມຢ້ານ ໃຫ້ເປັນເສດຖະສາດ.
Multi-cloud (ຍຸດທະສາດ ທຽບ ຂໍ້ເທັດຈິງ)	ການແລ່ນແບບເຄື່ອນຍ້າຍໄດ້ເທິງຫຼາຍ cloud ໂດຍເຈດຕະນາ (ແພງ, ຫາເຫດຜົນຮອງຮັບໄດ້ຍາກ) ທຽບກັບການມີ workload ຢູ່ຫຼາຍ cloud ເສີຍໆ (ປົກກະຕິ).

ບົດຝຶກຫັດ: (1) ແຕ້ມແຜນກະແສຂໍ້ມູນລູກຄ້າຂອງ ThaiTicket: ທຸກບ່ອນເກັບ, ທຸກສຳເນົາ (ຢ່າລືມ log, backup, pipeline ວິເຄາະ, ໄຟລ໌ export ຂອງທີມສະໜັບສະໜູນ), ທຸກຊາຍແດນ; ຈາກນັ້ນຂຽນເສັ້ນທາງການລຶບ. ຮູ້ສຶກເບິ່ງວ່າແຜນທີ່ຫາບັນຫາທີ່ແຜນວາດເຊື່ອງໄວ້ໄດ້ແນວໃດ. (2) ອອກແບບແຜນ strangler fig ໃຫ້ລະບົບຄຸ້ມຄອງສາງ on-prem ອາຍຸ 20 ປີ, ບອກຊື່ສາມການປອກທຳອິດ. (3) ຂຽນວັກ lock-in ແບບ DynamoDB (ປະໂຫຍດຕອນນີ້, ຕົ້ນທຶນການອອກພາຍຫຼັງ, ຍອມຮັບ ຫຼື ຫຼຸດຜ່ອນ) ສຳລັບສາມບໍລິການທີ່ທ່ານຈະໃຊ້ຈິງ.

Milestone — ຈົບໄລຍະທີ 4: ຕ່ານທົດສອບຂໍ້ຈຳກັດ: ໂຈດການອອກແບບອັນດຽວທີ່ຖືກທໍ້ທັງສາມເຂົ້າກັນ (“ບໍລິສັດປະກັນໄພໄທ, ຂໍ້ມູນລູກຄ້າ, ລະບົບກົມຫະບຽນເທິງ mainframe, ຄະນະກຳມະການກົງວິນເລືອງການ

ເພິງພາ AWS”) — ທ່ານຜະລິດແນວທາງໜຶ່ງໜ້າ ທີ່ແຕະທັງ residency, ການບູລະນາການ, ແລະ lock-in, ແຕ່ລະອັນມີ pattern ທີ່ບອກຊິໄດ້ ແລະ ຕົ້ນທຶນທີ່ຊື່ສັດ. ເມື່ອຂ້າຈຳກັດເຮັດໃຫ້ທ່ານຕື່ນເຕັ້ນຫຼາຍກວ່າ greenfield — ເພາະຂ້າຈຳກັດຄືບ່ອນທີ່ສະຖາປະນິກຫາໄດ້ຫຼາຍກວ່າຄົນແຕ້ມແຜນວາດ — ໄລຍະທີ 4 ຈົບແລ້ວ.

ໄລຍະທີ 5 — ຝົນວິຊາຊີບຂອງສະຖາປະນິກ (ເດືອນທີ 7–12)

ທຸກສິ່ງມາຮອດຕອນນີ້ ເຮັດໃຫ້ທ່ານ *ອອກແບບ* ໄດ້. ໄລຍະນີ້ເຮັດໃຫ້ທ່ານ *ເຮັດວຽກໃນຖານະສະຖາປະນິກ* ໄດ້ — ເອກະສານ, ການທົບທວນ, ການນຳສະເໜີ, ແລະ ຫຼັກຖານ ທີ່ບົດບາດນີ້ປະກອບຂຶ້ນຈິງ, ບວກກັບ certification ທີ່ພາທ່ານຜ່ານ HR, ແລະ capstone ສາມອັນທີ່ກາຍເປັນ portfolio ຂອງທ່ານ.

Module 12 (ເດືອນທີ 7–8): ຜົນງານສົ່ງມອບ — ADR, ຊຸດແຜນວາດ, ແລະ ການດຳເນີນການທົບທວນ

Architecture Decision Records (ADRs). ADR ຄືເອກະສານໜຶ່ງໜ້າທີ່ຈັບການຕັດສິນໃຈສຳຄັນອັນໜຶ່ງໄວ້: ພວກເຮົາເລືອກຫຍັງ, ບໍ່ເລືອກຫຍັງ, ແລະ ເປັນຫຍັງ — ຂຽນ *ຕອນທີ່ຕັດສິນໃຈ*, ໃສ່ໝາຍເລກ, ແລະ ເກັບໄວ້ໃນ repository ຂອງໂຄງການຕະຫຼອດໄປ. ເປັນຫຍັງປະກາດວຽກຈິງເອີ້ນຊື່ມັນ: ອີກສອງປີ, ຈະມີຄົນຖາມວ່າ “ເປັນຫຍັງອັນນີ້ຈຶ່ງເປັນ DynamoDB?” ແລະ ADR ຕອບໄດ້ໃນສາມສິບວິນາທີ — ພ້ອມບໍລິບົດ, ຂ້າຈຳກັດ, ແລະ ທາງເລືອກທີ່ຖືກພິຈາລະນາຢ່າງຊື່ສັດ. ທີ່ມີ ADR ບໍ່ຕ້ອງຮີດະດີເກົ່າມາຖຽງໃໝ່; ທີ່ມີບໍ່ມີ ຖຽງເປັນວົງກົມທຸກປີ. ແມ່ແບບ — ຈິໃຫ້ຂຶ້ນໃຈ:

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.

ຂຽນ ADR ໜຶ່ງສະບັບຕໍ່ໜຶ່ງການຕັດສິນໃຈສຳຄັນ ໃນທຸກ capstone. Portfolio ສຳພາດທີ່ມີ ADR ຈິ່ງ ຄືສິ່ງທີ່ຫາຍາກ ແລະ ມີພະລັງຮ້າຍກາດ.

ຊຸດແຜນວາດ — ໝຶງລະບົບ, ສາມລະດັບ zoom. ເອກະສານສະຖາປັດຕະຍະກຳຕົວຈິງຄື *ຊຸດ* (ແບບຈຳລອງ C4 ເປັນຜູ້ເຮັດໃຫ້ວິໄນນີ້ແພ່ຫຼາຍ): **context diagram** — ລະບົບເປັນກ່ອງດຽວ, ພ້ອມຜູ້ໃຊ້ ແລະ ລະບົບພາຍນອກ ທີ່ມັນສຳຜັດ; ມຸມມອງສຳລັບຜູ້ບໍລິຫານ/ຄົນເຂົ້າໃໝ່, ແລະ ເປັນອັນທີ່ທ່ານຈະນຳສະເໜີຕໍ່ຜູ້ນຳ. **Container diagram** — ຊຸມເຂົ້າ: ຂົນສ່ວນຫຼັກທີ່ແລ່ນຢູ່ (ເວັບແອັບ, API, database, queue, cache) ແລະ ວິທີທີ່ພວກມັນ ເວົ້າກັນ; ມຸມມອງວິສະວະກຳປະຈຳວັນ, ປະມານແຜນວາດຂອງທ່ານໃນໄລຍະທີ 1–3. **Deployment diagram** — ທຸກຢ່າງແລ່ນຢູ່ໃສ່ທາງກາຍະພາບ: region, AZ, VPC, subnet, scaling group; ມຸມມອງສຳລັບການທົບ ທວນ, ຄວາມປອດໄພ, ແລະ ປະຕິບັດການ. ໝຶງລະບົບ, ສາມກຸ່ມຜູ້ຟັງ, ສາມແຜນວາດ — ຮັກສາໃຫ້ທັນສະໄໝ, ໃສ່ວັນທີ, ຢູ່ໃນ version control ຂ້າງ ADR. ແຜນວາດທີ່ຫາບໍ່ພົບ ຫຼື ເຊື່ອບໍ່ໄດ້ ຄືແຜນວາດທີ່ບໍ່ມີຢູ່.

ການດຳເນີນ Well-Architected review. ທ່ານຮູ້ຫົກເສົາແລ້ວ (ໄລຍະທີ 2); ນິຕິຕົວກອງປະຊຸມເອງ. **ກ່ອນ:** ເລືອກ workload ແລະ ຂອບເຂດ, ເຮັດຊຸດແຜນວາດໃຫ້ທັນສະໄໝ, ເຊີນຄົນທີ່ *ປະຕິບັດງານ* ສິ່ງນັ້ນ (ບໍ່ແມ່ນ ສະເພາະຄົນອອກແບບ), ແລະ ວາງໂທນ — ນິຕິການກວດສຸຂະພາບເພື່ອປະໂຫຍດຂອງທີມ, ບໍ່ແມ່ນການກວດ ສອບເພື່ອເຂົ້າແຟ້ມໃຜ. **ລະຫວ່າງ (ເຄິ່ງມື):** ເດີນຜ່ານເສົາຫຼັກດ້ວຍຊຸດຄຳຖາມຂອງ framework (AWS ຕີພິມ ພວກມັນ, ພ້ອມ Well-Architected Tool ຟຣີໃນ console ທີ່ຈັດໂຄງສ້າງໃຫ້ທັງບົດຝຶກ — <https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html>); ສຳລັບແຕ່ລະຄຳ ຕອບ, ບັນທຶກສິ່ງທີ່ພົບ, ແລະ *ຈັດລຳດັບຄວາມສຳຄັນໄປພ້ອມກັນ* — ບັນຫາຄວາມສ່ຽງສູງບໍ່ຫຼາຍຂີ້, ບໍ່ແມ່ນຂີ້ຈຸກ ຈິກຮ້ອຍຂີ້. ໂທນຄຳຖາມຂອງທ່ານຄືຕົວການທົບທວນເອງ: “ຊ່ວຍໃຫ້ຂ້ອຍເຂົ້າໃຈວ່າເກີດຫຍັງເມື່ອຜູ້ໃຫ້ບໍລິການ ຂຳລະເງິນ timeout” ເປີດປະຕູ ຂະນະທີ່ “ພວກເຈົ້າບໍ່ຈັດການ timeout ບໍ?” ເປີດປະຕູດັ່ງປັ້ງ. **ຫຼັງ:** ບົດລາຍງານສັ້ນ — ສິ່ງທີ່ພົບອັນດັບຕົ້ນ, ແຕ່ລະອັນມີຄວາມສ່ຽງ, ແຮງງານ, ແລະ ຄຳແນະນຳ — ແລະ ລາຍການປັບປຸງຕ້ອງລົງໄປ ໃນ backlog ຈິງຂອງທີມ ພ້ອມເຈົ້າຂອງ, ບໍ່ດັ່ງນັ້ນການທົບທວນກໍເປັນລະຄອນ. ຝຶກ: ດຳເນີນການທົບທວນເຕັມ ຮູບແບບຕໍ່ການອອກແບບ Capstone 1 ຂອງທ່ານເອງ; ການພົບຂໍ້ບົກຜ່ອງຂອງຕົນເອງຢ່າງເປັນລະບົບ ຄືທັກສະທີ່ ທົບຕົ້ນທົບຕອກໄວທີ່ສຸດໃນຫຼັກສູດນີ້.

ບົດຝຶກຫັດ: (1) ຂຽນ ADR ຍ້ອນຫຼັງໃຫ້ການຕັດສິນໃຈໃຫຍ່ຫ້າອັນ ໃນການອອກແບບຈາກປຶ້ມບັນທຶກໄລຍະທີ 2–3 ຂອງທ່ານ — ທ່ານຈະພົບຢ່າງໜ້ອຍໜຶ່ງອັນທີ່ຕອນນີ້ທ່ານປົກປ້ອງບໍ່ໄດ້ອີກແລ້ວ, ຊຶ່ງນັ້ນເອງຄືບົດຮຽນ. (2) ຜະລິດຊຸດແຜນວາດສາມອັນຄົບຊຸດສຳລັບ ThaiTicket. (3) ດຳເນີນການທົບທວນຈຳລອງຂ້າງເທິງ, ຜະລິດບົດ ລາຍງານສິ່ງທີ່ພົບໜຶ່ງໜ້າ.

Milestone: ຄົນແປກໜ້າ (ຫຼື AI ທີ່ສະແດງເປັນຄົນແປກໜ້າ) ຢືບຊຸດ ThaiTicket ຂອງທ່ານຂຶ້ນມາ — ສາມ ແຜນວາດ + ຫ້າ ADR — ແລ້ວຕອບໄດ້ຢ່າງຖືກຕ້ອງວ່າ “ນີ້ແມ່ນຫຍັງ, ມັນແລ່ນແນວໃດ, ແລະ ເປັນຫຍັງຈຶ່ງສ້າງ ແບບນີ້?” ໂດຍທ່ານບໍ່ຢູ່ໃນຫ້ອງ. ຊຸດນັ້ນ ຄືຜົນງານສົ່ງມອບຂອງວຽກສະຖາປະນິກ.

Module 13 (ເດືອນທີ 8–9): ຝຶມ້ານມະນຸດ — ການນຳສະເໜີ, ການຄິດຕົ້ນຫຼິ້ນ, ການ ຮັບແຮງຕ້ານ, ແລະ ການເປັນພິລັງ

ການນຳສະເໜີຕໍ່ຜູ້ບໍລິຫານ ທຽບກັບວິສະວະກອນ — ໝຶງການອອກແບບ, ສອງພາສາ. ຜູ້ບໍລິຫານຊື່ຜົນລັບ; ວິສະວະກອນຊື່ກົນໄກ. ຕໍ່ຜູ້ບໍລິຫານ: ນຳດ້ວຍການຕັດສິນໃຈ ແລະ ຕົວເລກ (“ການອອກແບບນີ້ຮອງຮັບເປົ້າໝາຍ

ລູກຄ້າໜຶ່ງລ້ານຄົນ ທີ່ \$1.90 ຕໍ່ຄໍາສັ່ງຊື້, ແລະ ນິຕິສອງການຕັດສິນໃຈທີ່ຂ້ອຍຕ້ອງການຈາກທ່ານ”); ສະໄລດ໌ດຽວ, ໃຊ້ແຜນວາດ context, ນໍາສະເໜີຄວາມສ່ຽງໃນຮູບຄວາມສ່ຽງທາງທຸລະກິດ (ລາຍຮັບ, compliance, ຊື່ສຽງ), ແລະ ຢ່າເວົ້າຄຳວ່າ “Kubernetes” ເມື່ອຄຳວ່າ “ແຜລດຟອມ” ພຽງພໍ. ກຽມພ້ອມສໍາລັບສາມຄໍາຖາມທີ່ຜູ້ບໍລິຫານຖາມສະເໝີ: ມີຄ່າໃຊ້ຈ່າຍເທົ່າໃດ, ຫຍັງອາດຜິດພາດ, ເປັນຫຍັງບໍ່ເອົາທາງເລືອກທີ່ຖືກກວ່າ? ຕໍ່ວິສະວະກອນ: ນໍາດ້ວຍບັນຫາ ແລະ ຂໍ້ຈຳກັດ ກ່ອນ ວິທີແກ້ (ວິສະວະກອນທີ່ຮູ້ສຶກເຖິງບັນຫາ ຈະຍອມຮັບວິທີແກ້; ວິສະວະກອນທີ່ຖືກຍືນຄໍາຕັດສິນໃສ່ມື ຈະລ່າຂັບກຜ່ອງເປັນຫຼັກການ), ສະແດງແຜນວາດ container ແລະ deployment, ບອກຊື່ trade-off ແລະ ທາງເລືອກທີ່ຖືກປະຕິເສດດ້ວຍຕົນເອງ (ຄວາມໜ້າເຊື່ອຖືມາຈາກສິ່ງທີ່ທ່ານຍອມຮັບ), ແລະ ເປີດຊ່ອງຈິງໆໃຫ້ປ່ຽນການອອກແບບ — ຄໍາຖາມໃນຫ້ອງຄືການທົບທວນຟຣີ. ນິໄສທີ່ຮ້າຍທີ່ສຸດຂອງສະຖາປະນິກຄື ໃຊ້ສະໄລດ໌ຊຸດດຽວກັບຜູ້ຟັງທັງສອງກຸ່ມ; ນິໄສທີ່ດີທີ່ສຸດຄືຂຽນບົດສະຫຼຸບສໍາລັບຜູ້ບໍລິຫານ ກ່ອນ, ເພາະຖ້າການອອກແບບຢູ່ລອດການບົບອັດເປັນຫ້າປະໂຫຍກບໍ່ໄດ້, ມັນຍັງບໍ່ສໍາເລັດ.

ການປະເມີນຕົ້ນທຶນສໍາລັບຂໍ້ສະເໜີ. ວິທີການ: ແຍກການອອກແບບເປັນສ່ວນປະກອບທີ່ມີຕົ້ນທຶນ ~10 ອັນ; ຕີລາຄາແຕ່ລະອັນທີ່ໂຫຼດຄາດໝາຍ ໃນ pricing calculator; **ລະບຸສົມມຸດຕິຖານຂອງທ່ານເປັນລາຍລັກອັກສອນ** (ຄໍາຮ້ອງຂໍ/ມື້, ການເຕີບໂຕຂອງຂໍ້ມູນ, egress — ສົມມຸດຕິຖານຄິດໄລ່ປະເມີນ; ເມື່ອພວກມັນປ່ຽນ, ຕົວເລກກໍຂຶ້ນລົງດ້ວຍຈິດໃຈທີ່ສະບາຍ); ໃຊ້ຍຸດທະສາດການຈອງກັບສ່ວນທີ່ຄົງທີ່; ເພີ່ມສະຖານະການການເຕີບໂຕ (ມື້ນີ້, 2x, 10x — ຜູ້ບໍລິຫານຈິຕິວເລກ 10x); ແລະ ນໍາສະເໜີເປັນ *ຊ່ວງພ້ອມຕົວຂັບເຄື່ອນ* (“\$55–70k/ເດືອນ, ຂັບເຄື່ອນຫຼັກໂດຍ egress — ນິຕິຄັນໂຍກ”), ບໍ່ແມ່ນຕົວເລກດ່ຽວທີ່ແມ່ນຢ່າແບບຫຼອກ. ໃສ່ migration bubble ເຂົ້າໄປນໍາ ຖ້າມີ. ການປະເມີນຕໍາເລີຍໃຫ້ໄດ້ຮັບອະນຸມັດ ຄືບາບຄລາສສິກຂອງສະຖາປະນິກມືໃໝ່; ໂຄງການຈື່ໄດ້.

ການຮັບມືກັບແຮງຕ້ານ. ທ່ານຈະຖືກຍູ້ເລື່ອງຕົ້ນທຶນ (“ແພງເກີນ” → ກັບໄປຫາສາມຫຼ່ຽມ: “ພວກເຮົາຕັດ \$30k ໄດ້ຖ້າຍອມຮັບ single-AZ — ນິຕິເລກຄະນິດຂອງການຂັດຂ້ອງ; ທ່ານເປັນຜູ້ຕັດສິນ, ແລະ ຂ້ອຍຈະບັນທຶກໄວ້” — ການເຮັດໃຫ້ການແລກປ່ຽນຈະແຈ້ງ ແລະ *ຖືກບັນທຶກ* ປ່ຽນແຮງຕ້ານສ່ວນໃຫຍ່ໃຫ້ເປັນການເຫັນດີ ຫຼື ຄວາມສ່ຽງທີ່ຮັບຮູ້ແລ້ວຍອມຮັບ, ຊຶ່ງທັງສອງກໍດີ); ເລື່ອງລົດນິຍົມ (“ວິສະວະກອນມັກ stack ອື່ນ” → steel-man ມັນອອກສຽງ, ແລ້ວເອົາໄປສູ່ເກນ, ບໍ່ແມ່ນຄວາມມັກ: ຄວາມເໝາະສົມກັບ NFR, ທັກສະທິມ, ລະບົບນິເວດ, ຕົ້ນທຶນການອອກ — ແລະ ເມື່ອມັນສຸສິກັນແທ້, ໃຫ້ຄວາມມັກຂອງວິສະວະກອນຜູ້ລົງມືເປັນຝ່າຍຊະນະ, ຊື່ຄວາມມຸ່ງໝັ້ນມາຢ່າງຖືກ); ແລະ ເລື່ອງອໍານາດ (“ໝູ່ຂອງ CTO ບອກໃຫ້ໃຊ້ X” → ຢ່າສູ້ຄວາມເຫັນດ້ວຍຄວາມເຫັນ; ຂໍເກນມາ, ເອົາ X ຜ່ານການປະເມີນສາທາລະນະອັນດຽວກັນກັບທຸກຢ່າງ, ປ່ອຍໃຫ້ຕາຕະລາງເປັນຜູ້ຕອບຢ່າງສຸພາບ). ແລະ ເມື່ອທ່ານຜິດ — ທ່ານຈະອອກແບບບາງສິ່ງທີ່ລົ້ມເຫຼວ; ສະຖາປະນິກທຸກຄົນເປັນ — ທ່າທີຄື blameless post-mortem ນໍາໃຊ້ກັບຕົນເອງ, ຢູ່ທີ່ແຈ້ງ, ພ້ອມອັບເດດ ADR. ບໍ່ມີຫຍັງສ້າງຄວາມໜ້າເຊື່ອຖືລະດັບທົດສະວັດໄດ້ໄວກວ່ານີ້; ແລະ ບໍ່ມີຫຍັງທໍາລາຍມັນໄວກວ່າການປົກປ້ອງສິບ.

ການເປັນພີລິດວິສະວະກອນ. ວະລີໃນປະກາດວຽກຄື “mentor the engineers who build against your standards,” ແລະ ຮູບແບບການປະຕິບັດຄື: **ຊົ່ວໂມງໃຫ້ຄໍາປຶກສາການທົບທວນການອອກແບບ** (ປະຕູທີ່ເປີດປະຈຳ, ເພື່ອໃຫ້ຄໍາແນະນໍາເກີດກ່ອນໂຄດ, ບໍ່ແມ່ນຫຼັງ); **ການທົບທວນການອອກແບບຂອງເຂົາຢ່າງເມດຕາ** — ຄໍາຖາມກ່ອນຄໍາຕັດສິນ, ແລະ ສະເໜີຕ້ອງມີ *ເປັນຫຍັງ* ຢູ່ເບື້ອງຫຼັງມາດຕະຖານ (guardrail ທີ່ອະທິບາຍ ໄດ້ພັນທະມິດ; guardrail ທີ່ບັງຄັບ ໄດ້ວິທີຫຼົບຫຼີກ); **ການມອບການຕັດສິນໃຈຈິງ** ພ້ອມຕາໜ່າງນິລະໄພ (“ເຈົ້າເປັນເຈົ້າຂອງການອອກແບບ cache; ນິຕິຂໍ້ຈຳກັດ; ຂ້ອຍຈະທົບທວນ, ແລະ ຂ້ອຍຈະໝູນຫຼັງການຕັດສິນໃຈຂອງເຈົ້າ”); ແລະ **ການສອນຢູ່ທີ່ແຈ້ງ** — ທຸກ ADR, ທຸກບົດສະຫຼຸບການທົບທວນ, ທຸກກອງປະຊຸມແລກປ່ຽນຕອນທ່ຽງ

ຂະຫຍາຍທ່ານອອກໄປເກີນຊົ່ວໂມງຂອງທ່ານເອງ. ຄວາມຈິງກົງໆເລືອງອາຊີບ: ສະຖາປະນິກອັດສະລິຍະ ໂດດດ່ຽວມີເພດານ; ສະຖາປະນິກທີ່ບໍ່ມີວິສະວະກອນທ້າຄົນໃຫ້ກາຍເປັນເພື່ອນຮ່ວມງານທີ່ອອກແບບໄດ້ ຄືຜູ້ ດຳເນີນວິຊາຊີບນີ້.

ບົດຝຶກຫັດ: (1) ນຳສະເໜີ ThaiTicket ສອງເທື່ອ — ສະບັບຜູ້ບໍລິຫານ 5 ນາທີ ແລະ ສະບັບວິສະວະກຳ 20 ນາທີ — ບັນທຶກທັງສອງ, ແລ້ວຟັງຫາສັບແສງທີ່ຮິວເຂົ້າສະບັບທຳອິດ. (2) ຜະລິດຂໍ້ສະເໜີຕົ້ນທຶນເປັນລາຍລັກອັກສອນ ຄົບຖ້ວນ (ສົມມຸດຕິຖານ, ຊ່ວງ, ຕົວຂັບເຄື່ອນ, ສະຖານະການ $2 \times 10 \times$). (3) ລະຄອນແຮງຕ້ານກັບ AI: ສາມຮອບ — ການໃຈມຕິເລືອງຕົ້ນທຶນຈາກ CFO, ຄວາມມັກເລືອງ stack ຈາກວິສະວະກອນອາວຸໂສ, ການອ້າງອຳນາດແບບ “ໝູ່ຂອງ CTO” — ບັນທຶກວ່າຫຍັງໃຊ້ໄດ້ຜົນ. (4) ທົບທວນການອອກແບບຂອງນ້ອງໃໝ່ (AI ສ້າງອັນທີ່ມີຂໍ້ບົກຜ່ອງ ໄດ້) ເປັນລາຍລັກອັກສອນ, ຄຳຖາມມາກ່ອນ, ແລ້ວໃຫ້ AI ໃຫ້ຄະແນນໂທນຂອງທ່ານ.

Milestone: ບົດຄູ່: ນຳສະເໜີ pitch ຕໍ່ຜູ້ບໍລິຫານ ແລ້ວຢູ່ລອດການຖາມ-ຕອບແບບປະສົມທິດແຕ່ຍຸຕິທຳຊາວນາທີ່ (ຄະນະ AI: CFO ໝຶ່ງຄົນ, ວິສະວະກອນຂີ່ສິ່ງໄສໜຶ່ງຄົນ) ໂດຍບໍ່ຂ້າມສາຍສັບແສງ, ບໍ່ຕັ້ງທ່າປ້ອງກັນຕົວ, ແລະ ບໍ່ມີ trade-off ອັນໃດທີ່ບໍ່ໄດ້ຂຽນໄວ້.

Module 14 (ເດືອນທີ 9–12): Certification ແລະ capstone ສາມອັນ

ເສັ້ນທາງ certification, ຜ່ອມກຳນົດເວລາທີ່ຊື່ສັດ. Certification ບໍ່ໄດ້ເຮັດໃຫ້ທ່ານເປັນສະຖາປະນິກ — ສິບ ສາມ module ກ່ອນໜ້ານີ້ຕ່າງຫາກທີ່ເຮັດ — ແຕ່ພວກມັນເຮັດໃຫ້ທ່ານໄດ້ສຳພາດ, ແລະ ການກຽມສອບຝັ່ງ ບໍລິການຂອງຜູ້ໃຫ້ບໍລິການເຂົ້າສູ່ປາຍນິ້ວຂອງທ່ານ. ບັນໄດ, ທີ່ຈັງຫວະມີລະຫຼັງຊົ່ວໂມງ: **AWS Certified Solutions Architect – Associate (SAA-C03)** — 2–3 ເດືອນ; ໃບຢັ້ງຢືນສະຖາປະນິກທີ່ຖືກຮຽກຫາຫຼາຍ ທີ່ສຸດໃນປະກາດວຽກ, ແລະ ຫຼັງໄລຍະທີ່ 1–3 ຫຼາຍສ່ວນຂອງມັນຈະຮູ້ສຶກຄືການທວນຄືນທີ່ຕິດຊື່ບໍລິການເຂົ້າໄປ; ເອົາອັນນີ້ກ່ອນ. **AWS Certified Solutions Architect – Professional (SAP-C02)** — 4–8 ເດືອນ ຫຼັງ Associate; ອີງສະຖານະການ, ຍາກແທ້, ແລະ ເປັນແຖວດຽວທີ່ແຂງແຮງທີ່ສຸດໃນ resume ຂອງສາຍງານນີ້; ເນື້ອຫາ migration ແລະ multi-account ຂອງໄລຍະທີ່ 4 ຄືເຄິ່ງໜຶ່ງຂອງຫຼັກສູດສອບ. **Azure AZ-305 (Azure Solutions Architect Expert)** — ເພີ່ມຖ້າຕະຫຼາດຂອງທ່ານໜັກໄປທາງ Microsoft (ຕະຫຼາດວິສາຫະກິດ ສ່ວນໃຫຍ່ຢ່າງໜ້ອຍໃຊ້ສອງພາສາ); ຄາດ 2–3 ເດືອນ ໂດຍຄວາມຮູ້ AWS ໂອນມາໄດ້ຢ່າງຄຸ້ມຄ່າ (ເລີ່ມທີ່ <https://learn.microsoft.com/en-us/credentials/certifications/azure-solutions-architect/>). **TOGAF Foundation** — ທາງເລືອກ, 2–4 ອາທິດ; ວິສາຫະກິດຂະໜາດໃຫຍ່ບາງແຫ່ງຮຽກຫາ; ມັນຢັ້ງຢືນ *ວິທີການ* ແລະ ຄຳສັບຂອງ enterprise architecture ຫຼາຍກວ່າທັກສະ cloud. ລຳດັບສຳລັບຫຼັກສູດນີ້: ຮຽນ SAA ຄຽງຄູ່ ເດືອນທີ 7–8, ສອບປະມານເດືອນທີ 8; ຈາກນັ້ນເລືອກ SA Pro (ເສັ້ນທາງເລິກທາງເຕັກນິກ) ຫຼື AZ-305 (ເສັ້ນທາງ ກວ້າງ) ຕະຫຼອດເຖິງເດືອນທີ 12, ໂດຍ SA Pro ຈົບໃນເດືອນທີ 14–16 ຕາມຈັງຫວະທີ່ຊື່ສັດ.

Capstone ສາມອັນ. ແຕ່ລະອັນຄືຊຸດຄົບຖ້ວນລະດັບ portfolio: ຊຸດແຜນວາດສາມລະດັບ, ADR 5+ ສະບັບ, ການປະເມີນຕົ້ນທຶນຜ່ອມສົມມຸດຕິຖານ, ແລະ Well-Architected review ທີ່ທ່ານດຳເນີນເອງຜ່ອມສົ່ງທີ່ພົບ. ໃຊ້ ເວລາອັນລະ 3–4 ອາທິດ. ສາມຊິນງານນີ້, ບວກກັບປຶ້ມບັນທຶກຂອງທ່ານ, ຄື ຫຼັກຖານໃນການສຳພາດຂອງທ່ານ ເລື່ອງ “end-to-end solution design.”

Capstone 1 — greenfield: ແພລດຟອມອີຄອມເມີຊໄທ ສໍາລັບຜູ້ໃຊ້ 1 ລ້ານຄົນ. ໂຈດ: marketplace, ຜູ້ໃຊ້ລົງທະບຽນ 1 ລ້ານຄົນ, 50k ຜ່ອມກັນທີ່ຈຸດສູງສຸດຂອງ flash sale, mobile-first, ການເຊື່ອມຕໍ່ຊໍາລະເງິນແບບ PromptPay, ສອດຄ່ອງ PDPA, availability 99.9%, ງົບແບບ seed-stage ທີ່ຕ້ອງລະວັງ. ຕ້ອງມີ: ການເລືອກ pattern ຜ່ອມເສັ້ນທາງວິວັດ, ຍຸດທະສາດ caching, ບົດສິນທະນາ RTO/RPO ຂຽນເປັນບົດສິນທະນາ, ການປະເມີນ unit economics, ແຜນກະແສຂໍ້ມູນສ່ວນບຸກຄົນ.

Capstone 2 — brownfield: ການ migrate ບໍລິສັດ on-prem 40 server. ໂຈດ: ບໍລິສັດໂລຈິສຕິກໄທ; 40 server (ERP ເທິງ Oracle, ແອັບຄຸ້ມຄອງສາງອາຍຸ 12 ປີ, file share, AD, ແອັບຂອງພະແນກຕ່າງໆ — ສ້າງບັນຊີລາຍການເຕັມດ້ວຍ AI ແລ້ວຕຶງມັນໄວ້); ສັນຍາເຊົາ data center ໝົດອາຍຸໃນ 14 ເດືອນ; ຄະນະກຳມະການຢາກອອກ. ຕ້ອງມີ: ຕາຕະລາງບັນຊີລາຍການ 7-R ຄົບຖ້ວນ, ແຜນສາມຄື້ນຜ່ອມເຫດຜົນເລື່ອງ dependency, ການອອກແບບ landing zone, ເສັ້ນເວລາຕົ້ນທຶນຂອງ migration bubble, ແລະ ບົດບັນທຶກເຖິງ CFO.

Capstone 3 — ອັນຍາກ: multi-region DR ສໍາລັບ fintech. ໂຈດ: ບໍລິສັດຊໍາລະເງິນທີ່ຖືກກຳກັບໃຫ້ $RTO \leq 15$ ນາທີ / $RPO \approx 0$ ສໍາລັບບັນຊີແຍກປະເພດ, ຜ່ອມຂໍ້ຈຳກັດ data residency ຕໍ່ຂໍ້ມູນລູກຄ້າ ແລະ ການທົດສອບ failover ປະຈຳປີທີ່ຜູ້ຄຸມກົດເປັນພະຍານ. ຕ້ອງມີ: ການວິເຄາະ active-passive ທຽບ active-active ຜ່ອມຕົ້ນທຶນ, ການອອກແບບ replication ຜ່ອມແຜນ residency, runbook ຂອງ failover, ແລະ ແຜນການຊ້ອມ.

ເກນປະເມີນຕົນເອງ (ນຳໃຊ້ກັບແຕ່ລະ capstone, ຢ່າງຊື່ສັດ, ໃນປຶ້ມບັນທຶກ):

ມິຕິ	1 — ຍັງບໍ່ທັນ	3 — ໝັ້ນຄົງ	5 — ຈ້າງຄົນມີເລີຍ
Requirements	ໂດດໄປຫາວິທີແກ້	NFR ຖືກລະບຸ ແລະ ໂຍງເຖິງທາງເລືອກການອອກແບບ	ວາງຕໍາແໜ່ງເທິງສາມຫຼ່ຽມ trade-off, ຍົກຂໍ້ຂັດແຍ້ງໃຫ້ “ຝ່າຍທຸລະກິດ”, ສະກັດການຕັດສິນໃຈອອກມາໄດ້
ຄຸນນະພາບການອອກແບບ	ຍັງມີ SPOF; pattern ບໍ່ເໝາະ	Pattern ຖືກຕ້ອງ, multi-AZ, ຈັດການຄວາມລົ້ມເຫຼວແລ້ວ	ສະແດງເຫດຜົນ “ເມື່ອໃດບໍ່ຄວນໃຊ້”; ມີເສັ້ນທາງວິວັດ; ການອອກແບບທີ່ງ່າຍທີ່ສຸດທີ່ຕອບໂຈດ
ຫົກເສົາຫຼັກ	ບໍ່ສົນເສົາຫຼັກ	ແຕ່ລະເສົາຖືກແກ້ໄຂຢ່າງເບິ່ງເຫັນໄດ້	ການທົບທວນທີ່ຕົນດຳເນີນເອງພົບຂໍ້ບົກຜ່ອງຈິງ — ແລະ ການອອກແບບຖືກແກ້ໄຂຕາມນັ້ນ

ມິຕິ	1 — ຍັງບໍ່ທັນ	3 — ພັນຄົງ	5 — ຈ້າງຄົນນິເລີຍ
ຕົ້ນທຶນ	ບໍ່ມີຕົວເລກ	ການປະເມີນແຈກແຈງ ລາຍການ, ລະບຸສົມມຸດຕິ ຖານ	ຊ່ວງຜ່ອມຕົວຂັບເຄື່ອນ, unit economics, ສະຖານະການ 10×, ຍຸດທະສາດການຈອງ
ເອກະສານ	ມີແຕ່ແຜນວາດ	ຊຸດສາມລະດັບ + ADR, ສັນຍາລັກຖືກຕ້ອງ	ຄົ້ນແປກໜ້າຕອບ “ຫຍັງ/ ແນວໃດ/ເປັນຫຍັງ” ໄດ້ ຈາກຊຸດເອກະສານຢ່າງ ດຽວ
ການສື່ສານ	ຊຶ່ງງານດຽວສໍາລັບທຸກ ຜູ້ຟັງ	ມີສະບັບຜູ້ບໍລິຫານ ແລະ ສະບັບວິສະວະກອນ	ບົດສະຫຼຸບຫ້າປະໂຫຍກທີ່ ຢູ່ລອດ; ຕອບແຮງຕ້ານໄວ້ ລ່ວງໜ້າເປັນລາຍລັກ ອັກສອນ

ໃຫ້ຄະແນນທຸກມິຕິ 4+ ໃນທັງສາມ capstone — ແກ້ໄຂຈົນກວ່າຈະໄດ້ — ແລ້ວທ່ານກໍຈົບຫຼັກສູດນີ້.

ສືບຄໍາຖາມສໍາພາດສະຖາປະນິກ (ຜ່ອມຄໍາຕອບທີ່ແຂງແຮງ)

ຝຶກສົ່ງເຫຼົ່ານີ້ອອກສຽງ; ຄວາມແຂງແຮງຢູ່ທີ່ ໂຄງສ້າງ ຂອງແຕ່ລະຄໍາຕອບ, ຊຶ່ງຕອນນີ້ເປັນຂອງທ່ານແລ້ວ.

1. “Design a URL shortener / ticketing site / photo app for a million users.” (ອອກແບບຕົວຫຍໍ້ URL / ເວັບຂາຍປີ້ / ແອັບຮູບ ສໍາລັບຜູ້ໃຊ້ໜຶ່ງລ້ານຄົນ.) ຮູບແບບທີ່ແຂງແຮງ: requirement ກ່ອນ, ອອກສຽງ (“ການອໍານວຍການກວ່າການຂຽນບໍ່? ເປົ້າ availability? ງົບ?”) → ວາງຕໍາແໜ່ງເທິງສາມຫຼັງມ → ໂຄງກະດູກ three-tier ຫຼື serverless-first ຜ່ອມ multi-AZ, cache, CDN → ບອກຊື່ trade-off ໂດຍບໍ່ຕ້ອງໃຫ້ຖາມ → ຈົບດ້ວຍລໍາດັບຂະໜາດຂອງຕົ້ນທຶນ ແລະ ເສັ້ນທາງວິວັດ 10×. ຜູ້ສໍາພາດໃຫ້ຜ່ານດ້ວຍ ຄໍາຖາມທີ່ທ່ານຖາມ, ບໍ່ແມ່ນກ່ອງທີ່ທ່ານແຕ້ມ.

2. “When would you choose NoSQL over a relational database?” (ເມື່ອໃດທ່ານຈະເລືອກ NoSQL ແທນ relational database?) “ຂ້ອຍໃຊ້ SQL ເປັນຄໍາມາດຕະຖານ — ການຮັບປະກັນຄວາມຖືກຕ້ອງ ແລະ ທັກສະທີ່ມີຢູ່ຫົວໄປ — ຈົນກວ່າຈະມີເຫດຜົນທີ່ບອກຊື່ໄດ້ມາລົບລ້າງ: ການຂະຫຍາຍແນວນອນສຸດຂີດ, schema ຍືດຍຸ່ນ, ຫຼື ການເຂົ້າເຖິງແບບ key-value ລະດັບ millisecond ຫຼັກດຽວ. ຈາກນັ້ນຂ້ອຍເລືອກລົດຊາດ NoSQL ໃຫ້ຕິງກັບ access pattern, ແລະ ຂ້ອຍຂຽນ ADR ລວມທັງສິ່ງທີ່ພວກເຮົາຍອມເສຍ: join, ແລະ semantics ບາງຢ່າງຂອງ consistency.”

3. “Explain RTO and RPO, and how they drive design.” (ອະທິບາຍ RTO ແລະ RPO, ແລະ ພວກມັນຂັບເຄື່ອນການອອກແບບແນວໃດ.) ນິຍາມທັງສອງໃຫ້ຄົມ → “ພວກມັນຄືການຕັດສິນໃຈທາງທຸລະກິດທີ່ມີຢ່າຍລາຄາແບບເລກກຳລັງ” → ປັນໄດ: backup ກາງຄືນ → replication ຕໍ່ເນື່ອງ → warm standby → active-active, ຕື່ນຫິນສູງຂຶ້ນທຸກຂັ້ນ → “ວຽກຂອງຂ້ອຍຄືເຮັດໃຫ້ຝ່າຍທຸລະກິດເລືອກຢ່າງຮູ້ຕົວ, ແລ້ວອອກແບບໃຫ້ຕົງກັບຕົວເລກນັ້ນພໍດີ — ແລະ ຊ້ອມມັນ, ເພາະ failover ທີ່ບໍ່ໄດ້ທົດສອບຄືຄວາມຫວັງ.”

4. “Monolith or microservices?” (Monolith ຫຼື microservices?) ຄຳຕັດສິນຂອງ Module 8, ຄຳຕໍ່ຄຳ: ເລີ່ມແບບ modular-monolith; ແຍກອອກເມື່ອຄວາມເຈັບປວດຈາກການຂະຫຍາຍທີ່ມ ຫຼື ໂຫຼດມາເຖິງແທ້ໆ; microservices ຄືເຄື່ອງມືຂະຫຍາຍທີ່ມ ທີ່ເກັບພາສີລະບົບກະຈາຍ. ໄດ້ຄະແນນເພີ່ມສຳລັບ “ຂ້ອຍຢາກແລ່ນ monolith ທີ່ດີ ຫຼາຍກວ່າລະບົບກະຈາຍທີ່ບໍ່ດີ.”

5. “How do you handle a large cloud bill / cost optimization?” (ທ່ານຈັດການໃບບິນ cloud ກ້ອນໃຫຍ່ / ການເພີ່ມປະສິດທິພາບຕື່ນຫິນແນວໃດ?) “ຄວາມເບິ່ງເຫັນໄດ້ກ່ອນ — tagging ແລະ showback; ຈາກນັ້ນເກັບກ່ຽວຕາມລຳດັບແຮງງານ: ຂ້າຊັບພະຍາກອນທີ່ຕາຍແລ້ວ, rightsize ຈາກການວັດແທກ, ແບ່ງຂັ້ນ storage, ຈັດຕາຕະລາງໃຫ້ non-prod ຫຼັບ, ແລ້ວຈຶ່ງຈອງຖານໂຫຼດຄົງທີ່. ແລະ ຂ້ອຍລາຍງານ unit economics, ບໍ່ແມ່ນຍອດລວມ — ໃບບິນທີ່ໃຫຍ່ຂຶ້ນຜ່ອມຕື່ນຫິນຕໍ່ຄຳສັ່ງຊື້ທີ່ຫຼຸດລົງ ຄືຄວາມສຳເລັດ, ບໍ່ແມ່ນບັນຫາ.”

6. “How would you migrate a legacy on-prem application?” (ທ່ານຈະ migrate ແອັບພລິເຄຊັນ on-prem ເກົ່າແນວໃດ?) “ປະເມີນກ່ອນຢ່າຍ — ປັນຊີລາຍການ, dependency, ແລະ R ຕໍ່ workload ຈາກ 7 RS; ສ້າງ landing zone ກ່ອນຄືນທຳອິດ; ຄືນຮຽງງ່າຍກ່ອນ ຜ່ອມ cutover ທີ່ຊ້ອມແລ້ວ ແລະ ແຜນ rollback; ແລະ ສຳລັບ database ເພັດຍອດມົງກູດ, ໃຊ້ cutover ອີງ replication ທີ່ຂະໜາດພໍດີກັບ downtime ທີ່ຝ່າຍທຸລະກິດເຊັນຮັບຮອງ. ອີກຢ່າງ: ຄຳເຕືອນເລື່ອງ migration bubble ຢ່າງຊື່ສັດແຕ່ຫົວທີ່.”

7. “How do you secure a cloud architecture?” (ທ່ານເຮັດໃຫ້ສະຖາປັດຕະຍະກຳ cloud ປອດໄພແນວໃດ?) ເດີນຜ່ານທີ່ລະຊັ້ນ: ຕົວຕົນ (least privilege, MFA, ບໍ່ໃຊ້ root ປະຈຳວັນ) → ເຄືອຂ່າຍ (private subnet, ການແບ່ງສ່ວນ, attack surface ຕ່ຳສຸດ) → ຂໍ້ມູນ (ເຂົ້າລະຫັດຕອນເກັບ/ຕອນສົ່ງ, ການຈັດປະເພດ, residency) → ການກວດພິບ (audit log, ການແຈ້ງເຕືອນ) → “ແລະ ດ້ວຍການອອກແບບ, ບໍ່ແມ່ນຕິດເສີມພາຍຫຼັງ — ການຄວບຄຸມຄວາມປອດໄພທີ່ຖືກທົດສຸດ ຄືການຕັດສິນໃຈສະຖາປັດຕະຍະກຳທີ່ຕັດຄວາມສ່ຽງອອກໄປໝົດ, ເຊັ່ນ ການບໍ່ແຕະເລກບັດດິບເລີຍ.”

8. “Tell me about a design decision you got wrong.” (ເລົ່າໃຫ້ຝັງເລື່ອງການຕັດສິນໃຈອອກແບບທີ່ທ່ານເຮັດຜິດ.) ເຂົ້າທົດສອບອັດຕາ, ບໍ່ແມ່ນປະຫວັດສາດ. ຮູບແບບ: ຕົວຢ່າງຈິງ (capstone) → ສິ່ງທີ່ທ່ານເຊື່ອ → ສິ່ງທີ່ຄວາມເປັນຈິງບອກ → post-mortem, ADR ທີ່ອັບເດດ, pattern ທີ່ຕອນນີ້ທ່ານກວດຫາ. ສະຖາປະນິກທີ່ໃຫ້ຄຳຕອບນີ້ບໍ່ໄດ້ ຄືສະຖາປະນິກທີ່ບໍ່ເຄີຍຖືກທົບທວນ.

9. “How do you explain a complex technical decision to a non-technical executive?” (ທ່ານອະທິບາຍການຕັດສິນໃຈເຕັກນິກທີ່ຊັບຊ້ອນ ໃຫ້ຜູ້ບໍລິຫານທີ່ບໍ່ແມ່ນສາຍເຕັກນິກແນວໃດ?) “ການຕັດສິນໃຈ ແລະ ຕົວເລກທາງທຸລະກິດກ່ອນ, ກົນໄກເມື່ອຖືກຮ້ອງຂໍເຫຼົ່ານັ້ນ; ໃຊ້ context diagram, ບໍ່ແມ່ນ container; ຄວາມສ່ຽງໃນຄຳສັບຂອງລາຍຮັບ-compliance-ຊື່ສຽງ; ແລະ ຂ້ອຍນຳສອງການຕັດສິນໃຈທີ່ຂ້ອຍຕ້ອງການຈາກເຂົາມາ

ດ້ວຍ, ຂຽນເປັນທາງເລືອກພ້ອມປ້າຍລາຄາ — ຜູ້ບໍລິຫານຕັດສິນລະຫວ່າງທາງເລືອກ; ເຂົາບໍ່ອະນຸມັດຄວາມລຶກລັບ.”

10. “An engineer strongly disagrees with your design. What do you do?” (ວິສະວະກອນຄົນໜຶ່ງບໍ່ເຫັນດີກັບການອອກແບບຂອງທ່ານຢ່າງແຮງ. ທ່ານຈະເຮັດແນວໃດ?) “ກ່ອນອື່ນຂ້ອຍ steel-man ເຂົາອອກສຽງ — ເຂົາອາດຖືກ, ແລະ ການທົບທວນທີ່ປ່ຽນການອອກແບບຂອງຂ້ອຍ ຄືການທົບທວນທີ່ໄດ້ຜິນ. ຖ້າມັນສູງສຸດແທ້, ເກນເປັນຜູ້ຕັດສິນ (ຄວາມເໝາະສົມກັບ NFR, ທັກສະທີ່ມ, ຕົ້ນທຶນການອອກ), ບໍ່ແມ່ນອາວຸໂສ — ແລະ ຂ້ອຍໃຫ້ຄວາມມັກຂອງຜູ້ລົງມືເປັນຝ່າຍຊະນະເມື່ອສະເໝີກັນ, ເພາະຄວາມມຸ່ງໝັ້ນຄືຂອງຖືກທີ່ລາຄານັ້ນ. ບໍ່ວ່າທາງໃດການຕັດສິນໃຈ ແລະ ທາງເລືອກທີ່ຖືກປະຕິເສດ ຈະລົງໃນ ADR, ເພື່ອວ່າພວກເຮົາຈະບໍ່ຖຽງເລື່ອງນີ້ສອງເທື່ອ.”

ຫຼັກສູດຫຍໍ້ໜຶ່ງໜ້າ

ເມື່ອໃດ	Module	ຈຸດສຸມ	ຫຼັກຖານພາຍນອກ
ອາທິດທີ 1–2	1	ບົດທວນ cloud; ສາມຫຼ່ຽມ trade-off; NFR	ເລີ່ມ Design Journal
ອາທິດທີ 3–4	2	Compute/storage/database/network ໃນຖານະການຕັດສິນໃຈ	—
ອາທິດທີ 5–6	3	ການອ່ານ & ແຕ້ມແຜນວາດ; ຄວາມຄ່ອງ 3 ຊັ້ນ	Milestone ກະດານ ຂາວ 15 ນາທີ
ອາທິດທີ 7–8	4	Reliability: multi-AZ, auto-scaling, RTO/RPO	—
ອາທິດທີ 9–10	5	Security: least privilege, zero trust, ການແບ່ງສ່ວນ	—
ອາທິດທີ 11–12	6	Performance & cost: caching, CDN, ການຈອງ, unit economics	ການອອກແບບທີ່ ຕີລາຄາໃນ calculator
ອາທິດທີ 13–14	7	Ops excellence & sustainability; ບົດ ຝຶກຫຼົກເລີ່ມ	ການທົບທວນຫຼັກເລີ່ມ ຈຳລອງ

ເມື່ອໃດ	Module	ຈຸດສຸມ	ຫຼັກຖານພາຍນອກ
ອາທິດທີ 15–17	8	Pattern: monolith/microservices, event-driven, serverless	—
ອາທິດທີ 18–20	9	Pattern: data lake/warehouse, multi-region, hybrid	ດ່ານທິດສອບຄັ້ງລວມ
ອາທິດທີ 21–23	10	Migration: 7 Rs, ຄືນ, landing zone	Migration ເທິງເຈ້ຍ
ອາທິດທີ 24–26	11	ຂໍ້ຈຳກັດ: PDPA/GDPR, ລະບົບເກົ່າ, ເສດຖະສາດ lock-in	ດ່ານທິດສອບຂໍ້ຈຳກັດ
ເດືອນທີ 7–8	12	ADR, ຊຸດແຜນວາດ, ການດຳເນີນ Well-Architected review	ຊຸດ ThaiTicket; ສອບ SAA-C03 (ກຽມ 2–3 ດ.)
ເດືອນທີ 8–9	13	ການນຳສະເໜີ, ການຄິດຕົ້ນທຶນຂໍ້ສະເໜີ, ແຮງຕ້ານ, ການເປັນພິລັງງ	ບົດຄູ່ທີ່ບັນທຶກໄວ້
ເດືອນທີ 9–12	14	Capstone 1–3	Portfolio ຄົບ; SA Pro (4–8 ດ.) ຫຼື AZ-305 ກຳລັງດຳເນີນ

ຄຳສັ່ງຫ້າຍຈາກຄູຂອງທ່ານ. ຊາວຫົກອາທິດຜ່ານໄປ ທ່ານອອກແບບໄດ້; ສິບສອງເດືອນຜ່ານໄປ ທ່ານ *ປະກອບ ວິຊາຊີບ* ໄດ້ — ມັນມີຄວາມຕ່າງ, ແລະ ຄວາມຕ່າງນັ້ນເອງຄືສິ່ງທີ່ຫຼັກສູດນີ້ຖືກສ້າງມາເພື່ອປິດ. ນິໄສເຫຼົ່ານີ້ຕອນນີ້ຄືອາຊີບ: ແຕ້ມກ່ອນຖຽງ, ຂຽນ ADR ໃນມື້ທີ່ຕັດສິນໃຈ, ຕີລາຄາສິ່ງທີ່ທ່ານສະເໜີ, ບອກຊື່ trade-off ກ່ອນມີໃຜຖາມ, ແລະ ບົມວິສະວະກອນອ້ອມຕົວທ່ານ ຈົນມາດຕະຖານຂອງທ່ານຢູ່ຍາວກວ່າການມີຕົວທ່ານໃນຫ້ອງ. ສະຖາປັດຕະຍະກຳຄືວຽກສາຍເຕັກນິກທີ່ຫາຍາກ ຊຶ່ງ *ດີຂຶ້ນ* ເມື່ອທ່ານແກ່ຕົວໄປກັບມັນ, ເພາະວັດຖຸດິບຂອງມັນຄືວິຈາລະນະຍານ, ແລະ ວິຈາລະນະຍານທີ່ບໍ່ຕົນທົບຕອກ. ຮັກສາປຶ້ມບັນທຶກໄວ້ຕໍ່ໄປ. ອີກສິບສອງອອກແບບຈາກນີ້, ທ່ານຈະບໍ່ຕ້ອງການຫຼັກສູດນີ້ — ທ່ານຈະເປັນຄົນແກ້ໄຂມັນ.

ແຫຼ່ງອ້າງອີງ

ປະກາດຮັບສະໝັກວຽກ ແລະ ນິຍາມບົດບາດ ທີ່ໃຊ້ໃນການ mapping ຂໍ້ກຳນົດ (ດຶງມາເດືອນສິງຫາ 2026): [ແມ່ແບບຄຳບັນຍາຍວຽກ Cloud Architect ຂອງ KORE1](#) · [ຄຳບັນຍາຍວຽກ Cloud Architect ຂອງ 4 Corner Resources](#) · [ໜ້າທີ່ຮັບຜິດຊອບຂອງ Solution Architect ໃນ Azure Well-Architected Framework ຂອງ Microsoft](#) · [ປະກາດ Pre-Sales Solutions Architect – AWS ຂອງ Arpio](#) · [ປະກາດ Solution Architect –](#)

[Enterprise \(ຜ່ານ Built In\)](#) ຂອງ Intel · [ປະກາດ Cloud Domain Architect](#) ຂອງ Halliburton. ການປະເມີນ ເວລາກຽມສອບ certification: [ການສຳຫຼວດເວລາຮຽນ SAA-C03 ຂອງ CBT Nuggets](#) ແລະ [ຄຳແນະນຳການ ກຽມ SAP-C02 ຂອງ Whizlabs](#); ນິຍາມ certification ຈາກ [Microsoft Learn \(AZ-305\)](#) ແລະ [AWS Well-Architected Framework](#). ເປັນເຫຼັ້ມຄູ່ຂອງ *The Cloud Leader Course* ໃນຊຸດ B4LCILC.

ບົດຮຽນໃໝ່ — ສະບັບເດືອນສິງຫາ 2026

ໃນເດືອນສິງຫາ 2026, ບົດຮຽນໃໝ່ໄດ້ຖືກເພີ່ມເຂົ້າໃນສາມຫຼັກສູດແບບໂຕ້ຕອບທີ່ [b4lcilc.org](#) — ຫຼັກສູດຜູ້ນຳ ດ້ານ Cloud (Cloud Leader), ວິສະວະກອນ Cloud (Cloud Engineer), ແລະ ສະຖາປະນິກ Cloud (Cloud Architect). ເອກະສານເພີ່ມເຕີມສະບັບນີ້ ນຳສະບັບອ່ານໃຫ້ທັນສະໄໝ ດ້ວຍບົດສະຫຼຸບຫຍໍ້ທີ່ຊື່ສັດຕໍ່ເນື້ອຫາໃໝ່ ນັ້ນ, ໃນນ້ຳສຽງດຽວກັນກັບສ່ວນອື່ນຂອງປຶ້ມເຫຼັ້ມນີ້.

ຂໍເວົ້າຈັກຄຳກ່ຽວກັບສິ່ງທີ່ທ່ານຖືຢູ່ໃນມື. ໜ້າເຫຼົ່ານີ້ຄືສະບັບອ່ານ: ແນວຄິດ, ຄຳປຽບທຽບ, ແລະ ປະໂຫຍກທີ່ຄຸ້ມ ຄຳແກ່ການເວົ້າອອກສຽງ. ສະບັບເຕັມແບບໂຕ້ຕອບທາງອອນລາຍ ມີເນື້ອຫາຫຼາຍກວ່ານີ້ຫຼາຍ — ທຸກບົດຮຽນໃໝ່ ມາຜ່ອມກັບແບບທົດສອບ (quiz) ທີ່ກວດຄວາມເຂົ້າໃຈຂອງທ່ານກ່ອນຈະໃຫ້ກ້າວຕໍ່ໄປ, ຫ້ອງທົດລອງແບບລົງມື ເຮັດຈິງ (ລວມທັງ terminal ຝຶກຫັດ ທີ່ເຮັດວຽກຄືກັບຂອງແທ້ ແຕ່ບໍ່ສາມາດເສຍເງິນໄດ້), ວິດີໂອ, ບົດຝຶກຄຳສັບ, ແລະ — ສຳລັບເນື້ອຫາ capstone (ບົດຝຶກລວມຍອດ) — ບົດຝຶກອອກແບບແບບມີຄຳແນະນຳ ຜ່ອມລາຍການ ກວດຄວາມຕ້ອງການ ແລະ ຄຳຕອບຕົວຢ່າງທີ່ເຮັດໃຫ້ເບິ່ງ. ຈົ່ງອ່ານບົດສະຫຼຸບເຫຼົ່ານີ້ ເພື່ອໃຫ້ຮູ້ບ່ອນຂອງແຕ່ລະ ແນວຄິດເຂົ້າໄປຢູ່ໃນຫົວຂອງທ່ານ; ຈາກນັ້ນ, ເມື່ອມີໂອກາດ, ໃຫ້ນັ່ງເຮັດສະບັບໂຕ້ຕອບ ເພື່ອໃຫ້ຄວາມຮູ້ ຕິດແໜ້ນ.

ເນື້ອຫາທີ່ເພີ່ມແບ່ງອອກເປັນສາມກຸ່ມ. ສຳລັບຫຼັກສູດ **ຜູ້ນຳດ້ານ Cloud (Cloud Leader)**: ການແນະນຳ terminal ຢ່າງນຸ່ມນວນ, ກົດໝາຍຄວາມເປັນສ່ວນຕົວຂອງໄທເອງ, ຄູ່ມືຜູ້ນຳສຳລັບຂໍ້ສະເໜີ AI ແລະ ໃບບິນ AI, ແລະ ບົດຝຶກ capstone ທີ່ປະກອບຫຼັກສູດທັງໝົດເຂົ້າເປັນໜຶ່ງໜ້າ. ສຳລັບຫຼັກສູດ **ວິສະວະກອນ Cloud (Cloud Engineer)**: serverless computing — ຮູບແບບທີ່ປ່ຽນສະຄຣິບປະຕິບັດງານນ້ອຍໆ ໃຫ້ກາຍເປັນ function ທີ່ແລ່ນໂດຍບໍ່ເສຍຄ່າຫຍັງ. ສຳລັບຫຼັກສູດ **ສະຖາປະນິກ Cloud (Cloud Architect)**: ເລກຄະນິດຂອງ ຄຳສັນຍາຄວາມຜ່ອມໃຊ້ງານ, ຮູບແບບ Retrieval-Augmented Generation ທີ່ທຸກມື້ນິລູກຄ້າຖາມເຖິງທຸກ ອາທິດ, ແລະ ບົດຝຶກຄິດໄລ່ຕົ້ນທຶນເຕັມຮູບແບບ — ຕີລາຄາການອອກແບບ ກ່ອນທີ່ໃຜຈະລົງມືສ້າງ.

ສຳລັບຫຼັກສູດສະຖາປະນິກ Cloud

ເລກຄະນິດຂອງເລກເກົ້າ — SLA, SLO ແລະ error budget

ຄວາມຜ່ອມໃຊ້ງານ (availability) ຖືກອ້າງອີງເປັນຈຳນວນເລກເກົາ, ແລະ ສິ່ງທຳອິດທີ່ສະຖາປະນິກເຮັດກັບເລກເກົາ ຄືປ່ຽນມັນເປັນນາທີ — ເພາະນາທີຄືສິ່ງທີ່ທຸລະກິດຮູ້ສຶກໄດ້. ເດືອນ 30 ມີ 43,200 ນາທີ:

ຄວາມຜ່ອມໃຊ້ງານ	ເວລາຢຸດເຮັດວຽກທີ່ອະນຸຍາດຕໍ່ເດືອນ
99.9% (“ສາມເກົາ”)	ປະມານ 43 ນາທີ
99.99% (“ສີ່ເກົາ”)	ປະມານ 4.3 ນາທີ
99.999% (“ຫ້າເກົາ”)	ປະມານ 26 ວິນາທີ

ຫ້າເກົາ — ຕົວເລກທີ່ຜູ້ບໍລິຫານມັກເວົ້າອອກສຽງ — ເກີນຄວາມສາມາດຂອງການກູ້ຄືນທີ່ຄົນເປັນຜູ້ປະຕິບັດ; failover ຕ້ອງເປັນອັດຕະໂນມັດ ບໍ່ດັງນັ້ນຄຳສັນຍາກໍເປັນນິຍາຍ. ເລກເກົາແຕ່ລະຕົວທີ່ເພີ່ມເຂົ້າມາ ເສຍຄ່າວິສະວະກຳແພງຂຶ້ນປະມານສິບເທົ່າ; ການປ່ຽນເປັນນາທີ ຄືວິທີເຮັດໃຫ້ stakeholder ເລືອກຢ່າງຮູ້ຕົວ.

ສາມຄຳຫຍໍ້ນຳພາບົດສິນທະນາ. Service-Level Indicator (SLI — ຕົວຊີ້ວັດລະດັບບໍລິການ) ຄືການວັດແທກ. Service-Level Objective (SLO — ເປົ້າໝາຍລະດັບບໍລິການ) ຄືເປົ້າໝາຍພາຍໃນເທິງການວັດແທກນັ້ນ. Service-Level Agreement (SLA — ຂໍ້ຕົກລົງລະດັບບໍລິການ) ຄືສັນຍາພາຍນອກ, ມີເງິນຜູກຢູ່ກັບການພາດເປົ້າ. ວິໄນ: SLO ຕ້ອງເຂັ້ມງວດກວ່າ SLA ສະເໝີ — ທ່ານເລັ່ງເປົ້າໄວ້ທາງໃນເສັ້ນທີ່ທ່ານຖືກຈ້າງໃຫ້ຮັກສາ — ແລະ SLI ວັດແທກສິ່ງທີ່ຜູ້ໃຊ້ປະສົບຈິງ: server ທີ່ເປີດຢູ່ ແຕ່ຕອບກັບເປັນ error ຖືວ່າລົ້ມ ໃນທຸກຄວາມໝາຍທີ່ສຳຄັນ.

Error budget (ງົບຄວາມຜິດພາດ) ເຮັດໃຫ້ SLO ນຳໃຊ້ໄດ້ຈິງ: ຊ່ອງຫວ່າງລະຫວ່າງຄວາມສົມບູນແບບ ກັບເປົ້າໝາຍ, ຖືເປັນຊັບພະຍາກອນທີ່ໃຊ້ຈ່າຍໄດ້. SLO 99.9% ໃຫ້ຄວາມລົ້ມເຫຼວ 43 ນາທີຕໍ່ເດືອນ; ການ deploy ແລະ ການທົດລອງໃຊ້ຈ່າຍຈາກມັນ; ເມື່ອມັນໝົດ, ທີ່ມຸດປ່ອຍຂອງໃໝ່ ແລ້ວຫັນມາເຮັດວຽກດ້ານຄວາມເຊື່ອຖືໄດ້ — ຈົບການຖືກຖຽງທີ່ເກົາແກ່ທີ່ສຸດໃນວົງການ software, ປ່ອຍໄວ ຫຼື ໝັ້ນຄົງ, ດ້ວຍການປ່ຽນຄວາມເຫັນເປັນເລກຄະນິດ.

ບັດນິຄົກັບດັກ: ຄວາມຜ່ອມໃຊ້ງານຄູນກັນໄປຕາມລວງໄສ້. ສາມບໍລິການຕໍ່ກັນເປັນແຖວ (series), ແຕ່ລະອັນຢູ່ທີ່ 99.9% ຢ່າງຊື່ສັດ, ປະກອບກັນເປັນ $0.999 \times 0.999 \times 0.999 \approx 99.7\%$ — ປະມານ 130 ນາທີຕໍ່ເດືອນ, ສາມເທົ່າຂອງທີ່ບໍລິການໃດອັນດຽວຍອມຮັບ. ການສຳຮອງຊ້າ (redundancy) ເຮັດວຽກໄປທາງກົງກັນຂ້າມ: ສອງ instance 99.9% ທີ່ເປັນອິດສະຫຼະຈາກກັນ ຈັດຂະໜານກັນ (parallel) ຈະລົ້ມຜ່ອມກັນພຽງ $0.1\% \times 0.1\%$ ຂອງເວລາ — ນິຕິເຫດຜົນທີ່ multi-zone ຄຸ້ມຄ່າຫຼາຍ. ປະໂຫຍກລະດັບການສຳພາດວຽກ: ທ່ານບໍ່ສາມາດສັນຍາຄວາມຜ່ອມໃຊ້ງານ ເກີນຜົນຄູນຂອງ dependency ແບບ synchronous ຂອງທ່ານໄດ້, ດັ່ງນັ້ນ ຫຼີກຫຍ້ໄສ້ໃຫ້ສັນລົງ ຫຼືກໍຫຼຸດຄຳສັນຍາລົງ.

ສະຖາປັດຕະຍະກຳໃນຍຸກ AI — RAG ແລະ ຍາດພີ້ນ້ອງຂອງມັນ

ມາຮອດປີ 2026 ປຶ້ມລາຍການຮູບແບບ (pattern catalog) ມີລາຍການໃໝ່ ທີ່ລູກຄ້າຖາມເຖິງທຸກອາທິດ: ມັນແມ່ນຫຍັງ, ເມື່ອໃດມັນຄຸ້ມຄ່າ, ແລະ ເມື່ອໃດມັນຈະເຮັດໃຫ້ທ່ານອາຍ. **Large Language Model (LLM — ໂມເດວພາສາຂະໜາດໃຫຍ່)** API — Amazon Bedrock, Azure OpenAI Service — ໃນທາງສະຖາປັດຕະຍະກຳຄື *dependency* ທີ່ແພງ, ຊ້າ, ແລະ ບໍ່ຈື່ສະຖານະ (stateless). ມັນຖືກຄິດເງິນຕໍ່ **token** — ຕ່ອນຄຳຍາວປະມານສີ່ຕົວອັກສອນ. ມັນຕອບເປັນວິນາທີ, ບໍ່ແມ່ນ millisecond, ບໍ່ຈື່ຫຍັງລະຫວ່າງການເອີ້ນແຕ່ລະຄັ້ງ, ແລະ ບາງຄັ້ງກໍ່ຜິດຢ່າງໝັ້ນໃຈ — ອົງປະກອບທີ່ມີຮູບແບບຄວາມລຶມເຫຼວແປກປະຫຼາດ, ສົມຄວນໄດ້ຮັບ timeout, retry, circuit breaker, ແລະ ສັນຍານເຕືອນງົບປະມານ ຄືກັນກັບ dependency ໃດໆ, ເພາະ loop ການເອີ້ນທີ່ຫຼຸດຈາກການຄວບຄຸມ ເຜົາເງິນແທ້ເປັນລາຍຊົ່ວໂມງ.

ໂມເດວບໍ່ຮູ້ຈັກເອກະສານຂອງທ່ານ; ຄຳຕອບມາດຕະຖານຄື **Retrieval-Augmented Generation (RAG — ການສ້າງຄຳຕອບ ເສີມດ້ວຍການຄົ້ນຄົ້ນຂໍ້ມູນ)**: ແທນທີ່ຈະສອນຂໍ້ມູນຂອງທ່ານໃຫ້ໂມເດວ, ທ່ານ ຄົ້ນຫາ ຂໍ້ເທັດຈິງທີ່ກ່ຽວຂ້ອງ ໃນເວລາຖາມ. ເອກະສານຖືກຕັດເປັນຕ່ອນ (chunks), ແຕ່ລະຕ່ອນຖືກປ່ຽນເປັນ **embedding** — ລາຍການຕົວເລກທີ່ວາງຕຳແໜ່ງຄວາມໝາຍຂອງມັນໃນອະວະກາດ — ເກັບໄວ້ໃນ **vector database** (pgvector ເທິງ AWS, AI Search ເທິງ Azure), ເຊິ່ງສົ່ງຄືນຕ່ອນທີ່ໃກ້ຄຽງທີ່ສຸດ. ຕ່ອນເຫຼົ່ານັ້ນຖືກວາງໃສ່ໃນ prompt: ໂມເດວບໍ່ໄດ້ປຶກສາຂໍ້ມູນຂອງທ່ານ; ມັນກຳລັງສະຫຼຸບຂໍ້ຄວາມທີ່ຄົດມາ ທີ່ທ່ານໄປເອົາມາໃຫ້ມັນ.

ເທິງກະດານຂາວ, RAG ຄືສອງທໍ່ລຳລຽງ (pipeline): ເສັ້ນທາງນຳເຂົ້າ (ingestion path) ທີ່ແລ່ນລ່ວງໜ້າ (ເອກະສານຖືກຕັດຕ່ອນ, ຝັ່ງເປັນ embedding, ເກັບໄວ້) ແລະ **ເສັ້ນທາງສອບຖາມ (query path)** ທີ່ແລ່ນຕໍ່ແຕ່ລະ request — ສອງຮູບແບບທີ່ມີຢູ່ໃນປຶ້ມລາຍການແລ້ວ, ນິຕິເຫດຜົນທີ່ສະຖາປະນິກຮຽນເລື່ອງນີ້ໄດ້ໃນຕອນບ່າຍດຽວ.

ການແລກປ່ຽນຄືສາມຢ່າງມາດຕະຖານ, ແຕ່ຄົມກວ່າເກົ່າ: ຕື້ນຫີນ (ທຸກການສອບຖາມເສຍເງິນແທ້ — cache ຄຳຖາມທີ່ຊ້າກັນ, ສົ່ງຄຳຖາມງ່າຍໄປຫາໂມເດວທີ່ຖືກກວ່າ), latency (ວິນາທີ, ບໍ່ແມ່ນ millisecond — stream ຄຳຕອບອອກມາ), ແລະ ຄວາມເປັນສ່ວນຕົວ (ເອກະສານຂອງທ່ານອອກມາຂອບເຂດຂອງທ່ານ; ຖ້າ vector database ບໍ່ສົນໃຈການຄວບຄຸມການເຂົ້າເຖິງລະດັບເອກະສານ, chatbot ຈະສະຫຼຸບຕາຕະລາງເງິນເດືອນໃຫ້ໃຜກໍ່ຕາມທີ່ຖາມ ຢ່າງເຕັມອີກເຕັມໃຈ). ຈົ່ງຮູ້ວ່າເມື່ອໃດ *ບໍ່ຄວນ* ສ້າງມັນ: ຄຳຂໍ “ພວກເຮົາຕ້ອງການ chatbot” ຫຼາຍອັນ ທີ່ຈົ່ງແມ່ນກ່ອງຄືນຫາທີ່ນຸ່ງເຄື່ອງຕາມແຟຊັ້ນ, ແລະ ຄຳຕອບທີ່ແນ່ນອນ ກວດສອບໄດ້ — ຜົນບວກ, ຍອດເງິນ, ບັນຊີ — ແມ່ນວຽກທີ່ຜິດຟາຍິດຕົວ ສຳລັບລະບົບທີ່ຖືກ 97% ຂອງເວລາ. ປະໂຫຍກລະດັບການສຳພາດວຽກ: *RAG ຄືບັນຫາການຄົ້ນຫາ ທີ່ໜີບຕິດກັບ API ສະຫຼຸບຄວາມ.*

ຄິດໄລ່ຕື້ນຫີນການອອກແບບ ກ່ອນທີ່ໃຜຈະລົງມືສ້າງ

Module ເສົາຄ້າດ້ານປະສິດທິພາບ ແລະ ຕື້ນຫີນ (Pillars of Performance and Cost) ຂອງຫຼັກສູດ ບັດນິປົດ ທ້າຍດ້ວຍບົດຝຶກອອກແບບເຕັມຮູບແບບ: ການປະເມີນພາຍໃຕ້ຄວາມກົດດັນ. ໂຈດ: ຝ່າຍນຳຕ້ອງການປະເມີນຕື້ນຫີນລາຍເດືອນ ຂອງຜະລິດຕະພັນຈັດການເອກະສານ ກ່ອນຈະອະນຸມັດການສ້າງ. Workload — ຜູ້ໃຊ້ລົງທະບຽນ 200,000 ຄົນ, ໃຊ້ງານປະຈຳມື້ 40,000 ຄົນ, ແຕ່ລະຄົນສ້າງປະມານ 25 API request ຕໍ່ມື້ ແລະ upload ເອກະສານສະເລ່ຍ 2 megabyte ຕໍ່ເດືອນ, ເກັບຮັກສາໄວ້ເຈັດປີ. ການອອກແບບ: ຊັ້ນ API ທີ່ມີ load balancer ຂ້າມສອງ zone, ຖານຂໍ້ມູນ PostgreSQL ແບບ managed multi-zone, object storage, ການສົ່ງອອກ 50

gigabyte ທຸກຄືນໃຫ້ partner, ແລະ ສະພາບແວດລ້ອມ development ແລະ staging ແຍກຕ່າງຫາກ. ໜ້າ
ວຽກ: ການປະເມີນລາຍເດືອນໜຶ່ງໜ້າ ເປັນຕົວເລກມື້ໜຶ່ງ, ພ້ອມສົມມຸດຕິຖານທີ່ຂຽນເປັນລາຍລັກອັກສອນ ກ່ອນທີ່
ລາຄາໃດໆຈະປາກົດ, ສະແດງເລກຄິດໄລ່ຄວາມສາມາດຮອງຮັບ (ຜູ້ໃຊ້ → request ຕໍວິນາທີ → ຈຳນວນ
instance), ແລະ ສະຖານະການ ມື້ນີ້ / 2× / 10×.

ລາຍການກວດຂອງບົດຝຶກສອນວິໄນນີ້: ຄິດລາຄາ compute ຈາກ load ສູງສຸດ (peak), ບໍ່ແມ່ນຄ່າສະເລ່ຍ; ນັບ
standby ຂອງ database ນຳ; ຄິດ storage ເປັນການເຕີບໂຕຕະຫຼອດເຈັດປີ ພ້ອມຊັ້ນ lifecycle; ຄິດໄລ່ egress
ແທນທີ່ຈະເດົາມັນ — ແຖວທີ່ການປະເມີນສ່ວນຫຼາຍສົມ ແລະ ເປັນແຖວທີ່ຄອບງຳຜົນລວມເລື້ອຍທີ່ສຸດ; ຢ່າລືມ
NAT (Network Address Translation) gateway, ສະພາບແວດລ້ອມທີ່ບໍ່ແມ່ນ production, ການສຳຮອງຂໍ້ມູນ,
ການເຝົ້າຕິດຕາມ, ແລະ ແຜນ support; ນຳສ່ວນຫຼຸດແບບຜູກມັດໄລຍະຍາວ (commitment discounts) ໃຊ້ກັບ
ພຽງ baseline ທີ່ຄົງທີ່ເທົ່ານັ້ນ; ແລະ ຈົບດ້ວຍຊ່ວງຕົວເລກ ພ້ອມລະບຸປັດໄຈຂັບເຄື່ອນ ບວກກັບການກວດຄວາມ
ສົມເຫດສົມຜົນແບບ unit economics — ຕົ້ນທຶນຕໍ່ຜູ້ໃຊ້ທີ່ໃຊ້ງານ ຕໍ່ເດືອນ. ຕົວເລກແມ່ນຍ່າໂດດໆອັນດຽວ ຜິດ
ຕັ້ງແຕ່ມື້ທີ່ມັນຖືກເວົ້າອອກມາ.