

หลักสูตรวิศวกรคลาวด์ (The Cloud Engineer Course)

จากความรู้ศูนย์สู่วิศวกรคลาวด์ที่พร้อมทำงานจริง — หลักสูตรในชุด B4LCILC

13 สิงหาคม 2569

หลักสูตรวิศวกรคลาวด์ (The Cloud Engineer Course)

จากความรู้ศูนย์ สู่วิศวกรคลาวด์ที่พร้อมเข้าทำงานได้จริง

หลักสูตรนี้ทำงานอย่างไร

คุณกำลังฝึกเพื่อเป็นวิศวกรคลาวด์: คนที่สร้าง server (เซิร์ฟเวอร์) ด้วยโค้ด ดูแลระบบให้มีชีวิตอยู่แม้นอนดีสาม ใช้ระบบอัตโนมัติทำงานน่าเบื่อ และมองระบบที่พังแล้วค้นหาสาเหตุได้อย่างใจเย็น นั่นคืออาชีพช่างฝีมือที่ต้องลงมือทำจริง และหลักสูตรนี้ก็ปฏิบัติต่อมันแบบนั้น คุณจะใช้เวลาพิมพ์มากกว่าอ่านหลายเท่า — ทุกโมดูลมี lab (แล็บ — แบบฝึกปฏิบัติ) และ lab เหล่านั้นแหละคือตัวหลักสูตร การอ่านเรื่องคลาวด์สร้างได้แค่ความคุ้นเคย แต่การลงมือสร้างบนคลาวด์ต่างหากที่สร้างอาชีพ

หลักสูตรแบ่งเป็นห้าช่วง **ช่วงที่ 0 (สัปดาห์ 1–4): รากฐาน** — คอมพิวเตอร์และเครือข่ายทำงานกันอย่างไรจริง ๆ และ Linux ระบบปฏิบัติการที่คลาวด์ทั้งโลกรันอยู่ **ช่วงที่ 1 (สัปดาห์ 5–12): แกนกลางของคลาวด์ แบบลงมือทำ** — บริการ AWS หัวตัวที่ปรากฏในประกาศรับสมัครงานทุกใบ เรียนรู้ผ่าน lab ที่คุณสร้างขึ้นแล้วหรือที่ **ช่วงที่ 2 (สัปดาห์ 13–20): ระบบอัตโนมัติ** — การเขียนสคริปต์, Git, Terraform, CI/CD และ container: ทักษะที่แยกวิศวกรออกจากคนที่ได้แต่คลิกหน้าคอนโซล **ช่วงที่ 3 (สัปดาห์ 21–28): งานปฏิบัติการ** — การเฝ้าระวังระบบ การรับมือเหตุขัดข้อง ความปลอดภัย และต้นทุน: ทักษะที่เขาจ้างคุณมาทำจริง ๆ **ช่วงที่ 4 (เดือน 8–12): ความพร้อมเข้าทำงาน** — โปรเจกต์พอร์ตโฟลิโอสามชิ้น ใบรับรองสามใบ และการฝึกสัมภาษณ์งาน

กติกาตลอดหลักสูตร: เรียนวันละ 1.5–2 ชั่วโมง หกวันต่อสัปดาห์ — ความสม่ำเสมอชนะความหักโหม และนี่คืออาชีพช่างที่เรียนรู้ด้วยการทำซ้ำทุกวัน เหมือนการฝึกเครื่องดนตรี ทุกโมดูลจบด้วย **ตารางคำศัพท์** (คำที่คุณต้องเป็นเจ้าของให้ได้), **แบบฝึกพูดออกเสียง** (ประโยคที่คุณพูดซ้ำจนเป็นธรรมชาติ — การสัมภาษณ์งานคือการพูดไม่ใช้การเขียน), **แบบฝึกหัดลงมือทำ** และ **Milestone** (หมุดหมาย) ที่เป็นด่านกันความคืบหน้า: อย่าไปต่อจนกว่าจะทำได้ เพราะทุกช่วงยืนอยู่บนช่วงก่อนหน้า และตั้งแต่สัปดาห์ที่ 1 ให้จด **Engineering Journal** (สมุด

บันทึกงานวิศวกรรม): ทุก lab ทุกข้อความ error ทุกการแก้ไข เขียนอย่างซื่อสัตย์หนึ่งย่อหน้า พอถึงเดือนที่ 10 สมุดเล่มนั้นจะกลายเป็นวัตถุดิบของพอร์ตโฟลิโอและเรื่องเล่าในการสัมภาษณ์ของคุณ

คำสัญญาหนึ่งข้อเป็นการตอบแทน: ไม่มีส่วนไหนของหลักสูตรนี้ที่ตั่งสมมติฐานว่าคุณรู้อะไรมาก่อน ทุกคำศัพท์ถูกนิยามตั้งแต่ครั้งแรกที่ปรากฏ ถ้าคุณใช้เว็บเบราว์เซอร์เป็น และยอมพิมพ์คำสั่งที่รู้สึกแปลกหูแปลกตาในสองสัปดาห์แรกได้ คุณก็มีคุณสมบัติครบทุกข้อเท่าที่จำเป็นแล้ว

ประกาศรับสมัครงานที่คุณกำลังฝึกเพื่อให้ตรงตาม

ในเดือนสิงหาคม 2569 เราได้รวบรวมประกาศรับสมัครงาน Cloud Engineer จริง ๆ จากเทมเพลตของนายจ้างและคู่มือการจ้างงาน (Arc.dev, Wiz, DevsData, XOPA, Betterteam — รายการเต็มอยู่ในส่วน Sources) เมื่อลอกชื่อบริษัทออก ข้อกำหนดชุดเดิมก็ปรากฏซ้ำแล้วซ้ำเล่า ตารางนี้คือสัญญาะหว่างคุณกับหลักสูตร — ทุกหัวข้อที่นายจ้างขอ ถูกจับคู่กับช่วงที่สอนมัน:

สิ่งที่ประกาศรับสมัครงานจริงขอ (เกือบคำต่อคำ)	หลักสูตรนี้สอนตรงไหน
“Knowledge of Linux/Unix operating systems” (ความรู้ระบบปฏิบัติการ Linux/Unix)	ช่วงที่ 0, โมดูล 1
“Expertise in cloud networking including VPCs, subnets, load balancers, DNS” (เชี่ยวชาญเครือข่ายคลาวด์ รวมถึง VPC, subnet, load balancer, DNS)	ช่วงที่ 0 โมดูล 2 + ช่วงที่ 1 Lab 3
“Demonstrated expertise in core AWS services, including EC2, S3, RDS, VPC, IAM” (เชี่ยวชาญบริการหลักของ AWS ที่พิสูจน์ได้ รวมถึง EC2, S3, RDS, VPC, IAM)	ช่วงที่ 1 (Lab 1–5)
“Design, develop, and deploy cloud infrastructure using infrastructure as code tools such as Terraform, CloudFormation” (ออกแบบ พัฒนา และ deploy โครงสร้างพื้นฐานคลาวด์ด้วยเครื่องมือ infrastructure as code เช่น Terraform, CloudFormation)	ช่วงที่ 2, โมดูล 7
“Proficiency in scripting languages such as Python, Bash, PowerShell, or Go” (คล่องภาษาสคริปต์ เช่น Python, Bash, PowerShell หรือ Go)	ช่วงที่ 2, โมดูล 6
“Build and maintain CI/CD pipelines using tools like Jenkins, GitLab CI, or GitHub Actions” (สร้างและดูแล CI/CD pipeline ด้วยเครื่องมืออย่าง Jenkins, GitLab CI หรือ GitHub Actions)	ช่วงที่ 2, โมดูล 8

สิ่งที่ประกาศรับสมัครงานจริงขอ (เกือบคำต่อคำ)	หลักสูตรนี้สอนตรงไหน
“Experience with containerization technologies including Docker and Kubernetes” (ประสบการณ์เทคโนโลยี containerization รวมถึง Docker และ Kubernetes)	ช่วงที่ 2, โมดูล 8
“Monitor infrastructure health and performance using cloud-native monitoring tools” (เฝ้าระวังสุขภาพและประสิทธิภาพของโครงสร้างพื้นฐานด้วยเครื่องมือ monitoring แบบ cloud-native)	ช่วงที่ 3, โมดูล 9
“Participate in incident response, including log analysis”; “troubleshooting and analytical skills” (ร่วมมือเหตุขัดข้อง รวมถึงการวิเคราะห์ log; ทักษะการแก้ปัญหาและการวิเคราะห์)	ช่วงที่ 3, โมดูล 10
“Implement and enforce security controls including encryption, identity and access management”; “least-privilege access” (วางและบังคับใช้มาตรการความปลอดภัย รวมถึงการเข้ารหัสและการจัดการตัวตนกับสิทธิ์เข้าถึง; สิทธิ์แบบ least-privilege)	ช่วงที่ 1 Lab 5 + ช่วงที่ 3 โมดูล 10
“Manage cloud costs through rightsizing resources, implementing auto-scaling, resource tagging” (จัดการต้นทุนคลาวด์ด้วยการ rightsize ทรัพยากร ใช้ auto-scaling และติด tag ทรัพยากร)	ช่วงที่ 3, โมดูล 10
“Maintaining, testing and implementing disaster recovery procedures” (ดูแล ทดสอบ และดำเนินการตามขั้นตอน disaster recovery)	ช่วงที่ 3 + โพรเจกต์พอร์ตโฟลิโอ 3
“AWS certifications preferred” (มีใบรับรอง AWS จะพิจารณาเป็นพิเศษ)	ตารางสอบใบรับรองช่วงที่ 4 (CCP → SAA → Terraform Associate)
“Provide technical guidance and documentation”; “good communication and collaboration skills” (ให้คำแนะนำทางเทคนิคและจัดทำเอกสาร; ทักษะการสื่อสารและการทำงานร่วมกันที่ดี)	Journal + runbook + การเตรียมสัมภาษณ์ช่วงที่ 4

บริบทเรื่องเงินเดือน เพื่อให้รู้ว่าคุณกำลังมุ่งไปสู่อะไร: เงินเดือนวิศวกรคลาวด์ในสหรัฐฯ กระจุกตัวอยู่แถว **ค่ามัธยฐานใกล้ \$104,000** ต่อปี โดยช่วงทั่วไปคือ **\$85K–\$140K** และตำแหน่ง AWS ระดับอาวุโส/ผู้เชี่ยวชาญประกาศรับสูงกว่านั้นมาก ตำแหน่งระดับเริ่มต้นมีอยู่ภายใต้หลายชื่อ — cloud support associate, junior cloud engineer, cloud operations engineer — และหลักสูตรนี้เล็งเป้าไปที่ข้อกำหนดของตำแหน่งเหล่านั้นพอดี

ช่วงที่ 0 — รากฐาน (สัปดาห์ 1–4)

โมดูล 1 (สัปดาห์ 1–2): คอมพิวเตอร์ทำงานอย่างไร และ Linux — ภาษาของ server

แนวคิดใหญ่: server ก็คือคอมพิวเตอร์ที่มีหน้าที่ให้บริการคอมพิวเตอร์เครื่องอื่น แล็บที่อปตรงหน้าคุณกับเครื่องที่รัน Netflix ต่างกันที่ขนาดและความทนทาน ไม่ได้ต่างกันโดยชนิด: ทั้งคู่คือ **CPU** (ส่วนที่ทำงานประมวลผล), **memory/RAM** (หน่วยความจำ — พื้นที่ทำงานชั่วคราวความเร็วสูง ถูกล้างเมื่อรีสตาร์ท), **disk** (ดิสก์ — ที่เก็บข้อมูลระยะยาวที่ช้ากว่าแต่อยู่รอดผ่านการรีสตาร์ท) และ **network card** (การ์ดเครือข่าย — จุดเชื่อมต่อกับทุกสิ่ง) ทั้งหมดประสานงานโดย **operating system (OS)** (ระบบปฏิบัติการ) แล็บที่อปของคุณน่าจะรัน Windows หรือ macOS แต่ server แทบทั้งหมดรัน **Linux** — OS แบบโอเพนซอร์สที่ฟรี เสถียร เขียนสคริปต์ควบคุมได้ และสั่งงานทั้งหมดด้วยคำสั่งที่พิมพ์ ประเด็นอยู่ที่ข้อสุดท้ายนั้นแหละ: คุณทำให้การคลิกเมาส์เป็นอัตโนมัติไม่ได้ แต่ทำให้คำสั่งเป็นอัตโนมัติได้ และวิศวกรรมคลาวด์คือการทำให้เป็นอัตโนมัติ ดังนั้นความคล่องแรกของคุณคือ **terminal** (เทอร์มินัล)

เปิดมัน terminal ให้หายสับสน terminal (หรือ “shell” — โปรแกรมข้างในมันมักจะเป็น **Bash**) คือบทสนทนาแบบข้อความกับคอมพิวเตอร์ คุณพิมพ์คำสั่ง มันตอบกลับ แค่นั้นเอง เครื่องหมาย \$ ที่คุณเห็นในตัวอย่างคือ **prompt** (พรอมต์) — shell กำลังบอกว่า “ตาคุณแล้ว” ทุกอย่างใน Linux คือไฟล์ ไฟล์อาศัยอยู่ในต้นไม้เดียวที่เริ่มจากราก / และโพลเดอร์ส่วนตัวของคุณคือ /home/yourname (ชื่อเล่น: ~) ส่วน **path** (พาธ) คือที่อยู่ของไฟล์ในต้นไม้: /home/anna/notes.txt

หา Linux ไว้ฝึกสักเครื่อง (เลือกหนึ่งทาง ใช้เวลาสิบนาที): บน Windows ติดตั้ง **WSL** (Windows Subsystem for Linux — Ubuntu Linux ของจริงที่อยู่ใน Windows: <https://learn.microsoft.com/en-us/windows/wsl/install>); บน Mac แอป Terminal ที่ติดมากับเครื่องก็ใกล้เคียงพอจะเริ่มได้ (macOS เป็นญาติตระกูล Unix); หรือรอถึงสัปดาห์ที่ 5 ที่คุณจะเช่า Linux server ของจริงจาก AWS ได้ฟรี สำหรับคนส่วนใหญ่ WSL คือคำตอบที่ดีที่สุด

คำสั่งยอดนิยม 25 ตัว — ตารางนี้คือเนื้อหาสัปดาห์ 1–2 ทั้งหมด จงพิมพ์ทุกตัว หลาย ๆ รอบ:

คำสั่ง	ทำหน้าที่อะไร	ตัวอย่าง
pwd	Print working directory — “ตอนนี้ฉันอยู่ที่ไหน?”	pwd
ls	แสดงรายการไฟล์ตรงนี้ (-l รายละเอียดแบบยาว, -a รวมไฟล์ซ่อน)	ls -la
cd	Change directory — เดินทางไปมาในต้นไม้ไฟล์	cd /var/log

คำสั่ง	ทำหน้าที่อะไร	ตัวอย่าง
mkdir	สร้าง directory (โฟลเดอร์)	mkdir projects
touch	สร้างไฟล์เปล่า	touch notes.txt
cp	คัดลอกไฟล์ (-r สำหรับโฟลเดอร์)	cp a.txt backup.txt
mv	ย้ายหรือเปลี่ยนชื่อ	mv old.txt new.txt
rm	ลบ — ถาวร ไม่มีถังขยะ จงเคารพมัน	rm notes.txt
cat	พิมพ์เนื้อหาทั้งหมดของไฟล์ออกมา	cat notes.txt
less	อ่านไฟล์ยาวทีละหน้า (กด q เพื่อออก)	less /var/log/syslog
head / tail	บรรทัดแรก / บรรทัดท้ายของไฟล์ ส่วน tail -f เพื่อดู log สด ๆ — คลาสสิกของงาน ops	tail -f app.log
grep	ค้นหาข้อความตามรูปแบบ — คำสั่ง ops ที่ถูกใช้มากที่สุดเพียงหนึ่งเดียว	grep "ERROR" app.log
find	หาไฟล์ตามชื่อ/ขนาด/อายุ	find / -name "*.conf"
echo	พิมพ์ข้อความ (มักพิมพ์ลงไฟล์หรือตัวแปร)	echo "hello"
nano	โปรแกรมแก้ไขข้อความในเทอร์มินัลที่เป็นมิตร	nano notes.txt
man	คู่มือของทุกคำสั่ง (กด q เพื่อออก)	man grep
sudo	รันหนึ่งคำสั่งในฐานะแอดมินผู้ทรงอำนาจ ("root") ด้วยความเคารพ	sudo apt update
apt	ติดตั้ง/อัปเดตซอฟต์แวร์ (ตระกูล Ubuntu/Debian)	sudo apt install http
chmod	เปลี่ยน permission (สิทธิ์) ของไฟล์	chmod 644 notes.txt
chown	เปลี่ยนเจ้าของไฟล์	sudo chown anna file
ps	แสดง process ที่กำลังรัน (ps aux เพื่อดูทั้งหมด)	ps aux

คำสั่ง	ทำหน้าที่อะไร	ตัวอย่าง
top (หรือ htop)	แดชบอร์ดสดของ CPU/หน่วยความจำ/process — คำถาม “ทำไม server ช้า?” เริ่มที่นี่	top
df -h / du -sh	พื้นที่ดิสก์ที่เหลือ / พื้นที่ที่โพลเดอร์หนึ่งใช้ — คำถาม “ดิสก์เต็ม” เริ่มที่นี่	df -h
ssh	ล็อกอินเข้า terminal ของเครื่องอื่นผ่านเครือข่าย — ประตุน้ำบ้านของวิศวกรคลาวด์	ssh anna@server-ip
curl	ยิงคำขอเว็บจากเทอร์มินัล — “เว็บยังขึ้นอยู่ไหม?”	curl https://example.com

Permission ฉบับ 60 วินาที ทุกไฟล์มีเจ้าของและ mode หน้าตาแบบ `rwxr-xr--`: สามชุด ๆ ละสามตัว — เจ้าของ, กลุ่ม, คนอื่นทั้งหมด — ของสิทธิ์ **read** (อ่าน), **write** (เขียน), **execute** (รัน) ในรูปตัวเลข: `r=4, w=2, x=1` ดังนั้น `chmod 755 script.sh` แปลว่า “เจ้าของทำได้ทุกอย่าง ($7=4+2+1$); คนอื่นอ่านและรันได้ ($5=4+1$)” เมื่อโปรแกรม “ไม่มี permission” นั่นคือระบบกำลังปฏิเสธ — และตอนนี้คุณอ่านออกแล้วว่าเพราะอะไร

SSH ฉบับ 60 วินาที `ssh` เปิด terminal ระยะไกลที่ปลอดภัยบนเครื่องอื่น — จากเล็บที่ขอบของคุณเข้าไปใน server ที่เวอร์จิเนียราวกับคุณนั่งอยู่หน้าเครื่อง แทนที่จะใช้รหัสผ่าน มีอาชีวะใช้ **key pair** (คู่กุญแจ): private key (ไฟล์ลับบนเล็บที่ขอบของคุณ ห้ามแชร์เด็ดขาด) กับ public key (วางไว้บน server) ทั้งคู่สวมกันพอดีเหมือนลูกกุญแจกับแม่กุญแจ ทุก server บน AWS ที่คุณเปิดในช่วงที่ 1 จะยื่นสิ่งนี้ให้คุณเป๊ะ ๆ

Process และ service: process (โพรเซส) คือโปรแกรมที่กำลังรัน ส่วน **service** (เซอร์วิส หรือ “daemon”) คือ process ที่รันตลอดไปอยู่เบื้องหลัง — web server, database คำสั่ง `systemctl status nginx` คือการถาม Linux ว่า “service nginx แข็งแรงดีไหม?” — ประโยคที่คุณจะพิมพ์ในฐานะมืออาชีวะไปอีกหลายปี

คำศัพท์:

คำศัพท์	ความหมาย
Server	คอมพิวเตอร์ที่มีหน้าที่ให้บริการคอมพิวเตอร์เครื่องอื่นบนคลาวด์คือเครื่องที่คุณเช่า
CPU / RAM / disk	คนทำงาน, พื้นที่ทำงานชั่วคราวความเร็วสูง (ถูกล้างเมื่อรีสตาร์ท) และที่เก็บข้อมูลถาวรที่ช้ากว่า
Operating system (OS)	ซอฟต์แวร์ที่ควบคุมเครื่องและเป็นบ้านให้โปรแกรมต่าง ๆ server รัน Linux

คำศัพท์	ความหมาย
Linux / distribution	OS ฝั่ง server แบบโอเพนซอร์สที่ฟรี ส่วน “distro” (Ubuntu, Amazon Linux, Debian) คือรสชาติหนึ่งที่แพ็คเกจมาให้ใช้
Terminal / shell / Bash	หน้าจอข้อความสำหรับคุยกับ OS / โปรแกรมที่ดีความคำสั่งของคุณ / ชื่อของ shell มาตรฐาน
Prompt	เครื่องหมาย \$ — shell ที่กำลังรอคำสั่งของคุณ
Directory / path	โฟลเดอร์ / ที่อยู่เต็มของไฟล์ในต้นไม้เดียวที่เริ่มจาก /
Root (สองความหมาย)	ยอดบนสุดของต้นไม้ไฟล์ (/) และผู้ใช้แอดมินผู้ทรงอำนาจ บริบทจะบอกเองว่าหมายถึงตัวไหน
sudo	“Superuser do” — รันหนึ่งคำสั่งด้วยอำนาจแอดมิน
Permissions (rwx)	กฎรายไฟล์ว่าใครอ่าน เขียน รันได้บ้าง — แสดงเป็นสามชุดสำหรับเจ้าของ/กลุ่ม/คนอื่นทั้งหมด
Process / service (daemon)	โปรแกรมที่กำลังรัน / โปรแกรมที่รันตลอดไปอยู่เบื้องหลัง (web server, database)
SSH / key pair	การล็อกอินระยะไกลอย่างปลอดภัยเข้า terminal ของเครื่องอื่น / คู่ไฟล์กุญแจ private+public ที่ใช้แทนรหัสผ่าน
Log	ไฟล์ข้อความที่ซอฟต์แวร์จดว่าเกิดอะไรขึ้น — ที่แรกที่คุณดูเมื่ออะไรก็ตามพัง
Package manager	ตัวติดตั้งของ OS (apt, yum) — ได้ซอฟต์แวร์ด้วยคำสั่ง ไม่ใช่ด้วยหน้าเว็บดาวน์โหลด

วิดีโอประจำโมดูลนี้ (ตรวจสอบลิงก์แล้ว):

วิดีโอ	ช่อง	ความยาว	ลิงก์
Linux Operating System — Crash Course for Beginners	freeCodeCamp	~2 ชม.	https://www.youtube.com/watch?v=ROjZy1WbCIA
WSL install guide (เอกสารอ้างอิง ไม่ใช่วิดีโอ)	Microsoft Learn	—	https://learn.microsoft.com/en-us/windows/wsl/install

พูดออกเสียงจนเป็นธรรมชาติ: “Let me SSH in and check the logs.” (ขอ SSH เข้าไปเช็ค log ก่อนนะ) · “Grep the log for the error, then tail -f it while we retry.” (grep หา error ใน log แล้ว tail -f ดูสด ๆ ระหว่างเราลองใหม่) · “It’s a permissions problem — who owns the file and what’s the mode?” (เป็นปัญหา permission — ใครเป็นเจ้าของไฟล์ แล้ว mode คืออะไร?) · “Check top — is it CPU, memory, or disk?” (เช็ค top สิ — เป็นที่ CPU หน่วยความจำ หรือดิสก์?)

แบบฝึกหัด: (1) ใน terminal Linux ของคุณ สร้างต้นไม้โปรเจกต์เล็ก ๆ ด้วย mkdir กับ touch คัดลอกและย้ายสิ่งต่าง ๆ แล้วลบทิ้ง — พร้อมบรรยายแต่ละคำสั่งออกเสียงดัง ๆ (2) สร้างไฟล์ hello.sh ที่มีเนื้อหา echo "hello from \$(whoami)" ทำให้รันได้ด้วย chmod +x แล้วรันด้วย ./hello.sh (3) รัน tail -f กับไฟล์ log สักไฟล์ (บน Ubuntu: sudo tail -f /var/log/syslog) แล้วเฝ้าดูบรรทัดใหม่ทยอยมาถึง (4) ลงสมุดบันทึก: อธิบายให้เพื่อนในจินตนาการฟังว่าทำไม server ถึงรัน Linux ในสามประโยค

Milestone: โดยไม่ต้องเปิดเน็ต คุณเดินทางไปที่ไหนก็ได้ในต้นไม้ไฟล์ สร้าง/คัดลอก/ย้าย/ลบไฟล์ได้ อธิบาย chmod 755 ได้ และใช้ grep หาค่าในไฟล์ได้ ถ้าข้อไหนยังต้องเปิดหาข้อมูล ให้อยู่ตรงนี้อีกสองวัน โมดูลนี้คือเสารับน้ำหนักของทุกสิ่งตามมา

โมดูล 2 (สัปดาห์ 3–4): พื้นฐานเครือข่าย + บัญชี AWS ของคุณ

แนวคิดใหญ่: เครือข่ายก็คือคอมพิวเตอร์ที่ส่งของจดหมายจำหน้าถึงกัน ทุกเครื่องได้รับ **IP address** (เช่น 172.31.8.14 — ที่อยู่บ้านในรูปตัวเลข) ข้อมูลถูกสับเป็น **packet** (แพ็กเก็ต — ของจดหมาย) แล้วถูกส่งต่อทีละช่วงไปยังที่อยู่ปลายทาง เมื่อถึงที่หมาย หมายเลข **port** (พอร์ต) จะบอกว่าของนี้ส่งถึงโปรแกรมไหน — ตึกเดียวกัน แต่มีประตูนับพันบานที่ติดหมายเลขไว้: พอร์ต **22** คือ SSH, **80** คือเว็บไม่เข้ารหัส (**HTTP**), **443** คือเว็บเข้ารหัส (**HTTPS**), **5432** คือ PostgreSQL คำว่า “เปิดพอร์ต 443” จึงแปลว่า “อนุญาตของที่จำหน้าถึงประตู 443”

Private กับ public: บ้านของคุณและเครือข่ายคลาวด์ทุกแห่งใช้ **ช่วง private IP** ซ้ำกันได้ (10.x.x.x, 172.16–31.x.x, 192.168.x.x) ซึ่งใช้ได้เฉพาะภายในเครือข่ายท้องถิ่น ส่วน **public IP** เข้าถึงได้จากอินเทอร์เน็ตทั่วโลก การแบ่งแบบนี้คือรากฐานของความปลอดภัยบนคลาวด์: สิ่งที่ไม่จำเป็นต้องหันหน้าเข้าหาอินเทอร์เน็ต ก็ไม่ต้องมีที่อยู่ public เลย

DNS — สมุดโทรศัพท์ของอินเทอร์เน็ต มนุษย์ใช้ชื่อ (example.com) แต่ packet ต้องการตัวเลข **DNS** ทำหน้าที่แปล: เครื่องของคุณถาม DNS server ว่า “IP ของ example.com คืออะไร?” ได้ตัวเลขมา แล้วค่อยเชื่อมต่อ ครั้งหนึ่งของเหตุขัดข้องลึกลับทั้งหลายเกี่ยวข้องกับ DNS; มุกในวงการที่ว่า “it’s always DNS” (มันเป็นเพราะ DNS เสมอแหละ) มีอยู่จริงก็เพราะมันจริงบ่อย ๆ คำสั่ง nslookup example.com คือการทำ lookup ด้วยมือ

HTTP — วิธีที่เว็บพูดคุยกับ client (ไคลเอนต์ — เบราว์เซอร์) ส่งคำขอ — ประกอบด้วย **method** (GET = ดึงข้อมูล, POST = ส่งข้อมูล) บวก path — แล้ว **server** ตอบกลับด้วย **status code** (รหัสสถานะ): 200 OK, 301 ย้ายที่แล้ว, 403 ห้ามเข้า, 404 หาไม่เจอ, 500 server มีปัญหา, 502/503 “server ข้างหลังฉันพัง/รับงานล้นมือ” จงท่องหกด่วนนั้นให้ขึ้นใจ เพราะในฐานะวิศวกรคุณจะอ่านมันทุกวัน curl -I https://example.com แสดง status line จริง ๆ พร้อม header ให้คุณดู

Firewall: รายการกฎที่ตัดสินว่า packet ไหนผ่านได้ ตามต้นทาง ปลายทาง และพอร์ต — “อนุญาต 443 จากทุกที่; อนุญาต 22 เฉพาะจากออฟฟิศ; ที่เหลือปฏิเสธ” ใน AWS ตัว firewall รายเครื่องเรียกว่า **security group** และ security group ที่ตั้งค่าผิดคือช่องโหว่ความปลอดภัยอันดับ 1 ของมือใหม่ ปิดท้ายด้วยคำศัพท์ **latency** (ความหน่วง — ดีเลย์ หน่วยเป็นมิลลิวินาที) และ **bandwidth** (แบนด์วิดท์ — ความจุต่อวินาที): ระยะทางสร้าง latency นี้คือเหตุผลที่คลาวด์มี region กระจายทั่วโลก

คราวนี้ถึงบัญชี AWS ของคุณ — ครึ่งหลังของโมดูลนี้ ไปที่ <https://aws.amazon.com/free> แล้วสร้างบัญชี Free Tier (ใช้อีเมล เบอร์โทร บัตรเครดิต/เดบิตเพื่อยืนยันตัวตน — บัตรจะไม่ถูกเรียกเก็บอย่างมีนัยสำคัญถ้าคุณรักษาวินัยการรื้อถอน (teardown) ตามหลักสูตรนี้) **Free Tier** ให้โควตารายเดือนของบริการพื้นฐาน (รวมถึง EC2 server ขนาดเล็ก 750 ชั่วโมง/เดือนในปีแรก) และทุก lab ในหลักสูตรนี้ออกแบบมาให้อยู่ในโควตานี้ จากนั้น ก่อนทำอย่างอื่นใด มีสามขั้นตอนความปลอดภัยที่ต้องรองไม่ได้ — การทำสามข้อนี้คือแบบฝึกหัดแรกของอาชีพสายความปลอดภัยของคุณ:

- 1. **เปิด MFA ให้ root user** root user คือกุญแจดอกแม่ของบัญชี เพิ่ม multi-factor authentication (แอป authenticator บนโทรศัพท์) ใน IAM → Security credentials แล้วเลิกใช้ root สำหรับงานประจำวัน
- 2. **สร้าง IAM user ระดับแอดมิน** (Lab 5 จะอธิบาย IAM อย่างลึก; ตอนนี้เอาแค่: console → IAM → Users → สร้าง user พร้อมสิทธิ์แอดมิน) แล้วล็อกอินด้วย user นั้นตั้งแต่นี้เป็นต้นไป
- 3. **ตั้ง billing alert** (การแจ้งเตือนค่าใช้จ่าย) Console → Billing → Budgets → สร้าง **zero-spend budget** (เทมเพลตของ AWS ที่ส่งอีเมลถึงคุณทันทีที่มีอะไรก็ตามเริ่มคิดเงิน) และเปิด Free Tier usage alerts ด้วยวิศวกรที่คุมต้นทุนไม่ได้คือภาระ; คุณเพิ่งใช้บัญชีมา 20 นาทีแต่นำหน้ามืออาชีพหลายคนแล้ว (อ้างอิง: <https://docs.aws.amazon.com/cost-management/latest/userguide/budgets-managing-costs.html>)

คำศัพท์:

คำศัพท์	ความหมาย
IP address	ที่อยู่เครือข่ายในรูปตัวเลขของเครื่อง เช่น 172.31.8.14
Packet	ซองข้อมูลจำหน้าแล้วหนึ่งซอง; ทราฟฟิกทั้งหมดคือสายธารของมัน
Port	ประตูติดหมายเลขบนเครื่อง ระบุว่าทราฟฟิกส่งถึงโปรแกรมไหน: 22 SSH, 80 HTTP, 443 HTTPS
Private / public IP	ที่อยู่ที่ใช้ได้เฉพาะในเครือข่ายท้องถิ่น / ที่อยู่ที่สามารถเข้าถึงได้จากอินเทอร์เน็ตทั่วโลก
DNS	ระบบแปลงชื่อ (example.com) เป็น IP address — “It’s always DNS.”
HTTP / HTTPS	โพรโทคอลคำขอ-คำตอบของเว็บ / ตัวเดียวกันแต่เข้ารหัสด้วย TLS

คำศัพท์	ความหมาย
Status code	บทสรุปคำตอบของ server: 200 OK, 404 หาไม่เจอ, 500 server มีปัญหา, 503 รับงานล้มมือ
Firewall	รายการกฎที่ตัดสินว่า packet ไหนผ่านได้ ตามต้นทางปลายทาง พอร์ต
Security group	firewall รายเครื่องของ AWS การตั้งค่าผิดคือช่องโหว่คลาสสิกของมือใหม่
Latency / bandwidth	ความหน่วง (มิลลิวินาที) / ความจุ (ต่อวินาที) ระยะทางสร้าง latency
Client / server (บทบาท)	ผู้ถามและผู้ตอบในทุกบทสนทนาบนเครือข่าย
AWS Free Tier	โควตาฟรีรายเดือนของบัญชี AWS ใหม่ — งบประมาณทั้งหมดของหลักสูตรนี้
Root user	ตัวตนแม่ของบัญชี AWS เปิด MFA ให้มัน แล้วเลิกใช้
Billing alert / budget	อีเมลอัตโนมัติเมื่อค่าใช้จ่ายข้ามเกณฑ์ ของคุณตั้งไว้ที่ \$0

วิดีโอประจำโมดูลนี้:

วิดีโอ	ช่อง	ความยาว	ลิงก์
What is AWS?	Amazon Web Services (ช่องทางการ)	~2 นาที	https://www.youtube.com/watch?v=a9__D53WsUs
Top 50+ AWS Services Explained in 10 Minutes	Fireship	~10 นาที	https://www.youtube.com/watch?v=JlbiYCM48to
AWS Networking Basics — VPC & Subnets	KodeKloud	~30 นาที	https://www.youtube.com/watch?v=QM63dyA_4Pc

พูดออกเสียง: “What’s the IP, and is it public or private?” (IP คืออะไร แล้วเป็น public หรือ private?) · “Is port 443 open in the security group?” (พอร์ต 443 เปิดอยู่ใน security group หรือเปล่า?) · “Curl it — what status code do you get?” (curl ดูสิ — ได้ status code อะไร?) · “Did DNS resolve? Check with nslookup before blaming the server.” (DNS resolve ไหม? เช็คด้วย nslookup ก่อนไปโทษ server)

แบบฝึกหัด: (1) ping google.com (สังเกต latency), nslookup google.com (สังเกตว่า DNS คืบ IP มาหลายตัว), curl -I https://aws.amazon.com (อ่าน status line กับ header สามตัว แล้วค้นความหมายของแต่ละตัว) (2) หา private IP ของเครื่องคุณ (ip addr บน Linux) และ public IP ของคุณ (ค้นคำว่า “what is my IP”) — อธิบายลงสมุดบันทึกว่าทำไมมันต่างกัน (3) ทำการตั้งค่าบัญชี AWS ให้ครบ: MFA, admin user, zero-spend budget — แปะหน้าจอ budget เก็บลงสมุดบันทึก (4) วาดจากความจำ: แล็บที่อป → DNS lookup → คำขอ HTTPS ผ่านพอร์ต 443 → firewall → web server วาดสามรอบจนเป็นอัตโนมัติ

Milestone — จบช่วงที่ 0: คุณเล่าได้ว่าเกิดอะไรขึ้นเมื่อพิมพ์ URL แล้วกด Enter — DNS, IP, พอร์ต, firewall, คำขอ HTTP, status code — ภายในสองนาทีก่อน โดยใช้ทุกคำศัพท์อย่างถูกต้อง; และบัญชี AWS ของคุณมีอยู่จริงพร้อม MFA กับ budget alert ที่ \$0 ตอนนี้นี้คุณรู้เรื่องกลไกของอินเทอร์เน็ตมากกว่าคนส่วนใหญ่ที่ใช้มันหาकिनแล้ว ช่วงที่ 1 คือจุดที่คุณเริ่มสร้างต่อยอดจากมัน

ช่วงที่ 1 — แกนกลางของคลาวด์ แบบลงมือทำ (สัปดาห์ 5–12)

ช่วงนี้ทำงานอย่างไร บริการทั้งห้าด้านล่างแต่ละตัวคือหนึ่ง **Lab**: มีเป้าหมาย, โครงร่างขั้นตอน (ละเอียดพอให้ทำตาม สั้นพอให้ต้องคิดเอง — การคิดนั่นแหละคือการเรียนรู้), สิ่งที่ได้เรียนรู้ และ — เสมอ — **teardown** (การรื้อถอน) วินัยการรื้อถอนสำคัญสองต่อ: มันทำให้คุณอยู่ใน Free Tier และ “ไม่ทิ้งอะไรที่ไม่จำเป็นให้ร้นค้างไว้” คือปฏิกิริยาอัตโนมัติของมืออาชีพที่ผู้สัมภาษณ์จับจ้องสร้างแต่ละ lab ซ้ำอย่างน้อยสองรอบ: รอบแรกตามโครงร่าง รอบสองจากความจำ รอบที่สองคือตอนที่ความรู้อยู่เข้าไปอยู่ในมือคุณ ตั้งบเวลาราวหนึ่งสัปดาห์ครึ่งต่อ lab; เวลาที่เหลือไว้เผื่อของพัง เพราะของจะพังแน่ ๆ และการติบักมันคือการสอนที่ดีที่สุดที่หลักสูตรนี้เขียนบทให้ไม่ได้

ก่อนอื่น สามแนวคิดที่วางกรอบทุกสิ่งที่คุณกำลังจะสร้าง **Region และ Availability Zone**: Region คือกลุ่ม data center ของ AWS ตามพื้นที่ภูมิศาสตร์ (เลือกอันที่ใกล้คุณแล้วอยู่กับมัน — ทรัพยากรใน region หนึ่งมองไม่เห็นจากอีก region ซึ่งเป็นความงงอันดับ 1 ของ “server ฉันทายไปไหน?”) ส่วน **Availability Zone (AZ)** คือ data center ที่แยกอิสระภายใน region; ระบบที่จริงจังจะรันในสอง AZ เพื่อให้อาคารเดียวพังโดยไม่ล้มระบบ **Console กับ CLI**: console (<https://console.aws.amazon.com>) คือหน้าเว็บควบคุมของ AWS — เยี่ยมสำหรับการเรียนรู้และการดู; ส่วน **AWS CLI** (คำสั่ง aws ในเทอร์มินัล) ทำได้ทุกอย่างที่ console ทำได้ แต่เขียนสคริปต์ได้ คุณจะเริ่มที่ console แล้วเลื่อนขึ้นไป CLI เพราะช่วงที่ 2 จะทำทุกอย่างที่คุณทำมือในช่วงนี้ให้เป็นอัตโนมัติ

Lab 1 (สัปดาห์ 5–6): EC2 — server เครื่องแรกของคุณ

EC2 (Elastic Compute Cloud) ให้เช่าเครื่องเสมือน เรียกว่า **instance** (อินสแตนซ์) นี่คือนิยามของวิศวกรรมคลาวด์: Linux server ของจริง บนอินเทอร์เน็ต ใน 60 วินาที ฟรี

เป้าหมาย: เปิด Linux server, SSH เข้าไป, ทำให้มันเสิร์ฟหน้าเว็บให้คนทั้งโลก แล้วทำลายมันทิ้ง

โครงร่างขั้นตอน: 1. Console → EC2 → Launch instance ตั้งชื่อ เลือก **Amazon Linux 2023** เป็น AMI (Amazon Machine Image — ดิสก์แม่แบบที่ server ของคุณบูตขึ้นมา) และ **t2.micro** หรือ **t3.micro** เป็น **instance type** (ขนาดเครื่อง; สองตัวนี้อยู่ใน Free Tier) 2. สร้าง **key pair**; โฟล์ private key นามสกุล .pem จะถูกดาวน์โหลดมา นั่นคือกุญแจ SSH จากโมดูล 1 — เก็บให้ดี แล้ว `chmod 400` มัน 3. ในการตั้งค่าเครือข่าย อนุญาต SSH (พอร์ต 22) จาก “My IP” เท่านั้น — ตอนนี้คุณรู้เป๊ะแล้วว่ากฎ security group ข้อนี้แปลว่าอะไร — และอนุญาต HTTP (พอร์ต 80) จากทุกที่ 4. กด Launch รอสถานะ “running” คัดลอก public IP แล้วจากเทอร์มินัลของคุณ: `ssh -i mykey.pem ec2-user@<public-ip>` สูดหายใจลึก ๆ: คุณอยู่ข้างในคอมพิวเตอร์เครื่องหนึ่งใน data center ของ AWS แล้ว 5. บน server: `sudo dnf install -y nginx && sudo systemctl start nginx && sudo systemctl enable nginx` แล้วเปิดเบราว์เซอร์ไปที่ `http://<public-ip>` — นั่นคือ web server ของคุณที่กำลังตอบคนทั้งโลก 6. เปลี่ยนหน้าเริ่มต้น: `echo "<h1>Built by me, on EC2</h1>" | sudo tee /usr/share/nginx/html/index.html` รีเฟรช แคปหน้าจอลงสมุดบันทึก 7. สํารวจแบบวิศวกร: `top, df -h, sudo tail -f /var/log/nginx/access.log` ระหว่างรีเฟรชหน้าเว็บ — ฝ้าดูการเข้าชมของตัวเองทยอยปรากฏใน log

สิ่งที่ได้เรียนรู้: AMI, instance type, key pair, security group แบบใช้งานจริง, SSH เข้า server ของจริง, ติดตั้งและรัน Linux service, อ่าน log ของมัน — พุดง่าย ๆ คือท่วงท่าทางกายภาพประจำวันของอาชีพนี้

Teardown: EC2 → เลือก instance → Instance state → **Terminate** ยืนยันจนขึ้นว่า “terminated” Free Tier ให้ micro instance หนึ่งเครื่อง 750 ชั่วโมง/เดือน ดังนั้นต่อให้เปิดทิ้งไว้ก็ยังไม่ถูกคิดเงิน — แต่จงรื้อถอนอยู่ดี เพราะประเด็นคือการสร้างนิสัย

Lab 2 (สัปดาห์ 7): S3 — โฟล์ที่ไม่มีวันหาย

S3 (Simple Storage Service) คือ object storage (พื้นที่เก็บอ็อบเจกต์): ถึงเก็บไฟล์ไว้กันปั้งที่หนานระดับสิบเอ็ดเลขเก้า เป็นคำตอบมาตรฐานของ “เราจะเก็บไฟล์ไว้ที่ไหน?” และเมื่อถูกตั้งคำถาม ก็เป็นต้นตอของเหตุข้อมูลรั่วไหลที่โด่งดังที่สุดในประวัติศาสตร์ — นั่นคือเหตุผลที่ lab นี้เป็นเรื่อง storage ครึ่งหนึ่ง ความปลอดภัยอีกครึ่งหนึ่ง

เป้าหมาย: สร้าง bucket, ใช้งานมันผ่าน CLI, โฮสต์เว็บไซต์ static เล็ก ๆ และเข้าใจอย่างถ่องแท้ว่า “public bucket” แปลว่าอะไร

โครงร่างขั้นตอน: 1. Console → S3 → Create bucket (ชื่อ bucket ต้องไม่ซ้ำกันทั้งโลก — `yourname-lab-2026` ใช้ได้) สังเกตว่า **Block Public Access** เปิดอยู่เป็นค่าเริ่มต้น อัปโหลดไฟล์อะไรก็ได้ผ่าน console แล้วดาวน์โหลดกลับมา 2. ติดตั้ง AWS CLI บนเครื่องคุณ แล้วรัน `aws configure` ด้วย **access key** ที่คุณสร้างให้ IAM admin user ของคุณ (access key คือชื่อผู้ใช้+รหัสผ่านแบบโปรแกรมสำหรับ API — ปฏิบัติต่อมันเหมือนรหัสผ่าน ห้ามใส่ในโค้ดเด็ดขาด; คุณจะชิมชั้บกฏนี้เข้ากระดูกใน Lab 5) 3. จากเทอร์มินัลของคุณ: `aws s3 ls · aws s3 cp notes.txt s3://yourname-lab-2026/ · aws s3 sync ./myfolder s3://yourname-lab-2026/backup/` สัมผัสความต่าง: console คือการมาเยี่ยมชม ส่วน CLI คือการทำวิศวกรรม 4. เว็บไซต์ static: สร้าง bucket ใบที่สอง เปิด static website hosting อัปโหลด `index.html` แล้วเพิ่ม **bucket policy** แบบ public-read ตามเอกสาร (เอกสารสิทธิ์รูปแบบ JSON — อ่านทีละบรรทัด: ใครทำอะไรได้กับ bucket ใบไหน) ตอนนี้หน้าเว็บของคุณอยู่บนอินเทอร์เน็ตโดยไม่มี server ลักเครื่อง 5. สํารวจ

storage class (Standard → Infrequent Access → Glacier: ถูกลดต่อ GB แต่ดึงข้อมูลช้าลง/แพงขึ้น) และตั้ง **lifecycle rule** (“ย้าย object ไป IA หลัง 30 วัน”) — รสชาติแรกของการจัดการต้นทุนแบบอัตโนมัติ

สิ่งที่ได้เรียนรู้: object storage กับดิสก์ต่างกันอย่างไร, CLI และ access key, bucket policy และการเปิด public (แบบตั้งใจ ไม่ใช่พลาด), storage tier ในฐานะคั่นโยกลดต้นทุน

Teardown: เทของออกจาก bucket ทั้งสองใบ ลบทั้งสองใบ และ — สำคัญ — ปิดใช้งาน access key ที่ไม่ได้ใช้ โควตาฟรีของ S3 เล็ก (5 GB) แต่ไฟล์พวกนี้แค่หลักกิโลไบต์; ย้ำอีกครั้งว่าประเด็นคือวินัย

Lab 3 (สัปดาห์ 8–9): VPC — เครือข่ายที่คุณเป็นเจ้าของ

VPC (Virtual Private Cloud) คือส่วนแบ่งเครือข่ายส่วนตัวที่ล้อมรั้วไว้ของคุณในเครือข่ายของ AWS จนถึงตอนนี้คุณใช้ตัว default โดยไม่เคยมองเห็น; วิศวกรสร้างของตัวเอง เพราะประโยค “*database อยู่ใน private subnet ที่ไม่มีเส้นทางออกอินเทอร์เน็ต*” คือครึ่งหนึ่งของความปลอดภัยบนคลาวด์ และคุณกำลังจะทำให้มันเป็นจริงด้วยมือตัวเอง

เป้าหมาย: สร้างเครือข่ายสองชั้น — public subnet สำหรับ web server, private subnet สำหรับ database ในอนาคต — แล้วพิสูจน์ว่าครึ่ง private เข้าถึงจากอินเทอร์เน็ตไม่ได้

โครงร่างขั้นตอน: 1. Console → VPC → Create VPC กำหนดบล็อกที่อยู่ 10.0.0.0/16 — **CIDR notation** ที่ /16 แปลว่า “16 บิตแรกตายตัว ที่เหลือเป็นของฉัน”: ที่อยู่ private 65,536 ตัว 2. สร้าง **subnet** สองอัน: 10.0.1.0/24 (public, ใน AZ-a) และ 10.0.2.0/24 (private, ใน AZ-b) subnet คือบล็อกย่อยภายใน VPC ที่อาศัยอยู่ใน AZ เดียวเป๊ะ ๆ 3. สร้าง **Internet Gateway** (ประตูสู่อินเทอร์เน็ตของ VPC) แล้ว attach เข้าไป สร้าง **route table** ที่มีกฎ 0.0.0.0/0 → internet gateway (“อะไรที่ไม่ใช่ภายในให้ไปที่ประตูอินเทอร์เน็ต”) แล้ว associate กับ subnet ฝั่ง *public* เท่านั้น ฝั่ง private เก็บไว้แค่ route ภายใน — การไม่มี route นั้นแหละคือความปลอดภัย 4. เปิด micro EC2 instance หนึ่งเครื่องในแต่ละ subnet (ฝั่ง public มี public IP ฝั่ง private ไม่มี) 5. การพิสูจน์: SSH เข้าเครื่อง public — สำเร็จ ลองที่อยู่ของเครื่อง private จากเล็บที่ขอบของคุณ — ค้างตลอดกาล และตอนนี้คุณอธิบายได้เป๊ะว่าทำไม จากนั้น SSH จากเครื่อง public ไปยังเครื่อง private (มันเข้าถึงได้จากภายใน VPC): เครื่อง public กำลังทำหน้าที่เป็น **bastion host** (เครื่องป้อมปราการ) — แพตเทิร์นมาตรฐานอุตสาหกรรมที่คุณเพิ่งค้นพบด้วยการสร้างมันเอง 6. แนวคิดโบนัสให้ค้นและจดลงสมุด: **NAT gateway** ทำให้เครื่อง private ติดต่อออกไปข้างนอกได้ (เพื่ออัปเดต) โดยยังเข้าถึงจากข้างนอกไม่ได้ — แต่มันคิดเงินรายชั่วโมง ฉะนั้นอ่านเรื่องมัน อย่าสร้างมัน

สิ่งที่ได้เรียนรู้: CIDR, subnet, route table, internet gateway, การแบ่ง public/private, bastion host — หัวข้อ networking ที่อยู่ในทุก JD สายคลาวด์ ในรูปความจำของกลัมนั่น

Teardown: terminate ทั้งสอง instance ก่อน แล้วลบ VPC (ซึ่งจะกวาด subnet, route table และ gateway ไปด้วย) ตรวจสอบใน EC2 ว่าไม่มีอะไรขึ้นสถานะ “running”

Lab 4 (สัปดาห์ 10): RDS — database ที่คุณไม่ต้องเฝ้าประคบบประหม

RDS (Relational Database Service) คือ managed database: AWS รันเอนจิน database (PostgreSQL, MySQL...) และจัดการ backup, การแพตช์ และ failover ให้ ขณะที่คุณเป็นเจ้าของข้อมูลและ query “Managed service” คือข้อตกลงแกนกลางของคลาวด์ — แลกการควบคุมบางส่วนกับการปลดแอกจากงานแรงงานที่ไม่สร้างความต่าง — และ lab นี้คือจุดที่คุณจะได้สัมผัสข้อตกลงนั้น

เป้าหมาย: เปิด PostgreSQL database ใน private subnet, เชื่อมต่อจาก EC2 instance, เข้าใจ backup กับ multi-AZ — แล้วรื้อถอนทันที เพราะ RDS คือ lab ที่ถูกลิ้มเปิดทิ้งไว้ง่ายที่สุด

โครงร่างขั้นตอน: 1. สร้าง VPC ของ Lab 3 ใหม่อย่างรวดเร็ว (รอบสองจากความจำ — นี่คือ spaced repetition แบบจงใจ) โดยเพิ่ม subnet *private* อันที่สองในอีก AZ เพราะ RDS ต้องการ **subnet group** ที่พาดผ่านสอง AZ 2. Console → RDS → Create database → PostgreSQL → **Free tier template** (ตัวนี้จะเลือก db.t3.micro/db.t4g.micro แบบ single-AZ ให้อัตโนมัติ) ตั้งรหัสผ่าน master วางไว้ใน VPC ของคุณ, **Public access: No**, ใน security group ที่อนุญาตพอร์ต 5432 เฉพาะจาก security group ของ web server เท่านั้น — กฎที่อ้างถึงกฎอีกข้อแทนที่จะอ้าง IP สง่างาม และเป็นมาตรฐาน 3. เปิด micro EC2 ใน public subnet ติดตั้งไคลเอนต์ postgresql แล้วเชื่อมต่อ: `psql -h <rds-endpoint> -U postgres` ตัว **endpoint** เป็นชื่อ DNS ไม่ใช่ IP — AWS อาจย้ายเครื่องข้างใต้ แล้วชื่อจะตามไปเอง (ความรู้โมดูล 2 เริ่มจ่ายค่าเช่าแล้ว) 4. ที่พรอมต์ psql: สร้างตาราง insert สามแถว select กลับมาดู วันนี้คุณยังไม่ต้องรู้ SQL ลึก แต่ต้องเคยแตะมันแล้ว 5. เทียวชมแต่อย่าเปิดใช้: การตั้งค่า **automated backups** (กู้คืนแบบ point-in-time จาก snapshot รายคืน + log) และตัวเลือก **Multi-AZ** (เครื่อง standby แบบสด ๆ ในอีก AZ พร้อม **failover** อัตโนมัติ — ราคาราวสองเท่า จึงเป็นเหตุที่ prod ตอบรับแต่ lab ตอบปฏิเสธ) ทำ **snapshot** ด้วยมือหนึ่งอัน หามันใน console แล้วทำความเข้าใจว่าคุณกู้คืนเป็นเครื่องโคลนจากมันได้ 6. คำถามลงสมุดที่คุ้มค่าลิบนาท: AWS กำลังทำอะไรให้คุณกันแน่ ที่ไม่จั้นคุณต้องทำเองตอนตีสอง? (แพตช์, backup, failover, ฮาร์ดแวร์) คำตอบนั้นคือคำตอบเรื่อง managed services ในห้องสัมภาษณ์

สิ่งที่ได้เรียนรู้: managed database, subnet group, กฎ security-group-ถึง-security-group, endpoint, snapshot, multi-AZ/failover — บวกการสาธิตสด ๆ ว่าการจ่ายด้วยการควบคุมเพื่อซื้อความน่าเชื่อถือเป็นอย่างไร

Teardown: ลบ RDS instance (ปฏิเสธ final snapshot สำหรับ lab; จดไว้ว่า prod จะต้องทำ) ลบ snapshot ที่ทำด้วยมือ (snapshot คิดเงินค่าพื้นที่!) terminate EC2 ลบ VPC เช็กแดชบอร์ด billing ในวันถัดไป — การอ่านมันทุกสัปดาห์คือนิสัยของช่วงที่ 3 ที่เริ่มตั้งแต่ตอนนี้

Lab 5 (สัปดาห์ 11–12): IAM — ใครทำอะไรได้บ้าง

IAM (Identity and Access Management) ตัดสินว่าคนไหนและโปรแกรมไหนทำอะไรได้กับทรัพยากรตัวไหน มันไม่คิดเงิน ไม่สร้างทรัพยากรอะไรเลย — แต่เป็นบริการที่ถูกตรวจสอบมากที่สุด ถูกซักในสัมภาษณ์มากที่สุด และพัวพันกับเหตุข้อหลว่ร่มากที่สุดใน AWS หัวข้อความปลอดภัยใน JD (“least-privilege access controls,” “IAM policies”) หมายถึง lab นี้

เป้าหมาย: สร้าง user, group, policy และ — ตัวที่สำคัญที่สุด — **role** และชิมชั least privilege ด้วยการโดน AWS ปฏิเสธใส่หน้า

โครงสร้างขั้นตอน: 1. แนวคิดก่อน ทำหน้าที่: **user** คือตัวตนของคนหรือโปรแกรม; **group** มัดรวม user; **policy** คือเอกสาร JSON ที่มอบสิทธิ์ (“อนุญาต s3:GetObject บน arn:aws:s3:::my-bucket/*”); **role** คือตัวตนที่มี policy แต่ไม่มีรหัสผ่าน ซึ่งฝ่ายที่ได้รับความไว้วางใจจะ *assume* (สวมบทบาท) ชั่วคราว — วิธีที่ server และบริการต่าง ๆ ได้สิทธิ์โดยไม่มี secret เก็บอยู่ที่ไหนเลย 2. สร้าง user ชื่อ readonly-rita ในกลุ่มที่ติด policy ReadOnlyAccess ที่ AWS ดูแลให้ ล็อกอินเป็นเธอในหน้าต่างเบราว์เซอร์ private: เธอเห็นได้ทุกอย่าง แต่ปุ่มสร้าง/ลบทุกปุ่มล้มเหลวพร้อมคำปฏิเสธชัดเจน อ่าน error พวกนั้นสักอันให้จบ — การหัดแยกส่วนประโยค “not authorized to perform X on Y” คือทักษะงานรายวัน 3. เขียน custom policy แรกของคุณใน JSON editor: อนุญาต s3:ListBucket และ s3:GetObject บน bucket เจาะจงหนึ่งใบ ติดให้ user ใหม่; ตรวจสอบว่าเธออ่าน bucket นั้นได้และอย่างอื่นไม่ได้เลย ตอนนี้คุณได้ลงมือทำ least privilege แล้ว ไม่ใช่แค่เนียนมัน 4. Role — บทสรุปรางวัล: สร้าง role สำหรับ EC2 พร้อม AmazonS3ReadOnlyAccess เปิด micro instance โดย attach role นั้น SSH เข้าไป แล้วรัน aws s3 ls — มันทำงานโดยไม่มี access key อยู่บนเครื่องเลยแม้แต่ตัวเดียว instance กำลัง assume role และรับ credential อายุสั้นโดยอัตโนมัติ นี่คือแพตเทิร์นความปลอดภัยที่สำคัญที่สุดหนึ่งเดียวใน AWS: role สำหรับเครื่อง ไม่เอา key บนเครื่องเด็ดขาด พุดซ้ำสองรอบ 5. ตรวจสอบบัญชีตัวเองแบบมือโปร: root เปิด MFA หรือยัง (สัปดาห์ 3)? มี access key ไหนแก่เกิน 90 วัน? user คนไหนมีสิทธิ์มากกว่าที่ใช้จริง? รัน **credential report** ของ IAM แล้วอ่านมัน พิธีกรรมสับนาทีนี้ ทำรายเดือน คือเช็กนิสัยสุขอนามัย IAM ของช่วงที่ 3 ที่กำลังถือกำเนิด

สิ่งที่ได้เรียนรู้: user/group/policy/role, การอ่านและเขียน policy JSON, least privilege ที่บังคับใช้ผ่านการทดลอง, role-ไม่ใช่-key และการตรวจสอบความปลอดภัยครั้งแรกของคุณ

Teardown: ลบ user ทดสอบและ credential ของพวกเขา, terminate instance, เก็บความรู้เรื่อง role ไว้ตลอดกาล

คำศัพท์สำหรับช่วงที่ 1 (ตารางเดียว ครบทั้งทำ lab):

คำศัพท์	ความหมาย
Region / Availability Zone	กลุ่ม data center ของ AWS ตามภูมิศาสตร์ / data center ที่แยกอิสระหนึ่งแห่งข้างในนั้น ระบบจริงจึงพาดผ่านสอง AZ
EC2 / instance	บริการเช่าเครื่องเสมือน / server ที่เช่าไว้หนึ่งเครื่อง
AMI	Amazon Machine Image — ดิสก์แม่แบบที่ instance นูตขึ้นมา
Instance type	ขนาด/สเปกที่คุณเลือก (t3.micro = จี๋ว, Free Tier)
Security group	firewall รายทรัพย์สิน: พอร์ตไหน จากต้นทางไหน
Key pair	กุญแจ SSH คู่ private/public สำหรับเข้าถึง instance ของคุณ

คำศัพท์	ความหมาย
S3 / bucket / object	object storage / ภาชนะติดชื่อหนึ่งใบ / ไฟล์ที่เก็บไว้หนึ่งไฟล์
Storage class / lifecycle rule	ระดับชั้นราคา-ความเร็วของ object / กฎอัตโนมัติที่ย้ายมันไป tier ถูกลงตามอายุ
Bucket policy	เอกสาร JSON ระบุว่าใครทำอะไรได้กับ bucket ตัว public เคยขึ้นพาดหัวข่าวมาแล้ว
AWS CLI / access key	หน้าจอเทอร์มินัลของ AWS / credential แบบโปรแกรมสำหรับมัน (ปฏิบัติเหมือนรหัสผ่าน)
VPC / subnet	ส่วนแบ่งเครือข่ายส่วนตัวของคุณ / บล็อกย่อยของมันที่อยู่ในหนึ่ง AZ แบบ public หรือ private
CIDR	สัญกรณ์บล็อกที่อยู่: 10.0.0.0/16 = “16 บิตแรกตายตัว ที่อยู่ 65,536 ตัวเป็นของฉัน”
Internet gateway / route table	ประตูอินเทอร์เน็ตของ VPC / กฎที่ตัดสินว่าทราฟฟิกถูกส่งไปไหน ไม่มี route = เข้าไม่ถึง = ปลอดภัย
NAT gateway	ให้เครื่อง private โทรรอกได้โดยไม่ถูกโทรเข้า คิดเงินรายชั่วโมง — รู้จักมันไว้ อย่าเปิดทิ้ง
Bastion host	เครื่อง public ที่เสริมเกราะไว้ให้คุณ SSH ผ่านเพื่อเข้าถึงเครื่อง private
RDS / endpoint	บริการ relational database แบบ managed / ชื่อ DNS ที่คุณใช้เชื่อมต่อ
Snapshot / Multi-AZ / failover	สำเนา ณ จุดเวลา / เครื่อง standby สด ๆ ใน AZ ที่สอง / การสลับไปทามันโดยอัตโนมัติ
IAM user / group / policy / role	ตัวตน / มัดรวมของตัวตน / เอกสารมอบสิทธิ์ JSON / ตัวตนสวมได้ที่ไม่มีรหัสผ่าน — วิธีที่เครื่องได้สิทธิ์
Least privilege	กฎหมายคำ: สิทธิ์ขั้นต่ำที่ทำงานได้สำเร็จ ไม่มากไปกว่านั้น
Managed service	AWS รับงานแรงงานที่ไม่สร้างความต่าง (แพตช์, backup, failover); คุณเก็บข้อมูลและการตัดสินใจไว้

วิดีโอประจำช่วงนี้:

วิดีโอ	ช่อง	ความยาว	ลิงก์
Getting Started with EC2	AWS Developers (ช่องทางการ)	27 นาที	https://www.youtube.com/watch?v=nJ-djerESW0
Introduction to Amazon S3	Amazon Web Services (ช่องทางการ)	~5 นาที	https://www.youtube.com/watch?v=ecv-19sYL3w
AWS Networking Basics — VPC & Subnets	KodeKloud (ดูซ้ำหลัง Lab 3 — คราวนี้มันจะเข้าหัวคนละแบบ)	~30 นาที	https://www.youtube.com/watch?v=QM63dyA_4Pc
The AWS Shared Responsibility Model	Digital Cloud Training	4 นาที	https://www.youtube.com/watch?v=ESPBBEK-cvo

พูดออกเสียง: “It’s in a private subnet; there’s no route to the internet gateway.” (มันอยู่ใน private subnet ไม่มี route ไป internet gateway) · “The security group only allows 5432 from the web tier’s security group.” (security group อนุญาต 5432 เฉพาะจาก security group ของ web tier) · “The instance uses a role — there are no keys on the box.” (instance ใช้ role — ไม่มี key อยู่บนเครื่อง) · “Terminate it when you’re done; nothing idles in this account.” (เสร็จแล้ว terminate เลย บัญชีนี้ไม่มีอะไรเปิดทิ้งเฉย ๆ)

Milestone — จบช่วงที่ 1: ด้านทดสอบรวบยอด (gauntlet build) ในการนั่งครั้งเดียว จากความจำ: VPC ที่มี public/private subnet → EC2 web server (public, ติด role, เสิร์ฟหน้าเว็บ) → RDS PostgreSQL (private, เข้าถึงได้จาก web server เท่านั้น) → S3 bucket ที่ instance อ่านได้ผ่าน role ของมัน — แล้วรื้อถอนทั้งหมดจนสะอาด ทำได้ในไม่เกินสามชั่วโมงแปลว่าคุณพร้อมสำหรับช่วงที่ 2 อีกอย่าง: **เริ่มเตรียมสอบ CLF-C02 ตั้งแต่ตอนนี้** (คอร์สเต็มของ freeCodeCamp — <https://www.youtube.com/watch?v=7HKot-brXFE> — เปิดความเร็ว 1.25× เป็นการทบทวน; ฉบับคลาสสิก 14 ชั่วโมงคือ <https://www.youtube.com/watch?v=NhDYbskXRgc>) และเข้าสอบ **AWS Cloud Practitioner** ราวสัปดาห์ที่ 14–16

ช่วงที่ 2 — ระบบอัตโนมัติ (สัปดาห์ 13–20)

แนวคิดของช่วงนี้: ทุกอย่างที่คุณคลิกในช่วงที่ 1 ตอนนี้คุณจะทำด้วยโค้ด คำเหินเบา ๆ ของวงการสำหรับการคลิกหน้าคอนโซลคือ **ClickOps**; ส่วนวลีในประกาศงานสำหรับสิ่งที่มาแทนคือ “*design, develop, and deploy cloud infrastructure using infrastructure as code*” ช่วงนี้คือความต่างระหว่างคนที่เคยใช้ AWS กับคนที่บริษัทยอมจ่ายเงินให้ดูแลมัน

โมดูล 6 (สัปดาห์ 13–15): การเขียนสคริปต์ — Bash และ Python สำหรับ ops บวกด้วย Git

การเขียนสคริปต์ Bash คือการเอาคำสั่งจากโมดูล 1 ใส่ลงไฟล์ เพื่อให้คอมพิวเตอร์ทำซ้ำได้อย่างสมบูรณ์แบบเรียนตามลำดับนี้: **ตัวแปร** (NAME="web-1"), **command substitution** (TODAY=\$(date +%F)), **if** (if [-f "\$FILE"]; then ... fi), **ลูป** (for f in *.log; do gzip "\$f"; done), **exit code** (ทุกคำสั่งคืนค่า 0 เมื่อสำเร็จ ไม่ใช่ศูนย์เมื่อล้มเหลว; && ต่อโซ่คำสั่งเมื่อสำเร็จ — สคริปต์ตัดสินใจด้วยสิ่งนี้) และการอ่าน **argument** (\$1, \$2) นั่นคือ 90% ของ Bash งาน ops จริง ๆ เขียนสคริปต์ที่มีประโยชน์จริงตัวแรกของคุณ สัปดาห์นี้: backup.sh — tar โพลเดอร์หนึ่ง ตั้งชื่อไฟล์บีบอัดด้วยวันที่วันนี้ aws s3 cp ขึ้น bucket ลบไฟล์บีบอัดในเครื่องที่เก่ากว่า 7 วัน และ echo บรรทัดสำเร็จ/ล้มเหลวลงไฟล์ log สคริปต์เดียวนั้นคือตัวแปร, substitution, เงื่อนไข, exit code และ CLI ในชั้นงานเดียว — และเป็นหนึ่งบรรทัดบนเรซูเม่ของคุณ (“automated backups to S3”)

Python คือพี่คนโตของ Bash: ดีกว่าสำหรับทุกอย่างที่มีตรรกะ ข้อมูล หรือการคุยกับ API คุณต้องการ Python ระดับใช้งาน ops ได้ ไม่ใช่ความลึกระดับ software engineering: ตัวแปรและชนิดข้อมูล, list และ dictionary, if/for, ฟังก์ชัน, อ่าน/เขียนไฟล์, จัดการ error ด้วย try/except และติดตั้งไลบรารีด้วย pip จากนั้นทำความรู้จัก **boto3** ไลบรารี AWS สำหรับ Python: import boto3; ec2 = boto3.client("ec2"); ec2.describe_instances() — ความรู้ช่วงที่ 1 ของคุณกลายเป็นสิ่งที่เขียนโปรแกรมสั่งได้ในปริบทาเขียน audit.py: แสดง EC2 instance ทุกตัวในบัญชีพร้อมชื่อ ชนิด สถานะ และเวลาเปิดเครื่อง แล้วพิมพ์บรรทัดเตือนสำหรับอะไรก็ตามที่รันนานเกิน 24 ชั่วโมง ยินดีด้วย — คุณเพิ่งเขียนเครื่องมือควบคุมต้นทุนของจริงที่นายจ้างจ่ายเงินซื้อ

Git คือระบบควบคุมเวอร์ชันที่งานวิศวกรรมทั้งหมดอาศัยอยู่ แนวคิด: **repository** (โพลเดอร์ที่ประวัติทั้งหมดถูกบันทึก), **commit** (สแนปช็อตที่บันทึกไว้หนึ่งครั้งพร้อมข้อความ), **branch** (สายงานคู่ขนาน), **remote** (สำเนาบน GitHub) และ **pull request** (การขอให้ branch ถูกรีวิวและ merge) ลูปประจำวันคือหาคำสั่ง: git init / git clone, git status, git add, git commit -m "message", git push, git pull สร้างบัญชี GitHub วันนี้ สร้าง repo ชื่อ cloud-journey และจากนี้ไป **ทุกสคริปต์และทุก config ที่คุณเขียนในหลักสูตรนี้ต้อง commit เข้าไป** ในอีกสัปดาห์ ประวัติ commit นั้นคือหลักฐานสาธารณะที่มีวันที่กำกับของทุกสิ่งทีหลักสูตรนี้อ้างว่าคุณทำได้

คำศัพท์:

คำศัพท์	ความหมาย
Script	ไฟล์คำสั่งที่รันจากบนลงล่าง — อะตอมของระบบอัตโนมัติ
Variable / argument	ค่าที่ตั้งชื่อไว้ในสคริปต์ / ค่าที่ส่งเข้ามาตอนรัน (\$1)
Exit code	สัญญาณสำเร็จ (0) หรือล้มเหลว (ไม่ใช่ศูนย์) ของทุกคำสั่ง — วิธีที่สคริปต์ตัดสินใจ
Cron	ตัวตั้งเวลาของ Linux: รันสคริปต์ทุกคืนตีสอง ตลอดไป
Python / pip	ภาษาโปรแกรมโปรดของโลก ops / ตัวติดตั้งแพ็คเกจของมัน

คำศัพท์	ความหมาย
boto3	ไลบรารี Python สำหรับขับเคลื่อน AWS — คอนโซลในรูปแบบของโค้ด
try/except	“ลองทำอันนี้ ถ้าพังให้ทำอันนั้นแทน” ของ Python — วิธีที่สคริปต์ล้มเหลวอย่างมีมารยาท
Git / repository / commit	ระบบควบคุมเวอร์ชัน / ประวัติที่บันทึกไว้ของหนึ่งโปรเจกต์ / สแนปช็อตหนึ่งครั้งพร้อมข้อความ
Branch / merge	สายงานคู่ขนาน / การนำมันกลับเข้าสู่สายหลัก
GitHub / remote / pull request	เว็บไซต์ / สำเนาบนคลาวด์ของ repo คุณ / คำขอ merge branch ที่ผ่านการรีวิวและถกเถียง
README	เอกสารหน้าแรกของ repo อธิบายว่ามันคืออะไรและรันอย่างไร recruiter อ่านสิ่งเหล่านี้จริง ๆ

วิดีโอประจำโมดูลนี้:

วิดีโอ	ช่อง	ความยาว	ลิงก์
Learn Python — Full Course for Beginners	freeCodeCamp	~4.5 ชม. (ทยอยดูตลอด 3 สัปดาห์)	https://www.youtube.com/watch?v=rfscVS0vtbw

พูดออกเสียง: “I’ll script it — it’ll be wrong once, then never again.” (เดี่ยวเขียนเป็นสคริปต์ — มันจะผิดแค่ครั้งเดียว แล้วไม่ผิดอีกเลย) · “Check the exit code before the next step runs.” (เช็ก exit code ก่อนให้ขั้นถัดไปรัน) · “Commit it with a message that says why, not what.” (commit พร้อมข้อความที่บอกว่าทำไม ไม่ใช่ทำอะไร) · “It’s in the repo — pull the latest.” (อยู่ใน repo แล้ว — pull ตัวล่าสุดมา)

แบบฝึกหัด: (1) backup.sh ตามสเปกข้างบน ตั้งเวลารันทุกคืนด้วย cron บน EC2 instance สำหรับ lab (2) audit.py ตามสเปกข้างบน (3) commit ทั้งคู่เข้า cloud-journey พร้อม README อธิบายแต่ละตัว (4) จงใจทำสคริปต์ตัวเองพัง (path ผิด, bucket หาย) แล้วทำให้มันล้มเหลวแบบเสียงดังและชัดเจน — การจัดการ error คือภาษารักของชาว ops

Milestone: สคริปต์ backup ของคุณรอดจากการถูกรันสองครั้งติดกันและจากโพลเดอร์ที่หายไป (ไม่แคระช มีข้อความชัดเจน); Python audit ของคุณรันกับบัญชีจริงของคุณ; repo ของคุณมี commit ต่อเนื่องหนึ่งสัปดาห์

โมดูล 7 (สัปดาห์ 16–17): Terraform — infrastructure as code

แนวคิดใหญ่: แทนที่จะคลิกให้ทรัพยากรถือกำเนิด คุณเขียนไฟล์ข้อความที่ประกาศว่าอะไรควรมีอยู่ — “VPC หนึ่งตัว, subnet สองอัน, instance หนึ่งเครื่อง, tag ชุดนี้” — แล้ว **Terraform** ทำให้ AWS ตรงกับไฟล์

Declarative (เชิงประกาศ) ไม่ใช่ imperative (เชิงคำสั่ง): คุณระบุจุดหมาย ไม่ใช่ทางเลี้ยว เหตุผลที่ทุก JD ขอ มัน: ไฟล์อยู่ใน Git ดังนั้นโครงสร้างพื้นฐานจึงถูก**รีวิว**ได้ (pull request ก่อนการเปลี่ยนแปลง), **ทำซ้ำได้** (ไฟล์เดียวกันสร้าง dev, staging และ prod เหมือนกันเป๊ะ), **ย้อนกลับได้** (roll back ด้วยการ revert commit) และ**ตรวจสอบย้อนหลังได้** (ประวัติบอกใครเปลี่ยนอะไร เมื่อไหร่ ทำไม) คอนโซลคือโรงช่าง; Terraform คือโรงงาน

รูปแบบกลางที่คุณจะรันหลายร้อยรอบ: terraform init (ดาวน์โหลด **provider** ของ AWS — ปลั๊กอินที่แปลไฟล์ของคุณเป็นการเรียก API) → terraform plan (การซ้อมแห้งที่พิมพ์ออกมาเป๊ะ ๆ ว่าจะสร้าง/เปลี่ยน/ทำลายอะไร — อ่านทุกบรรทัด; วิศวกรที่ apply โดยไม่อ่าน plan คือคนก่อเหตुरะบบล่ม) → terraform apply (ทำให้เป็นจริง) → terraform destroy (ถอนคืนทั้งหมด — teardown ในคำสั่งเดียว ซึ่งถึงตอนนี้คุณคงเห็นว่ามันงดงาม) Terraform จัดสิ่งที่มันสร้างไว้ใน **state file** — ความทรงจำต่อความเป็นจริงของมัน; ทำหาย หรือแอบแก้ความเป็นจริงลับหลังมัน (“drift”) แล้วความเจ็บปวดจะตามมา ทรัพยากรถูกประกาศใน **HCL** ภาษา config ที่อ่านง่าย:

```
resource "aws_instance" "web" {
  ami           = "ami-0abcdef1234567890"
  instance_type = "t3.micro"
  tags = { Name = "web-1", Project = "lab" }
}
```

Lab (นี่แหละคือตัวโมดูล): สร้างด่านรายยอดของช่วงที่ 1 ขึ้นใหม่ด้วย Terraform แบบทีละขั้น (1) init repo ชื่อ terraform-labs; เขียน config แค S3 bucket ติด tag หนึ่งใบ; plan, อ่าน, apply, เช็กใน console — มันอยู่ตรงนั้นจริง; destroy (2) ขยาย config: VPC, subnet สองอัน, internet gateway, route table — งานสร้างของ Lab 3 ตอนนี้อยู่เหลือ HCL ราว 60 บรรทัด (3) เพิ่ม security group และ EC2 instance พร้อม variables.tf (region, ขนาด instance) และ outputs.tf (พิมพ์ public IP หลัง apply) (4) เปลี่ยน instance type ในไฟล์แล้ว apply ใหม่ — ฝ้าดู Terraform คำนวณผลต่างและเปลี่ยนเฉพาะตรงนั้น พฤติกรรมขับเคลื่อนด้วย diff นี่คือเวทมนตร์ทั้งหมด (5) destroy ทุกอย่าง ยืนยันว่า console ว่างเปล่า commit แล้วติดแท็ก repo เป็น v1.0 เอกสารอ้างอิงอยู่ที่ <https://developer.hashicorp.com/terraform> — นึกมาร์กไว้; การอ่านเอกสาร provider คือครึ่งหนึ่งของงาน Terraform จริง

คำศัพท์:

คำศัพท์	ความหมาย
IaC (Infrastructure as Code)	การประกาศโครงสร้างพื้นฐานในไฟล์ข้อความที่มีเวอร์ชัน แล้วให้เครื่องมือเนรมิตเป็นความจริง
Terraform / HCL	เครื่องมือ IaC ข้ามคลาวด์ที่ครองตลาด / ภาษา configuration ของมัน
Provider	ปลั๊กอินที่แปลไฟล์ของคุณเป็นการเรียก API ของคลาวด์หนึ่งเจ้า (ที่นี่: AWS)

คำศัพท์	ความหมาย
plan / apply / destroy	diff แบบซ้อมแห้ง / ทำให้เป็นจริง / ถอนคืนทั้งหมด อ่านทุก plan
State file	บันทึกของ Terraform ว่ามันสร้างอะไรไว้ — ความทรงจำต่อความเป็นจริง ปกป้องมัน
Drift	ความเป็นจริงเปลี่ยนไปหลังจาก Terraform (มีคนไปคลิก) คัดรุตัวณากจ
Variable / output / module	ค่าป้อนเข้า config / ผลลัพธ์ที่พิมพ์ออกมา (IP, URL) / ก้อน config สำเร็จรูปที่นำกลับมาใช้ได้
Declarative vs imperative	ระบุจุดหมาย vs เขียนสคริปต์บอกทางเลี้ยว Terraform เป็นแบบ declarative
CloudFormation	บริการ IaC ของ AWS เอง — แนวคิดเดียวกัน ใช้ได้เฉพาะ AWS JD รับผิดชอบ; เรียน Terraform ก่อน

วิดีโอประจำโมดูลนี้:

วิดีโอ	ช่อง	ความยาว	ลิงก์
Terraform explained in 15 mins	TechWorld with Nana	18 นาที	https://www.youtube.com/watch?v=I5k1ai_GBDE
Terraform Course — Automate your AWS cloud infrastructure	freeCodeCamp	~2.5 ชม.	https://www.youtube.com/watch?v=SLB_c_ayRMO

พูดออกเสียง: “Nothing changes in prod except through Terraform.” (ไม่มีอะไรเปลี่ยนใน prod นอกจากผ่าน Terraform) · “Show me the plan before you apply.” (ขอดู plan ก่อน apply) · “That was clicked in manually — that’s drift; let’s import it or remove it.” (อันนั้นถูกคลิกสร้างด้วยมือ — นั่นคือ drift; import เข้ามาหรือเอาออกกันเถอะ) · “It’s in the module; reuse it, don’t rewrite it.” (มันอยู่ใน module แล้ว; เอามาใช้ซ้ำ อย่าเขียนใหม่)

แบบฝึกหัด: lab ข้างบน บวกด้วย: (1) จงใจสร้าง drift — แก้ tag ของ instance ด้วยมือใน console, รัน terraform plan แล้วเฝ้าดู Terraform สังเกตเห็น; จดลงสมุดว่ามันเสนออะไร (2) เขียนย่อหน้าภาษาคนธรรมดาใน README ของ repo: “ทำไมวิธีนี้ชนะการคลิก” ถ้าเขียนไม่ได้ แปลว่าคุณยังไม่ได้มันจริง

Milestone: จากใดเรกทอรีว่างเปล่า คุณยกสแตก VPC + instance ขึ้นมาด้วย init/plan/apply และถอนคืนด้วย destroy ได้ พร้อมอธิบายออกเสียงว่าแต่ละคำสั่งกำลังทำอะไร — โดยไม่เปิดเน็ต

โมดูล 8 (สัปดาห์ 18–20): CI/CD, Docker และแรกพบกับ Kubernetes

CI/CD (Continuous Integration / Continuous Delivery) คือสายพานหุ่นยนต์ที่ต่อเข้ากับ Git repo ของคุณ: ทุกครั้งที่ push มันจะตรวจ ทดสอบ build และ — เมื่อถูกตั้งค่าไว้ — deploy การเปลี่ยนแปลงของคุณโดยอัตโนมัติ จุดหมายคือการปล่อยของแบบเล็ก บ่อย และน่าเบื่อ แทนที่จะนาน ๆ ครั้งแล้วนำหาวดพว **GitHub Actions** คือระบบ CI/CD ที่ฝังใน GitHub: **workflow** คือไฟล์ YAML ใน .github/workflows/ ที่บอกว่า “เมื่อ push ให้รันขั้นตอนเหล่านี้บน **runner** ใหม่เอี่ยม (VM ชั่วคราว)” Lab: เพิ่ม workflow ให้ terraform-labs ที่รัน terraform fmt -check และ terraform validate ทุกครั้งที่ push — push ไฟล์ที่จงใจเขียนผิดรูปแบบแล้วดู X สีแดง แก้แล้วดู ✓ สีเขียว ตอนนี้คุณมี pipeline แล้ว; เล็กก็จริง แต่มันคือสปีชีส์เดียวกับที่อยู่ในทุก JD (ขั้นที่สอง ในโปรเจกต์ 1: workflow ที่รัน terraform plan บน pull request เพื่อให้การเปลี่ยนแปลงโครงสร้างพื้นฐานที่ถูกเสนอโชว์ diff ของมันในการรีวิว)

Docker บรรจุแอปพลิเคชันพร้อมทุกสิ่งที่มันต้องใช้ลงใน **container** — กล่องข้าวปิดผนึกที่รันเหมือนกันเป๊ะบนแล็ปท็อปของคุณ บน EC2 และทุกที่ ปิดฉากโรคระบาดโบราณ “works on my machine” (บนเครื่องผมมันรันได้นะ) **image** คือสูตรที่ถูกแช่แข็ง (build จาก **Dockerfile** ไฟล์ข้อความสั้น ๆ ของขั้นตอน build: เริ่ม FROM base image, COPY แอปของคุณเข้าไป, กำหนดคำสั่งเริ่มต้น); **container** คือ instance ที่กำลังรันหนึ่งตัวของ image; **registry** (Docker Hub หรือ **ECR** ของ AWS) คือที่ที่ image ถูก push และ pull container *ไม่ใช่* VM: มันแชร์ Linux kernel ของเครื่องแม่ จึงสตาร์ทในราวหนึ่งวินาที และคุณรันได้หลายสิบตัวบน micro instance เครื่องเดียว Lab: บน EC2 instance ติดตั้ง Docker, docker run hello-world, ต่อด้วย docker run -d -p 80:8080 <a sample web image> แล้วเปิดเบราว์เซอร์ไปดู; จากนั้นเขียน Dockerfile ทำบรรทัดที่เสิร์ฟหน้า static ของ Lab 2 จาก base image nginx, build, run แล้ว push ขึ้น Docker Hub เรียนรู้คำกริยารายวัน: docker ps, docker logs, docker exec -it <id> bash (shell ข้างใน container), docker stop

Kubernetes (K8s) — สำหรับหลักสูตรนี้ ทักษะระดับอ่านออก ป้ายกำกับอย่างซื้อสัตย์ เมื่อบริษัทรัน container หลายร้อยตัวข้ามหลายเครื่อง ต้องมีบางสิ่งจัดการทำงาน รัสตาร์ดตัวที่แคระข ขยายตัวทำงานล้น และกำหนดเส้นทางกราฟฟิกระหว่างพวกมัน: ตัวจัดวางดนตรี (orchestrator) นั่นคือ Kubernetes เรียนแผนที่แนวคิดตอนนี้ — **cluster** ของ **node** รัน **pod** (หน่วย deploy เล็กที่สุด มักเป็นหนึ่ง container); **deployment** ประกาศว่า “รักษา pod นี้ให้มี 3 replica มีชีวิตอยู่” แล้ว cluster ทำให้เป็นจริงอย่างต่อเนื่อง (declarative อีกแล้ว — Kubernetes คือปรัชญาของ Terraform ที่ประยุกต์กับซอฟต์แวร์ที่กำลังรัน); **service** ให้ที่อยู่ที่มันคงแก่ pod บริการ managed ของ AWS คือ **EKS** JD ระดับ junior ต้องการแค่ความรู้อ่านออกระดับนี้บวกความคล่อง Docker; ทักษะปฏิบัติการ K8s จริง (และใบรับรอง CKA) คือเป้าหมายขั้นดีของปีที่ 2 ไม่ใช่เดือนที่ 5 หลักสูตรนี้บอกคุณตรง ๆ ว่าอันไหนเป็นอันไหน

คำศัพท์:

คำศัพท์	ความหมาย
CI/CD	pipeline อัตโนมัติที่ตรวจ build ทดสอบ และส่งมอบทุกการเปลี่ยนแปลง

คำศัพท์	ความหมาย
GitHub Actions / workflow / runner	CI/CD ในตัวของ GitHub / ไฟล์ YAML ที่นิยาม pipeline / VM ชั่วคราวที่รันมัน
YAML	ฟอร์แมต config อิงย่อหน้าเยื้องของ pipeline และ Kubernetes การเยื้องคือความหมาย — ระวังให้ดี
Docker / image / container	ชุดเครื่องมือ container / สูตรแช่แข็งเป็นชั้น ๆ / instance ที่กำลังรันหนึ่งตัวของมัน
Dockerfile	ไฟล์ข้อความสั้น ๆ ของขั้นตอนที่ build image
Registry / ECR	ที่ที่ image ถูก push และ pull / registry ของ AWS
Kubernetes (K8s) / cluster / node	ตัวจัดวางดนตรีของ container / กลุ่มเครื่องของมัน / เครื่องหนึ่งในนั้น
Pod / deployment / service	หน่วย deploy เล็กที่สุด / “รักษา N replica ให้มีชีวิต” บังคับใช้ต่อเนื่อง / ที่อยู่มั่นคงหน้ากลุ่ม pod
EKS	Kubernetes control plane แบบ managed ของ AWS
Rollback	การถอยกลับเวอร์ชันก่อนหน้าอย่างรวดเร็วเมื่อรีลีสมีปัญหา — ตาข่ายนิรภัยที่ CI/CD ทำให้เป็นเรื่องปกติ

วิดีโอประจำโมดูลนี้:

วิดีโอ	ช่อง	ความยาว	ลิงก์
DevOps CI/CD Explained in 100 Seconds	Fireship	2 นาที	https://www.youtube.com/watch?v=scEDHsr3APg
What is DevOps? REALLY understand it	TechWorld with Nana	~15 นาที	https://www.youtube.com/watch?v=0yWAtQ6wYNM
Docker in 100 Seconds	Fireship	2 นาที	https://www.youtube.com/watch?v=Gjnup-PuquQ
Docker Tutorial for Beginners [FULL COURSE in 3 Hours]	TechWorld with Nana	3 ชม.	https://www.youtube.com/watch?v=3c-iBn73dDE
Kubernetes explained in 15 mins	TechWorld with Nana	~16 นาที	https://www.youtube.com/watch?v=VnvRFRk_51k

พูดออกเสียง: “Don’t merge until the pipeline is green.” (อย่า merge จนกว่า pipeline จะเขียว) · “It’s containerized — same image in dev and prod.” (มันอยู่ใน container แล้ว — image เดียวกันทั้ง dev และ prod) · “Exec into the container and check its logs.” (exec เข้าไปใน container แล้วเช็ค log ของมัน) · “K8s keeps three replicas up; kill one and watch it come back.” (K8s รักษา replica ไว้สามตัว; ฆ่าไปหนึ่งแล้วดูมันกลับมา)

แบบฝึกหัด: สอง lab ข้างบน บวกด้วย: (1) จดคำตอบหนึ่งย่อหน้าลงสมุดสำหรับ “VM vs container vs serverless — เมื่อไหร่ใช้อันไหน?” (Serverless in 100 Seconds ของ Fireship — https://www.youtube.com/watch?v=W_VV2Fx32_Y — เติมเต็มตัวเลือกที่สาม) (2) terminate instance ที่ใช้ทำ Docker lab; ยืนยันว่า image ของคุณยังมีชีวิตอยู่ใน registry — สังเกตว่า *artifact* (ชิ้นงาน) ตอนนี้อายุยืนกว่า *server* ซึ่งคือโลกทัศน์การ deploy สมัยใหม่ทั้งใบในหนึ่งข้อสังเกต

Milestone — จบช่วงที่ 2: GitHub ของคุณโชนั: repo ของสคริปต์ ops Bash + Python ที่ใช้งานได้, repo Terraform ที่สร้างและทำลายสแต็กจริง, workflow ของ Actions ที่เขียว และ Dockerfile ที่ push ขึ้น registry — และคุณอธิบายทุกไฟล์ในห้องสัมภาษณ์ได้ โปรไฟล์ GitHub นั้นไม่ใช่ของนักเรียนอีกต่อไป มันเป็นของวิศวกรระดับ junior แล้ว

ช่วงที่ 3 — งานปฏิบัติการ (สัปดาห์ 21–28)

แนวคิดของช่วงนี้: การสร้างระบบทำให้คุณได้งาน; การดูแลระบบต่างหากคืองานจริง หัวข้อ JD ที่ช่วงนี้ตอบคือ ครึ่งฝั่งปฏิบัติการ — “monitor infrastructure health,” “participate in incident response, including log analysis,” “identifying, analyzing, and resolving infrastructure vulnerabilities,” “manage cloud costs” สัปดาห์ของวิศวกรคลาวด์ส่วนใหญ่คือช่วงนี้

โมดูล 9 (สัปดาห์ 21–24): Monitoring, logging และ incident

Monitoring — รู้ก่อนผู้ใช้จะรู้ CloudWatch คือบริการ observability (การสังเกตการณ์ระบบ) ในตัวของ AWS สามหน่วยพื้นฐาน: **metric** (เมตริก — ตัวเลขตามช่วงเวลา — CPU %, ดิสก์ %, จำนวนคำขอ, จำนวน error), **alarm** (กฎที่เฝ้าดู metric: “ถ้า CPU > 80% นาน 5 นาที → แจ้งเตือน”) และ **dashboard** (แดชบอร์ด — metric จัดเรียงบนจอเดียว) การแจ้งเตือนไหลผ่าน **SNS** (Simple Notification Service — topic ที่คุณ publish ใส่ แล้ว subscriber จะได้รับอีเมล/เพจ) ฝีมือเชิงช่างอยู่ที่เลือกอะไรมาตั้ง alarm: เพจหามนุษย์เฉพาะสิ่งที่ต้องการมนุษย์ ไม่เช่นนั้นจะหัดเมินเพจเจอร์ (**alert fatigue** — ความล้าจากการแจ้งเตือน — โหมดลืมหูลืมตาที่นำหน้าเหตุลุ่มโถมดังมาแล้วหลายครั้ง) สี่สัญญาณทองคำที่ควรท่องจำ: **latency, traffic, errors, saturation** — ซ้าแค่ไหน ยุ่งแค่ไหน พังแค่ไหน เต็มแค่ไหน

Logging — รู้ว่าทำไม Metric บอกว่ามีบางอย่างผิดปกติ; log บอกว่าเกิดอะไรขึ้น **CloudWatch Logs** รวมศูนย์มัน: **agent** (เอเจนต์) บนแต่ละ instance ส่งไฟล์อย่าง `/var/log/nginx/access.log` ขึ้นไปยัง **log group**

ที่ซึ่ง **Logs Insights** ให้คุณ query ข้ามเครื่องได้ (“นับ response 5xx เป็นรายนาที่ของช่วงเวลาที่ผ่านมา”) การรวมศูนย์สำคัญเพราะตอนนี้ server เป็นของใช้แล้วทิ้ง (คุณพิสูจน์แล้วในโมดูล 8) — log ต้องอายุยืนกว่าเครื่องที่เขียนมัน

Lab (สัปดาห์ 21–22): ตั้ง web server ขนาดเล็กที่ถูกเฝ้าระวังครบวงจร ทั้งหมดใน Terraform (สแตกจากโมดูล 7 ของคุณ ขยายเพิ่ม): EC2 + nginx + CloudWatch agent; SNS topic ที่ส่งอีเมลถึงคุณ; alarm หนึ่งตัวจับ CPU สูง อีกตัวจับ instance status-check ล้มเหลว จากนั้นโจมตีตัวเอง: SSH เข้าไปแล้วรันตัวเผา CPU (yes > /dev/null & สักสองสามครั้ง); เฝ้าดู metric ใต้ขึ้น, alarm ทำงาน, อีเมลมาถึง; ฆ่า process; เฝ้าดูระบบฟื้น แล้ว query access log ของตัวเองใน Logs Insights ตอนนี้คุณได้เห็นลูปเต็ม detect → notify → diagnose → resolve บนระบบที่คุณสร้างเองแล้ว

Incident response — ครั้งที่ เป็นมนุษย์ incident (เหตุขัดข้อง) คือ “ระบบไม่โอเค” แบบไม่ได้วางแผน; ระดับ **severity** (ความร้ายแรง — sev-1 = ลูกค้าใช้งานไม่ได้ ทุกคนลงมือ) กำหนดสเกลการตอบสนอง รูปแบบมีอาชีพ: **detect** (จาก alarm ไม่ใช่จากอีเมลลูกค้า — ในอุดมคติ) → **triage** (แยแ่ไหน ต้องการใคร) → **mitigate** (ห้ามเลือดก่อน — roll back, restart, failover; สาเหตุรากไว้ทีหลัง) → **resolve** → **post-mortem**: รีวิวเป็นลายลักษณ์อักษรแบบ *ไม่กล่าวโทษใคร* (blameless) ว่าเกิดอะไร ทำไม และอะไรจะป้องกันไม่ให้เกิดซ้ำ blameless ไม่ใช่ความใจอ่อน; มันคือวิศวกรรม: คนที่ถูกลงโทษจะซ่อนข้อมูล และข้อมูลที่ถูกซ่อนคือต้นเหตุของเหตุลุ่มซ้ารอบ **on-call** คือเวรหมุนเวียนว่าใครถือเพจเจอร์; JD พูดถึงมัน ผู้สัมภาษณ์ถามถึงมัน และคำตอบข้อสัต์ยของคุณหลังจบโมดูลนี้คือ “ผมจำลองมันมาแล้ว และผมรู้จักลูปนี้”

Runbook — หัวข้อ “documentation” ใน JD ในรูปธรรม runbook คือสูตรที่ละขั้นสำหรับสถานการณ์ปฏิบัติการหนึ่งอย่าง เขียนให้คนที่กำลังเครียดตอนตีสามทำตามได้: อาการ → การตรวจ (คำสั่งเป๊ะ ๆ) → การแก้ (คำสั่งเป๊ะ ๆ) → การส่งต่อ (จะปลูกใครถ้ายังไม่หาย) **Lab (สัปดาห์ 23–24):** เขียน runbook สองเล่มลง repo — “web server down” และ “disk filling up” — แล้ว *ซ้อมจริง*: พังของตัวเอง ทำตามเอกสารของตัวเองแบบตรงตัวอักษร แล้วแก้ทุกขั้นที่พิสูจน์ว่าคลุ้มเครือ จากนั้นจัด **game day** (วันซ้อมรบ) เต็มรูปแบบหนึ่งครั้ง: ให้เพื่อน (หรือ AI) แอบพังสแตก lab ของคุณ; คุณถูกเพจ วินิจฉัยจาก metric และ log, mitigate แล้วเขียน post-mortem ลงสมุดบันทึก post-mortem นั่นคือเรื่องเล่าในห้องสัมภาษณ์ และเป็นเรื่องที่ดีด้วย

คำศัพท์:

คำศัพท์	ความหมาย
CloudWatch	บริการ monitoring ของ AWS: metric, alarm, dashboard, log
Metric / alarm / dashboard	ตัวเลขตามช่วงเวลา / กฎที่ทำงานเมื่อค่าผิดเกณฑ์ / หนึ่งจอที่รวมพวกมัน
SNS	บริการแจ้งเตือนที่ alarm ส่งข้อความเข้า — อีเมล, SMS, เพจเจอร์
Golden signals	Latency, traffic, errors, saturation — ตัวเลขสี่ตัวที่บรรยายสุขภาพของบริการใดก็ตาม

คำศัพท์	ความหมาย
Alert fatigue	เพจที่ไม่ต้องลงมือมากเกินไป → เพจถูกเฝ้า → พลาดของจริง ศัตรูของการออกแบบ alarm
Log group / Logs Insights	ที่ที่ log รวมศูนย์ไปลงเอย / ภาษา query เนื่อพวกมัน
Incident / severity	ความเสื่อมถอยแบบไม่ได้วางแผน / ระดับความแย่งที่จัดอันดับ (sev-1 = แย่สุด)
Triage / mitigate / resolve	ประเมินเร็ว / ห้ามเลือดก่อน / แก้จริง
Post-mortem	รีวิวลายลักษณ์อักษรแบบไม่กล่าวโทษ: อะไร ทำไม อะไร ป้องกันการเกิดซ้ำ
Runbook	สูตรกันดีสามสำหรับหนึ่งสถานการณ์: อาการ การตรวจ การแก้ การส่งต่อ
On-call / game day	เวรถือเพจเจอร์ / เหตุขัดข้องซ้อมรบแบบจริงจัง
MTTR	Mean time to recovery — เมตริกฝั่ง ops ที่ทีมโตแล้วมุ่งปรับปรุง

พูดออกเสียง: “Did we find out from the alarm or from a customer?” (เรารู้จาก alarm หรือจากลูกค้า?) · “Mitigate first — root cause after we’re stable.” (mitigate ก่อน — สาเหตุรากไว้หลังระบบนิ่ง) · “Is there a runbook for this? There will be by tomorrow.” (เรื่องนี้มี runbook ไหม? พรุ่งนี้จะมีแน่นอน) · “What did the post-mortem conclude, and what’s the prevention item?” (post-mortem สรุปว่าอย่างไร แล้วมาตรการป้องกันคืออะไร?)

Milestone: post-mortem จาก game day ของคุณมีอยู่จริง เป็นแบบ blameless และระบุมาตรการป้องกันที่เป็นรูปธรรมหนึ่งข้อที่คุณลงมือทำจริงแล้ว (เพิ่ม alarm, แก้ runbook)

โมดูล 10 (สัปดาห์ 25–28): งานปฏิบัติการด้านความปลอดภัย และการจัดการต้นทุน

Security operations — สุขอนามัย ไม่ใช่วีรกรรม เริ่มที่ **shared responsibility model** (โมเดลความรับผิดชอบร่วม — ดูซ้ำ: <https://www.youtube.com/watch?v=ESPBBEK-cvo>): AWS ดูแลความปลอดภัยของตัวคลาวด์เอง; *คุณ* ดูแลความปลอดภัยของสิ่งที่คุณใส่เข้าไป — และเหตุข้อมูลร้ายจริงส่วนใหญ่คือการตั้งค่าผิดของลูกค้า ไม่ใช่ความล้มเหลวของ AWS เช็กลิสต์ความปลอดภัยเชิงปฏิบัติการของคุณ ฝึกจนน่าเบื่อ:

1. **สุขอนามัย IAM** (รายเดือน): MFA ทุกที่; ไม่มี access key อายุยาวตรงที่ role ทำแทนได้; รัน credential report; ลบ user และ key ที่ไม่ใช้; ตั้งคำถามกับทุกเครื่องหมาย * ในทุก policy คุณสร้างปฏิกริยานี้ไว้แล้วใน Lab 5 — ตอนนี้นั่นคือนัดในปฏิทิน

2. **การแพตช์:** ซอฟต์แวร์ที่ไม่แพตช์คือจุดเริ่มของการเจาะระบบส่วนใหญ่ บน instance ของคุณ: อัปเดตตามตาราง (และรู้ไว้ว่า **SSM Patch Manager** ทำให้อัปเดตโน้ตได้ทั้งฝูง); ในโลก container การแพตช์มักหมายถึงการ build image ใหม่จาก base ที่อัปเดตแล้ว — โยงกลับไปโมดูล 8
3. **Backup — แบบที่ทดสอบแล้ว:** backup ที่ไม่เคยทดสอบคือความหวัง ไม่ใช่แผน RDS snapshot, S3 **versioning** (ทุกการเขียนทับเก็บเวอร์ชันเอาไว้ — เกราะกันมือลั่นและกัน ransomware) Lab: snapshot database ใน lab ของคุณ, *กู้คืน* เป็น instance ใหม่, ตรวจสอบแถวข้อมูล, รื้อถอน ตอนนี้คุณพูดคำว่า “tested” ในห้องสัมภาษณ์ได้แล้ว
4. **Guardrail (รู้กัน):** เปิด **CloudTrail** (บันทึกตรวจสอบของการเรียก API ในบัญชี — ใครทำอะไร เมื่อไหร่) แล้วไล่อ่านเหตุการณ์ของเมื่อวานสักหนึ่งรอบ; เปิด free trial ของ **GuardDuty** (การตรวจจับภัยคุกคามอัตโนมัติ) แล้วอ่านว่ามันเฝ้าดูอะไรบ้าง รู้จักวลี **encryption at rest and in transit** (การเข้ารหัสทั้งตอนพักและตอนเดินทาง) และรู้ว่าใน AWS มันส่วนใหญ่คือช่องดักที่มี **KMS** หนุนหลัง — เดิมพันขั้นต่ำของวงการ เปิดเสมอ

การจัดการต้นทุน — ทักษะที่ทำให้ junior ดูเป็น senior หัวข้อใน JD เขียนไว้คำต่อคำ: “manage cloud costs through rightsizing resources, implementing auto-scaling, resource tagging” มิเตอร์เดินอย่างไร: compute คิดเงินต่อวินาทีที่มันเปิดอยู่ (ว่าง ≠ ฟรี — “เราลิมิตมัน” คือความสูญเปล่าคลาสสิก), storage คิดต่อ GB-เดือน และ **egress** (ข้อมูลออกจาก AWS) คิดต่อ GB ขณะที่ข้อมูลขาเข้าฟรี — ตัวเซอร์ไพรส์บิลชื่อดัง คำนโยกต่าง ๆ เรียงตามลำดับที่ junior ดึงได้: **ปิดมันซะ** (ระบบ dev ตอนกลางคืน; นิสัย teardown ของคุณ ยกกระต๊อบเป็นอุตสาหกรรม) → **rightsizing** (server ส่วนใหญ่สเปกเกินจำเป็น; เช็คประวัติ CPU ใน CloudWatch แล้วหด) → **ติด tag ทุกอย่าง** (Project, Owner, Environment — ค่าใช้จ่ายไร้ tag คือค่าใช้จ่ายไร้เจ้าภาพ; บังคับ tag ใน Terraform ของคุณ) → **จัด tier ให้ storage** (lifecycle rule จาก Lab 2) → รู้ว่ามี **reserved capacity/Savings Plans** (ผูกมัด 1–3 ปี ลด 30–70%) และ **Spot** (ลดได้ถึง 90% แต่ถูกเรียกคืนได้) สำหรับ workload คงที่และงาน batch ตามลำดับ เครื่องมือ: **Cost Explorer** (ค่าใช้จ่ายของบัญชีในรูปกราฟ — อ่านรายสัปดาห์ ตลอดไป) และ **AWS Budgets** (alert \$0 ของคุณจากสัปดาห์ที่ 3 ตอนนี้เข้าใจแล้วว่าเป็นสมาชิกตัวเล็กสุดของครอบครัวที่จริงจัง) วินัยภาพใหญ่นี้มีชื่อว่า **FinOps** — คุ่มค่าสองนาที่: https://www.youtube.com/watch?v=Y-c_xw9bHFW

คำศัพท์:

คำศัพท์	ความหมาย
Shared responsibility model	AWS ดูแลความปลอดภัยของคลาวด์; คุณดูแลสิ่งที่ใส่เข้าไป เหตุร้ายส่วนใหญ่อยู่ครึ่งหลัง
CloudTrail	บันทึกตรวจสอบของบัญชี: ทุกการเรียก API โดยใคร เมื่อไหร่
GuardDuty	การตรวจจับภัยคุกคามอัตโนมัติของ AWS จาก log และทราฟฟิก
Patching / SSM	การลงอัปเดตความปลอดภัย / Systems Manager ของ AWS ที่ทำให้อัปเดตโน้ตทั้งฝูง

คำศัพท์	ความหมาย
S3 versioning	เก็บทุกเวอร์ชันที่ถูกเขียนทับ — เกราะกันมือลั่น กัน ransomware
KMS / encryption at rest & in transit	บริการจัดการกุญแจของ AWS / ข้อมูลถูกเข้ารหัสบน ดิสก์และบนสาย เปิดเสมอ
Egress	ข้อมูลที่ออกจาก AWS — คิดเงินต่อ GB; ขาเข้าฟรี ตัว เซอร์ไพรส์บิลคลาสสิก
Rightsizing	หัดทรัพยากรที่สเปกเกินให้พอดีกับความต้องการที่วัดจริง เงินฟรี
Tagging	ติดป้ายทุกทรัพยากรด้วย owner/project/environment เพื่อให้ทุกค่าใช้จ่ายมีเจ้าภาพ
Savings Plans / Spot	ผูกมัด 1–3 ปีลด 30–70% สำหรับโหลดคงที่ / กำลังว่างที่ เรียกคืนได้ ลดถึง 90% สำหรับงาน batch
Cost Explorer / Budgets	กราฟค่าใช้จ่ายที่คุณอ่านรายสัปดาห์ / การแจ้งเตือนที่ทำให้คุณไม่ต้องเรียนรู้จากใบแจ้งหนี้
FinOps	วินัยของการทำให้ค่าใช้จ่ายคลาวด์มองเห็นได้ จัดสรรได้ และถูกปรับให้ดีขึ้นต่อเนื่อง

พูดออกเสียง: “Who has access to prod, and when did we last review the list?” (ใครเข้าถึง prod ได้บ้าง แล้วเราทบทวนรายชื่อครั้งสุดท้ายเมื่อไหร่?) · “When did we last *restore* a backup?” (ครั้งสุดท้ายที่เรา*กู้คืน* backup จริง ๆ คือเมื่อไหร่?) · “What’s untagged, and what died but is still billing?” (อะไรที่ยังไม่มี tag และอะไรที่ตายแล้วแต่ยังคิดเงินอยู่?) · “It’s over-provisioned — the CPU history says we can halve it.” (มันสเปกเกิน — ประวัติ CPU บอกว่าเราลดลงครึ่งหนึ่งได้)

แบบฝึกหัด: (1) lab ทดสอบกู้คืนข้างบน (2) ไฟล์เช็กलिस्टความปลอดภัยรายเดือนใน repo ของคุณ แล้วรันมันกับ บัญชีจริง (3) ใน Cost Explorer หาสิ่งที่แพงที่สุดในประวัติบัญชีของคุณ แล้วอธิบายในหนึ่งประโยคลงสมุด (4) เพิ่ม default tag ให้ทุกทรัพยากรใน Terraform repo ของคุณ

Milestone — จบช่วงที่ 3: รันการตรวจสอบตัวเองเต็มรูปแบบหนึ่งรอบ — เช็กलिस्टความปลอดภัย, รีวิวต้นทุน, ทดสอบ alarm, กู้คืน backup — แล้วเขียนรายงานหนึ่งหน้า ตอนนี้คุณทำตัวงานได้แล้ว ไม่ใช่แค่การสร้าง สิ่งที่เหลือคือพิสูจน์ให้คนแปลกหน้าเห็น: ช่วงที่ 4

ช่วงที่ 4 — ความพร้อมเข้าทำงาน (เดือน 8–12)

โมดูล 11 (เดือน 8–10): โปรเจกต์พอร์ตโฟลิโอสามชิ้น

ใบรับรองบอกว่าคุณเรียนมา; โปรเจกต์พิสูจน์ว่าคุณสร้างเป็น สามชิ้น แต่ละชิ้นอยู่ใน GitHub repo ของตัวเอง แต่ละชิ้นมีแผนภาพสถาปัตยกรรม, README ที่เขียนเพื่อ hiring manager (คืออะไร ทำไม รันอย่างไร ต้นทุน เท่าไหร่) และสคริปต์ teardown สร้าง → แคปหน้าจอ/อัดวิดีโอ → ทำลาย — repo คือชิ้นงาน ไม่ใช่บิลที่เดินอยู่

โปรเจกต์ 1 — เว็บแอปสามชั้น อัตโนมัติเต็มรูปแบบ (ชิ้นเอก) Terraform สร้างทุกอย่าง: VPC ที่มี public/private subnet พาดสอง AZ; **Application Load Balancer** (อุปกรณ์กระจายกราฟฟิก — ของใหม่สำหรับคุณ และเป็น ก้าวต่อโดยธรรมชาติจาก Lab 3) หน้า **Auto Scaling Group** ของ web instance (AWS เพิ่ม/ลด instance ตาม โหลด — คั่นดู แล้วต่อสายมัน); RDS ใน private subnet; S3 สำหรับ static asset; CloudWatch alarm และ runbook หนึ่งเล่ม GitHub Actions รัน terraform plan ทุก pull request และ apply ตอน merge — pipeline โครงสร้างพื้นฐานของจริง โปรเจกต์เดียวนี้สาธิตหัวข้อ JD สิบจากสิบสี่ข้อในตารางจับคู่; คาดหวังได้เลยว่าทุกสัมภาษณ์จะไล่ดูมัน และซ่อมเล่าเรื่องมันในห้านาทีให้คล่อง

โปรเจกต์ 2 — serverless data pipeline (ความหลากหลาย) ไม่มี server ลักเครื่อง: ไฟล์ที่ตกลงใน S3 bucket จะ trigger **Lambda function** (Python ของคุณ ที่ AWS รันให้ต่อเหตุการณ์ คิดเงินต่อการเรียก) ให้ประมวลผล มัน — parse ไฟล์ CSV, สรุปมัน, เขียนผลลัพธ์ลง bucket ที่สองหรือตาราง DynamoDB — พร้อมความล้มเหลว ที่ถูกจับ, log เข้า CloudWatch และ alarm เข้ากล่องอีเมลของคุณ deploy ด้วย Terraform มันโชว์ Python, การ คิดแบบ event-driven และความกว้างที่พ้นจาก EC2 ปฐมนิเทศสองนาทีก่อน:

https://www.youtube.com/watch?v=W_VV2Fx32_Y

โปรเจกต์ 3 — โഴว์เคสงานปฏิบัติการระดับ production (ตัวสร้างความต่าง) เอาโปรเจกต์ 1 มาแล้วดูแลมัน เหมือนมันสำคัญจริง: CloudWatch dashboard ของ golden signals; alarm พร้อมนโยบายการเพิกที่จัดทำเป็น เอกสาร; runbook สามเล่ม; ขั้นตอน backup-and-restore ที่ทดสอบแล้วพร้อมหลักฐาน; รอบ security-hardening (ตรวจ IAM, บันทึกรการแพตช์, เปิด CloudTrail) เขียนสรุป; บทวิเคราะห์ต้นทุน (“สแตกนี้ราคา \$X/ เดือน; นี่คือการเปลี่ยนแปลงที่จะลดลงครึ่งหนึ่ง”); และ post-mortem จาก game day หนึ่งฉบับ ผู้สมัคร junior แทบไม่มีใครมีสิ่งนี้ มันตอบคำถามเดียวที่การสัมภาษณ์ถามจริง ๆ — “ไว้ใจคนนี้กับ production ได้ไหม?” — ด้วยเอกสาร ไม่ใช่คำคุณศัพท์

โมดูล 12 (เดือน 8–12 คู่ขนาน): ตารางสอบใบรับรอง

ใบรับรองเปิดฟิลเตอร์ของ recruiter และจัดโครงสร้างการเรียนรู้ของคุณ เส้นทางมาตรฐานปี 2569 (ค.ศ. 2026) สำหรับตำแหน่งนี้ พร้อมเวลาเตรียมตัวที่สมจริงตามจังหวะของหลักสูตรนี้ — สังเกตว่าแต่ละการสอบมาถึงพอดี หลังช่วงของหลักสูตรที่สอนเนื้อหาของมัน ซึ่งเป็นเหตุที่เวลาเหล่านี้สั้นกว่าที่อินเทอร์เน็ตบอก:

ลำดับ	ใบรับรอง	พิสูจน์อะไร	เวลาเตรียมตัวจากจุดที่คุณจะยืนอยู่	สอบเมื่อไหร่
1	AWS Certified Cloud Practitioner (CLF-C02)	คำศัพท์คลาวด์, billing, shared responsibility	3–4 สัปดาห์ (คู่มือต่าง ๆ ก็ว่าเท่านี้สำหรับมือใหม่; ช่วงที่ 0–1 ครอบคลุมเนื้อหาส่วนใหญ่แล้ว)	~เดือน 4
2	AWS Solutions Architect Associate (SAA-C03)	การออกแบบสถาปัตยกรรม AWS จริง — ใบรับรองที่ JD ใช้กรองจริง ๆ	เตรียมเข้มข้น 6–8 สัปดาห์ (ค่าประมาณมาตรฐานอุตสาหกรรม หลังจบ CCP)	เดือน 8–9
3	HashiCorp Terraform Associate (003)	ความคล่อง IaC แบบมีตารางประทับ	2–3 สัปดาห์ — หลังโมดูล 7 และโปรเจกต์ 1 คุณแค่ทบทวนเป็นหลัก (เตรียมตัวทางการ: https://developer.hashicorp.com/terraform/tutorials/certification-003)	เดือน 10–11

ใบที่สี่แบบเลือกได้ ถ้าเล็งตำแหน่งซื่อสาย ops: **AWS SysOps Administrator / CloudOps Associate** — เนื้อหาของมันคือช่วงที่ 3 แบบตรงตัว สำหรับ CLF-C02 คอร์สเต็มของ freeCodeCamp (<https://www.youtube.com/watch?v=7HKot-brXFE>, ฉบับคลาสสิก <https://www.youtube.com/watch?v=NhDYbskXRgc>) บวกข้อสอบฝึกหัดคือเส้นทางฟรีที่คนเดินกันจนเจียน; สำหรับ SAA-C03 จับคู่คอร์สกับข้อสอบจับเวลาหลาย ๆ ชุด — ข้อสอบเป็นแบบสถานการณ์และความยืดหยุ่น

โมดูล 13 (เดือน 11–12): เรซูเม่ การสัมภาษณ์ และลิบคำถาม

เรซูเม่: หนึ่งหน้า นำด้วยทักษะ (AWS, Terraform, Python/Bash, Docker, CI/CD, CloudWatch — สะท้อนถ้อยคำของ JD เป๊ะ ๆ; ฟิเตอร์อัตโนมัติจับคู่คีย์เวิร์ด) และโปรเจกต์สามชิ้น แต่ละชิ้นเป็นสองหัวข้อในรูปแบบ *ทำ X ด้วย Y ได้ผล Z*: “Deployed a three-tier web app on AWS with Terraform and GitHub Actions; zero-downtime deploys via ALB + Auto Scaling” ใส่ใบรับรองพร้อมวันที่ ลิงก์ GitHub — แล้วสมมติเลยว่าเขาจะเปิดดูจริง เพราะบริษัทดี ๆ ทำ และของคุณตอนนี้คุ้มค่าการมาเยือน

การเตรียมสัมภาษณ์: สามารถหาที่ต้องซ้อม — คำถามความรู้ (ตารางคำศัพท์ของแต่ละช่วง), สถานการณ์ (lab ที่คุณลงมือทำมาแล้วจริง ๆ) และเชิงพฤติกรรม (เรื่องเล่าจากสมุดบันทึกของคุณ เล่าในรูปแบบ **STAR**: Situation, Task, Action, Result) ฝึกออกเสียง ทุกวัน สองสัปดาห์ — ดีที่สุดคือกับผู้สัมภาษณ์ AI แบบไม่ปราณี ลิบคำถามด้านล่างครอบคลุมของคลาสสิก; คำตอบตัวอย่างใจเขียนกระชับ — จงขยายแต่ละข้อด้วยรายละเอียด lab ของคุณเอง เพราะประโยค “...และตอนที่ผมสร้างอันนี้ สิ่งที่เกิดขึ้นจริงคือ...” คือประโยคที่แยกคุณออกจากผู้สมัครที่แค่อ่านมา

สืบคำถาม พร้อมคำตอบที่แข็งแรง:

1. **“Explain the difference between a public and a private subnet.” (อธิบายความต่างระหว่าง public กับ private subnet)** route table ของ public subnet มี route ไป internet gateway ทรัพยากรของมันจึงถูกเข้าถึงจากอินเทอร์เน็ตได้; private subnet ไม่มี route แบบนั้น web tier อยู่ public, database อยู่ private และการเข้าถึงเชิงแอดมินไปถึงเครื่อง private ผ่าน bastion หรือ SSM ใน lab ของผม ผมพิสูจน์แล้วว่า instance ฝั่ง private เข้าไม่ถึงจากแล็บที่ออฟแต่เข้าถึงได้จาก tier public — route ที่หายไปนั่นแหละคือความปลอดภัย
2. **“What is IAM, and what is least privilege?” (IAM คืออะไร และ least privilege คืออะไร)** IAM ควบคุมว่าตัวตนไหนทำการกระทำไหนได้กับทรัพยากรไหน ผ่าน user, group, role และ policy รูปแบบ JSON least privilege คือการให้สิทธิ์ขั้นต่ำที่ทำงานสำเร็จ — user ของสคริปต์ audit ของผมมีสิทธิ์อ่านอย่างเดียวกับบริการเดียวเป๊ะ ๆ และ EC2 instance ของผมใช้ role แทน key ที่เก็บไว้ จึงไม่มี credential อายุยาวให้รั่ว
3. **“An EC2 web server stopped responding. Walk me through your troubleshooting.” (EC2 web server หยุดตอบสนอง ไล่ขั้นตอนการแก้ปัญหาของคุณให้ฟังหน่อย)** เริ่มจากขอบเขต: เครื่องเดียวหรือทั้งหมด — เช็ค CloudWatch dashboard และ alarm ที่มี จากนั้นไล่ขั้นตอนตามลำดับ: instance status check; security group — 80/443 เปิดจริงไหม; SSH เข้าไป — nginx รันอยู่ไหม (systemctl status), ดิสก์เต็มไหม (df -h), CPU ตันไหม (top); แล้วก็ log (tail ที่ error log) mitigate ก่อน — restart service หรือเปลี่ยนเครื่อง — แล้วค่อยหาสาเหตุรากหลังระบบนิ่ง นั่นคือลำดับที่ผมซ้อมไว้ใน runbook ของผม
4. **“Why Terraform instead of clicking in the console?” (ทำไมต้อง Terraform แทนการคลิกในคอนโซล)** คอนโซลไม่ทิ้งบันทึกที่รีวิวได้และสร้างอะไรซ้ำไม่ได้ Terraform เป็น declarative และมีเวอร์ชัน: การเปลี่ยนแปลงผ่าน pull request, config เดียวกันสร้าง environment ที่เหมือนกันเป๊ะ, การเปลี่ยนแปลงที่เลวร้าย roll back ได้ด้วยการ revert commit และ plan โชว์ diff เป๊ะ ๆ ก่อนอะไรจะเกิดขึ้น pipeline ของผมรัน plan ทุก PR ให้ diff เป็นส่วนหนึ่งของการรีวิว
5. **“What’s the difference between a container and a virtual machine?” (container ต่างจาก virtual machine อย่างไร)** VM จำลองฮาร์ดแวร์และแบก OS เต็มตัว ใช้เวลาบูตเป็นนาที; container แอร์ kernel ของเครื่องแม่และบรรจุแค่แอปกับ dependency ของมัน สตาร์ทในราวหนึ่งวินาที container ให้พฤติกรรมเหมือนกันข้าม environment — image เดียวกันรันบนแล็บที่ออฟผมและบน EC2 ส่วน VM ยังสำคัญสำหรับ isolation และสำหรับ workload ที่ไม่ใช่ Linux
6. **“How would you reduce our AWS bill?” (คุณจะลดบิล AWS ของเราอย่างไร)** เรียงตามแรงที่ใช้: หาสิ่งที่รันอยู่ทั้งที่ไม่ควร — instance วางงาน, volume ที่ไม่ถูก attach, snapshot เก่า; Cost Explorer กับ tag ทำให้มองเห็น จากนั้น rightsize เทียบประวัติ CloudWatch จากนั้นตั้งเวลาปิด environment dev นอกเวลางาน จากนั้น lifecycle ข้อมูล S3 เก่าไป tier ถูกลง จากนั้นผูก workload คงที่กับ Savings Plans และวางงาน batch ที่ขัดจังหวะได้บน Spot และจับตา egress — ข้อมูลขาออกคือบรรทัดเซอร์ไพรส์คลาสสิก
7. **“What happens when you type a URL and press Enter?” (เกิดอะไรขึ้นเมื่อพิมพ์ URL แล้วกด Enter)** DNS แปลงชื่อเป็น IP; เบราร์เซอร์เปิดการเชื่อมต่อ TCP ไปพอร์ต 443 และเจรจา TLS; ส่ง HTTP GET; ฝั่งโน้นมันไปเตะ load balancer ซึ่งส่งต่อให้ instance ที่แข็งแรงใน public subnet; แอปอาจ query database ใน private subnet; คำตอบกลับมาพร้อม status code — 200 ถ้าแข็งแรง — แล้วเบราร์เซอร์เรนเดอร์ ผมเจาะลึกได้ทุกช่วงต่อ และผมสร้างมาแล้วทุกอัน

8. “A teammate’s change broke production. What happens next?” (การเปลี่ยนแปลงของเพื่อนร่วมทีมทำ production พัง แล้วอะไรต่อ) mitigate ก่อน — roll back ผ่าน pipeline; ในวินาทีนั้นการกู้คืนชนะการวินิจฉัย จากนั้น post-mortem แบบ blameless: เกิดอะไร ทำไมระบบยอมให้มันเกิด และ guardrail ตัวไหนที่ขาด — การทดสอบ, การรีวิว plan, alarm สักตัว blameless สำคัญเชิงปฏิบัติ: คนที่กลัวการลงโทษจะซ่อนข้อมูล และข้อมูลที่ถูกซ่อนทำให้เกิดซ้ำ
9. “What is the shared responsibility model?” (shared responsibility model คืออะไร) AWS ดูแลความปลอดภัยของคลาวด์ — data center, ฮาร์ดแวร์, hypervisor; ลูกค้าดูแลสิ่งที่อยู่ข้างใน — ข้อมูล, IAM, กฎเครือข่าย, การแพตช์, การตั้งค่าเข้ารหัส เหตุรั่วไหลบนคลาวด์จริงส่วนใหญ่อยู่ฝั่งลูกค้า เป็นการตั้งค่าผิดอย่าง bucket สาธารณะหรือ key หลุด ดังนั้น ‘AWS ปลอดภัย’ กับ ‘เราปลอดภัยบน AWS’ เป็นคนละประโยคกัน และประโยคที่สองคืองานของผม
10. “Tell me about a problem you debugged.” (เล่าปัญหาที่คุณเคยดีบั๊กให้ฟังหน่อย) (ของตัวเอง จากสมุดบันทึก ในรูป STAR โครงตัวอย่าง:) ระหว่าง game day ด้าน monitoring ของผม alarm ทำงานเพราะ CPU อิมตัว (S) ผมต้องหาและหยุดต้นเหตุก่อนที่ ‘ผู้ใช้’ จะสังเกต (T) metric ชี้จุดเริ่มของ spike; Logs Insights โยงมันเข้ากับ process ที่หลุดควบคุม; ผม SSH เข้าไป ยืนยันด้วย top ข้ามมันทิ้ง แล้วเฝ้าดูการฟื้นตัวบน dashboard (A) แก่จบในสิบเอ็ดนาทีก และ post-mortem ให้กำเนิด alarm นับจำนวน process ที่ผมลงมือทำจริงใน Terraform (R)

กลไกการทำงาน: สมัครตั้งแต่เดือน 11 — หลังได้ SAA อย่างรวดเร็ว “พร้อม” เพราะการสัมภาษณ์คือการฝึกซ้อม ตำแหน่งเป้าหมาย: cloud engineer (junior/associate), cloud support engineer, cloud operations engineer, junior DevOps engineer, AWS support associate ทุกการถูกปฏิเสธที่มีสัมภาษณ์คือบทเรียนฟรี; จดลงสมุดว่า เขาถามอะไร

คำศัพท์:

คำศัพท์	ความหมาย
Portfolio	หลักฐานสาธารณะพร้อมเอกสารว่าคุณสร้างเป็น — สำหรับอาชีพนี้คือ GitHub repo พร้อมแผนภาพและ README
Application Load Balancer (ALB)	ตัวกระจายทราฟฟิกของ AWS: แจกคำขอให้ instance ที่แข็งแรง ตัดตัวป่วยทิ้ง
Auto Scaling Group	รักษา instance ไว้ N เครื่องและปรับ N ตามโหลด — self-healing กับความยืดหยุ่นในตัวเดียว
Lambda / serverless	โค้ดที่ AWS รันต่อเหตุการณ์ คิดเงินต่อการเรียก — ไม่มี server ให้ดูแลเลย
DynamoDB	ตาราง NoSQL แบบ serverless ของ AWS — เข้าคู่กับ Lambda โดยธรรมชาติ

คำศัพท์	ความหมาย
STAR	Situation, Task, Action, Result — รูปทรงของเรื่องเล่า สัมภาษณ์ที่ดี
ATS	Applicant tracking system — ฟิลเตอร์คีย์เวิร์ดที่เรซูเม่ หน้าเดียวของคุณต้องผ่าน
CLF-C02 / SAA-C03 / Terraform Associate 003	สามการสอบของคุณ: คำศัพท์, สถาปัตยกรรม, IaC
SysOps / CloudOps Associate	ใบ associate สาย ops ของ AWS แบบเลือกได้ — ช่วงที่ 3 ในรูปข้อสอบ

Milestone — จบหลักสูตร: สาม repo ที่คนแปลกหน้าเข้าใจได้, ใบรับรองสามใบที่นัดสอบหรือสอบผ่านแล้ว, คำตอบสิบข้อที่ซ่อมออกเสียงแล้ว, ใบสมัครที่ยื่นแล้ว คุณไม่ใช่คนที่ “หวังจะได้เข้าวงการคลาวด์” คุณคือวิศวกรคลาวด์ระดับ junior ที่มีหลักฐาน และกำลังสัมภาษณ์อยู่

หลักสูตรย่อหน้า

เมื่อไหร่	จุดเน้น	หลักฐานจากการลงมือ	หลักฐานภายนอก
สัปดาห์ 1–2	คอมพิวเตอรืทำงาน อย่างไร; Linux, terminal, permission, SSH	จัดการไฟล์ด้วยสคริปต์; เซลล์สคริปต์แรก	—
สัปดาห์ 3–4	เครือข่าย: IP, DNS, พอร์ต, HTTP, firewall; บัญชี AWS	บัญชี + MFA + budget \$0	—
สัปดาห์ 5–6	Lab 1: EC2	web server บน อินเทอร์เน็ต แล้วรีบูต	—
สัปดาห์ 7	Lab 2: S3 + CLI	เว็บ static; ความคล่อง CLI	—
สัปดาห์ 8–9	Lab 3: VPC	เครือข่าย public/private, พิสูจน์ bastion	—
สัปดาห์ 10	Lab 4: RDS	database ส่วนตัว, snapshot, teardown	—

เมื่อไหร่	จุดเน้น	หลักฐานจากการลงมือ	หลักฐานภายนอก
สัปดาห์ 11–12	Lab 5: IAM; สร้างด่าน รบบยอตช่วงที่ 1 ใหม่	สแตกเต็มจากความจำ, <3 ชม.	นัดสอบ CCP
สัปดาห์ 13–15	Bash, Python/boto3, Git	backup.sh, audit.py, ประวัติ repo	สอบ CCP (~เดือน 4)
สัปดาห์ 16–17	Terraform	สแตกในรูปโค้ด, plan/apply/destroy	—
สัปดาห์ 18–20	CI/CD, Docker, อ่าน K8s ออก	pipeline เขียน; image ใน registry	—
สัปดาห์ 21–24	CloudWatch, log, incident, runbook	alarm ทำงานจริง + post- mortem จาก game day	—
สัปดาห์ 25–28	security ops; การจัดการ ต้นทุน	รายงานตรวจสอบตัวเอง; ทดสอบกู้คืน	—
เดือน 8–10	โปรเจกต์พอร์ตโพลีโอ 1– 3	สาม repo พร้อมเอกสาร	SAA-C03 (เดือน 8–9)
เดือน 10–11	เตรียม Terraform Associate	—	Terraform Associate
เดือน 11–12	เรซูเม่, สัมภาษณ์, สมัคร งาน	คำตอบสลิปข้อ ออกเสียง	ข้อเสนองานแรก

คำส่งท้ายจากครูของคุณ สิบสองเดือนนั้นสั้นสำหรับการเปลี่ยนอาชีพ และยาวสำหรับนิสัยรายวัน ฉะนั้นนี่คือข้อตกลงที่ตรงไปตรงมา: คนที่เรียนจบหลักสูตรนี้ไม่ใช่คนที่ฉลาดที่สุด — แต่คือคนที่พิมพ์คำสั่งแม้ในวันที่เหนื่อยด้วย ทุกข้อความ error ที่คุณเจอคือหลักสูตรที่กำลังทำงาน ไม่ใช่ล้มเหลว; วิศวกรที่เคยพังและซ่อมของเล็ก ๆ มาร้อยอย่างคือสิ่งที่ hiring manager หมายถึงเวลาพูดคำว่า “ประสบการณ์” เป๊ะ ๆ จดสมุดบันทึกต่อไป รื้อถอนสิ่งที่สร้าง อย่าอ้างในห้องสัมภาษณ์ในสิ่งที่ repo ของคุณหนุนหลังไม่ได้ — แล้วอีกหนึ่งปีจากนี้ เมื่อเพจเจอร์ดังตอนตีสาม คุณจะรู้สึกถึงบางอย่างที่คาดไม่ถึงอยู่ใต้อะดรีนาลีน: ความสามารถ ไปสร้างกันเถอะ

Sources

งานวิจัยประกาศรับสมัครงาน (สิงหาคม 2569): [Arc.dev — AWS Cloud Engineer Job Description](#) · [Wiz — Cloud Engineer Job Description Guide](#) · [DevsData — AWS Cloud Engineer JD Template](#) · [XOPA — Cloud Engineer JD Template 2026](#) · [Betterteam — Cloud Engineer Job Description](#) เส้นทางใบรับรองและเวลาเตรียมตัว: [StudyTech — AWS Certification Roadmap 2026](#) · [Cloud Evolvers — Cloud Engineer Roadmap](#)

[2026 · HashiCorp — Terraform Associate 003_prep](#) บริบทเงินเดือน (มัธยฐานสหรัฐฯ ≈ \$104K, ช่วง \$85K–\$140K): Wiz ด้านบน ลิงก์ YouTube ทั้งหมดตรวจสอบผ่าน metadata ของ YouTube ณ เวลาที่เขียน

หลักสูตรในชุด B4LCILC — เล่มคู่หูของ “The Cloud Leader Course”

บทเรียนใหม่ — ฉบับเดือนสิงหาคม 2026

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

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

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

สำหรับหลักสูตรวิศวกรคลาวด์ (Cloud Engineer)

serverless แท้จริงแล้วคืออะไร

ทุกสิ่งที่คุณทำให้เป็นอัตโนมัติมาจนถึงตอนนี้ ยังตั้งอยู่บนสมมติฐานว่ามีเครื่องอยู่หนึ่งเครื่อง: cron job ในห้องปฏิบัติการของโมดูลนี้รันวันละสองนาทีก่อน virtual machine ที่คุณจ่ายเงินทั้งเดือน ต้อง patch ต้องเฝ้าดู และสักวันหนึ่งต้อง debug ตอนดีสาม serverless คือคำตอบของคลาวด์ต่อความไม่ลงรอยนั้น server ยังคงมีอยู่จริง —

ชื่อนี้เป็นเรื่องการตลาด — แต่มันกลายเป็นปัญหาของผู้ให้บริการ คุณส่งมอบฟังก์ชันหนึ่งตัว ผู้ให้บริการรันมันเมื่อมีบางสิ่งมากระตุ้น และคุณจ่ายตามจำนวนครั้งที่เรียกใช้ คิดเป็นรายมิลลิวินาที โมเดลนี้เรียกว่า **FaaS — Function as a Service**: บน AWS คือ **Lambda** บน Azure คือ **Azure Functions** แนวคิดถ่ายโอนกันได้ชื่อต่อชื่อ

กายวิภาคของมันวางบนฝ่ามือเดียวได้ **handler** คือฟังก์ชันที่แพลตฟอร์มเรียกใช้ และมันรับ **event** เข้ามา — คำบรรยายในรูป JSON (JavaScript Object Notation) ของสิ่งที่เพิ่งเกิดขึ้น: มีไฟล์ตกลงมาใน bucket มี HTTP request เข้ามา หรือตารางเวลาหนึ่งดังขึ้น นั่งตัวตรงฟังข้อสุดท้ายให้ดี: “ตารางเวลาดังขึ้น” คือสิ่งเดียวกับที่ cron ทำเป๊ะ ๆ — เพียงแต่ Lambda ที่รันตามตารางเวลาไม่ต้องมีเครื่องเปิดรออยู่ระหว่างรอบ การแปลงสคริปต์งานปฏิบัติการตัวหนึ่งของโมเดลนี้จึงเป็นโปรเจกต์ serverless ชั้นแรกสุดคลาสสิก

เศรษฐศาสตร์คือเหตุผลที่โมเดลนี้ถือกำเนิดขึ้น และมันคมทั้งสองด้าน free tier ของ AWS ให้การเรียก Lambda หนึ่งล้านครั้งต่อเดือน ตลอดไป — ฟังก์ชันในห้องปฏิบัติการของคุณจะรันทุกนาทีตลอดทั้งปีก็ไม่เสียเงิน ทำให้ serverless เป็นบ้านที่ถูกที่สุดสำหรับงานการประสานระบบปฏิบัติการ แต่ราคารายมิลลิวินาทีที่เป็นแค่เศษสตางค์เมื่อปริมาณต่ำ จะกลายเป็นเงินจริงเมื่อปริมาณสูง; ฟังก์ชันที่รันตลอดเวลาจะถูกกว่าถ้าอยู่ในรูป container หรือ VM ทักษะของมืออาชีพคือรู้วางงานชิ้นหนึ่งอยู่ปากไหนของเส้นแบ่งนั้น

ข้อควรระวังตรงไปตรงมาสองข้อ ข้อแรก **cold start**: ฟังก์ชันที่ไม่ได้รันมาสักพักต้องใช้เวลาไม่กี่ร้อยมิลลิวินาทีในการตื่น — ไม่มีนัยสำคัญสำหรับรายงานประจำคืน แต่เป็นข้อพิจารณาในการออกแบบสำหรับ API ที่อ่อนไหวต่อ latency ข้อสอง ฟังก์ชันยังคงรันในนามของตัวตนหนึ่ง: Lambda สวม role ของ IAM (Identity and Access Management) เหมือนที่ EC2 instance ทำทุกประการ และฟังก์ชันที่ได้สิทธิ์เกินจำเป็นก็คือความผิดพลาดด้านความปลอดภัยแบบเดียวกัน เพียงแต่อยู่ในห่อที่เล็กลง
