

ຫຼັກສູດວິສະວະກອນ Cloud (The Cloud Engineer Course)

ຈາກຄວາມຮູ້ສູນ ສູ່ວິສະວະກອນ cloud ທີ່ພ້ອມເຮັດວຽກແທ້ຈິງ — ຫຼັກສູດ B4LCILC

13 ສິງຫາ 2026

ຫຼັກສູດວິສະວະກອນ Cloud (The Cloud Engineer Course)

ຈາກຄວາມຮູ້ສູນ ສູ່ຄວາມພ້ອມເຮັດວຽກຢ່າງແທ້ຈິງ ໃນຖານະວິສະວະກອນ cloud.

ຫຼັກສູດນີ້ເຮັດວຽກແນວໃດ

ທ່ານກຳລັງຝຶກເພື່ອ ເປັນ ວິສະວະກອນ cloud: ຄົນທີ່ສ້າງ server (ເຊັ່ນເວີ — ຄອມພິວເຕີແມ່ຂ່າຍ) ດ້ວຍໂຄດ, ຮັກສາລະບົບ ໃຫ້ມີຊີວິດຢູ່ຕອນຕິດສາມ, ໃຊ້ອັດຕະໂນມັດກຳຈັດວຽກໜ້າເບື້ອງ, ແລະ ສາມາດເບິ່ງລະບົບທີ່ຟັງແລ້ວຄ່ອຍໆຊອກຫາສາເຫດຢ່າງ ສະຫງົບ. ນັ້ນຄືວິຊາຊີບທີ່ຕ້ອງລົງມືເຮັດຈິງ, ແລະ ຫຼັກສູດນີ້ກໍປະຕິບັດຕໍ່ມັນແບບນັ້ນ. ທ່ານຈະໃຊ້ເວລາພິມຫຼາຍກວ່າອ່ານຫຼາຍ ເທົ່າ — ທຸກ module ມີ lab (ບົດທົດລອງ), ແລະ lab ນັ້ນແຫຼະຄືຕົວຫຼັກສູດ. ການອ່ານກ່ຽວກັບ cloud ສ້າງພຽງການຈື່ຈຳ; ການລົງມືສ້າງໃນ cloud ຈຶ່ງສ້າງອາຊີບ.

ຫຼັກສູດນີ້ມີຫ້າໄລຍະ. **ໄລຍະທີ 0 (ອາທິດທີ 1–4): ພື້ນຖານ** — ຄອມພິວເຕີ ແລະ ເຄືອຂ່າຍເຮັດວຽກແທ້ຈິງແນວໃດ, ແລະ Linux, ລະບົບປະຕິບັດການທີ່ cloud ແລ່ນຢູ່ເທິງມັນ. **ໄລຍະທີ 1 (ອາທິດທີ 5–12): ແຖມ cloud, ລົງມືເຮັດຈິງ** — ຫ້າບໍລິການ AWS ທີ່ປາກົດໃນທຸກປະກາດຮັບສະໝັກວຽກ, ຮຽນຜ່ານ lab ທີ່ທ່ານສ້າງຂຶ້ນແລ້ວຮື້ຖອນ. **ໄລຍະທີ 2 (ອາທິດທີ 13–20): ອັດຕະໂນມັດ** — ການຂຽນ script, Git, Terraform, CI/CD, ແລະ container: ຫັກສະທີ່ແຍກວິສະວະກອນອອກຈາກຄົນຄລິກ console. **ໄລຍະທີ 3 (ອາທິດທີ 21–28): ການປະຕິບັດງານ** — ການຕິດຕາມກວດກາ, ເຫດການສຸກເສີນ, ຄວາມປອດໄພ, ແລະ ຕົ້ນທຶນ: ຫັກສະທີ່ເຂົ້າຈ້າງທ່ານມາເຮັດແທ້ໆ. **ໄລຍະທີ 4 (ເດືອນທີ 8–12): ຄວາມພ້ອມສູ່ວຽກ** — ສາມໂຄງການ portfolio, ສາມໃບຢັ້ງຢືນ (certification), ແລະ ການຝຶກສຳພາດ.

ກົດລະບຽບຕະຫຼອດຫຼັກສູດ: ຮຽນ **ມີລະ 1.5–2 ຊົ່ວໂມງ, ຫົກມື້ຕໍ່ອາທິດ** — ຄວາມສະໝັກສະເໝີຊະນະຄວາມໝັກໝ່ວງ, ແລະ ນິຕິວິຊາຊີບທີ່ຮຽນຮູ້ດ້ວຍການເຮັດຊ້າທຸກມື້ ຄືກັບເຄື່ອງດົນຕີ. ທຸກ module ຈົບລົງດ້ວຍ **ຕາຕະລາງຄຳສັບ** (ຄຳທີ່ທ່ານຕ້ອງ ເປັນເຈົ້າຂອງ), **ບົດຝຶກເວົ້າອອກສຽງ** (ປະໂຫຍກທີ່ທ່ານເວົ້າຊ້າຈົນເປັນທຳມະຊາດ — ການສຳພາດວຽກແມ່ນເວົ້າ, ບໍ່ແມ່ນ ຂຽນ), **ບົດຝຶກຫັດ** ແບບລົງມືເຮັດ, ແລະ **Milestone** (ຫຼັກໝາຍ) ທີ່ເປັນປະຕູກັ້ນທາງ: ຢ່າກ້າວຕໍ່ໄປຈົນກວ່າທ່ານຈະເຮັດມັນ ໄດ້, ເພາະທຸກໄລຍະຕັ້ງຢູ່ເທິງໄລຍະກ່ອນໜ້າ. ຕັ້ງແຕ່ອາທິດທີ 1, ໃຫ້ຮັກສາ **Engineering Journal** (ປຶ້ມບັນທຶກວິສະວະກຳ): ທຸກ lab, ທຸກຂໍ້ຄວາມ error, ທຸກການແກ້ໄຂ, ໜຶ່ງວັກແບບຊື້ສັດ. ຮອດເດືອນທີ 10 ປຶ້ມບັນທຶກນັ້ນຈະກາຍເປັນວັດຖຸດິບຂອງ portfolio ແລະ ເລື່ອງເລົ່າໃນການສຳພາດຂອງທ່ານ.

ຄຳໝັ້ນສັນຍາໜຶ່ງຂໍ້ເປັນການຕອບແທນ: ບໍ່ມີສິ່ງໃດໃນຫຼັກສູດນີ້ ທີ່ສົມມຸດວ່າທ່ານຮູ້ຫຍັງມາກ່ອນ. ທຸກຄຳສັບຖືກນິຍາມໃນຄັ້ງທຳອິດທີ່ມັນປາກົດ. ຖ້າທ່ານໃຊ້ web browser ເປັນ ແລະ ເຕັມໃຈພິມຄຳສັງເກດຮູ້ສຶກແປກປະຫຼາດໃນສອງອາທິດທຳອິດ, ທ່ານກໍມີເງື່ອນໄຂເບື້ອງຕົ້ນຄົບຖ້ວນຢ່າງແລ້ວ.

ປະກາດຮັບສະໝັກວຽກ ທີ່ທ່ານກຳລັງຝຶກໄປຫາ

ໃນເດືອນສິງຫາ 2026 ພວກເຮົາໄດ້ດຶງປະກາດຮັບສະໝັກ Cloud Engineer ຕົວຈິງ ຈາກແມ່ແບບຂອງນາຍຈ້າງ ແລະ ຄູ່ມືການຈ້າງງານ (Arc.dev, Wiz, DevsData, XOPA, Betterteam — ລາຍການເຕັມຢູ່ໃນ Sources). ເມື່ອປອກຊີບລິສັດອອກ ຂໍກຳນົດດຽວກັນກໍປາກົດຊ້ຳແລ້ວຊ້ຳອີກ. ຕາຕະລາງນິຕິສັນຍາຂອງທ່ານກັບຫຼັກສູດ — ທຸກຂໍ້ຫຼັກສູດຮຽກຮ້ອງ ຊື່ໄປຫາໄລຍະທີ່ສອນມັນ:

ສິ່ງທີ່ປະກາດຮັບສະໝັກຕົວຈິງຮຽກຮ້ອງ (ເກືອບຄຳຕໍ່ຄຳ)	ຫຼັກສູດນິສອນມັນຢູ່ໃສ
“Knowledge of Linux/Unix operating systems” (ຄວາມຮູ້ລະບົບປະຕິບັດການ Linux/Unix)	ໄລຍະທີ 0, Module 1
“Expertise in cloud networking including VPCs, subnets, load balancers, DNS” (ຄວາມຊ່ຽວຊານເຄືອຂ່າຍ cloud ລວມທັງ VPC, subnet, load balancer, DNS)	ໄລຍະທີ 0 Module 2 + ໄລຍະທີ 1 Lab 3
“Demonstrated expertise in core AWS services, including EC2, S3, RDS, VPC, IAM” (ຄວາມຊ່ຽວຊານທີ່ພິສູດໄດ້ ໃນບໍລິການແກນຂອງ AWS ລວມທັງ EC2, S3, RDS, VPC, IAM)	ໄລຍະທີ 1 (Labs 1–5)
“Design, develop, and deploy cloud infrastructure using infrastructure as code tools such as Terraform, CloudFormation” (ອອກແບບ, ພັດທະນາ, ແລະ deploy ໂຄງລ່າງ cloud ດ້ວຍເຄື່ອງມື infrastructure as code ເຊັ່ນ Terraform, CloudFormation)	ໄລຍະທີ 2, Module 7
“Proficiency in scripting languages such as Python, Bash, PowerShell, or Go” (ຄວາມຄ່ອງແຄ້ວໃນພາສາ script ເຊັ່ນ Python, Bash, PowerShell, ຫຼື Go)	ໄລຍະທີ 2, Module 6
“Build and maintain CI/CD pipelines using tools like Jenkins, GitLab CI, or GitHub Actions” (ສ້າງ ແລະ ຮັກສາ CI/CD pipeline ດ້ວຍເຄື່ອງມືເຊັ່ນ Jenkins, GitLab CI, ຫຼື GitHub Actions)	ໄລຍະທີ 2, Module 8
“Experience with containerization technologies including Docker and Kubernetes” (ປະສົບການເຕັກໂນໂລຊີ container ລວມທັງ Docker ແລະ Kubernetes)	ໄລຍະທີ 2, Module 8

ສິ່ງທີ່ປະກາດຮັບສະໝັກຕົວຈິງຮຽກຮ້ອງ (ເກືອບຄໍາຕໍ່ຄໍາ)	ຫຼັກສູດນິສົນນັ້ນຢູ່ໃສ
“Monitor infrastructure health and performance using cloud-native monitoring tools” (ຕິດຕາມສຸຂະພາບ ແລະ ປະສິດທິພາບຂອງໂຄງລ່າງ ດ້ວຍເຄື່ອງມື monitoring ແບບ cloud-native)	ໄລຍະທີ 3, Module 9
“Participate in incident response, including log analysis”; “troubleshooting and analytical skills” (ຮ່ວມ ຮັບມືເຫດການສຸກເສີນ ລວມທັງການວິເຄາະ log; ຫຼັກສະແກ້ໄຂ ບັນຫາ ແລະ ການວິເຄາະ)	ໄລຍະທີ 3, Module 10
“Implement and enforce security controls including encryption, identity and access management”; “least-privilege access” (ຈັດຕັ້ງ ແລະ ບັງຄັບໃຊ້ການຄວບຄຸມຄວາມ ປອດໄພ ລວມທັງການເຂົ້າລະຫັດ, ການຄຸ້ມຄອງຕົວຕົນ ແລະ ການເຂົ້າເຖິງ; ການເຂົ້າເຖິງແບບສິດໜ້ອຍທີ່ສຸດ)	ໄລຍະທີ 1 Lab 5 + ໄລຍະທີ 3 Module 10
“Manage cloud costs through rightsizing resources, implementing auto-scaling, resource tagging” (ຄຸ້ມຄອງ ຕົ້ນທຶນ cloud ຜ່ານການ rightsizing ຊັບພະຍາກອນ, ການໃຊ້ auto-scaling, ການຕິດ tag ຊັບພະຍາກອນ)	ໄລຍະທີ 3, Module 10
“Maintaining, testing and implementing disaster recovery procedures” (ຮັກສາ, ທົດສອບ ແລະ ຈັດຕັ້ງຂັ້ນຕອນ ກັບໄພພິບັດ)	ໄລຍະທີ 3 + ໂຄງການ Portfolio ທີ 3
“AWS certifications preferred” (ຕ້ອງການຜູ້ມີໃບຢັ້ງຢືນ AWS)	ຕາຕະລາງ cert ໄລຍະທີ 4 (CCP → SAA → Terraform Associate)
“Provide technical guidance and documentation”; “good communication and collaboration skills” (ໃຫ້ຄໍາແນະນຳ ທາງເຕັກນິກ ແລະ ເອກະສານ; ຫຼັກສະສື່ສານ ແລະ ຮ່ວມມືທີ່ດີ)	ປຶ້ມບັນທຶກ + runbook + ການກຽມສຳພາດໄລຍະທີ 4

ບໍລິບົດເງິນເດືອນ, ເພື່ອໃຫ້ທ່ານຮູ້ວ່າກຳລັງພະຍາຍາມໄປຫາຫຍັງ: ເງິນເດືອນວິສະວະກອນ cloud ໃນສະຫະລັດ ກະຈຸກຕົວຢູ່ **ຄ່າກາງໃກ້ \$104,000**, ຊ່ວງປົກກະຕິ **\$85K–\$140K**, ແລະ ຕໍ່າແໜ້ງ AWS ອາວຸໂສ/ຊ່ຽວຊານ ປະກາດສູງກວ່ານັ້ນຫຼາຍ. ຕໍ່າແໜ້ງລະດັບເລີ່ມຕົ້ນມີຢູ່ພາຍໃຕ້ຫຼາຍຊື່ — cloud support associate, junior cloud engineer, cloud operations engineer — ແລະ ຫຼັກສູດນີ້ເລັ່ງໃສ່ຂໍ້ກຳນົດຂອງພວກມັນໂດຍກົງ.

ໄລຍະທີ 0 — ພື້ນຖານ (ອາທິດທີ 1–4)

Module 1 (ອາທິດທີ 1–2): ຄອມພິວເຕີເຮັດວຽກແນວໃດ, ແລະ Linux — ພາສາຂອງ server

ແນວຄິດຫຼັກ: server ກໍຄືຄອມພິວເຕີທີ່ມີໜ້າທີ່ຮັບໃຊ້ຄອມພິວເຕີອື່ນ. ໃນດບຸກຕໍ່ໜ້າທ່ານ ກັບເຄື່ອງທີ່ແລ່ນ Netflix ຕ່າງກັນທີ່ຂະໜາດ ແລະ ຄວາມທົນທານ, ບໍ່ແມ່ນທີ່ຊະນິດ: ທັງສອງມີ **CPU** (ສ່ວນທີ່ເຮັດວຽກປະມວນຜົນ), **memory/RAM** (ບ່ອນເຮັດວຽກຊົ່ວຄາວຄວາມໄວສູງ, ຖືກລ້າງເມື່ອ restart), **disk** (ບ່ອນຈັດເກັບຖາວອນທີ່ຊ້າກວ່າ ແຕ່ຢູ່ລອດຜ່ານການ restart), ແລະ **network card** (ການເຊື່ອມຕໍ່ຫາທຸກສິ່ງອື່ນ), ທັງໝົດປະສານງານໂດຍ **operating system (OS)** (ລະບົບປະຕິບັດການ). ໃນດບຸກຂອງທ່ານຄົງແລ່ນ Windows ຫຼື macOS. ແຕ່ server ເກືອບທັງໝົດແລ່ນ **Linux** — OS ຟຣີ open-source ທີ່ໝັ້ນຄົງ, ຂຽນ script ໄດ້, ແລະ ຄວບຄຸມທັງໝົດດ້ວຍຄໍາສັ່ງທີ່ພິມເຂົ້າໄປ. ຈຸດສຸດທ້າຍນັ້ນຄືປະເດັນສໍາຄັນ: ທ່ານເຮັດອັດຕະໂນມັດການຄລິກເມົາສິບໄດ້, ແຕ່ເຮັດອັດຕະໂນມັດຄໍາສັ່ງໄດ້, ແລະ ວິສະວະກໍາ cloud ຄືອັດຕະໂນມັດ. ດັ່ງນັ້ນຄວາມຄ່ອງແຄ້ວທໍາອິດຂອງທ່ານແມ່ນ **terminal** (ເທີມິນອລ — ໜ້າຈໍພິມຄໍາສັ່ງ).

Terminal, ຖອດຄວາມລຶກລັບອອກ. Terminal (ຫຼື “shell” — ໂປຣແກຣມພາຍໃນມັນ ປົກກະຕິແມ່ນ **Bash**) ຄືການສົນທະນາດ້ວຍຕົວໜັງສືກັບຄອມພິວເຕີ. ທ່ານພິມຄໍາສັ່ງ; ມັນຕອບ. ພຽງເທົ່ານັ້ນ. ເຄື່ອງໝາຍ \$ ທີ່ທ່ານຈະເຫັນໃນຕົວຢ່າງຄື **prompt** — shell ກໍາລັງບອກວ່າ “ເຖິງຕາທ່ານແລ້ວ.” ທຸກສິ່ງໃນ Linux ເປັນໄຟລ໌, ໄຟລ໌ຢູ່ໃນຕົ້ນໄມ້ດຽວທີ່ເລີ່ມຈາກ root /, ແລະ ໂຟນເດີສ່ວນຕົວຂອງທ່ານແມ່ນ /home/yourname (ຊື່ຫຼິ້ນ: ~). **path** ຄືທີ່ຢູ່ຂອງໄຟລ໌ໃນຕົ້ນໄມ້ນັ້ນ: /home/anna/notes.txt.

ຫາ Linux ໄວ້ຝຶກ (ເລືອກໜຶ່ງ, ສືບນາທີ): ໃນ Windows, ຕິດຕັ້ງ **WSL** (Windows Subsystem for Linux — Ubuntu Linux ຕົວຈິງພາຍໃນ Windows: <https://learn.microsoft.com/en-us/windows/wsl/install>); ໃນ Mac, ແອັບ Terminal ທີ່ຕິດມາກັບເຄື່ອງ ໃກ້ຄຽງພໍທີ່ຈະເລີ່ມໄດ້ (macOS ເປັນຍາດຂອງ Unix); ຫຼື ລໍ່ຖ້າອາທິດທີ 5 ຕອນທ່ານຈະເຊົ້າ Linux server ຕົວຈິງຈາກ AWS ໂດຍບໍ່ເສຍຄ່າ. WSL ຄືຄໍາຕອບທີ່ດີທີ່ສຸດສໍາລັບຄົນສ່ວນໃຫຍ່.

25 ຄໍາສັ່ງສໍາຄັນສູງສຸດ — ຕາຕະລາງມືຄືອາທິດທີ 1–2. ພິມທຸກຄໍາສັ່ງ, ຫຼາຍໆເທື່ອ:

ຄໍາສັ່ງ	ໜ້າທີ່ຂອງມັນ	ຕົວຢ່າງ
pwd	Print working directory — “ຂ້ອຍຢູ່ໃສ?”	pwd
ls	ລາຍການໄຟລ໌ຢູ່ບ່ອນນີ້ (-l ລາຍລະອຽດຍາວ, -a ລວມໄຟລ໌ເຊືອງ)	ls -la
cd	Change directory — ຍ້າຍໄປມາໃນຕົ້ນໄມ້	cd /var/log
mkdir	ສ້າງ directory (ໂຟນເດີ)	mkdir projects
touch	ສ້າງໄຟລ໌ເປົ່າ	touch notes.txt
cp	ສໍາເນົາໄຟລ໌ (-r ສໍາລັບໂຟນເດີ)	cp a.txt backup.txt
mv	ຍ້າຍ ຫຼື ປ່ຽນຊື່	mv old.txt new.txt
rm	ລຶບ — ຖາວອນ, ບໍ່ມີຖັງຂີ້ເຫຍື້ອ. ຈົ່ງເຄົາລົບມັນ.	rm notes.txt
cat	ພິມເນື້ອໃນທັງໝົດຂອງໄຟລ໌	cat notes.txt

ຄຳສັ່ງ	ໜ້າທີ່ຂອງມັນ	ຕົວຢ່າງ
less	ອ່ານໄຟລ໌ຍາວທີ່ລະໜ້າ (q ເພື່ອອອກ)	less /var/log/syslog
head / tail	ແຖວທຳອິດ / ແຖວສຸດທ້າຍຂອງໄຟລ໌. tail -f ເພື່ອເບິ່ງ log ສົດໆ — ຄລາສສິກຂອງ ops	tail -f app.log
grep	ຄົ້ນຫາຕົວໜັງສືຕາມ pattern — ຄຳສັ່ງ ops ທີ່ຖືກໃຊ້ຫຼາຍທີ່ສຸດ	grep "ERROR" app.log
find	ຊອກຫາໄຟລ໌ຕາມຊື່/ຂະໜາດ/ອາຍຸ	find / -name "*.conf"
echo	ພິມຂໍ້ຄວາມ (ມັກໃສ່ລົງໄຟລ໌ ຫຼື ຕົວແປ)	echo "hello"
nano	ຕົວແກ້ໄຂຂໍ້ຄວາມໃນ terminal ທີ່ເປັນມິດ	nano notes.txt
man	ຄູ່ມືຂອງທຸກຄຳສັ່ງ (q ເພື່ອອອກ)	man grep
sudo	ແລ່ນຄຳສັ່ງດຽວໃນຖານະ admin ຜູ້ຊົງອຳນາດ ("root"). ດ້ວຍຄວາມເຄົາລົບ.	sudo apt update
apt	ຕິດຕັ້ງ/ອັບເດດ software (ຕະກູນ Ubuntu/Debian)	sudo apt install http
chmod	ປ່ຽນສິດອະນຸຍາດ (permissions) ຂອງໄຟລ໌	chmod 644 notes.txt
chown	ປ່ຽນເຈົ້າຂອງໄຟລ໌	sudo chown anna file
ps	ລາຍການ process ທີ່ກຳລັງແລ່ນ (ps aux ສຳລັບທັງໝົດ)	ps aux
top (ຫຼື htop)	ໜ້າປັດສິດຂອງ CPU/memory/process — “ເປັນຫຍັງ server ຂ້າ?” ເລີ່ມທີ່ນີ້	top
df -h / du -sh	ຝົນທີ່ດີສວ່າງ / ຝົນທີ່ດີໂຟນເດີໃຊ້ — “ຕິດເຕັມ” ເລີ່ມທີ່ນີ້	df -h
ssh	ເຂົ້າສູ່ terminal ຂອງເຄື່ອງອື່ນຜ່ານເຄືອຂ່າຍ — ປະຕູໜ້າ ຂອງແທ້ ຂອງວິສະວະກອນ cloud	ssh anna@server-ip
curl	ສົ່ງຄຳຮ້ອງຂໍເວັບຈາກ terminal — “ເວັບໄຊຍັງອອນລາຍບໍ່?”	curl https://example.com

Permissions, ສະບັບ 60 ວິນາທີ. ທຸກໄຟລ໌ມີເຈົ້າຂອງ ແລະ mode ຄື `rw-r--r--`: ສາມຊຸດສາມໂຕ — ເຈົ້າຂອງ, ກຸ່ມ, ທຸກຄົນ — ຂອງ `read` (ອ່ານ), `write` (ຂຽນ), `execute` (ແລ່ນ). ເປັນຕົວເລກ: `r=4, w=2, x=1`, ດັ່ງນັ້ນ `chmod 755 script.sh` ໝາຍວ່າ “ເຈົ້າຂອງເຮັດໄດ້ທຸກຢ່າງ ($7=4+2+1$); ຄົນອື່ນອ່ານ ແລະ ແລ່ນໄດ້ ($5=4+1$).” ເມື່ອໂປຣແກຣມ “ບໍ່ມີສິດອະນຸຍາດ,” ນີ້ຄືລະບົບກຳລັງເວົ້າວ່າບໍ່ — ແລະ ຕອນນີ້ທ່ານອ່ານໄດ້ແລ້ວວ່າຍ້ອນຫຍັງ.

SSH, ສະບັບ 60 ວິນາທີ. `ssh` ເປີດ terminal ໄລຍະໄກທີ່ປອດໄພເທິງເຄື່ອງອື່ນ — ຈາກໂນດບຸກຂອງທ່ານ ເຂົ້າສູ່ server ຢູ່ Virginia ຄືກັບວ່າທ່ານນັ່ງຢູ່ໜ້າມັນ. ແທນທີ່ຈະໃຊ້ລະຫັດຜ່ານ, ມີອາຊີບໃຊ້ **key pair** (ຄູ່ກະແຈ): private key (ໄຟລ໌ລັບໃນໂນດບຸກຂອງທ່ານ, ບໍ່ແບ່ງປັນເດັດຂາດ) ແລະ public key (ວາງໄວ້ໃນ server). ພວກມັນເຂົ້າກັນຄືກະແຈກັບແມ່ກະແຈ. ທຸກ AWS server ທີ່ທ່ານເປີດໃນໄລຍະທີ 1 ຈະມອບສິ່ງນີ້ໃຫ້ທ່ານແທ້ໆ.

Process ແລະ service: process ຄືໂປຣແກຣມທີ່ກຳລັງແລ່ນ; **service** (ຫຼື “daemon”) ຄື process ທີ່ແລ່ນຕະຫຼອດໄປໃນເບື້ອງຫຼັງ — web server, database. `systemctl status nginx` ຖາມ Linux ວ່າ “service nginx ແຂງແຮງດີບໍ?” — ປະໂຫຍກທີ່ທ່ານຈະພົບແບບມີອາຊີບໄປອີກຫຼາຍປີ.

ຄໍາສັບ:

ຄໍາສັບ	ຄວາມໝາຍ
Server	ຄອມພິວເຕີທີ່ມີໜ້າທີ່ຮັບໃຊ້ຄອມພິວເຕີອື່ນ. ໃນ cloud, ແມ່ນເຄື່ອງທີ່ທ່ານເຊົາ.
CPU / RAM / disk	ຜູ້ເຮັດວຽກ, ບ່ອນເຮັດວຽກຊົ່ວຄາວຄວາມໄວສູງ (ຖືກລ້າງເມື່ອ restart), ແລະ ບ່ອນຈັດເກັບຖາວອນທີ່ຊ້າກວ່າ.
Operating system (OS)	Software ທີ່ຄວບຄຸມເຄື່ອງ ແລະ ຮອງຮັບໂປຣແກຣມຕ່າງໆ. Server ແລ່ນ Linux.
Linux / distribution	OS ຂອງ server ທີ່ຟຣີ ແລະ open-source. “Distro” (Ubuntu, Amazon Linux, Debian) ຄືລົດຊາດທີ່ຫຸ້ມຫໍ່ໄວ້ແບບໜຶ່ງຂອງມັນ.
Terminal / shell / Bash	ໜ້າຕໍ່ຕົວໜຶ່ງສື່ສູ່ OS / ໂປຣແກຣມທີ່ຄວາມຄຳສັ່ງຂອງທ່ານ / ຊື່ຂອງ shell ມາດຕະຖານ.
Prompt	ເຄື່ອງໝາຍ \$ — shell ກຳລັງລໍຖ້າຄຳສັ່ງຂອງທ່ານ.
Directory / path	ໂຟນເດີ / ຫີຢູ່ເຕັມຂອງໄຟລ໌ ໃນຕົ້ນໄມ້ດຽວທີ່ເລີ່ມຈາກ /.
Root (ສອງຄວາມໝາຍ)	ຍອດຂອງຕົ້ນໄມ້ໄຟລ໌ (/) ແລະ ຜູ້ໃຊ້ admin ຜູ້ຊົງອຳນາດ. ບໍລິບົດຈະບອກທ່ານເອງວ່າແມ່ນອັນໃດ.
sudo	“Superuser do” — ແລ່ນຄຳສັ່ງດຽວດ້ວຍອຳນາດ admin.
Permissions (rwx)	ກົດຕໍ່ໄຟລ໌ວ່າໃຜອ່ານ, ຂຽນ, ແລ່ນໄດ້ — ສະແດງເປັນຊຸດສາມໂຕ ສຳລັບເຈົ້າຂອງ/ກຸ່ມ/ທຸກຄົນ.

ຄຳສັບ	ຄວາມໝາຍ
Process / service (daemon)	ໂປຣແກຣມທີ່ກຳລັງແລ່ນ / ໂປຣແກຣມທີ່ແລ່ນຕະຫຼອດໄປໃນເບື້ອງຫຼັງ (web server, database).
SSH / key pair	ການເຂົ້າສູ່ລະບົບໄລຍະໄກແບບປອດໄພ ສູ່ terminal ຂອງເຄື່ອງອື່ນ / ໄຟລ໌ກະແຈ private+public ທີ່ແທນລະຫັດຜ່ານ.
Log	ໄຟລ໌ຕົວໜັງສືທີ່ software ຂຽນບັນທຶກສິ່ງທີ່ເກີດຂຶ້ນ — ບ່ອນທຳອິດທີ່ທ່ານເບິ່ງ ເມື່ອສິ່ງໃດຜິດ.
Package manager	ຕົວຕິດຕັ້ງຂອງ OS (apt, yum) — ໄດ້ software ດ້ວຍຄຳສັ່ງ, ບໍ່ແມ່ນດ້ວຍໜ້າດາວໂຫຼດ.

ວິທີໂອສຳລັບ module ນີ້ (ລົງກວດສອບແລ້ວ):

ວິທີໂອ	ຊ່ອງ	ຄວາມຍາວ	ລິ້ງ
Linux Operating System — Crash Course for Beginners	freeCodeCamp	~2 ຊົ່ວໂມງ	https://www.youtube.com/watch?v=ROjZy1WbCIA
WSL install guide (ເອກະສານອ້າງອີງ, ບໍ່ແມ່ນວິທີໂອ)	Microsoft Learn	—	https://learn.microsoft.com/en-us/windows/wsl/install

ເວົ້າອອກສຽງຈົນເປັນທຳມະຊາດ: “Let me SSH in and check the logs.” (ຂ້ອຍຈະ SSH ເຂົ້າໄປກວດເບິ່ງ log.) · “Grep the log for the error, then tail -f it while we retry.” (Grep ຫາ error ໃນ log, ແລ້ວ tail -f ມັນໄວ້ ຂະນະທີ່ເຮົາລອງໃໝ່.) · “It’s a permissions problem — who owns the file and what’s the mode?” (ມັນເປັນບັນຫາ permissions — ໃຜເປັນເຈົ້າຂອງໄຟລ໌ ແລະ mode ແມ່ນຫຍັງ?) · “Check top — is it CPU, memory, or disk?” (ກວດ top ເບິ່ງ — ແມ່ນ CPU, memory, ຫຼື disk?)

ບົດຝຶກຫັດ: (1) ໃນ terminal Linux ຂອງທ່ານ, ສ້າງຕົ້ນໄມ້ໂຄງການນ້ອຍໆດ້ວຍ mkdir ແລະ touch, ສຳເນົາ ແລະ ຍ້າຍສິ່ງຕ່າງໆ, ແລ້ວລຶບມັນຖິ້ມ — ພ້ອມບັນຍາຍແຕ່ລະຄຳສັ່ງອອກສຽງ. (2) ສ້າງ hello.sh ທີ່ບັນຈຸ echo "hello from \$(whoami)", ເຮັດໃຫ້ມັນແລ່ນໄດ້ດ້ວຍ chmod +x, ແລ່ນມັນດ້ວຍ ./hello.sh. (3) ແລ່ນ tail -f ໃສ່ໄຟລ໌ log (ໃນ Ubuntu: sudo tail -f /var/log/syslog) ແລ້ວເບິ່ງແຖວໃໝ່ໄຫຼເຂົ້າມາ. (4) ປຶ້ມບັນທຶກ: ອະທິບາຍໃຫ້ໝູ່ໃນຈົນຕະນາການຟັງ ວ່າເປັນຫຍັງ server ຈຶ່ງແລ່ນ Linux, ໃນສາມປະໂຫຍກ.

Milestone: ໂດຍບໍ່ເບິ່ງບັນທຶກ, ທ່ານສາມາດທ່ອງໄປທຸກບ່ອນໃນຕົ້ນໄມ້ໄຟລ໌, ສ້າງ/ສຳເນົາ/ຍ້າຍ/ລຶບໄຟລ໌, ອະທິບາຍ chmod 755, ແລະ ໃຊ້ grep ຊອກຫາຄຳໃນໄຟລ໌ໄດ້. ຖ້າອັນໃດອັນໜຶ່ງຍັງຕ້ອງເປີດຄົ້ນຫາ, ໃຫ້ຢູ່ບ່ອນນີ້ຕື່ມອີກສອງມື້. Module ນີ້ຄືເສົາຄ້າຂອງທຸກສິ່ງທີ່ຕາມມາ.

Module 2 (ອາທິດທີ 3–4): ພື້ນຖານເຄືອຂ່າຍ + ບັນຊີ AWS ຂອງທ່ານ

ແນວຄິດຫຼັກ: ເຄືອຂ່າຍຄືຄອມພິວເຕີທີ່ສົ່ງຊອງຈິດໝາຍທີ່ຈຳໜ່າຍເຖິງກັນ. ທຸກເຄື່ອງໄດ້ຮັບ **IP address** (ເຊັ່ນ 172.31.8.14 — ທີ່ຢູ່ຖະໜົນເປັນຕົວເລກ). ຂໍ້ມູນຖືກຫັນເປັນ **packet** (ຊອງຈິດໝາຍ) ແລ້ວຖືກສົ່ງຕໍ່ເປັນຂັ້ນໆ ໄປຫາທີ່ຢູ່ປາຍທາງ. ເມື່ອໄປຮອດ, ໝາຍເລກ **port** ບອກວ່າຊອງນັ້ນສຳລັບໂປຣແກຣມໃດ — ອາຄານດຽວກັນ, ປະຕູຕິດເລກນັບພັນບານ: port 22 ຄື SSH, 80 ຄືເວັບບໍເຂົ້າລະຫັດ (HTTP), 443 ຄືເວັບເຂົ້າລະຫັດ (HTTPS), 5432 ຄື PostgreSQL. “ເປີດ port 443” ໝາຍວ່າ “ອະນຸຍາດຊອງທີ່ຈຳໜ່າຍເຖິງປະຕູ 443.”

Private ກັບ public: ບ້ານຂອງທ່ານ ແລະ ທຸກເຄືອຂ່າຍ cloud ໃຊ້ຊື່ **ຊ່ວງ IP ແບບ private** (10.x.x.x, 172.16–31.x.x, 192.168.x.x) ທີ່ໃຊ້ໄດ້ພຽງພາຍໃນເຄືອຂ່າຍທ້ອງຖິ່ນ; **public IP** ຄືທີ່ຢູ່ທີ່ເຂົ້າເຖິງໄດ້ຈາກອິນເຕີເນັດທັງໝົດ. ການແບ່ງນິຄົຮາກຖານຂອງຄວາມປອດໄພ cloud: ສິ່ງທີ່ບໍ່ຈຳເປັນຕ້ອງຫັນໜ້າຫາອິນເຕີເນັດ ບໍ່ໄດ້ຮັບທີ່ຢູ່ public ເລີຍ.

DNS — ປຶ້ມໂທລະສັບຂອງອິນເຕີເນັດ. ມະນຸດໃຊ້ຊື່ (example.com); packet ຕ້ອງການຕົວເລກ. **DNS** ເປັນຜູ້ແປ: ເຄື່ອງຂອງທ່ານຖາມ DNS server ວ່າ “IP ຂອງ example.com ແມ່ນຫຍັງ?”, ໄດ້ຕົວເລກມາ, ແລ້ວຈຶ່ງເຊື່ອມຕໍ່. ເຄິ່ງໜຶ່ງຂອງການລິມະລະບົບລຶກລັບທັງໝົດ ກ່ຽວຂ້ອງກັບ DNS; ຄຳຕະຫຼົກຂອງວົງການ “it’s always DNS” ມີຢູ່ກໍເພາະມັນມັກເປັນຄວາມຈິງ. nslookup example.com ເຮັດການຄົ້ນຫານັ້ນດ້ວຍມື.

HTTP — ວິທີທີ່ເວັບສົນທະນາ. **client** (browser) ສົ່ງຄຳຮ້ອງຂໍ — **method** (GET = ດຶງມາ, POST = ສົ່ງໄປ) ບວກກັບ **path** — ແລ້ວ **server** ຕອບກັບດ້ວຍ **status code**: 200 OK, 301 ຍ້າຍແລ້ວ, 403 ຫ້າມເຂົ້າ, 404 ບໍ່ພົບ, 500 server ຜິດພາດ, 502/503 “server ທາງຫຼັງຂ້ອຍຟັງໂຫຼດເກີນ.” ຈິທີກອນນັ້ນໄວ້; ໃນຖານະວິສະວະກອນ ທ່ານຈະອ່ານພວກມັນທຸກມື້. curl -I https://example.com ສະແດງ status line ແລະ header ສົດໆໃຫ້ທ່ານເບິ່ງ.

Firewall: ບັນຊີກົດລະບຽບທີ່ຕັດສິນວ່າ packet ໃດຜ່ານໄດ້, ຕາມຕື້ນທາງ, ປາຍທາງ, ແລະ port — “ອະນຸຍາດ 443 ຈາກທຸກບ່ອນ; ອະນຸຍາດ 22 ຈາກຫ້ອງການເທົ່ານັ້ນ; ປະຕິເສດທີ່ເຫຼືອ.” ໃນ AWS, firewall ລະດັບ server ເອີ້ນວ່າ **security group**, ແລະ ການຕັ້ງຄຳມັນຜິດ ຄືຮູ້ຄວາມປອດໄພອັນດັບ 1 ຂອງມືໃໝ່. **Latency** (ຄວາມຊັກຊ້າ, ເປັນ millisecond) ແລະ **bandwidth** (ຄວາມຈຸຕໍ່ວິນາທີ) ເຮັດໃຫ້ຄຳສັບຄົບຊຸດ: ໄລຍະທາງສ້າງ latency, ຈຶ່ງເປັນເຫດຜົນທີ່ cloud ມີ region ຫົວໂລກ.

ຕໍ່ໄປ, ບັນຊີ AWS ຂອງທ່ານ — ເຄິ່ງທີ່ສອງຂອງ module ນີ້. ເຂົ້າໄປທີ່ <https://aws.amazon.com/free> ແລ້ວສ້າງບັນຊີ Free Tier (ອີເມວ, ເປີໂທລະສັບ, ບັດເຄຣດິດ/ເດບິດເພື່ອຢັ້ງຢືນຕົວຕົນ — ບັດຈະບໍ່ຖືກຮຽກເກັບຢ່າງມີນ້ຳໜັກ ຖ້າທ່ານປະຕິບັດວິໄນການຮືຖອນຂອງຫຼັກສູດນີ້). **Free Tier** (ຂັ້ນໃຊ້ຟຣີ) ໃຫ້ໂຄຕ້າລາຍເດືອນຂອງບໍລິການພື້ນຖານ (ລວມທັງ 750 ຊົ່ວໂມງ/ເດືອນ ຂອງ EC2 server ຂະໜາດນ້ອຍໃນປີທຳອິດ) ແລະ ທຸກ lab ໃນຫຼັກສູດນີ້ຖືກອອກແບບໃຫ້ຢູ່ພາຍໃນມັນ. ຈາກນັ້ນ, ກ່ອນສິ່ງອື່ນໃດ, ສາມຂັ້ນຕອນຄວາມປອດໄພທີ່ຕໍ່ລອງບໍ່ໄດ້ — ການເຮັດສິ່ງເຫຼົ່ານີ້ ຄືບົດຝຶກຫັດທຳອິດຂອງອາຊີບຄວາມປອດໄພຂອງທ່ານ:

1. **MFA ໃສ່ root user.** Root user ຄືກະແຈແມ່ຂອງບັນຊີ. ເພີ່ມ multi-factor authentication (ແອັບ authenticator ໃນໂທລະສັບ) ໃນ IAM → Security credentials. ຈາກນັ້ນເຂົ້າໃຊ້ root ສຳລັບວຽກປະຈຳວັນ.
2. **ສ້າງ admin IAM user** (Lab 5 ຈະອະທິບາຍ IAM ຢ່າງເລິກ; ຕອນນີ້: console → IAM → Users → ສ້າງ user ພ້ອມສິດ admin) ແລ້ວເຂົ້າສູ່ລະບົບເປັນ user ນັ້ນຕັ້ງແຕ່ນີ້ໄປ.
3. **ການແຈ້ງເຕືອນຄ່າໃຊ້ຈ່າຍ (billing alert).** Console → Billing → Budgets → ສ້າງ **zero-spend budget** (ແມ່ແບບຂອງ AWS ທີ່ສົ່ງອີເມວຫາທ່ານທັນທີ ທີ່ມີການຮຽກເກັບໃດໆ). ເປີດ Free Tier usage alerts ນຳ. ວິສະວະກອນທີ່ຄວບຄຸມຕື້ນຫິນບໍ່ໄດ້ ຄືພາລະ; ທ່ານຫາກໍໃຊ້ບັນຊີໄດ້ 20 ນາທີ ແຕ່ກໍລ້າໜ້າມີອາຊີບຫຼາຍຄົນແລ້ວ. (ອ້າງອີງ: <https://docs.aws.amazon.com/cost-management/latest/userguide/budgets-managing-costs.html>)

ຄຳສັບ:

ຄໍາສັບ	ຄວາມໝາຍ
IP address	ທີ່ຢູ່ເຄືອຂ່າຍເປັນຕົວເລກຂອງເຄື່ອງ, ເຊັ່ນ 172.31.8.14.
Packet	ຊອງຂໍ້ມູນທີ່ຈຳໜ່າແລ້ວໜຶ່ງຊອງ; traffic ທັງໝົດຄືສາຍທານຂອງພວກມັນ.
Port	ປະຕູຕິດເລກເທິງເຄື່ອງ ທີ່ບອກວ່າ traffic ສຳລັບໂປຣແກຣມໃດ: 22 SSH, 80 HTTP, 443 HTTPS.
Private / public IP	ທີ່ຢູ່ທີ່ໃຊ້ໄດ້ພຽງພາຍໃນເຄືອຂ່າຍທ້ອງຖິ່ນ / ທີ່ຢູ່ທີ່ເຂົ້າເຖິງໄດ້ຈາກອິນເຕີເນັດທັງໝົດ.
DNS	ລະບົບແປຊື່ (example.com) ໃຫ້ເປັນ IP address. "It's always DNS."
HTTP / HTTPS	ໂປຣໂຕຄອນຮ້ອງຂໍ-ຕອບກັບຂອງເວັບ / ອັນດຽວກັນ, ເຂົ້າລະຫັດດ້ວຍ TLS.
Status code	ບົດສະຫຼຸບຄຳຕອບຂອງ server: 200 OK, 404 ບໍ່ພົບ, 500 server ຜິດພາດ, 503 ໂຫຼດເກີນ.
Firewall	ບັນຊີກົດລະບຽບທີ່ຕັດສິນວ່າ packet ໃດຜ່ານໄດ້, ຕາມຕົ້ນທາງ, ປາຍທາງ, port.
Security group	Firewall ລະດັບ server ຂອງ AWS. ການຕັ້ງຄຳຜິດຄືຮູດລາສສິກຂອງມື້ໃໝ່.
Latency / bandwidth	ຄວາມຊັກຊ້າ (ms) / ຄວາມຈູ (ຕໍ່ວິນາທີ). ໄລຍະທາງສ້າງ latency.
Client / server (ບົດບາດ)	ຜູ້ຖາມ ແລະ ຜູ້ຕອບ ໃນທຸກການສົນທະນາເຄືອຂ່າຍ.
AWS Free Tier	ໂຄຕ້າຟຣີລາຍເດືອນຂອງບັນຊີ AWS ໃໝ່ — ງົບປະມານທັງໝົດຂອງຫຼັກສູດນີ້.
Root user	ຕົວຕົນແມ່ຂອງບັນຊີ AWS. ໃສ່ MFA ໃຫ້ມັນ, ແລ້ວເຊົາໃຊ້ມັນ.
Billing alert / budget	ອີເມວອັດຕະໂນມັດເມື່ອລາຍຈ່າຍຂ້າມຂີດກຳນົດ. ຂອງທ່ານຕັ້ງໄວ້ທີ່ \$0.

ວິດີໂອສຳລັບ module ນີ້:

ວິດີໂອ	ຊ່ອງ	ຄວາມຍາວ	ລິງ
What is AWS?	Amazon Web Services (official)	~2 ນາທີ	https://www.youtube.com/watch?v=a9__D53WsUs

ວິທີ	ຊ່ອງ	ຄວາມຍາວ	ລິງ
Top 50+ AWS Services Explained in 10 Minutes	Fireship	~10 ນາທີ	https://www.youtube.com/watch?v=JlbiYCM48to
AWS Networking Basics — VPC & Subnets	KodeKloud	~30 ນາທີ	https://www.youtube.com/watch?v=QM63dyA_4Pc

ເວົ້າອອກສຽງ: “What’s the IP, and is it public or private?” (IP ແມ່ນຫຍັງ, ແລະ ມັນເປັນ public ຫຼື private?) · “Is port 443 open in the security group?” (Port 443 ເປີດຢູ່ໃນ security group ບໍ່?) · “Curl it — what status code do you get?” (Curl ມັນເບິ່ງ — ໄດ້ status code ຫຍັງ?) · “Did DNS resolve? Check with nslookup before blaming the server.” (DNS resolve ໄດ້ບໍ່? ກວດດ້ວຍ nslookup ກ່ອນຈະໂທດ server.)

ບົດຝຶກຫັດ: (1) ping google.com (ສັງເກດ latency), nslookup google.com (ສັງເກດ DNS ຕອບກັບຫຼາຍ IP), curl -I https://aws.amazon.com (ອ່ານ status line ແລະ header ສາມອັນ, ແລ້ວຄົ້ນຫາຄວາມໝາຍຂອງແຕ່ລະອັນ). (2) ຊອກຫາ private IP ຂອງເຄື່ອງທ່ານ (ip addr ໃນ Linux) ແລະ public IP ຂອງທ່ານ (ຄົ້ນຫາ “what is my IP”) — ອະທິບາຍໃນປຶ້ມບັນທຶກວ່າເປັນຫຍັງມັນຈຶ່ງຕ່າງກັນ. (3) ເຮັດການຕັ້ງບັນຊີ AWS ໃຫ້ຄົບ: MFA, admin user, zero-spend budget — ຖ່າຍ screenshot ຂອງ budget ໄວ້ໃນປຶ້ມບັນທຶກ. (4) ແຕ້ມຈາກຄວາມຈຳ: ໂນດບຸກ → ການຄົ້ນຫາ DNS → ຄຳຮ້ອງຂໍ HTTPS ຜ່ານ port 443 → firewall → web server. ສາມເທື່ອ, ຈົນມັນອັດຕະໂນມັດ.

Milestone — ຈົບໄລຍະທີ 0: ທ່ານສາມາດບັນຍາຍສິ່ງທີ່ເກີດຂຶ້ນ ເມື່ອທ່ານພິມ URL ແລ້ວກົດ Enter — DNS, IP, port, firewall, ຄຳຮ້ອງຂໍ HTTP, status code — ພາຍໃນສອງນາທີ, ໃຊ້ທຸກຄຳສັບຢ່າງຖືກຕ້ອງ; ແລະ ບັນຊີ AWS ຂອງທ່ານມີຢູ່ແລ້ວ ພ້ອມ MFA ແລະ ການເຄື່ອນ budget \$0. ຕອນນີ້ທ່ານຮູ້ວິທີເຮັດວຽກຂອງອິນເຕີເນັດ ຫຼາຍກວ່າຄົນສ່ວນໃຫຍ່ທີ່ໃຊ້ມັນຫາລ້ຽງຊິບ. ໄລຍະທີ 1 ຄົບອັນທີ່ທ່ານເລີ່ມສ້າງເທິງມັນ.

ໄລຍະທີ 1 — ແຖມ CLOUD, ລົງມືເຮັດຈິງ (ອາທິດທີ 5–12)

ໄລຍະມີເຮັດວຽກແນວໃດ. ຫ້າບໍລິການຂ້າງລຸ່ມນີ້ ແຕ່ລະອັນຄື Lab ໜຶ່ງບົດ: ເປົ້າໝາຍ, ໂຄງຮ່າງຂັ້ນຕອນ (ລະອຽດພໍທີ່ຈະເຮັດຕາມ, ສັນພິທີທ່ານຕ້ອງຄິດເອງ — ການຄິດນັ້ນແຫຼະຄືການຮຽນ), ສິ່ງທີ່ທ່ານໄດ້ຮຽນຮູ້, ແລະ — ສະເໝີ — **ການຮືຖອນ (teardown)**. ວິໄນການຮືຖອນສຳຄັນສອງຊັ້ນ: ມັນຮັກສາທ່ານໃຫ້ຢູ່ໃນ Free Tier, ແລະ “ບໍ່ປະສົງໃດແລ່ນຄ້າງໄວ້ໂດຍບໍ່ຈຳເປັນ” ຄືປະຕິກິລິຍາຂອງມືອາຊີບ ທີ່ຜູ້ສຳພາດເຈາະຖາມແທ້ໆ. ສ້າງແຕ່ລະ lab ຄົ້ນຢ່າງໜ້ອຍສອງເທື່ອ: ເທື່ອໜຶ່ງຕາມໂຄງຮ່າງ, ເທື່ອໜຶ່ງຈາກຄວາມຈຳ. ການສ້າງເທື່ອທີສອງ ຄືຕອນທີ່ຄວາມຮູ້ຍ້າຍເຂົ້າສູ່ມືຂອງທ່ານ. ວາງງົບເວລາປະມານອາທິດເຄິ່ງຕໍ່ lab; ໃຊ້ເວລາເຫຼືອສຳລັບການຟັງ, ເພາະສິ່ງຕ່າງໆຈະຟັງແນ່ນອນ, ແລະ ການ debug ພວກມັນ ຄືການສອນທີ່ດີທີ່ສຸດ ທີ່ຫຼັກສູດນີ້ຂຽນບົດໃຫ້ບໍ່ໄດ້.

ກ່ອນອື່ນ, ສາມແນວຄິດທີ່ວາງກອບໃຫ້ທຸກສິ່ງທີ່ທ່ານກຳລັງຈະສ້າງ. **Region ແລະ Availability Zone:** Region (ພາກພື້ນ) ຄືກຸ່ມ data center ຂອງ AWS ຕາມພູມສາດ (ເລືອກອັນທີ່ໃກ້ທ່ານ ແລ້ວຢູ່ໃນນັ້ນ — ຊັບພະຍາກອນໃນ region ໜຶ່ງ ເບິ່ງບໍ່ເຫັນຈາກ region ອື່ນ, ຄວາມສັບສົນອັນດັບ 1 ຂອງ “server ຂ້ອຍຫາຍໄປໃສ?”). **Availability Zone (AZ)** ຄື data center ທີ່ແຍກອິດສະຫຼະ ພາຍໃນ region; ລະບົບທີ່ຈຶ່ງຈັດແລ່ນຢູ່ສອງ AZ ເພື່ອບໍ່ໃຫ້ຄວາມລົ້ມເຫຼວຂອງອາຄານດຽວດຶງພວກມັນລົງ.

Console ກັບ CLI: console (<https://console.aws.amazon.com>) ຄືໜ້າເວັບຄວບຄຸມຂອງ AWS — ດີສໍາລັບການຮຽນ ແລະ ການເບິ່ງ; **AWS CLI** (aws ໃນ terminal ຂອງທ່ານ) ເຮັດໄດ້ທຸກຢ່າງທີ່ console ເຮັດ, ແບບຂຽນ script ໄດ້. ທ່ານຈະ ເລີ່ມໃນ console ແລ້ວກ້າວຂຶ້ນສູ່ CLI, ເພາະໄລຍະທີ 2 ຈະເຮັດອັດຕະໂນມັດທຸກສິ່ງທີ່ທ່ານເຮັດດ້ວຍມືຢູ່ບ່ອນນີ້.

Lab 1 (ອາທິດທີ 5–6): EC2 — server ທໍາອິດຂອງທ່ານ

EC2 (Elastic Compute Cloud) ໃຫ້ເຊົາເຄື່ອງຈັກເສມືອນ, ເອີ້ນວ່າ **instance**. ນິຕິການກະທໍາບຸກເບີກຂອງວິສະວະກໍາ cloud: Linux server ຕົວຈິງ, ຢູ່ເທິງອິນເຕີເນັດ, ພາຍໃນ 60 ວິນາທີ, ໂດຍບໍ່ເສຍຄ່າ.

ເປົ້າໝາຍ: ເປີດ Linux server, SSH ເຂົ້າໄປ, ໃຫ້ມັນເສີບໜ້າເວັບຕໍ່ໂລກ, ແລ້ວທໍາລາຍມັນ.

ໂຄງຮ່າງຂັ້ນຕອນ: 1. Console → EC2 → Launch instance. ຕັ້ງຊື່ມັນ. ເລືອກ **Amazon Linux 2023** ເປັນ **AMI** (Amazon Machine Image — ດິສແມ່ແບບທີ່ server ຂອງທ່ານ boot ຈາກມັນ) ແລະ **t2.micro** ຫຼື **t3.micro** ເປັນ **instance type** (ຂະໜາດ; ພວກມີຢູ່ໃນ Free Tier). 2. ສ້າງ **key pair**; ໄຟລ໌ private key ນາມສະກຸນ .pem ຈະດາວໂຫຼດ ລົງມາ. ນັ້ນຄືກະແຈ SSH ຈາກ Module 1 — ຮັກສາມັນໄວ້ໃຫ້ດີ, chmod 400 ມັນ. 3. ໃນການຕັ້ງຄ່າເຄືອຂ່າຍ, ອະນຸຍາດ SSH (port 22) ຈາກ “My IP” ເທົ່ານັ້ນ — ຕອນນີ້ທ່ານຮູ້ແທ້ໆແລ້ວວ່າກົດ security group ນີ້ໝາຍຄວາມວ່າຫຍັງ — ແລະ ອະນຸຍາດ HTTP (port 80) ຈາກທຸກບ່ອນ. 4. Launch, ລໍຖ້າສະຖານະ “running,” ສໍາເນົາ public IP, ແລ້ວຈາກ terminal ຂອງທ່ານ: `ssh -i mykey.pem ec2-user@<public-ip>`. ຫາຍໃຈເລີກໆ: ທ່ານກໍາລັງຢູ່ພາຍໃນຄອມພິວເຕີ ໃນ data center ຂອງ AWS. 5. ເທິງ server: `sudo dnf install -y nginx && sudo systemctl start nginx && sudo systemctl enable nginx`. ຈາກນັ້ນເປີດ browser ໄປທີ່ `http://<public-ip>` — ນັ້ນຄື web server ຂອງທ່ານກໍາລັງຕອບໂລກ. 6. ປ່ຽນໜ້າເວັບມາດຕະຖານ: `echo "<h1>Built by me, on EC2</h1>" | sudo tee /usr/share/nginx/html/index.html`. Refresh. ຖ່າຍ screenshot ໄວ້ໃນປຶ້ມ ບັນທຶກ. 7. ສໍາຫຼວດແບບວິສະວະກອນ: `top`, `df -h`, `sudo tail -f /var/log/nginx/access.log` ຂະນະທີ່ ທ່ານ refresh ໜ້າເວັບ — ເບິ່ງການເຂົ້າຊົມຂອງທ່ານເອງ ໄຫຼເຂົ້າມາໃນ log.

ສິ່ງທີ່ທ່ານໄດ້ຮຽນຮູ້: AMI, instance type, key pair, security group ໃນສະໜາມຈິງ, SSH ສູ່ server ຕົວຈິງ, ການຕິດຕັ້ງ ແລະ ແລ່ນ service ຂອງ Linux, ການອ່ານ log ຂອງມັນ — ນັ້ນຄື ທ່າທາງທາງກາຍປະຈຳວັນຂອງອາຊີບນີ້.

ການຮີຖອນ: EC2 → ເລືອກ instance → Instance state → **Terminate**. ຢືນຢັນວ່າມັນຂຶ້ນ “terminated.” Free Tier ໃຫ້ 750 ຊົ່ວໂມງ/ເດືອນ ຂອງ micro instance ໜຶ່ງໜ່ວຍ, ດັ່ງນັ້ນເຖິງຈະປະໄວ້ກໍຄົງບໍ່ຖືກຮຽກເກັບ — ແຕ່ຮີຖອນມັນຢູ່ດີ. ນິໄສຄື ປະເດັນສໍາຄັນ.

Lab 2 (ອາທິດທີ 7): S3 — ໄຟລ໌ທີ່ບໍ່ມີວັນຫາຍ

S3 (Simple Storage Service) ຄື object storage (ບ່ອນເກັບແບບວັດຖຸ): ຖັງບັນຈຸໄຟລ໌ທີ່ບໍ່ມີກິ້ນ, ທິນທານລະດັບ eleven nines. ມັນຄືຄໍາຕອບມາດຕະຖານຂອງ “ພວກເຮົາຈະເອົາໄຟລ໌ໄປໄວ້ໃສ່?” ແລະ, ເມື່ອຕັ້ງຄ່າຜິດ, ຄືແຫຼ່ງກໍາເນີດຂອງການຮົ່ວໄຫຼຂໍ້ມູນທີ່ໂດ່ງດັງທີ່ສຸດໃນປະຫວັດສາດ — ຈຶ່ງເປັນເຫດຜົນທີ່ lab ນີ້ ເຄິ່ງໜຶ່ງແມ່ນ storage, ເຄິ່ງໜຶ່ງແມ່ນຄວາມປອດໄພ.

ເປົ້າໝາຍ: ສ້າງ bucket, ໃຊ້ງານມັນຈາກ CLI, host ເວັບໄຊ static ນ້ອຍໆ, ແລະ ເຂົ້າໃຈຢ່າງແທ້ຈິງວ່າ “public bucket” ໝາຍຄວາມວ່າຫຍັງ.

ໂຄງຮ່າງຂັ້ນຕອນ: 1. Console → S3 → Create bucket (ຊື່ຕ້ອງບໍ່ຊ້ຳກັນຫົວໂລກ — `yourname-lab-2026` ໃຊ້ໄດ້). ສັງເກດວ່າ **Block Public Access** ເປີດຢູ່ໂດຍຄໍາມາດຕະຖານ. ອັບໂຫຼດໄຟລ໌ໃດກໍໄດ້ຜ່ານ console; ດາວໂຫຼດມັນກັບຄືນ.

2. ຕິດຕັ້ງ AWS CLI ໃນເຄື່ອງທ່ານ ແລ້ວແລ່ນ `aws configure` ດ້ວຍ **access key** ທີ່ທ່ານສ້າງໃຫ້ IAM admin user ຂອງທ່ານ (access key ຄື username+password ແບບໂປຣແກຣມສໍາລັບ API — ປະຕິບັດຕໍ່ມັນຄືລະຫັດຜ່ານ, ຢ່າເອົາໃສ່ໃນໂຄດເດັດຂາດ; ທ່ານຈະຊົມຊັບກົດນີ້ໃນ Lab 5). 3. ຈາກ terminal ຂອງທ່ານ: `aws s3 ls · aws s3 cp notes.txt s3://yourname-lab-2026/ · aws s3 sync ./myfolder s3://yourname-lab-2026/backup/`. ຮູ້ສຶກເຖິງຄວາມແຕກຕ່າງ: console ຄືການໄປຢາມ; CLI ຄືວິສະວະກໍາ. 4. ເວັບໄຊ static: ສ້າງ bucket ທີ່ສອງ, ເປີດ static website hosting, ອັບໂຫຼດ `index.html`, ແລ້ວເພີ່ມ **bucket policy** ແບບ public-read ຕາມເອກະສານ (ເອກະສານສິດອະນຸຍາດແບບ JSON — ອ່ານມັນທີ່ລະແຖວ: *ໃຜເຮັດ ຫຍັງ ໄດ້ ຕໍ່ bucket ໃດ*). ຕອນນີ້ໜ້າເວັບຂອງທ່ານຢູ່ເທິງອິນເຕີເນັດ ໂດຍບໍ່ມີ server ເລີຍ. 5. ສໍາຫຼວດ **storage classes** (Standard → Infrequent Access → Glacier: ຖືກກວ່າ 1 GB, ຊໍາກວ່າ/ແພງກວ່າໃນການດຶງຄືນ) ແລ້ວຕັ້ງ **lifecycle rule** (“ຍ້າຍ object ໄປ IA ຫຼັງ 30 ວັນ”) — ລົດຊາດທໍາອິດຂອງການຄຸ້ມຄອງຕົ້ນທຶນແບບອັດຕະໂນມັດ.

ສິ່ງທີ່ທ່ານໄດ້ຮຽນຮູ້: object storage ທຽບກັບດິສ, CLI ແລະ access key, bucket policy ແລະ ການເປີດ public (ໂດຍເຈດຕະນາ, ບໍ່ແມ່ນໂດຍບັງເອີນ), ຂັ້ນ storage ໃນຖານະຄັນໂຍກຕິນທິນ.

ການຮຶກຮຸ້ນ: ສ້າງ bucket ທັງສອງໃຫ້ເປົ້າ, ລຶບ bucket ທັງສອງ, ແລະ — ສໍາຄັນ — ປິດການໃຊ້ access key ໃດໆທີ່ທ່ານບໍ່ໄດ້ໃຊ້. ໂຄຕ້າຟຣີຂອງ S3 ນ້ອຍ (5 GB) ແຕ່ໄຟລ໌ພວກມີເປັນລະດັບ kilobyte; ວິໄນ, ອີກເທື່ອໜຶ່ງ, ຄືປະເດັນສໍາຄັນ.

Lab 3 (ອາທິດທີ 8–9): VPC — ເຄືອຂ່າຍທີ່ທ່ານເປັນເຈົ້າຂອງ

VPC (Virtual Private Cloud) ຄືສ່ວນແບ່ງສ່ວນຕົວ ມີຮີວກັນ ຂອງທ່ານ ໃນເຄືອຂ່າຍຂອງ AWS. ມາຮອດຕອນນີ້ທ່ານໃຊ້ອັນມາດຕະຖານໂດຍບໍ່ໄດ້ເບິ່ງມັນ; ວິສະວະກອນສ້າງຂອງຕົນເອງ, ເພາະປະໂຫຍກ “*the database sits in a private subnet with no internet route*” (database ຢູ່ໃນ private subnet ທີ່ບໍ່ມີເສັ້ນທາງສູ່ອິນເຕີເນັດ) ຄືເຄິ່ງໜຶ່ງຂອງຄວາມປອດໄພ cloud, ແລະ ທ່ານກໍາລັງຈະເຮັດໃຫ້ມັນເປັນຈິງດ້ວຍມືຂອງທ່ານເອງ.

ເປົ້າໝາຍ: ສ້າງເຄືອຂ່າຍສອງຊັ້ນ — public subnet ສໍາລັບ web server, private subnet ສໍາລັບ database ໃນອະນາຄົດ — ແລ້ວພິສູດວ່າເຄິ່ງ private ເຂົ້າເຖິງບໍ່ໄດ້ຈາກອິນເຕີເນັດ.

ໂຄງຮ່າງຂັ້ນຕອນ: 1. Console → VPC → Create VPC. ໃຫ້ block ທີ່ຢູ່ `10.0.0.0/16` — **CIDR notation**, ບ່ອນທີ່ /16 ໝາຍວ່າ “16 bit ທໍາອິດຄົງທີ່, ສ່ວນທີ່ເຫຼືອເປັນຂອງຂ້ອຍ”: 65,536 ທີ່ຢູ່ private. 2. ສ້າງສອງ **subnet**: `10.0.1.0/24` (public, ໃນ AZ-a) ແລະ `10.0.2.0/24` (private, ໃນ AZ-b). Subnet ຄື block ນ້ອຍກວ່າພາຍໃນ VPC, ອາໄສຢູ່ໃນ AZ ດຽວແທ້ໆ. 3. ສ້າງ **Internet Gateway** (ປະຕູສູ່ອິນເຕີເນັດຂອງ VPC) ແລ້ວ attach ມັນ. ສ້າງ **route table** ດ້ວຍກົດ `0.0.0.0/0 → internet gateway` (“ອິນເຕີເນັດທີ່ບໍ່ແມ່ນທ້ອງຖິ່ນ ໄປປະຕູອິນເຕີເນັດ”) ແລ້ວ associate ມັນກັບ subnet *public* ເທົ່ານັ້ນ. Subnet private ຮັກສາໄວ້ພຽງເສັ້ນທາງທ້ອງຖິ່ນ — ການບໍ່ມີເສັ້ນທາງນັ້ນເອງ ຄືຄວາມປອດໄພ. 4. ເປີດ micro EC2 instance ໜຶ່ງໜ່ວຍໃນແຕ່ລະ subnet (ອັນ public ມີ public IP, ອັນ private ບໍ່ມີ). 5. ການພິສູດ: SSH ຫາ instance public — ໄດ້. ລອງທີ່ຢູ່ຂອງ instance private ຈາກໂນດບຸກຂອງທ່ານ — ຄ້າງຕະຫຼອດການ, ແລະ ຕອນນີ້ທ່ານບອກໄດ້ຢ່າງແມ່ນຢ່າວ່າຍ້ອນຫຍັງ. ຈາກນັ້ນ SSH ຈາກ instance public ໄປຫາອັນ private (ມັນເຂົ້າເຖິງໄດ້ຈາກພາຍໃນ VPC): ເຄື່ອງ public ກໍາລັງເຮັດໜ້າທີ່ເປັນ **bastion host**, ຮູບແບບມາດຕະຖານຂອງວົງການ ທີ່ທ່ານຫາກໍຄົ້ນພົບດ້ວຍການສ້າງມັນ. 6. ແນວຄິດເສີມໃຫ້ໄປຄືນຫາ ແລະ ຈິດບັນທຶກ: **NAT gateway** ໃຫ້ເຄື່ອງ private ຕິດຕໍ່ *ອອກໄປ* ໄດ້ (ເພື່ອອັບເດດ) ໃນຂະນະທີ່ຍັງເຂົ້າເຖິງຈາກນອກບໍ່ໄດ້ — ແຕ່ມັນຄິດເງິນເປັນຊົ່ວໂມງ, ດັ່ງນັ້ນອ່ານກ່ຽວກັບມັນ, ຢ່າສ້າງມັນ.

ສິ່ງທີ່ທ່ານໄດ້ຮຽນຮູ້: CIDR, subnet, route table, internet gateway, ການແບ່ງ public/private, bastion host — ຂໍ້ເຄືອຂ່າຍໃນທຸກ JD ຂອງສາຍ cloud ແທ້ໆ, ໃນຖານະຄວາມຈໍາຂອງກ້າມເນື້ອ.

ການຮືຖອນ: terminate ທັງສອງ instance ກ່ອນ, ແລ້ວລຶບ VPC (ຊຶ່ງຈະກວາດ subnet, route table, ແລະ gateway ໄປ ພ້ອມ). ກວດໃນ EC2 ວ່າບໍ່ມີຫຍັງຂຶ້ນ “running.”

Lab 4 (ອາທິດທີ 10): RDS — database ທີ່ທ່ານບໍ່ຕ້ອງລ້ຽງເບິ່ງແຍງ

RDS (Relational Database Service) ຄື database ແບບ managed: AWS ແລ່ນ database engine (PostgreSQL, MySQL...) ແລະ ຈັດການ backup, patch, ແລະ failover, ໃນຂະນະທີ່ທ່ານເປັນເຈົ້າຂອງຂໍ້ມູນ ແລະ query. “Managed service” ຄືຂໍ້ຕົກລົງແກນຂອງ cloud — ແລກການຄວບຄຸມສ່ວນໜຶ່ງ ກັບແຮງງານທີ່ບໍ່ສ້າງຄວາມແຕກຕ່າງຈຳນວນຫຼາຍ — ແລະ lab ນີ້ຄືບ່ອນທີ່ທ່ານຈະຮູ້ສຶກເຖິງຂໍ້ຕົກລົງນັ້ນ.

ເປົ້າໝາຍ: ເປີດ PostgreSQL database ໃນ private subnet, ເຊື່ອມຕໍ່ຫາມັນຈາກ EC2 instance, ແລະ ເຂົ້າໃຈ backup ກັບ multi-AZ — ແລ້ວຮືຖອນມັນທັນທີ, ເພາະ RDS ຄື lab ທີ່ຖືກປະໃຫ້ແລ່ນຢ່າງໂດຍບັງເອີນງ່າຍທີ່ສຸດ.

ໂຄງຮ່າງຂັ້ນຕອນ: 1. ສ້າງ VPC ຂອງ Lab 3 ຄືນຢ່າງໄວ (ການສ້າງເທື່ອທີສອງຈາກຄວາມຈຳ — ນີ້ຄື spaced repetition ໂດຍເຈດຕະນາ), ເພີ່ມ subnet *private* ທີ່ສອງໃນ AZ ອື່ນ, ເພາະ RDS ຕ້ອງການ **subnet group** ທີ່ກວມສອງ AZ. 2. Console → RDS → Create database → PostgreSQL → **Free tier template** (ອັນນີ້ເລືອກ db.t3.micro/db.t4g.micro, single-AZ ໄວ້ລ່ວງໜ້າ). ຕັ້ງລະຫັດຜ່ານ master. ວາງມັນໃນ VPC ຂອງທ່ານ, **Public access: No**, ໃນ security group ທີ່ອະນຸຍາດ port 5432 ຈາກ security group ຂອງ web server ເທົ່ານັ້ນ — ກົດທີ່ອ້າງອີງກົດອື່ນ ແທນທີ່ຈະອ້າງອີງ IP. ສະຫງ່າງາມ, ແລະ ເປັນມາດຕະຖານ. 3. ເປີດ micro EC2 ໃນ public subnet, ຕິດຕັ້ງ client postgresql, ແລ້ວເຊື່ອມຕໍ່: `psql -h <rds-endpoint> -U postgres`. **endpoint** ຄືຊື່ DNS, ບໍ່ແມ່ນ IP — AWS ອາດຢ້າຍເຄື່ອງທາງລຸ່ມ, ແລະ ຊື່ຈະຕິດຕາມມັນໄປ. (Module 2 ກຳລັງຈ່າຍຄ່າເຊົາຄືນໃຫ້ທ່ານແລ້ວ.) 4. ທີ່ prompt ຂອງ psql: ສ້າງຕາຕະລາງ, insert ສາມແຖວ, select ພວກມັນກັບຄືນ. ມື້ນີ້ທ່ານບໍ່ຕ້ອງການຄວາມເລິກຂອງ SQL; ທ່ານຕ້ອງການໄດ້ແຕ່ຕ້ອງມັນ. 5. ຢ້ຽມຊົມ, ຢ່າເປີດໃຊ້: ການຕັ້ງຄ່າ **automated backups** (ການຮັບຄືນແບບ point-in-time ຈາກ snapshot ຄືນລະເທື່ອ + log), ແລະ ທາງເລືອກ **Multi-AZ** (ເຄື່ອງສຳຮອງສິດໃນ AZ ອື່ນ ພ້ອມ **failover** ອັດຕະໂນມັດ — ຕື່ນຫິນປະມານສອງເທົ່າ, ຈຶ່ງເປັນເຫດຜົນທີ່ prod ຕອບຕົກລົງ ແລະ lab ຕອບບໍ່). ຖ່າຍ **snapshot** ດ້ວຍມື, ຊອກຫາມັນໃນ console, ເຂົ້າໃຈວ່າທ່ານສາມາດ restore ສຳເນົາຈາກມັນໄດ້. 6. ຄຳຖາມປຶ້ມບັນທຶກທີ່ຮຸ້ນຄ່າສິບນາທີ: ແທ້ຈິງແລ້ວ AWS ກຳລັງເຮັດຫຍັງໃຫ້ທ່ານຢູ່ບ່ອນນີ້ ທີ່ບໍ່ດັ່ງນັ້ນທ່ານຈະຕ້ອງເຮັດເອງຕອນຕົສອງ? (Patch, backup, failover, hardware.) ຄຳຕອບມັນຄືຄຳຕອບສຳພາດເລື່ອງ managed services.

ສິ່ງທີ່ທ່ານໄດ້ຮຽນຮູ້: managed database, subnet group, ກົດແບບ security-group-ຫາ-security-group, endpoint, snapshot, multi-AZ/failover — ບວກກັບການສາທິດສິດ ຂອງການຈ່າຍດ້ວຍການຄວບຄຸມ ເພື່ອຊື່ຄວາມທົນທານ.

ການຮືຖອນ: ລຶບ RDS instance (ປະຕິເສດ snapshot ສຸດທ້າຍສຳລັບ lab; ຈື່ໄວ້ວ່າ prod ຈະຖ່າຍໄວ້), ລຶບ snapshot ທີ່ຖ່າຍດ້ວຍມື (snapshot ຄິດເງິນຄ່າ storage!), terminate EC2, ລຶບ VPC. ກວດ billing dashboard ຂອງທ່ານໃນມື້ຖັດໄປ — ການອ່ານມັນທຸກອາທິດ ຄືນີ້ໄສຂອງໄລຍະທີ່ 3 ທີ່ເລີ່ມຕັ້ງແຕ່ຕອນນີ້.

Lab 5 (ອາທິດທີ 11–12): IAM — ໃຜເຮັດຫຍັງໄດ້

IAM (Identity and Access Management) ຕັດສິນວ່າຄົນໃດ ແລະ ໂປຣແກຣມໃດ ເຮັດຫຍັງໄດ້ ຕໍ່ຊັບພະຍາກອນໃດ. ມັນບໍ່ຄິດເງິນ, ບໍ່ provision ຫຍັງ — ແຕ່ມັນຄືບໍລິການທີ່ຖືກກວດສອບຫຼາຍທີ່ສຸດ, ຖືກເຈາະຖາມໃນການສຳພາດຫຼາຍທີ່ສຸດ, ແລະ ພົວພັນກັບການເຈາະລະບົບຫຼາຍທີ່ສຸດໃນ AWS. ຂໍ້ຄວາມປອດໄພໃນ JD (“least-privilege access controls,” “IAM policies”) ໝາຍເຖິງ lab ນີ້.

ເປົ້າໝາຍ: ສ້າງ user, group, policy, ແລະ — ອັນສໍາຄັນທີ່ສຸດ — **role**, ແລ້ວຊຶມຊັບ least privilege ດ້ວຍການຮູ້ສຶກ AWS ປະຕິເສດທ່ານ.

ໂຄງຮ່າງຂັ້ນຕອນ: 1. ແນວຄິດກ່ອນ, ຫ້າມາທີ່: **user** ຄືຕົວຕົນຂອງມະນຸດ ຫຼື ໂປຣແກຣມ; **group** ມັດ user ເຂົ້າກັນ; **policy** ຄືເອກະສານ JSON ທີ່ໃຫ້ສິດອະນຸຍາດ (“allow s3:GetObject on arn:aws:s3:::my-bucket/*”); **role** ຄືຕົວຕົນທີ່ມີ policy ແຕ່ *ບໍ່ມີລະຫັດຜ່ານ* ຊຶ່ງຝ່າຍທີ່ໄວ້ໃຈໄດ້ *assume* (ສວມ) ຊົ່ວຄາວ — ນີ້ຄືວິທີທີ່ server ແລະ service ໄດ້ຮັບສິດອະນຸຍາດ ໂດຍບໍ່ມີຄວາມລັບເກັບໄວ້ໃດໆ. 2. ສ້າງ user readonly-rita ໃນ group ທີ່ມີ policy ReadOnlyAccess ຂອງ AWS. ເຂົ້າສູ່ລະບົບເປັນນາງໃນ browser window ແບບ private: ນາງເຫັນທຸກຢ່າງ, ແຕ່ທຸກປຸ່ມ create/delete ລົ້ມເຫຼວ ພ້ອມການປະຕິເສດຢ່າງຈະແຈ້ງ. ອ່ານ error ໜຶ່ງອັນນັ້ນໃຫ້ຄົບ — ການຮຽນອ່ານ “not authorized to perform X on Y” ຄື ຫັກສະວຽກປະຈຳວັນ. 3. ຂຽນ custom policy ທໍາອິດຂອງທ່ານໃນ JSON editor: ອະນຸຍາດ s3:ListBucket ແລະ s3:GetObject ຕໍ່ bucket ສະເພາະອັນດຽວ. Attach ມັນໃສ່ user ໃໝ່; ຍືນຢັນວ່ານາງອ່ານ bucket ນັ້ນໄດ້ ແລະ *ບໍ່ມີ ຫຍັງອື່ນເລີຍ*. ຕອນນີ້ທ່ານໄດ້ຈັດຕັ້ງ least privilege ແລ້ວ, ບໍ່ແມ່ນພຽງນິຍາມມັນ. 4. Role, ຜົນຕອບແທນ: ສ້າງ role ສໍາລັບ EC2 ພ້ອມ AmazonS3ReadOnlyAccess, ເປີດ micro instance ທີ່ attach role ນັ້ນ, SSH ເຂົ້າໄປ, ແລ້ວແລ່ນ aws s3 ls — ມັນເຮັດວຽກ ໂດຍ **ບໍ່ມີ access key ຢູ່ບ່ອນໃດເທິງເຄື່ອງເລີຍ**. Instance ກໍາລັງ assume role ແລະ ໄດ້ຮັບ credential ອາຍຸສັ້ນໂດຍອັດຕະໂນມັດ. ນີ້ຄືຮູບແບບຄວາມປອດໄພສໍາຄັນທີ່ສຸດອັນດຽວໃນ AWS: *role ສໍາລັບເຄື່ອງຈັກ, ຢ່າມີ key ເທິງເຄື່ອງຈັກ*. ເວົ້າມັນສອງເທື່ອ. 5. ກວດສອບບັນຊີຂອງທ່ານເອງແບບມືອາຊີບ: root ມີ MFA ບໍ່ (ອາທິດທີ 3)? ມີ access key ໃດອາຍຸເກີນ 90 ວັນບໍ່? ມີ user ໃດມີສິດຫຼາຍກວ່າທີ່ໃຊ້ບໍ່? ແລ່ນ **credential report** ຂອງ IAM ແລ້ວອ່ານມັນ. ພິທີກໍາສືບນາທີ່ນີ້, ເຕືອນລະເທື່ອ, ຄື checklist ສຸຂະອະນາໄມ IAM ຂອງໄລຍະທີ 3 ທີ່ກໍາລັງເກີດຂຶ້ນ.

ສິ່ງທີ່ທ່ານໄດ້ຮຽນຮູ້: user/group/policy/role, ການອ່ານ ແລະ ຂຽນ policy JSON, least privilege ທີ່ຍືນຢັນດ້ວຍການທົດລອງ, role ບໍ່ແມ່ນ key, ແລະ ການກວດສອບຄວາມປອດໄພຄັ້ງທໍາອິດຂອງທ່ານ.

ການຮື້ຖອນ: ລຶບ user ທີ່ດສອບ ແລະ credential ຂອງພວກເຂົາ, terminate instance, ຮັກສາຄວາມຮູ້ເລື່ອງ role ໄວ້ ຕະຫຼອດໄປ.

ຄໍາສັບສໍາລັບໄລຍະທີ 1 (ຕາຕະລາງດຽວ, ທັງຫ້າ lab):

ຄໍາສັບ	ຄວາມໝາຍ
Region / Availability Zone	ກຸ່ມ data center ຂອງ AWS ຕາມພູມສາດ / data center ທີ່ແຍກອິດສະຫຼະໜຶ່ງແຫ່ງພາຍໃນມັນ. ລະບົບທີ່ຈົ່ງຈັງກວມສອງ AZ.
EC2 / instance	ບໍລິການເຊົາເຄື່ອງຈັກເສມືອນ / server ທີ່ເຊົາໜຶ່ງໜ່ວຍ.
AMI	Amazon Machine Image — ດິສແມ່ແບບທີ່ instance boot ຈາກມັນ.
Instance type	ຂະໜາດ/ສະເປັກທີ່ທ່ານເລືອກ (t3.micro = ນ້ອຍສຸດ, Free Tier).
Security group	Firewall ຕໍ່ຊັບພະຍາກອນ: port ໃດ, ຈາກຕົ້ນທາງໃດ.
Key pair	ກະແຈ SSH private/public ສໍາລັບເຂົ້າຫາ instance ຂອງທ່ານ.

ຄຳສັບ	ຄວາມໝາຍ
S3 / bucket / object	Object storage / ພາຊະນະບັນຈຸທີ່ມີຊື່ໜຶ່ງອັນ / ໄຟລ໌ທີ່ເກັບໄວ້ໜຶ່ງໄຟລ໌.
Storage class / lifecycle rule	ຂັ້ນລາຄາ-ຄວາມໄວຂອງ object / ກົດອັດຕະໂນມັດທີ່ຍ້າຍພວກມັນໄປຂັ້ນຖືກກວ່າຕາມອາຍຸ.
Bucket policy	ເອກະສານ JSON ທີ່ບອກວ່າໃຜເຮັດຫຍັງໄດ້ຕໍ່ bucket. ອັນທີ່ເປັນ public ຂຶ້ນຫົວຂ່າວ.
AWS CLI / access key	ໜ້າຕໍ່ terminal ສູ່ AWS / credential ແບບໂປຣແກຣມສຳລັບມັນ (ປະຕິບັດຄືລະຫັດຜ່ານ).
VPC / subnet	ສ່ວນແບ່ງເຄືອຂ່າຍສ່ວນຕົວຂອງທ່ານ / block ນ້ອຍກວ່າຂອງມັນຢູ່ໃນ AZ ດຽວ, public ຫຼື private.
CIDR	ສັນຍາລັກ block ທີ່ຢູ່: 10.0.0.0/16 = “16 bit ຫຳອິດຄົງທີ່, 65,536 ທີ່ຢູ່ເປັນຂອງຂ້ອຍ.”
Internet gateway / route table	ປະຕູອິນເຕີເນັດຂອງ VPC / ກົດທີ່ຕັດສິນວ່າ traffic ຖືກສົ່ງໄປໃສ່. ບໍ່ມີເສັ້ນທາງ = ເຂົ້າເຖິງບໍ່ໄດ້ = ປອດໄພ.
NAT gateway	ໃຫ້ເຄື່ອງ private ໂທອອກໄດ້ ໂດຍບໍ່ຖືກເຂົ້າເຖິງຈາກນອກ. ຄິດເງິນເປັນຊົ່ວໂມງ — ຮູ້ຈັກມັນ, ຢ່າປະມັນແລ່ນຫຼິ້ນ.
Bastion host	ເຄື່ອງ public ທີ່ເສີມຄວາມແຂງ ທີ່ທ່ານ SSH ຜ່ານ ເພື່ອໄປຫາເຄື່ອງ private.
RDS / endpoint	ບໍລິການ relational database ແບບ managed / ຊື່ DNS ທີ່ທ່ານເຊື່ອມຕໍ່ຫາ.
Snapshot / Multi-AZ / failover	ສຳເນົາ ໃນຈຸດເວລາໜຶ່ງ / ເຄື່ອງສຳຮອງສິດໃນ AZ ທີ່ສອງ / ການສະຫຼັບໄປຫາມັນໂດຍອັດຕະໂນມັດ.
IAM user / group / policy / role	ຕົວຕົນ / ມັດຂອງຕົວຕົນ / ການໃຫ້ສິດແບບ JSON / ຕົວຕົນທີ່ສວມໄດ້ ບໍ່ມີລະຫັດຜ່ານ — ວິທີທີ່ເຄື່ອງຈັກໄດ້ຮັບສິດ.
Least privilege	ກົດຄຳ: ສິດອະນຸຍາດໜ້ອຍທີ່ສຸດທີ່ເຮັດວຽກໄດ້, ບໍ່ຫຼາຍກວ່ານັ້ນ.
Managed service	AWS ແບກແຮງງານທີ່ບໍ່ສ້າງຄວາມແຕກຕ່າງ (patch, backup, failover); ທ່ານຮັກສາຂໍ້ມູນ ແລະ ການຕັດສິນໃຈ.

ວິທີໂອສຳລັບໄລຍະນີ້:

ວິດີໂອ	ຊ່ອງ	ຄວາມຍາວ	ລິງ
Getting Started with EC2	AWS Developers (official)	27 ນາທີ	https://www.youtube.com/watch?v=nJ-djerESW0
Introduction to Amazon S3	Amazon Web Services (official)	~5 ນາທີ	https://www.youtube.com/watch?v=ecv-19sYL3w
AWS Networking Basics — VPC & Subnets	KodeKloud (ເບິ່ງຄືນຫຼັງ Lab 3 — ມັນຈະຊຶມເຂົ້າຕ່າງຈາກເດີມ)	~30 ນາທີ	https://www.youtube.com/watch?v=QM63dyA_4Pc
The AWS Shared Responsibility Model	Digital Cloud Training	4 ນາທີ	https://www.youtube.com/watch?v=ESPBBEK-cvo

ເວົ້າອອກສຽງ: “It’s in a private subnet; there’s no route to the internet gateway.” (ມັນຢູ່ໃນ private subnet; ບໍ່ມີເສັ້ນທາງໄປຫາ internet gateway.) · “The security group only allows 5432 from the web tier’s security group.” (Security group ອະນຸຍາດ 5432 ຈາກ security group ຂອງຊັ້ນ web ເທົ່ານັ້ນ.) · “The instance uses a role — there are no keys on the box.” (Instance ໃຊ້ role — ບໍ່ມີ key ຢູ່ເທິງເຄື່ອງ.) · “Terminate it when you’re done; nothing idles in this account.” (Terminate ມັນເມື່ອເຮັດແລ້ວ; ບໍ່ມີຫຍັງແລ່ນຫຼິ້ນຢູ່ໃນບັນຊີນີ້.)

Milestone — ຈົບໄລຍະທີ 1: ດ່ານທົດສອບການສ້າງ. ໃນການນັ່ງເທື່ອດຽວ, ຈາກຄວາມຈໍາ: VPC ຜ່ອມ subnet public/private → EC2 web server (public, attach role, ເສີບໜ້າເວັບ) → RDS PostgreSQL (private, ເຂົ້າເຖິງໄດ້ຈາກ web server ເທົ່ານັ້ນ) → S3 bucket ທີ່ instance ອ່ານໄດ້ຜ່ານ role ຂອງມັນ — ແລ້ວຮືຖອນທັງໝົດໃຫ້ສະອາດ. ຕໍາກວ່າສາມຊົ່ວໂມງ ໝາຍວ່າທ່ານພ້ອມສໍາລັບໄລຍະທີ 2. ນອກນັ້ນ: **ເລີ່ມກຽມ CLF-C02 ດຽວນີ້** (ຄອສເຕັມຂອງ freeCodeCamp — <https://www.youtube.com/watch?v=7HKot-brXFE> — ທີ່ຄວາມໄວ 1.25× ເປັນການທວນຄືນ; ສະບັບຄລາສສິກ 14 ຊົ່ວໂມງແມ່ນ <https://www.youtube.com/watch?v=NhDYbskXRgc>), ແລ້ວສອບ **AWS Cloud Practitioner** ປະມານອາທິດທີ 14–16.

ໄລຍະທີ 2 — ອັດຕະໂນມັດ (ອາທິດທີ 13–20)

ແນວຄິດຂອງໄລຍະນີ້: ທຸກຢ່າງທີ່ທ່ານຄລິກໃນໄລຍະທີ 1, ຕອນນີ້ທ່ານຈະເຮັດດ້ວຍໂຄດ. ຄໍາເວົ້າແດກດັນເບົາໆຂອງວົງການ ຕໍາການຄລິກ console ຄື **ClickOps**; ວະລີໃນປະກາດຮັບສະໝັກ ສໍາລັບສິ່ງທີ່ມາແທນມັນຄື “*design, develop, and deploy cloud infrastructure using infrastructure as code.*” ໄລຍະນີ້ຄືຄວາມແຕກຕ່າງ ລະຫວ່າງຄົນທີ່ເຄີຍ ໃຊ້ AWS ກັບຄົນທີ່ບໍລິສັດຈະຈ່າຍເງິນໃຫ້ ແລ່ນ ມັນ.

Module 6 (ອາທິດທີ 13–15): ການຂຽນ script — Bash ແລະ Python ສໍາລັບ ops, ບວກ Git

ການຂຽນ Bash script ຄືການເອົາຄໍາສັ່ງຈາກ Module 1 ໃສ່ໄຟລ໌ ເພື່ອໃຫ້ຄອມພິວເຕີເຮັດຊ້າພວກມັນຢ່າງສົມບູນແບບ. ຮຽນ, ຕາມລໍາດັບນີ້: **variables** (ຕົວແປ — NAME="web-1"), **command substitution** (TODAY=\$(date +%F)), **if**

(if [-f "\$FILE"]; then ... fi), **loops** (for f in *.log; do gzip "\$f"; done), **exit codes** (ທຸກຄຳສັ່ງສົ່ງຄືນ 0 ເມື່ອສຳເລັດ, ບໍ່ແມ່ນສູນເມື່ອລົ້ມເຫຼວ; && ຕໍ່ໄສ້ເມື່ອສຳເລັດ — script ຕັດສິນໃຈດ້ວຍສົ່ງນີ້), ແລະ ການອ່ານ **arguments** (\$1, \$2). ນັ້ນຄື 90% ຂອງ Bash ໃນ ops ຕົວຈິງ. ຂຽນ script ທີ່ມີປະໂຫຍດແທ້ໆອັນທຳອິດຂອງທ່ານໃນອາທິດນີ້: backup.sh — tar directory ໜຶ່ງອັນ, ຕັ້ງຊື່ archive ດ້ວຍວັນທີມື້ນີ້, aws s3 cp ມັນຂຶ້ນ bucket, ລຶບ archive ໃນເຄື່ອງທີ່ເກົ່າກວ່າ 7 ວັນ, ແລະ echo ແຖວສຳເລັດ/ລົ້ມເຫຼວລົງໄຟລ໌ log. Script ອັນດຽວນັ້ນ ຄື variables, substitution, conditionals, exit codes, ແລະ CLI ໃນຊຸມງານດຽວ — ແລະ ເປັນຂໍ້ໜັງໃນ résumé ຂອງທ່ານ (“automated backups to S3”).

Python ຄືອ້າຍໃຫຍ່ຂອງ Bash: ດຶກວ່າສຳລັບສິ່ງໃດກໍຕາມທີ່ກ່ຽວກັບ logic, ຂໍ້ມູນ, ຫຼື ການສົນທະນາກັບ API. ທ່ານຕ້ອງການ ops-Python ທີ່ໃຊ້ງານໄດ້, ບໍ່ແມ່ນຄວາມເລິກຂອງ software engineering: ຕົວແປ ແລະ ຊະນິດຂໍ້ມູນ, list ແລະ dictionary, if/for, function, ການອ່ານ/ຂຽນໄຟລ໌, ການຈັດການ error ດ້ວຍ try/except, ແລະ ການຕິດຕັ້ງ library ດ້ວຍ pip. ຈາກນັ້ນພົບກັບ **boto3**, library ຂອງ AWS ສຳລັບ Python: import boto3; ec2 = boto3.client("ec2"); ec2.describe_instances() — ທັນໃດນັ້ນຄວາມຮູ້ໄລຍະທີ 1 ຂອງທ່ານກໍຂຽນເປັນໂປຣແກຣມໄດ້. ຂຽນ audit.py: ລາຍການທຸກ EC2 instance ໃນບັນຊີ ພ້ອມຊື່, type, state, ແລະ ເວລາ launch, ແລ້ວພິມແຖວເຕືອນສຳລັບອັນໃດທີ່ແລ່ນດົນກວ່າ 24 ຊົ່ວໂມງ. ຊົມເຊີຍ — ທ່ານໄດ້ຂຽນເຄື່ອງມືຄວບຄຸມຕົນທຶນຕົວຈິງ ທີ່ນາຍ໋າງຈ່າຍເງິນຊື້.

Git ຄືລະບົບຄວບຄຸມເວີຊັນ ທີ່ວຽກວິສະວະກຳທັງໝົດອາໄສຢູ່ໃນມັນ. ແນວຄິດ: **repository** (ໂຟນເດີທີ່ປະຫວັດທັງໝົດຖືກບັນທຶກ), **commit** (ພາບບັນທຶກໜຶ່ງອັນ ພ້ອມຂໍ້ຄວາມ), **branch** (ເສັ້ນວຽກຂະໜານ), **remote** (ສຳເນົາເທິງ GitHub), ແລະ **pull request** (ການຂໍໃຫ້ branch ຖືກທົບທວນ ແລະ merge). ວົງຈອນປະຈຳວັນຄືຫົກຄຳສັ່ງ: git init / git clone, git status, git add, git commit -m "message", git push, git pull. ສ້າງບັນຊີ GitHub ມື້ນີ້, ສ້າງ repo ຊື່ cloud-journey, ແລະ ຈາກນີ້ໄປ **ທຸກ script ແລະ config ທີ່ທ່ານຂຽນໃນຫຼັກສູດນີ້ ຖືກ commit ໃສ່ມັນ**. ໃນສິບເດືອນ, ປະຫວັດ commit ນັ້ນຄືຫຼັກຖານສາທາລະນະ ມີວັນທີ່ກຳກັບ ຂອງທຸກສິ່ງທີ່ຫຼັກສູດນີ້ອ້າງວ່າທ່ານເຮັດໄດ້.

ຄຳສັບ:

ຄຳສັບ	ຄວາມໝາຍ
Script	ໄຟລ໌ຄຳສັ່ງທີ່ແລ່ນຈາກເທິງລົງລຸ່ມ — ອະຕອມຂອງອັດຕະໂນມັດ.
Variable / argument	ຄຳທີ່ມີຊື່ໃນ script / ຄຳທີ່ສົ່ງເຂົ້າໄປຕອນແລ່ນມັນ (\$1).
Exit code	ສັນຍານສຳເລັດ (0) ຫຼື ລົ້ມເຫຼວ (ບໍ່ແມ່ນສູນ) ຂອງທຸກຄຳສັ່ງ — ວິທີທີ່ script ຕັດສິນໃຈ.
Cron	ຕົວກຳນົດເວລາຂອງ Linux: ແລ່ນ script ທຸກຄືນຕອນຕື່ສອງ, ຕະຫຼອດໄປ.
Python / pip	ພາສາຂຽນໂປຣແກຣມທີ່ໂລກ ops ມັກໃຊ້ / ຕົວຕິດຕັ້ງ package ຂອງມັນ.
boto3	Library ຂອງ Python ສຳລັບຂັບ AWS — console, ໃນຮູບໂຄດ.

ຄໍາສັບ	ຄວາມໝາຍ
try/except	“ລອງອັນນີ; ຖ້າລົ້ມເຫຼວ, ເຮັດອັນນັ້ນແທນ” ຂອງ Python — ວິທີທີ່ script ລົ້ມເຫຼວຢ່າງສະຫງ່າງາມ.
Git / repository / commit	ການຄວບຄຸມເວີຊັນ / ປະຫວັດທີ່ບັນທຶກໄວ້ຂອງໂຄງການໜຶ່ງ / ພາບບັນທຶກໜຶ່ງອັນພ້ອມຂໍ້ຄວາມ.
Branch / merge	ເສັ້ນວຽກຂະໜານ / ການເອົາມັນກັບຄືນເຂົ້າເສັ້ນຫຼັກ.
GitHub / remote / pull request	ເວັບໄຊ hosting / ສໍານຳເນົາ cloud ຂອງ repo ທ່ານ / ຄໍາຂໍ merge branch ທີ່ຖືກທົບທວນ ແລະ ສົນທະນາ.
README	ເອກະສານໜ້າປົກຂອງ repo ທີ່ອະທິບາຍວ່າມັນແມ່ນຫຍັງ ແລະ ແລ່ນແນວໃດ. Recruiter ອ່ານພວກນີ້.

ວິດີໂອສໍາລັບ module ນີ້:

ວິດີໂອ	ຊ່ອງ	ຄວາມຍາວ	ລິງ
Learn Python — Full Course for Beginners	freeCodeCamp	~4.5 ຊົ່ວໂມງ (ເບິ່ງແບ່ງຕະຫຼອດ 3 ອາທິດ)	https://www.youtube.com/watch?v=rfscVS0vtbw

ເວົ້າອອກສຽງ: “I’ll script it — it’ll be wrong once, then never again.” (ຂ້ອຍຈະຂຽນເປັນ script — ມັນຈະຜິດເທື່ອດຽວ, ແລ້ວບໍ່ຜິດອີກຈັກເທື່ອ.) · “Check the exit code before the next step runs.” (ກວດ exit code ກ່ອນຂັ້ນຕອນຖັດໄປຈະແລ່ນ.) · “Commit it with a message that says why, not what.” (Commit ມັນພ້ອມຂໍ້ຄວາມທີ່ບອກວ່າຍ້ອນຫຍັງ, ບໍ່ແມ່ນເຮັດຫຍັງ.) · “It’s in the repo — pull the latest.” (ມັນຢູ່ໃນ repo — pull ສະບັບຫຼ້າສຸດມາ.)

ບົດຝຶກຫັດ: (1) `backup.sh`, ຕາມທີ່ລະບຸຂ້າງເທິງ, ຕັ້ງເວລາແລ່ນທຸກຄືນດ້ວຍ cron ເທິງ EC2 instance ຂອງ lab. (2) `audit.py`, ຕາມທີ່ລະບຸຂ້າງເທິງ. (3) ທັງສອງຖືກ commit ໃສ່ `cloud-journey` ພ້ອມ README ອະທິບາຍແຕ່ລະອັນ. (4) ທໍາລາຍ script ຂອງທ່ານເອງໂດຍເຈດຕະນາ (path ຜິດ, bucket ຫາຍ) ແລ້ວເຮັດໃຫ້ມັນລົ້ມເຫຼວ *ຢ່າງດັງ ແລະ ຈະແຈ້ງ* — ການຈັດການ error ຄືພາສາຮັກຂອງ ops.

Milestone: backup script ຂອງທ່ານລອດການຖືກແລ່ນສອງເທື່ອຕິດກັນ ແລະ ກໍລະນີໂຟນເດີຫາຍ (ບໍ່ crash, ຂໍ້ຄວາມຈະແຈ້ງ); Python audit ຂອງທ່ານແລ່ນຕໍ່ບັນຊີຈິງຂອງທ່ານ; repo ຂອງທ່ານສະແດງ commit ຕິດຕໍ່ໜຶ່ງອາທິດ.

Module 7 (ອາທິດທີ 16–17): Terraform — infrastructure as code

ແນວຄິດຫຼັກ: ແທນທີ່ຈະຄລິກໃຫ້ຊັບພະຍາກອນເກີດຂຶ້ນ, ທ່ານຂຽນໄຟລ໌ຕົວໜັງສືທີ່ *ປະກາດ* ວ່າຄວນມີຫຍັງຢູ່ — “VPC ໜຶ່ງ, subnet ສອງ, instance ໜຶ່ງ, tag ພວກນີ້” — ແລ້ວ **Terraform** ເຮັດໃຫ້ AWS ກົງກັບໄຟລ໌. Declarative (ປະກາດ), ບໍ່ແມ່ນ imperative (ສັ່ງເປັນຂັ້ນ): ທ່ານບອກຈຸດໝາຍປາຍທາງ, ບໍ່ແມ່ນທາງລ້ຽວ. ເຫດຜົນທີ່ທຸກ JD ຮຽກຮ້ອງມັນ: ໄຟລ໌ອາໄສຢູ່ໃນ Git, ດັ່ງນັ້ນໂຄງລ່າງຈຶ່ງຖືກ **ທົບທວນ** (pull request ກ່ອນການປ່ຽນແປງ), **ເຮັດຊ້ຳໄດ້** (ໄຟລ໌ດຽວກັນສ້າງ dev, staging, ແລະ prod ຄືກັນທຸກປະການ), **ຍ້ອນກັບໄດ້** (roll back ດ້ວຍການ revert commit), ແລະ **ກວດສອບໄດ້** (ປະຫວັດບອກວ່າໃຜປ່ຽນຫຍັງ, ເມື່ອໃດ, ຍ້ອນຫຍັງ). Console ຄືໂຮງຊ່າງ; Terraform ຄືໂຮງງານ.

ວິງຈອນແກນທີ່ທ່ານຈະແລ່ນຫຼາຍຮ້ອຍເທື່ອ: terraform init (ຕາວໂຫຼດ **provider** ຂອງ AWS — plugin ທີ່ແປໄຟລ໌ຂອງທ່ານເປັນການເອີ້ນ API) → terraform plan (ການແລ່ນຊ້ອມ ພິມອອກມາແທ້ໆວ່າຫຍັງຈະຖືກສ້າງ/ປ່ຽນ/ທຳລາຍ — ອ່ານທຸກແຖວ; ວິສະວະກອນທີ່ apply ໂດຍບໍ່ອ່ານ plan ກໍ່ໃຫ້ເກີດການລົ້ມລະບົບ) → terraform apply (ເຮັດໃຫ້ມັນເປັນຈິງ) → terraform destroy (ຍົກເລີກມັນ — ການຮືຖອນເປັນຄຳສັ່ງດຽວ, ຊຶ່ງມາຮອດຕອນນີ້ທ່ານຈະເຫັນວ່າມັນງາມ). Terraform ຕິດຕາມສິ່ງທີ່ມັນສ້າງ ໃນ **state file** — ຄວາມຈຳຂອງມັນຕໍ່ຄວາມເປັນຈິງ; ເຮັດມັນເສຍ ຫຼື ໄປແກ້ຄວາມເປັນຈິງລັບຫຼັງມັນ (“drift”) ແລ້ວຄວາມເຈັບປວດຈະຕາມມາ. ຊັບພະຍາກອນຖືກປະກາດໃນ **HCL**, ພາສາ config ທີ່ອ່ານງ່າຍ:

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

Lab (ນິຕິຕົວ module): ສ້າງດ່ານທົດສອບຂອງໄລຍະທີ 1 ຄົ້ນໃໝ່ໃນ Terraform, ເປັນຂັ້ນໆ. (1) Init repo terraform-labs; ຂຽນ config ສຳລັບພຽງ S3 bucket ທີ່ຕິດ tag; plan, ອ່ານ, apply, ກວດ console — ມັນຢູ່ຫັ້ນ; destroy. (2) ຂະຫຍາຍ config: VPC, ສອງ subnet, internet gateway, route table — ການສ້າງຂອງ Lab 3, ຕອນນີ້ປະມານ 60 ແຖວຂອງ HCL. (3) ເພີ່ມ security group ແລະ EC2 instance ພ້ອມ variables.tf (region, ຂະໜາດ instance) ແລະ outputs.tf (ພິມ public IP ຫຼັງ apply). (4) ປ່ຽນ instance type ໃນໄຟລ໌ ແລ້ວ re-apply — ເບິ່ງ Terraform ຄິດໄລ່ຄວາມແຕກຕ່າງ ແລະ ປ່ຽນ *ພຽງອັນນັ້ນ*. ພິດຕິກຳຂັບເຄື່ອນດ້ວຍ diff ນີ້ ຄືເວດມົນທົງໝົດ. (5) Destroy ທຸກຢ່າງ, ຍືນຢັນວ່າ console ຫວ່າງເປົ້າ, commit, ແລ້ວ tag repo ເປັນ v1.0. ເອກະສານອ້າງອີງຢູ່ທີ່ <https://developer.hashicorp.com/terraform> — bookmark ມັນໄວ້; ການອ່ານເອກະສານ provider ຄືເຄິ່ງໜຶ່ງຂອງວຽກ Terraform ຕົວຈິງ.

ຄຳສັບ:

ຄຳສັບ	ຄວາມໝາຍ
IaC (Infrastructure as Code)	ການປະກາດໂຄງລ່າງໃນໄຟລ໌ຕົວໜັງສືທີ່ມີເວີຊັນ ຊຶ່ງເຄື່ອງມືປ່ຽນເປັນຄວາມຈິງ.
Terraform / HCL	ເຄື່ອງມື IaC ຫຼາຍ cloud ທີ່ຄອງຕະຫຼາດ / ພາສາ configuration ຂອງມັນ.
Provider	Plugin ທີ່ແປໄຟລ໌ຂອງທ່ານ ເປັນການເອີ້ນ API ຂອງ cloud ໜຶ່ງ (ທີ່ນີ້: AWS).
plan / apply / destroy	Diff ແບບແລ່ນຊ້ອມ / ເຮັດໃຫ້ເປັນຈິງ / ຍົກເລີກທັງໝົດ. ອ່ານທຸກ plan.
State file	ບັນທຶກຂອງ Terraform ວ່າມັນສ້າງຫຍັງໄວ້ — ຄວາມຈຳຂອງມັນຕໍ່ຄວາມເປັນຈິງ. ປົກປ້ອງມັນ.
Drift	ຄວາມເປັນຈິງປ່ຽນລັບຫຼັງ Terraform (ມີຄົນຄລິກ). ສັດຕູ.

ຄຳສັບ	ຄວາມໝາຍ
Variable / output / module	ຂາຂໍ້ຂອງ config / ຜົນທີ່ຜົມອອກ (IP, URL) / ຂຶ້ນ config ຫຼັກທີ່ໃຊ້ຊ້າໄດ້.
Declarative vs imperative	ບອກຈຸດໝາຍປາຍທາງ ທຽບກັບ ຂຽນບົດທາງລ້ຽວ. Terraform ເປັນ declarative.
CloudFormation	ບໍລິການ IaC ຂອງ AWS ເອງ — ແນວຄິດດຽວກັນ, ສະເພາະ AWS. JD ຮັບທັງສອງ; ຮຽນ Terraform ກ່ອນ.

ວິດີໂອສຳລັບ module ນີ້:

ວິດີໂອ	ຊ່ອງ	ຄວາມຍາວ	ລິງ
Terraform explained in 15 mins	TechWorld with Nana	18 ນາທີ	https://www.youtube.com/watch?v=l5k1ai_GBDE
Terraform Course — Automate your AWS cloud infrastructure	freeCodeCamp	~2.5 ຊົ່ວໂມງ	https://www.youtube.com/watch?v=SLB_c_ayRMO

ເວົ້າອອກສຽງ: “Nothing changes in prod except through Terraform.” (ບໍ່ມີຫຍັງປ່ຽນໃນ prod ນອກຈາກຜ່ານ Terraform.) · “Show me the plan before you apply.” (ສະແດງ plan ໃຫ້ຂ້ອຍເບິ່ງກ່ອນເຈົ້າ apply.) · “That was clicked in manually — that’s drift; let’s import it or remove it.” (ອັນນັ້ນຖືກຄລິກໃສ່ດ້ວຍມື — ນັ້ນຄື drift; ມາ import ຫຼື ເອົາມັນອອກກັນ.) · “It’s in the module; reuse it, don’t rewrite it.” (ມັນຢູ່ໃນ module; ໃຊ້ຊ້າ, ຢ່າຂຽນໃໝ່.)

ບົດຝຶກຫັດ: lab ຂ້າງເທິງ, ບວກ: (1) ສ້າງ drift ໂດຍເຈດຕະນາ — ປ່ຽນ tag ຂອງ instance ດ້ວຍມືໃນ console, ແລ່ນ terraform plan, ແລ້ວເບິ່ງ Terraform ສັງເກດເຫັນ; ຈົດບັນທຶກວ່າມັນສະເໜີຫຍັງ. (2) ຂຽນວັກພາສາທຳມະດາໃນ README ຂອງ repo: “ເປັນຫຍັງອັນນີ້ຈຶ່ງຕົກວ່າການຄລິກ.” ຖ້າທ່ານຂຽນມັນບໍ່ໄດ້, ທ່ານກໍຍັງບໍ່ທັນມີມັນ.

Milestone: ຈາກ directory ເບົາ, ທ່ານສາມາດຍົກ stack VPC + instance ຂຶ້ນດ້ວຍ init/plan/apply ແລະ ເອົາມັນອອກດ້ວຍ destroy, ຜ່ອມອະທິບາຍອອກສຽງວ່າແຕ່ລະຄຳສັ່ງກຳລັງເຮັດຫຍັງ — ໂດຍບໍ່ເບິ່ງບັນທຶກ.

Module 8 (ອາທິດທີ 18–20): CI/CD, Docker, ແລະ ການແນມເບິ່ງ Kubernetes ຄັ້ງທຳອິດ

CI/CD (Continuous Integration / Continuous Delivery) ຄືສາຍພານລຳລຽງຫຸ່ນຍົນ ທີ່ຕິດຢູ່ກັບ Git repo ຂອງທ່ານ: ໃນທຸກ push, ມັນກວດ, ທົດສອບ, build, ແລະ — ເມື່ອຕັ້ງຄ່າໄວ້ — deploy ການປ່ຽນແປງຂອງທ່ານໂດຍອັດຕະໂນມັດ. ຈຸດປະສົງຄືການ release ນ້ອຍ, ຖີ່, ໜ້າເບື້ອ ແທນທີ່ຈະຫາຍາກ ແລະ ໜ້າຢ້ານ. **GitHub Actions** ຄືລະບົບ CI/CD ທີ່ຕິດມາກັບ GitHub: **workflow** ຄືໄຟລ໌ YAML ໃນ .github/workflows/ ທີ່ບອກວ່າ “ເມື່ອ push, ແລ່ນຂັ້ນຕອນເຫຼົ່ານີ້ ເທິງ runner (VM ຊົ່ວຄາວ) ເຄື່ອງໃໝ່.” Lab: ເພີ່ມ workflow ໃສ່ terraform-labs ທີ່ແລ່ນ terraform fmt -check ແລະ terraform validate ໃນທຸກ push — push ໄຟລ໌ທີ່ຜິດຮູບແບບໂດຍເຈດຕະນາ ແລ້ວເບິ່ງ ✗ ສີແດງ, ແກ້ມັນ ແລ້ວເບິ່ງ ✓ ສີຂຽວ. ຕອນນີ້ທ່ານມີ pipeline ແລ້ວ; ນ້ອຍປານໃດກໍຕາມ, ມັນເປັນສາຍພັນດຽວກັບພວກທີ່ຢູ່ໃນທຸກ JD. (ຂັ້ນທີສອງ,

ໃນ Project 1: workflow ທີ່ແລ່ນ terraform plan ໃນ pull request, ເພື່ອໃຫ້ການປ່ຽນແປງໂຄງລ່າງທີ່ສະເໝີ ສະແດງ diff ຂອງມັນໃນການທົບທວນ.)

Docker ຫຸ້ມຫໍ່ແອັບພລິເຄຊັນພ້ອມທຸກສິ່ງທີ່ມັນຕ້ອງການ ເຂົ້າໃນ **container** — ກ່ອງເຂົ້າທ່ຽງທີ່ຜະນຶກແລ້ວ ຊຶ່ງແລ່ນຄືກັນທຸກປະການ ໃນໂນດບຸກຂອງທ່ານ, ເທິງ EC2, ແລະ ທຸກບ່ອນອື່ນ, ຢຸດໂລກລະບາດບູຮານຂອງ “works on my machine.” **image** ຄືສູດອາຫານແຊ່ແຂງ (build ຈາກ **Dockerfile**, ໄຟລ໌ຕົວໜັງສືສັນຂອງຂັ້ນຕອນ build: ເລີ່ມ FROM base image, COPY ແອັບຂອງທ່ານເຂົ້າໄປ, ກໍານົດຄໍາສັ່ງເລີ່ມຕົ້ນ); **container** ຄື instance ທີ່ກໍາລັງແລ່ນຂອງ image ໜຶ່ງອັນ; **registry** (Docker Hub, ຫຼື **ECR** ຂອງ AWS) ຄືບ່ອນ push ແລະ pull image. Container ບໍ່ແມ່ນ VM: ພວກມັນແບ່ງໃຊ້ Linux kernel ຂອງເຄື່ອງ host, ຈຶ່ງເລີ່ມຕົ້ນໃນປະມານໜຶ່ງວິນາທີ ແລະ ທ່ານແລ່ນຫຼາຍສິບອັນເທິງ micro instance ໄດ້. Lab: ເທິງ EC2 instance, ຕິດຕັ້ງ Docker, docker run hello-world, ຈາກນັ້ນ docker run -d -p 80:8080 <a sample web image> ແລ້ວເປີດ browser ຫາມັນ; ຈາກນັ້ນຂຽນ Dockerfile ຫ້າແຖວ ທີ່ເສີບໜ້າ static ຂອງ Lab 2 ຈາກ base image nginx, build ມັນ, ແລ່ນມັນ, ແລ້ວ push ຂຶ້ນ Docker Hub. ຮຽນຄໍາກີລິຍາປະຈຳວັນ: docker ps, docker logs, docker exec -it <id> bash (shell ພາຍໃນ container), docker stop.

Kubernetes (K8s) — ສໍາລັບຫຼັກສູດນີ້, ທັກສະລະດັບການອ່ານ, ຕິດປ້າຍຢ່າງຊື່ສັດ. ເມື່ອບໍລິສັດແລ່ນ container ຫຼາຍຮ້ອຍອັນ ຂ້າມຫຼາຍເຄື່ອງ, ຕ້ອງມີບາງສິ່ງມາຈັດຕາຕະລາງພວກມັນ, restart ອັນທີ່ crash, ຂະຫຍາຍອັນທີ່ວຽກໜັກ, ແລະ ນໍາທາງ traffic ລະຫວ່າງພວກມັນ: ຕົວປະສານວົງນັ້ນຄື Kubernetes. ຮຽນແຜນທີ່ແນວຄິດຕອນນີ້ — **cluster** ຂອງ **node** ແລ່ນ **pod** (ຫົວໜ່ວຍ deploy ນ້ອຍທີ່ສຸດ, ປົກກະຕິ container ດຽວ); **deployment** ປະກາດວ່າ “ຮັກສາ 3 replica ຂອງ pod ນີ້ໃຫ້ມີຊີວິດ” ແລ້ວ cluster ເຮັດໃຫ້ມັນເປັນຈິງຢ່າງຕໍ່ເນື່ອງ (declarative ອີກແລ້ວ — Kubernetes ຄືປັດຊະຍາຂອງ Terraform ນໍາໃຊ້ກັບ software ທີ່ກໍາລັງແລ່ນ); **service** ໃຫ້ pod ມີທີ່ຢູ່ໝັ້ນຄົງ. ບໍລິການ managed ຂອງ AWS ຄື **EKS**. JD ລະດັບ junior ຕ້ອງການຄວາມຮູ້ອ່ານອອກຂຽນໄດ້ນີ້ແທ້ໆ ບວກຄວາມຄ່ອງ Docker; ທັກສະປະຕິບັດງານ K8s ຕົວຈິງ (ແລະ cert CKA) ຄືເປົ້າໝາຍທີ່ດີຂອງປີທີ 2, ບໍ່ແມ່ນເດືອນທີ 5. ຫຼັກສູດນີ້ບອກທ່ານຢ່າງຊື່ສັດວ່າອັນໃດເປັນອັນໃດ.

ຄໍາສັບ:

ຄໍາສັບ	ຄວາມໝາຍ
CI/CD	Pipeline ອັດຕະໂນມັດທີ່ກວດ, build, ທົດສອບ, ແລະ ສົ່ງທຸກການປ່ຽນແປງ.
GitHub Actions / workflow / runner	CI/CD ທີ່ຕິດມາກັບ GitHub / ໄຟລ໌ YAML ທີ່ນິຍາມ pipeline / VM ຊຶ່ງຄາວທີ່ປະຕິບັດມັນ.
YAML	ຮູບແບບ config ອົງການຫຍໍ້ໜ້າ ຂອງ pipeline ແລະ Kubernetes. ການຫຍໍ້ໜ້າຄືຄວາມໝາຍ — ລະວັງມັນ.
Docker / image / container	ຊຸດເຄື່ອງມື container / ສູດອາຫານແຊ່ແຂງເປັນຊັ້ນ / instance ທີ່ກໍາລັງແລ່ນຂອງມັນໜຶ່ງອັນ.
Dockerfile	ໄຟລ໌ຕົວໜັງສືສັນຂອງຂັ້ນຕອນ ທີ່ build image.
Registry / ECR	ບ່ອນ push ແລະ pull image / registry ຂອງ AWS.
Kubernetes (K8s) / cluster / node	ຕົວປະສານວົງ container / ກຸ່ມເຄື່ອງຂອງມັນ / ເຄື່ອງໜຶ່ງໜ່ວຍໃນນັ້ນ.

ຄໍາສັບ	ຄວາມໝາຍ
Pod / deployment / service	ຫົວໜ່ວຍ deploy ນ້ອຍທີ່ສຸດ / “ຮັກສາ N replica ໃຫ້ມີຊີວິດ,” ບັງຄັບໃຊ້ຢ່າງຕໍ່ເນື່ອງ / ທີ່ຢູ່ພື້ນຄົງໝ້າ pod.
EKS	Kubernetes control plane ແບບ managed ຂອງ AWS.
Rollback	ການກັບຄືນເວີຊັນກ່ອນຢ່າງໄວ ເມື່ອ release ມີບັນຫາ — ຕາໜ່າງນິລະໄຍທີ CI/CD ເຮັດໃຫ້ເປັນເລື່ອງທໍາມະດາ.

ວິດີໂອສໍາລັບ module ນີ້:

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

ເລື່ອງອອກສຽງ: “Don’t merge until the pipeline is green.” (ຢ່າ merge ຈົນກວ່າ pipeline ຈະຂຽວ.) · “It’s containerized — same image in dev and prod.” (ມັນເປັນ container ແລ້ວ — image ດຽວກັນໃນ dev ແລະ prod.) · “Exec into the container and check its logs.” (Exec ເຂົ້າ container ແລ້ວກວດ log ຂອງມັນ.) · “K8s keeps three replicas up; kill one and watch it come back.” (K8s ຮັກສາສາມ replica ໄວ້; ຂ້າອັນໜຶ່ງ ແລ້ວເບິ່ງມັນກັບຄືນມາ.)

ບົດຝຶກຫັດ: ສອງ lab ຂ້າງເທິງ, ບວກ: (1) ຈົດບັນທຶກຄໍາຕອບໝັ້ງວັກ ຕໍ່ “VM vs container vs serverless — ໃຊ້ອັນໃດເມື່ອໃດ?” (Serverless in 100 Seconds ຂອງ Fireship — https://www.youtube.com/watch?v=W_VV2Fx32_Y — ເຕີມທາງເລືອກທີ່ສາມໃຫ້ຄົບ). (2) Terminate instance ຂອງ lab Docker; ຢືນຢັນວ່າ image ຂອງທ່ານຍັງມີຊີວິດຢູ່ໃນ registry — ສັງເກດວ່າ *ຊີ້ນງານ* (artifact) ຕອນນີ້ອາຍຸຍືນກວ່າ server, ຊຶ່ງຄືໂລກຫັດການ deploy ສະໄໝໃໝ່ທັງໝົດ ໃນການສັງເກດດຽວ.

Milestone — ຈົບໄລຍະທີ 2: GitHub ຂອງທ່ານສະແດງ: repo ຂອງ Bash + Python ops script ທີ່ໃຊ້ງານໄດ້, repo Terraform ທີ່ສ້າງ ແລະ ທໍາລາຍ stack ຕົວຈິງ, workflow Actions ສີຂຽວ, ແລະ Dockerfile ທີ່ push ຂຶ້ນ registry — ແລະ

ທ່ານອະທິບາຍທຸກໄຟລ໌ໃນການສຳພາດໄດ້. Profile GitHub ນັ້ນບໍ່ແມ່ນຂອງນັກຮຽນອີກຕໍ່ໄປ. ມັນເປັນຂອງວິສະວະກອນ junior.

ໄລຍະທີ 3 — ການປະຕິບັດງານ (ອາທິດທີ 21–28)

ແນວຄິດຂອງໄລຍະນີ້: ການສ້າງລະບົບເຮັດໃຫ້ທ່ານຖືກຈ້າງ; ການ ແລ່ນ ພວກມັນຄົວຽກຕົວຈິງ. ຂໍ JD ທີ່ໄລຍະນີ້ຕອບ ຄືເຄິ່ງ ດ້ານປະຕິບັດງານ — “monitor infrastructure health,” “participate in incident response, including log analysis,” “identifying, analyzing, and resolving infrastructure vulnerabilities,” “manage cloud costs.” ອາທິດຂອງ ວິສະວະກອນ cloud ສ່ວນໃຫຍ່ຄືໄລຍະນີ້.

Module 9 (ອາທິດທີ 21–24): ການຕິດຕາມກວດກາ, ການບັນທຶກ log, ແລະ ເຫດການສຸກເສີນ

Monitoring — ຮູ້ກ່ອນຜູ້ໃຊ້ຈະຮູ້. CloudWatch ຄືບໍລິການ observability ທີ່ຕິດມາກັບ AWS. ສາມາດຈັດປະກອບພື້ນຖານ: **metrics** (ຕົວເລກຕາມເວລາ — CPU %, ດິສ %, ຈຳນວນຄຳຮ້ອງຂໍ, ຈຳນວນ error), **alarms** (ກົດທີ່ເຝົ້າເບິ່ງ metric: “ຖ້າ CPU > 80% ເປັນເວລາ 5 ນາທີ → ແຈ້ງເຕືອນ”), ແລະ **dashboards** (metric ຈັດວາງເທິງໜ້າຈໍດຽວ). ການແຈ້ງເຕືອນໄຫຼ ຜ່ານ **SNS** (Simple Notification Service — topic ທີ່ທ່ານ publish ໃສ່, ຜູ້ subscribe ໄດ້ຮັບອີເມວ/page). ຝົນຢູ່ທີ່ **alarm** ໃສ່ຫຍັງ: page ຫາມະນຸດສະເພາະສິ່ງທີ່ຕ້ອງການມະນຸດ, ບໍ່ດັ່ງນັ້ນຄົນຈະຮຽນເມື່ນເສີຍ pager (**alert fatigue** — ໄໝດຄວາມລົມເຫຼວທີ່ນຳໜ້າການລົມລະບົບໂດ່ງດັງຫຼາຍຄັ້ງ). ສິສັນຍານທອງຄຳທີ່ຄວນຈິ: **latency, traffic, errors, saturation** — ຊ້າປານໃດ, ຫຍຸ້ງປານໃດ, ພັງປານໃດ, ເຕັມປານໃດ.

Logging — ຮູ້ ຍ້ອນຫຍັງ. Metric ບອກວ່າມີບາງຢ່າງຜິດ; log ບອກວ່າເກີດຫຍັງຂຶ້ນ. CloudWatch Logs ລວມສູນພວກມັນ: **agent** ເທິງແຕ່ລະ instance ສົ່ງໄຟລ໌ຄື `/var/log/nginx/access.log` ໄປຫາ **log groups**, ບ່ອນທີ່ **Logs Insights** ໃຫ້ທ່ານ query ຂ້າມຫຼາຍເຄື່ອງ (“ນັບ 5xx response ຕໍ່ນາທີ ໃນຊົ່ວໂມງຜ່ານມາ”). ການລວມສູນສຳຄັນ ເພາະ ຕອນນີ້ server ເປັນຂອງໃຊ້ແລ້ວຖືມ (ທ່ານພິສູດແລ້ວໃນ Module 8) — log ຕ້ອງອາຍຸຍືນກວ່າເຄື່ອງທີ່ຂຽນມັນ.

Lab (ອາທິດທີ 21–22): ຕັ້ງ web server ນ້ອຍທີ່ຖືກຕິດຕາມ, ທັງໝົດໃນ Terraform (stack ຂອງທ່ານຈາກ Module 7, ຂະຫຍາຍຕໍ່): EC2 + nginx + CloudWatch agent; SNS topic ທີ່ສົ່ງອີເມວຫາທ່ານ; alarm ໜຶ່ງໃສ່ CPU ສູງ ແລະ ອີກອັນໃສ່ຄວາມລົມເຫຼວຂອງ instance status check. ຈາກນັ້ນໂຈມຕີຕົນເອງ: SSH ເຂົ້າໄປ ແລ້ວແລ່ນຕົວເຜົາ CPU (`yes > /dev/null & ຈັກສອງສາມເທື່ອ`); ເບິ່ງ metric ໄຕ່ຂຶ້ນ, alarm ດັ່ງ, ອີເມວມາຮອດ; ຂ້າ process ພວກນັ້ນ; ເບິ່ງມັນຟື້ນຕົວ. ຈາກນັ້ນ query access log ຂອງທ່ານເອງໃນ Logs Insights. ຕອນນີ້ທ່ານໄດ້ເປັນພະຍານວົງຈອນເຕັມ `detect → notify → diagnose → resolve` ເທິງລະບົບທີ່ທ່ານສ້າງເອງ.

Incident response — ເຄິ່ງທີ່ເປັນມະນຸດ. incident ຄື “ລະບົບບໍ່ປົກກະຕິ” ທີ່ບໍ່ໄດ້ວາງແຜນ; ລະດັບ **severity** (sev-1 = ລູກຄ້າໃຊ້ບໍ່ໄດ້, ທຸກຄົນລົງມື) ກຳນົດຂະໜາດການຮັບມື. ວົງຈອນມື້ອາຊີບ: **detect** (ຈາກ alarm, ບໍ່ແມ່ນຈາກອີເມວລູກຄ້າ, ຢ່າງທີ່ຄວນ) → **triage** (ຮ້າຍແຮງປານໃດ, ຕ້ອງການໃຜ) → **mitigate** (ຢຸດເລືອດໄຫຼ ກ່ອນ — roll back, restart, failover; ຫາສາເຫດຮາກເຫງົ້າພາຍຫຼັງ) → **resolve** → **post-mortem**: ການທົບທວນເປັນລາຍລັກອັກສອນ ແບບບໍ່ໂຫດກັ້ນ (*blameless*) ວ່າເກີດຫຍັງຂຶ້ນ, ຍ້ອນຫຍັງ, ແລະ ຫຍັງຈະປ້ອງກັນການເກີດຊ້າ. Blameless ບໍ່ແມ່ນຄວາມອ່ອນໄຫວ; ມັນຄື ວິສະວະກຳ: ຄົນທີ່ຖືກລົງໂທດຈະເຊື່ອງຂໍ້ມູນ, ແລະ ຂໍ້ມູນທີ່ຖືກເຊື່ອງກໍໃຫ້ເກີດການລົມລະບົບຊ້າ. **On-call** ຄືການຜັດປ່ຽນວ່າ

ໃຜຖື pager; JD ກ່າວເຖິງມັນ, ຜູ້ສຳພາດຖາມກ່ຽວກັບມັນ, ແລະ ຄຳຕອບຊີ້ສັດຂອງທ່ານຫຼັງ module ນີ້ຄື “ຂ້ອຍໄດ້ຈຳລອງມັນ ແລະ ຂ້ອຍຮູ້ວິງຈອນ.”

Runbook — ຂໍ້ເອກະສານໃນ JD, ເຮັດໃຫ້ເປັນຮູບປະທຳ. Runbook ຄືສູດເປັນຂັ້ນຕອນສຳລັບສະຖານະການປະຕິບັດງານໜຶ່ງອັນ, ຂຽນໃຫ້ຄົນທີ່ຄຽດຕອນຕິສາມເຮັດຕາມໄດ້: ອາການ → ການກວດ (ຄຳສັ່ງແທ້ໆ) → ການແກ້ (ຄຳສັ່ງແທ້ໆ) → ການສົງຄໍ (ຈະປຸກໃຜຖ້າມັນບໍ່ໄດ້ຜົນ). **Lab (ອາທິດທີ 23–24):** ຂຽນສອງ runbook ໃນ repo ຂອງທ່ານ — “web server ລົ້ມ” ແລະ “ດິສກຳລັງເຕັມ” — ແລ້ວ *ຝຶກຊ້ອມ* ພວກມັນ: ທຳລາຍສິ່ງນັ້ນ, ເຮັດຕາມເອກະສານຂອງທ່ານເອງຢ່າງຕາມໂຕອັກສອນ, ແລ້ວແກ້ທຸກຂັ້ນຕອນທີ່ພິສູດວ່າມີວ. ຈາກນັ້ນຈັດ **game day** ເຕັມຮູບແບບໜຶ່ງເທື່ອ: ໃຫ້ໝູ່ (ຫຼື AI) ທຳລາຍ stack lab ຂອງທ່ານແບບລັບໆ; ທ່ານຖືກ page, ວິນິດໄສຈາກ metric ແລະ log, mitigate, ແລ້ວຂຽນ post-mortem ໃນປຶ້ມບັນທຶກ. Post-mortem ນັ້ນຄືເລື່ອງເລົ່າສຳພາດ, ແລະ ເປັນເລື່ອງທີ່ດີ.

ຄຳສັບ:

ຄຳສັບ	ຄວາມໝາຍ
CloudWatch	ບໍລິການ monitoring ຂອງ AWS: metrics, alarms, dashboards, logs.
Metric / alarm / dashboard	ຕົວເລກຕາມເວລາ / ກິດທີ່ດັງໃສ່ມັນ / ໜ້າຈໍດຽວຂອງພວກມັນ.
SNS	ບໍລິການແຈ້ງເຕືອນທີ່ alarm publish ໃສ່ — ອີເມວ, SMS, pager.
Golden signals	Latency, traffic, errors, saturation — ສີ່ຕົວເລກທີ່ບັນຍາຍສຸຂະພາບຂອງທຸກ service.
Alert fatigue	Page ທີ່ເຮັດຫຍັງບໍ່ໄດ້ຫຼາຍໄພດ → page ຖືກເມີນເສີຍ → ພາດອັນຈິງ. ສັດຕູຂອງການອອກແບບ alarm.
Log group / Logs Insights	ບ່ອນທີ່ log ລວມສູນຕົກລົງ / ພາສາ query ເໝືອພວກມັນ.
Incident / severity	ການເລື່ອມສະພາບທີ່ບໍ່ໄດ້ວາງແຜນ / ຄວາມຮ້າຍແຮງທີ່ຈັດອັນດັບ (sev-1 = ຮ້າຍສຸດ).
Triage / mitigate / resolve	ປະເມີນໄວ / ຢຸດເລືອດໄຫຼກ່ອນ / ແກ້ໄຂແທ້ໆ.
Post-mortem	ການທົບທວນເປັນລາຍລັກອັກສອນແບບບໍ່ໂທດກັນ: ຫຍັງ, ຍ້ອນຫຍັງ, ຫຍັງປ້ອງກັນການເກີດຊ້ຳ.
Runbook	ສູດກັນຕິສາມ ສຳລັບສະຖານະການດຽວ: ອາການ, ການກວດ, ການແກ້, ການສົງຄໍ.
On-call / game day	ການຜັດປ່ຽນຖື pager / ເຫດການສຸກເສີນຝຶກຊ້ອມໂດຍເຈດຕະນາ.
MTTR	Mean time to recovery — ຕົວຊີ້ວັດ ops ທີ່ທົມແກ່ກ້າເຝີມປະສິດທິພາບ.

ເວົ້າອອກສຽງ: “Did we find out from the alarm or from a customer?” (ພວກເຮົາຮູ້ຈາກ alarm ຫຼືຈາກລູກຄ້າ?) · “Mitigate first — root cause after we’re stable.” (Mitigate ກ່ອນ — ຫາສາເຫດຮາກເຫງົ້າ ຫຼັງເຮົາໝັ້ນຄົງແລ້ວ.) · “Is there a runbook for this? There will be by tomorrow.” (ມີ runbook ສໍາລັບອັນນີ້ບໍ່? ມີອັນຈະມີ.) · “What did the post-mortem conclude, and what’s the prevention item?” (Post-mortem ສະຫຼຸບວ່າແນວໃດ, ແລະ ລາຍການປ້ອງກັນແມ່ນຫຍັງ?)

Milestone: post-mortem ຈາກ game day ຂອງທ່ານມີຢູ່ຈິງ, ເປັນແບບບໍ່ໂທດກັນ, ແລະ ລະບຸການປ້ອງກັນຮູບປະທໍາໜຶ່ງອັນທີ່ທ່ານໄດ້ຈັດຕັ້ງແທ້ຈາກນັ້ນ (alarm ທີ່ເພີ່ມ, runbook ທີ່ແກ້).

Module 10 (ອາທິດທີ 25–28): ການປະຕິບັດງານດ້ານຄວາມປອດໄພ ແລະ ການຄຸ້ມຄອງຕົ້ນທຶນ

Security operations — ສຸຂະອະນາໄມ, ບໍ່ແມ່ນວິລະກໍາ. Shared responsibility model (ແບບຈຳລອງຄວາມຮັບຜິດຊອບຮ່ວມ) ກ່ອນ (ເບິ່ງຄືນ: <https://www.youtube.com/watch?v=ESPBEEK-cvo>): AWS ຮັກສາຄວາມປອດໄພຂອງຕົວ cloud ເອງ; ທ່ານ ຮັກສາຄວາມປອດໄພຂອງສິ່ງທີ່ທ່ານເອົາໃສ່ໃນມັນ — ແລະ ການເຈາະລະບົບຕົວຈິງສ່ວນໃຫຍ່ ຄືການຕັ້ງຄ່າຜິດຂອງລູກຄ້າ, ບໍ່ແມ່ນຄວາມລຶມເຫຼວຂອງ AWS. Checklist ຄວາມປອດໄພປະຕິບັດງານຂອງທ່ານ, ຝຶກຈົນໜ້າເບື້ອງ:

1. **ສຸຂະອະນາໄມ IAM** (ເດືອນລະເທື່ອ): MFA ທຸກບ່ອນ; ບໍ່ມີ access key ອາຍຸຍາວ ບ່ອນທີ່ role ໃຊ້ແທນໄດ້; ແລ່ນ credential report; ລຶບ user ແລະ key ທີ່ບໍ່ໃຊ້; ຕັ້ງຄ່າຖາມຕໍ່ທຸກ * ໃນທຸກ policy. ທ່ານສ້າງປະຕິກິລິຍານີ້ໃນ Lab 5 — ຕອນນີ້ມັນເປັນລາຍການໃນປະຕິທິນ.
2. **Patching:** software ທີ່ບໍ່ patch ຄືວິທີທີ່ການເຈາະເຂົ້າສ່ວນໃຫຍ່ເລີ່ມຕົ້ນ. ເທິງ instance ຂອງທ່ານ: ຕິດຕັ້ງອັບເດດຕາມຕາຕະລາງ (ແລະ ຮູ້ວ່າ **SSM Patch Manager** ເຮັດອັດຕະໂນມັດຫົວທັງຝູງ); ໃນໂລກ container, ການ patch ມັກໝາຍເຖິງການ build image ຄືນຈາກ base ທີ່ອັບເດດແລ້ວ — ເຊື່ອມອັນນັ້ນກັບ Module 8.
3. **Backups — ແບບທີ່ທົດສອບແລ້ວ:** backup ທີ່ບໍ່ໄດ້ທົດສອບ ຄືຄວາມຫວັງ, ບໍ່ແມ່ນແຜນ. RDS snapshot, S3 versioning (ທຸກການຂຽນທັບຮັກສາເວີຊັນເກົ່າໄວ — ການຄວບຄຸມກັນພາດມື ແລະ ກັນ ransomware). Lab: snapshot database lab ຂອງທ່ານ, *restore* ມັນ ໃສ່ instance ໃໝ່, ຢືນຢັນແຖວຂໍ້ມູນ, ຮີຖອນ. ຕອນນີ້ທ່ານມີສິດເວົ້າ “tested” ໃນການສໍາພາດແລ້ວ.
4. **Guardrails (ຮາວກັນ):** ເປີດ **CloudTrail** (log ກວດສອບຂອງທຸກການເອີ້ນ API ໃນບັນຊີ — ໃຜເຮັດຫຍັງ, ເມື່ອໃດ) ແລ້ວກວດເບິ່ງເຫດການມີວານໜຶ່ງເທື່ອ; ເປີດ free trial ຂອງ **GuardDuty** (ການກວດຈັບໄພຂົ່ມຂູ່ອັດຕະໂນມັດ) ແລ້ວອ່ານວ່າມັນເຝົ້າເບິ່ງຫຍັງ. ຮູ້ວ່າລິ **encryption at rest and in transit** (ການເຂົ້າລະຫັດຂະນະຈັດເກັບ ແລະ ຂະນະສົ່ງ) ແລະ ວ່າໃນ AWS ມັນສ່ວນໃຫຍ່ເປັນ checkbox ທີ່ຮອງຮັບໂດຍ **KMS** — ເປັນມາດຕະຖານຂັ້ນຕໍ່າ, ເປີດສະເໝີ.

ການຄຸ້ມຄອງຕົ້ນທຶນ — ຫັກສະທີ່ເຮັດໃຫ້ junior ເບິ່ງຄື senior. ຂໍ JD ຄໍາຕໍ່ຄໍາ: “manage cloud costs through rightsizing resources, implementing auto-scaling, resource tagging.” ມີເຕີແລ່ນແນວໃດ: compute ຄິດເງິນຕໍ່ວິນາທີທີ່ມັນ *ເປີດ* (idle ≠ ຟຣີ — “ພວກເຮົາປະມັ້ນແລ່ນໄວ້” ຄືການສູນເສຍຄລາສສິກ), storage ຕໍ່ GB-ເດືອນ, ແລະ egress (ຂໍ້ມູນອອກ ຈາກ AWS) ຕໍ່ GB ໃນຂະນະທີ່ຂໍ້ມູນເຂົ້າຟຣີ — ຄວາມແປກໃຈໃນໃບບິນທີ່ໂດ່ງດັງ. ຄັນໂຍກ, ຕາມລໍາດັບທີ່ junior ດຶງໄດ້: **ປິດມັນ** (ລະບົບ dev ຕອນກາງຄືນ; ນິໄສຮີຖອນຂອງທ່ານ, ຍົກລະດັບເປັນອຸດສາຫະກໍາ) → **rightsized** (server ສ່ວນໃຫຍ່ຂະໜາດເກີນ; ກວດປະຫວັດ CPU ໃນ CloudWatch ແລ້ວຫົດມັນ) → **tag ທຸກຢ່າງ** (Project, Owner, Environment — ລາຍຈ່າຍບໍ່ມີ tag ຄືລາຍຈ່າຍທີ່ບໍ່ມີໃຜຮັບຜິດຊອບ; ບັງຄັບ tag ໃນ Terraform ຂອງທ່ານ) → **ຈັດຂັ້ນ storage** (lifecycle rule ຈາກ Lab 2) → ຮູ້ວ່າ **reserved capacity/Savings Plans** (ຜູກມັດ 1–3 ປີ, ປະຢັດ 30–70%) ແລະ **Spot** (ຫຼຸດເຖິງ 90%, ຖືກຂັດຈັງຫວະໄດ້) ມີໄວ້ສໍາລັບ workload ຄົງທີ່ ແລະ batch ຕາມລໍາດັບ. ເຄື່ອງມື: **Cost Explorer** (ລາຍຈ່າຍຂອງບັນຊີ, ເປັນເສັ້ນສະແດງ — ອ່ານມັນທຸກອາທິດ, ຕະຫຼອດໄປ) ແລະ **AWS Budgets** (ການເຕືອນ \$0 ຂອງທ່ານຈາກອາທິດທີ 3,

ຕອນນີ້ເຂົາໃຈແລ້ວວ່າເປັນສະມາຊິກນ້ອຍສຸດ ຂອງຄອບຄົວທີ່ຈິງຈັງ). ວິໄນທີ່ກວ້າງກວ່ານີ້ເອີ້ນວ່າ **FinOps** — ຄຸ້ມຄ່າສອງນາທີ: https://www.youtube.com/watch?v=Y-c_xw9bHFw.

ຄໍາສັບ:

ຄໍາສັບ	ຄວາມໝາຍ
Shared responsibility model	AWS ຮັກສາຄວາມປອດໄພຂອງ cloud; ທ່ານຮັກສາຄວາມປອດໄພຂອງສິ່ງທີ່ໃສ່ໃນມັນ. ການເຈາະສ່ວນໃຫຍ່ຢູ່ເທິງທີ່ສອງ.
CloudTrail	Log ກວດສອບຂອງບັນຊີ: ທຸກການເອີ້ນ API, ໂດຍໃຜ, ເມື່ອໃດ.
GuardDuty	ການກວດຈັບໄພຂົ່ມຂູ່ອັດຕະໂນມັດຂອງ AWS ເໝືອ log ແລະ traffic.
Patching / SSM	ການຕິດຕັ້ງອັບເດດຄວາມປອດໄພ / Systems Manager ຂອງ AWS, ທີ່ເຮັດອັດຕະໂນມັດທົ່ວທັງຝູງ.
S3 versioning	ຮັກສາທຸກເວີຊັນທີ່ຖືກຂຽນທັບ — ການຄວບຄຸມກັນພາດມື, ກັນ ransomware.
KMS / encryption at rest & in transit	ບໍລິການກະແຈຂອງ AWS / ຂໍ້ມູນຖືກປົນເທິງດິສ ແລະ ເທິງສາຍ. ເປີດສະເໝີ.
Egress	ຂໍ້ມູນອອກຈາກ AWS — ຄິດເງິນຕໍ່ GB; ຂາເຂົ້າຟຣີ. ຄວາມແປກໃຈຄລາສສິກໃນໃບບິນ.
Rightsizing	ການຫົດຊັບພະຍາກອນຂະໜາດເກີນ ລົງສູ່ຄວາມຕ້ອງການທີ່ຈິງແທ້ແລ້ວ. ເງິນຟຣີ.
Tagging	ການຕິດປ້າຍທຸກຊັບພະຍາກອນ ດ້ວຍເຈົ້າຂອງ/ໂຄງການ/ສະພາບແວດລ້ອມ ເພື່ອໃຫ້ທຸກລາຍຈ່າຍຊີ້ຕົວໄດ້.
Savings Plans / Spot	ຜູກມັດ 1–3 ປີ ຫຼຸດ 30–70% ສໍາລັບໂຫຼດຄົງທີ່ / ຄວາມຈູເຫຼືອທີ່ຖືກຮຽກຄືນໄດ້ ຫຼຸດເຖິງ 90% ສໍາລັບ batch.
Cost Explorer / Budgets	ເສັ້ນສະແດງລາຍຈ່າຍທີ່ທ່ານອ່ານທຸກອາທິດ / ການເຕືອນທີ່ໝາຍວ່າທ່ານບໍ່ຕ້ອງຮຽນຈາກໃບແຈ້ງໜີ້ຈັກເໝືອ.
FinOps	ວິໄນຂອງການເຮັດໃຫ້ລາຍຈ່າຍ cloud ເບິ່ງເຫັນໄດ້, ຈັດສັນໄດ້, ແລະ ຖືກປັບປຸງຢ່າງຕໍ່ເນື່ອງ.

ເວົ້າອອກສຽງ: “Who has access to prod, and when did we last review the list?” (ໃຜເຂົ້າເຖິງ prod ໄດ້, ແລະ ເຮົາທົບທວນລາຍຊື່ຄັ້ງສຸດທ້າຍເມື່ອໃດ?) · “When did we last *restore* a backup?” (ເຮົາ *restore* backup ຄັ້ງສຸດທ້າຍເມື່ອໃດ?) · “What’s untagged, and what died but is still billing?” (ຫຍັງບໍ່ມີ tag, ແລະ ຫຍັງຕາຍແລ້ວແຕ່ຍັງຄິດເງິນຢູ່?) · “It’s over-provisioned — the CPU history says we can halve it.” (ມັນ provision ເກີນ — ປະຫວັດ CPU ບອກວ່າເຮົາຫຼຸດເຄິ່ງໄດ້.)

ບົດຝຶກຫັດ: (1) Lab ທົດສອບ restore ຂ້າງເທິງ. (2) ໄຟລ໌ checklist ຄວາມປອດໄພລາຍເດືອນໃນ repo ຂອງທ່ານ, ແລ້ວແລ່ນມັນຕໍ່ບັນຊີຂອງທ່ານແທ້ໆ. (3) ໃນ Cost Explorer, ຊອກຫາສິ່ງທີ່ແພງທີ່ສຸດໃນປະຫວັດບັນຊີຂອງທ່ານ ແລ້ວອະທິບາຍມັນໃນປຶ້ມບັນທຶກໜຶ່ງປະໂຫຍກ. (4) ເພີ່ມ tag ມາດຕະຖານໃສ່ທຸກຊັບພະຍາກອນໃນ repo Terraform ຂອງທ່ານ.

Milestone — ຈົບໄລຍະທີ 3: ແລ່ນການກວດສອບຕົນເອງເຕັມຮູບແບບໜຶ່ງເທື່ອ — checklist ຄວາມປອດໄພ, ການທົບທວນຕົນທຶນ, ການທົດສອບ alarm, ການ restore backup — ແລ້ວຂຽນລາຍງານໜຶ່ງໜ້າ. ຕອນນີ້ທ່ານເຮັດ *ວຽກ* ໄດ້ແລ້ວ, ບໍ່ແມ່ນພຽງການສ້າງ. ສິ່ງທີ່ເຫຼືອຄືການພິສູດມັນຕໍ່ຄົນແປກໜ້າ: ໄລຍະທີ 4.

ໄລຍະທີ 4 — ຄວາມພ້ອມສູ່ວຽກ (ເດືອນທີ 8–12)

Module 11 (ເດືອນທີ 8–10): ສາມໂຄງການ portfolio

Certification ບອກວ່າທ່ານໄດ້ຮຽນ; ໂຄງການພິສູດວ່າທ່ານສ້າງໄດ້. ສາມອັນ, ແຕ່ລະອັນຢູ່ໃນ GitHub repo ຂອງມັນເອງ, ແຕ່ລະອັນມີແຜນວາດສະຖາປັດຕະຍະກຳ, README ທີ່ຂຽນສຳລັບ hiring manager (ຫຍັງ, ຍ້ອນຫຍັງ, ແລ່ນແນວໃດ, ມີຄ່າໃຊ້ຈ່າຍເທົ່າໃດ), ແລະ script ຮີຖອນ. ສ້າງ → screenshot/ບັນທຶກວິດີໂອ → *ທຳລາຍ* — repo ຄືຊື່ງານ, ບໍ່ແມ່ນໃບບິນທີ່ແລ່ນຢູ່.

ໂຄງການທີ 1 — Web app ສາມຊັ້ນ, ອັດຕະໂນມັດເຕັມຮູບແບບ (ຊິນເອກ). Terraform ສ້າງທຸກຢ່າງ: VPC ພ້ອມ subnet public/private ຂ້າມສອງ AZ; **Application Load Balancer** (ອຸປະກອນກະຈາຍ traffic — ໃໝ່ສຳລັບທ່ານ, ແລະ ເປັນບາດກ້າວຕໍ່ໄປແບບທຳມະຊາດຈາກ Lab 3) ຢູ່ໜ້າ **Auto Scaling Group** ຂອງ web instance (AWS ເພີ່ມ/ຫຼຸດ instance ຕາມໂຫຼດ — ຄົ້ນຫາມັນ, ຕໍ່ສາຍມັນ); RDS ໃນ subnet private; S3 ສຳລັບ static asset; alarm CloudWatch ແລະ runbook ໜຶ່ງອັນ. GitHub Actions ແລ່ນ terraform plan ໃນທຸກ pull request ແລະ apply ເມື່ອ merge — pipeline ໂຄງລ່າງຕົວຈິງ. ໂຄງການດຽວນີ້ສາທິດສືບໃນສືບສີຂີ້ JD ໃນຕາຕະລາງແຜນທີ່; ຄາດວ່າທຸກການສຳພາດຈະຍ່າງຜ່ານມັນ, ແລະ ຝຶກຊ້ອມບັນຍາຍມັນໃນຫ້ານາທິ.

ໂຄງການທີ 2 — Pipeline ຂໍ້ມູນແບບ serverless (ຄວາມກວ້າງ). ບໍ່ມີ server ເລີຍ: ໄຟລ໌ທີ່ຕົກລົງໃນ S3 bucket ກະຕຸ້ນ **Lambda function** (Python ຂອງທ່ານ, ແລ່ນໂດຍ AWS ຕໍ່ event, ຄິດເງິນຕໍ່ການເອີ້ນ) ທີ່ປະມວນຜົນມັນ — parse CSV, ສະຫຼຸບມັນ, ຂຽນຜົນລົງ bucket ທີ່ສອງ ຫຼື ຕາຕະລາງ DynamoDB — ພ້ອມຄວາມລຶ້ມເຫຼວທີ່ຖືກຈັບ, ບັນທຶກລົງ CloudWatch, ແລະ alarm ຫາກອີເມວຂອງທ່ານ. Deploy ມັນດ້ວຍ Terraform. ມັນສະແດງ Python, ການຄິດແບບ event-driven, ແລະ ຄວາມກວ້າງນອກເໜືອ EC2. ປັບທິດທາງສອງນາທີກ່ອນ: https://www.youtube.com/watch?v=W_VV2Fx32_Y.

ໂຄງການທີ 3 — ການສະແດງ ops ແບບ production (ຕົວສ້າງຄວາມແຕກຕ່າງ). ເອົາໂຄງການທີ 1 ມາປະຕິບັດງານຄືກັບມັນສຳຄັນແທ້: dashboard CloudWatch ຂອງ golden signals; alarm ພ້ອມນະໂຍບາຍ paging ທີ່ມີເອກະສານ; ສາມ runbook; ຂັ້ນຕອນ backup-ແລະ-restore ທີ່ທົດສອບແລ້ວພ້ອມຫຼັກຖານ; ຮອບການເສີມຄວາມແຂງດ້ານຄວາມປອດໄພ (ການກວດສອບ IAM, ບັນທຶກ patching, CloudTrail ເປີດ) ຂຽນສະຫຼຸບໄວ້; ການວິເຄາະຕົ້ນທຶນ (“stack ນີ້ມີຄ່າໃຊ້ຈ່າຍ \$X/ເດືອນ; ນີ້ຄືສາມການປ່ຽນແປງທີ່ຈະຫຼຸດມັນເຄິ່ງໜຶ່ງ”); ແລະ post-mortem ຈາກ game day ໜຶ່ງອັນ. ເກືອບບໍ່ມີຜູ້ສະໝັກ junior ຄົນໃດມີສິ່ງນີ້. ມັນຕອບຄຳຖາມດຽວທີ່ການສຳພາດຖາມແທ້ໆ — “ຄົນນີ້ໄວ້ໃຈໃຫ້ຖື production ໄດ້ບໍ່?” — ດ້ວຍເອກະສານ ແທນຄຳຄຸນນາມ.

Module 12 (ເດືອນທີ 8–12, ຂະໜານກັນ): ຕາຕະລາງ certification

Certification ເປີດຕົວກອງຂອງ recruiter ແລະ ຈັດໂຄງສ້າງການຮຽນຂອງທ່ານ. ເສັ້ນທາງມາດຕະຖານປີ 2026 ສໍາລັບ ຕໍາແໜ່ງນີ້, ພ້ອມເວລາກຽມທີ່ສົມເຫດສົມຜົນຕາມຈັງຫວະຂອງຫຼັກສູດນີ້ — ສັງເກດວ່າແຕ່ລະການສອບຕົກຢູ່ຫຼັງໄລຍະຂອງ ຫຼັກສູດທີ່ສອນເນື້ອຫາຂອງມັນພໍດີ, ຈຶ່ງເປັນເຫດຜົນທີ່ເວລາເຫຼົ່ານີ້ສັນກວ່າຂອງອິນເຕີເນັດ:

ລໍາດັບ	Certification	ມັນພິສູດຫຍັງ	ເວລາກຽມຈາກຈຸດທີ່ທ່ານຈະຢືນຢູ່	ສອບ ເມື່ອ ໃດ
1	AWS Certified Cloud Practitioner (CLF-C02)	ຄໍາສັບ cloud, ການຄິດເງິນ, shared responsibility	3–4 ອາທິດ (ຄູ່ມືຕ່າງໆບອກຄືກັນສໍາລັບມືໃໝ່; ໄລຍະທີ 0–1 ກວມ ສ່ວນໃຫຍ່ຂອງມັນ)	~ເດືອນ ທີ 4
2	AWS Solutions Architect Associate (SAA-C03)	ການອອກແບບ ສະຖາປັດຕະຍະກຳ AWS ຕົວຈິງ — ໃບຢັ້ງຢືນທີ່ JD ໃຊ້ ກັນຕອງແທ້ໆ	6–8 ອາທິດຂອງການກຽມແບບເນັ້ນ (ການປະເມີນມາດຕະຖານວົງການ ຫຼັງ CCP)	ເດືອນ ທີ 8–9
3	HashiCorp Terraform Associate (003)	ຄວາມຄ່ອງ IaC, ຢັ້ງຢືນແລ້ວ	2–3 ອາທິດ — ຫຼັງ Module 7 ແລະ ໂຄງການທີ 1 ທ່ານສ່ວນໃຫຍ່ແມ່ນ ທວນຄືນ (ການກຽມທາງການ: https://developer.hashicorp.com/terraform/tutorials/certification-003)	ເດືອນ ທີ 10–11

ອັນທີສິ້ເປັນທາງເລືອກ, ຖ້າເລັ່ງຕໍາແໜ່ງສາຍ ops: **AWS SysOps Administrator / CloudOps Associate** — ເນື້ອຫາ ຂອງມັນຄືໄລຍະທີ 3 ຕາມໂຕອັກສອນ. ສໍາລັບ CLF-C02, ຄອສເຕັມຂອງ freeCodeCamp (<https://www.youtube.com/watch?v=7HKot-brXFE>, ສະບັບຄລາສສິກ <https://www.youtube.com/watch?v=NhDYbskXRgc>) ບວກຂໍ້ສອບຝຶກຫັດ ຄືເສັ້ນທາງຟຣີທີ່ຄົນຢ່າງຫຼາຍ; ສໍາລັບ SAA-C03, ຈັບຄູ່ຄອສກັບຂໍ້ສອບຝຶກຈັບເວລາ ຫຼາຍໆຊຸດ — ຂໍ້ສອບເປັນແບບ scenario ແລະ ຄວາມອຶດຫິນສໍາຄັນ.

Module 13 (ເດືອນທີ 11–12): Résumé, ການສໍາພາດ, ແລະ ສືບຄໍາຖາມ

Résumé: ໜຶ່ງໜ້າ. ນໍາໜ້າດ້ວຍຫັກສະ (AWS, Terraform, Python/Bash, Docker, CI/CD, CloudWatch — ສະທ້ອນຄໍາ ຂອງ JD ເອງ; ຕົວກອງອັດຕະໂນມັດຈັບຄູ່ keyword) ແລະ ສາມໂຄງການ, ແຕ່ລະອັນເປັນສອງຂໍ້ໃນຮູບ *ເຮັດ X ດ້ວຍ Y ບັນລຸ Z*: “Deployed a three-tier web app on AWS with Terraform and GitHub Actions; zero-downtime deploys via ALB + Auto Scaling.” ລາຍການ certification ພ້ອມວັນທີ. ໃສ່ລິ້ງ GitHub — ແລ້ວສົມມຸດວ່າເຂົາຈະເປີດມັນແທ້ໆ, ເພາະ ຄົນເກັ່ງໆເປີດ, ແລະ ຂອງທ່ານຕອນນີ້ໃຫ້ລາງວັນແກ່ການເຂົ້າຢາມ.

ການກຽມສໍາພາດ: ສາມລົດຊາດໃຫ້ຝຶກຊ້ອມ — ຄໍາຖາມຄວາມຮູ້ (ຕາຕະລາງຄໍາສັບຂອງແຕ່ລະໄລຍະ), scenario (lab ທີ່ ທ່ານໄດ້ເຮັດແທ້), ແລະ behavioral (ເລື່ອງເລົ່າຈາກປຶ້ມບັນທຶກຂອງທ່ານ, ເລົ່າໃນຮູບ **STAR**: Situation, Task, Action,

Result). ຝຶກ ອອກສຽງ, ທຸກມື້, ສອງອາທິດ — ດີສຸດກັບ AI interviewer, ແບບບໍ່ປານີ. ສືບຄໍາຖາມຂ້າງລຸ່ມກວມຂັດລາສສິກ; ຄໍາຕອບຕົວຢ່າງກະທັດຮັດໂດຍເຈດຕະນາ — ຂະຫຍາຍແຕ່ລະອັນດ້ວຍລາຍລະອຽດ lab ຂອງທ່ານເອງ, ເພາະ “...ແລະ ຕອນ ຂ້ອຍສ້າງອັນນີ້, ສິ່ງທີ່ເກີດຂຶ້ນແທ້ຄື...” ຄືປະໂຫຍກທີ່ແຍກທ່ານອອກຈາກຜູ້ສະໝັກທີ່ພຽງອ່ານ.

ສືບຄໍາຖາມ, ຜ່ອມຄໍາຕອບທີ່ແຂງແຮງ:

1. **“Explain the difference between a public and a private subnet.” (ອະທິບາຍຄວາມແຕກຕ່າງລະຫວ່າງ public subnet ກັບ private subnet.)** Route table ຂອງ public subnet ມີເສັ້ນທາງໄປຫາ internet gateway, ຊັບພະຍາກອນຂອງມັນຈຶ່ງຖືກເຂົ້າເຖິງໄດ້ຈາກອິນເຕີເນັດ; private subnet ບໍ່ມີເສັ້ນທາງແບບນັ້ນ. ຊັ້ນ web ໄປ public, database ໄປ private, ແລະ ການເຂົ້າເຖິງແບບ admin ໄປຫາເຄື່ອງ private ຜ່ານ bastion ຫຼື SSM. ໃນ lab ຂອງ ຂ້ອຍ ຂ້ອຍພິສູດວ່າ instance private ເຂົ້າເຖິງບໍ່ໄດ້ຈາກໂນດບຸກຂອງຂ້ອຍ ແຕ່ເຂົ້າເຖິງໄດ້ຈາກຊັ້ນ public — ເສັ້ນທາງ ທີ່ຫາຍໄປນັ້ນເອງຄືຄວາມປອດໄພ.
2. **“What is IAM, and what is least privilege?” (IAM ແມ່ນຫຍັງ, ແລະ least privilege ແມ່ນຫຍັງ?)** IAM ຄວບຄຸມວ່າຕົວຕົນໃດປະຕິບັດການກະທຳໃດຕໍ່ຊັບພະຍາກອນໃດໄດ້, ຜ່ານ user, group, role, ແລະ policy ແບບ JSON. Least privilege ໝາຍເຖິງການໃຫ້ໜ້ອຍທີ່ສຸດທີ່ເຮັດວຽກໄດ້ — user ຂອງ audit script ຂອງຂ້ອຍ ມີສິດອ່ານ ຢ່າງດຽວຕໍ່ບໍລິການດຽວແທ້ໆ, ແລະ EC2 instance ຂອງຂ້ອຍໃຊ້ role ແທນ key ທີ່ເກັບໄວ້, ຈຶ່ງບໍ່ມີ credential ອາຍຸ ຍາວໃຫ້ຮົວ.
3. **“An EC2 web server stopped responding. Walk me through your troubleshooting.” (EC2 web server ຢຸດຕອບສະໜອງ. ພາຂ້ອຍຜ່ານການແກ້ໄຂບັນຫາຂອງເຈົ້າ.)** ກ່ອນອື່ນກຳນົດຂອບເຂດ: server ດຽວ ຫຼື ທັງໝົດ — ກວດ dashboard CloudWatch ແລະ alarm ໃດໆ. ຈາກນັ້ນໄລ່ຊັ້ນຕາມລຳດັບ: instance status check; security group — 80/443 ເປີດແທ້ບໍ່; SSH ເຂົ້າໄປ — nginx ແລ່ນຢູ່ບໍ່ (systemctl status), ດິສເຕັມບໍ່ (df - h), CPU ຕັນບໍ່ (top); ຈາກນັ້ນ log (tail error log). Mitigate ກ່ອນ — restart service ຫຼື ປ່ຽນ instance — ແລ້ວຫາສາເຫດຮາກເຫງົ້າຫຼັງເຮົາໝັ້ນຄົງ. ນັ້ນຄືລຳດັບທີ່ຂ້ອຍຝຶກຊ້ອມໃນ runbook ຂອງຂ້ອຍ.
4. **“Why Terraform instead of clicking in the console?” (ເປັນຫຍັງ Terraform ແທນການຄລິກໃນ console?)** Console ບໍ່ປະຢັນທີ່ກະທຳທຸກໆໄດ້ ແລະ ສ້າງຫຍັງຄືນບໍ່ໄດ້. Terraform ເປັນ declarative ແລະ ມີເວີຊັນ: ການປ່ຽນແປງຜ່ານ pull request, config ດຽວກັນສ້າງສະພາບແວດລ້ອມຄືກັນທຸກປະການ, ການປ່ຽນທີ່ຜິດ roll back ດ້ວຍການ revert commit, ແລະ plan ສະແດງ diff ແທ້ໆກ່ອນສົ່ງໃດຈະເກີດ. Pipeline ຂອງຂ້ອຍແລ່ນ plan ໃນທຸກ PR ເພື່ອໃຫ້ diff ເປັນສ່ວນໜຶ່ງຂອງການທົບທວນ.
5. **“What’s the difference between a container and a virtual machine?” (Container ກັບ virtual machine ຕ່າງກັນແນວໃດ?)** VM ຈຳລອງ hardware ແລະ ແບກ OS ເຕັມຊຸດ, ໃຊ້ເວລາ boot ເປັນນາທີ; container ແບ່ງໃຊ້ kernel ຂອງ host ແລະ ຫຸ້ມຫໍ່ພຽງແອັບກັບ dependencies ຂອງມັນ, ເລີ່ມຕົ້ນໃນປະມານໜຶ່ງວິນາທີ. Container ໃຫ້ພຶດຕິກຳຄືກັນທຸກສະພາບແວດລ້ອມ — image ດຽວກັນແລ່ນເທິງໂນດບຸກຂອງຂ້ອຍ ແລະ ເທິງ EC2. VM ຍັງສຳຄັນສຳລັບ isolation ແລະ ການແລ່ນ workload ທີ່ບໍ່ແມ່ນ Linux.
6. **“How would you reduce our AWS bill?” (ເຈົ້າຈະຫຼຸດໃບບິນ AWS ຂອງເຮົາແນວໃດ?)** ຕາມລຳດັບຄວາມ ພະຍາຍາມ: ຊອກຫາສິ່ງທີ່ແລ່ນຢູ່ໂດຍບໍ່ຄວນ — instance idle, volume ບໍ່ຕິດກັບຫຍັງ, snapshot ເກົ່າ; Cost Explorer ແລະ tag ເຮັດໃຫ້ມັນເບິ່ງເຫັນໄດ້. ຈາກນັ້ນ rightsize ຕາມປະຫວັດ CloudWatch. ຈາກນັ້ນຕັ້ງເວລາປິດ dev environment ນອກໂມງເຮັດວຽກ. ຈາກນັ້ນ lifecycle ຂໍ້ມູນ S3 ເກົ່າໄປຂັ້ນຖືກກວ່າ. ຈາກນັ້ນຜູກມັດ workload ຄົງທີ່ໃສ່ Savings Plans ແລະ ວາງ batch ທີ່ຂັດຈັ່ງຫວະໄດ້ໃສ່ Spot. ແລະ ເຝົ້າເບິ່ງ egress — ຂໍ້ມູນຂາອອກຄືແຖວ ແປກໃຈຄລາສສິກ.
7. **“What happens when you type a URL and press Enter?” (ເກີດຫຍັງຂຶ້ນເມື່ອເຈົ້າພິມ URL ແລ້ວກົດ Enter?)** DNS ແປຊີເປັນ IP; browser ເປີດການເຊື່ອມຕໍ່ TCP ຫາ port 443 ແລ້ວເຈລະຈາ TLS; ມັນສົ່ງ HTTP GET; ຝັ່ງນັ້ນມັນແຕະ load balancer, ຊົງສົ່ງຕໍ່ຫາ instance ທີ່ແຂງແຮງໃນ public subnet; ແອັບອາດ query

database ໃນ private subnet; ຄໍາຕອບກັບຄືນຜ່ານ status code — 200 ຖ້າແຂງແຮງ — ແລ້ວ browser ສະແດງຜົນມັນ. ຂ້ອຍເຈາະລົງທຸກຂັ້ນໄດ້, ແລະ ຂ້ອຍໄດ້ສ້າງແຕ່ລະອັນມາແລ້ວ.

8. “A teammate’s change broke production. What happens next?” (ການປ່ຽນແປງຂອງເພື່ອນຮ່ວມທີມທໍາລາຍ production. ຈະເກີດຫຍັງຕໍ່ໄປ?) Mitigate ກ່ອນ — roll back ຜ່ານ pipeline; ການຟື້ນຕົວຊະນະການວິນິດໄສໃນຊ່ວງເວລານັ້ນ. ຈາກນັ້ນ post-mortem ແບບບໍ່ໂຫດກັກ: ເກີດຫຍັງຂຶ້ນ, ເປັນຫຍັງລະບົບຈຶ່ງອະນຸຍາດມັນ, ແລະ ຮາວກັນອັນໃດຫາຍໄປ — test, ການທົບທວນ plan, alarm. Blameless ສໍາຄັນໃນທາງປະຕິບັດ: ຄົນທີ່ຢ້ານການລົງໂທດຈະເຊື່ອງຂໍ້ມູນ, ແລະ ຂໍ້ມູນທີ່ຖືກເຊື່ອງກໍໃຫ້ເກີດການເກີດຊ້າ.
9. “What is the shared responsibility model?” (Shared responsibility model ແມ່ນຫຍັງ?) AWS ຮັກສາຄວາມປອດໄພຂອງ cloud — data center, hardware, hypervisor; ລູກຄ້າຮັກສາຄວາມປອດໄພຂອງສິ່ງທີ່ຢູ່ໃນມັນ — ຂໍ້ມູນ, IAM, ກົດເຄືອຂ່າຍ, patching, ການຕັ້ງຄ່າ encryption. ການເຈາະ cloud ຕົວຈິງສ່ວນໃຫຍ່ຢູ່ຝັ່ງລູກຄ້າ, ການຕັ້ງຄ່າຜິດເຊັ່ນ bucket public ຫຼື key ຮົວ. ດັ່ງນັ້ນ ‘AWS ປອດໄພ’ ກັບ ‘ພວກເຮົາປອດໄພເທິງ AWS’ ຄືສອງປະໂຫຍກຕ່າງກັນ, ແລະ ອັນທີສອງຄືວຽກຂອງຂ້ອຍ.
10. “Tell me about a problem you debugged.” (ເລົ່າໃຫ້ຝັ່ງກ່ຽວກັບບັນຫາທີ່ເຈົ້າ debug.) (ຂອງທ່ານເອງ, ຈາກປຶ້ມບັນທຶກ, ໃນຮູບ STAR. ໂຄງກະດູກຕົວຢ່າງ:) ໃນ game day ດ້ານ monitoring ຂອງຂ້ອຍ, alarm ຂອງຂ້ອຍດັງຍ້ອນ CPU saturation (S). ຂ້ອຍຕ້ອງຊອກຫາ ແລະ ຢຸດສາເຫດ ກ່ອນ ‘ຜູ້ໃຊ້’ ຈະສັງເກດ (T). Metric ສະແດງຈຸດເລີ່ມຂອງ spike; Logs Insights ຜູກມັນກັບ process ທີ່ແລ່ນເກີນຄວບຄຸມ; ຂ້ອຍ SSH ເຂົ້າໄປ, ຢືນຢັນດ້ວຍ top, ຂ້າພວກມັນ, ແລ້ວເບິ່ງການຟື້ນຕົວເທິງ dashboard (A). ແກ້ໄຂໄດ້ໃນສິບເອັດນາທີ, ແລະ post-mortem ອອກຜົນເປັນ alarm ນັບ process ທີ່ຂ້ອຍຈັດຕັ້ງຕໍ່ມາໃນ Terraform (R).

ກົນໄກການຊອກວຽກ: ສະໝັກຕັ້ງແຕ່ເດືອນທີ 11 — ຫຼັງ SAA, ຢ່າລໍຖ້າຄວາມ “ພ້ອມ,” ເພາະການສໍາພາດຄືການຝຶກ. ຊື່ຕໍາແໜ່ງເປົ້າໝາຍ: cloud engineer (junior/associate), cloud support engineer, cloud operations engineer, junior DevOps engineer, AWS support associate. ທຸກການປະຕິເສດທີ່ມີການສໍາພາດ ຄືບົດຮຽນຟຣີ; ຈິດບັນທຶກວ່າເຂົາຖາມຫຍັງ.

ຄໍາສັບ:

ຄໍາສັບ	ຄວາມໝາຍ
Portfolio	ຫຼັກຖານສາທາລະນະ ມີເອກະສານ ວ່າທ່ານສ້າງໄດ້ — ສໍາລັບວິຊາຊີບນີ້, GitHub repo ພ້ອມແຜນວາດ ແລະ README.
Application Load Balancer (ALB)	ຕົວກະຈາຍ traffic ຂອງ AWS: ແຈກຄໍາຮ້ອງຂໍໃຫ້ instance ທີ່ແຂງແຮງ, ຕັດອັນທີ່ບໍ່ສະບາຍອອກ.
Auto Scaling Group	ຮັກສາ N instance ໃຫ້ມີຊີວິດ ແລະ ປັບ N ຕາມໂຫຼດ — self-healing ແລະ elasticity ໃນອັນດຽວ.
Lambda / serverless	ໂຄດທີ່ AWS ແລ່ນຕໍ່ event, ຄິດເງິນຕໍ່ການເອີ້ນ — ບໍ່ມີ server ໃຫ້ຄຸ້ມຄອງເລີຍ.
DynamoDB	ຕາຕະລາງ NoSQL ແບບ serverless ຂອງ AWS — ຈັບຄູ່ກັບ Lambda ຢ່າງທໍາມະຊາດ.

ຄຳສັບ	ຄວາມໝາຍ
STAR	Situation, Task, Action, Result — ຮູບຮ່າງຂອງເລື່ອງເລົ່າສຳພາດທີ່ດີ.
ATS	Applicant tracking system — ຕົວກອງ keyword ທີ່ résumé ໜຶ່ງໜ້າຂອງທ່ານຕ້ອງຜ່ານ.
CLF-C02 / SAA-C03 / Terraform Associate 003	ສາມການສອບຂອງທ່ານ: ຄຳສັບ, ສະຖາປັດຕະຍະກຳ, IaC.
SysOps / CloudOps Associate	AWS associate ສາຍ ops ທີ່ເປັນທາງເລືອກ — ໄລຍະທີ 3, ໃນຮູບຂັ້ນສອບ.

Milestone — ຈົບຫຼັກສູດ: ສາມ repo ທີ່ຄົນແປກໜ້າເຂົ້າໃຈໄດ້, ສາມ certification ທີ່ນັດສອບ ຫຼື ຜ່ານແລ້ວ, ສືບຄຳຕອບຝຶກຊ້ອມອອກສຽງແລ້ວ, ໃບສະໝັກສົ່ງແລ້ວ. ທ່ານບໍ່ແມ່ນ “ຄົນຫວັງເຂົ້າສາຍ cloud.” ທ່ານຄິວິສະວະກອນ cloud ລະດັບ junior ພ້ອມຫຼັກຖານ, ກຳລັງສຳພາດຢູ່.

ຫຼັກສູດຫຍໍ້ໜຶ່ງໜ້າ

ເມື່ອໃດ	ຈຸດເນັ້ນ	ຫຼັກຖານລົງມືເຮັດ	ຫຼັກຖານພາຍນອກ
ອາທິດທີ 1–2	ຄອມພິວເຕີເຮັດວຽກແນວໃດ; Linux, terminal, permissions, SSH	ການຈັດການໄຟລ໌ດ້ວຍ script; shell script ຫ້າອິດ	—
ອາທິດທີ 3–4	ເຄືອຂ່າຍ: IP, DNS, port, HTTP, firewall; ບັນຊີ AWS	ບັນຊີ + MFA + budget \$0	—
ອາທິດທີ 5–6	Lab 1: EC2	Web server ເທິງອິນເຕີເນັດ, ຮື້ຖອນແລ້ວ	—
ອາທິດທີ 7	Lab 2: S3 + CLI	ເວັບໄຊ static; ຄວາມຄ່ອງ CLI	—
ອາທິດທີ 8–9	Lab 3: VPC	ເຄືອຂ່າຍ public/private, ການພິສູດ bastion	—
ອາທິດທີ 10	Lab 4: RDS	Database private, snapshot, ການຮື້ຖອນ	—
ອາທິດທີ 11–12	Lab 5: IAM; ການສ້າງດ່ານ ທົດສອບໄລຍະ 1 ຄົ້ນ	Stack ເຕັມຈາກຄວາມຈຳ, <3 ຊົ່ວໂມງ	ນັດສອບ CCP

ເມື່ອໃດ	ຈຸດເນັ້ນ	ຫຼັກຖານລົງມືເຮັດ	ຫຼັກຖານພາຍນອກ
ອາທິດທີ 13–15	Bash, Python/boto3, Git	backup.sh, audit.py, ປະຫວັດ repo	ສອບ CCP (~ເດືອນທີ 4)
ອາທິດທີ 16–17	Terraform	Stack ເປັນໂຄດ, plan/apply/destroy	—
ອາທິດທີ 18–20	CI/CD, Docker, ຄວາມຮູ້ K8s	Pipeline ຂງວ; image ໃນ registry	—
ອາທິດທີ 21–24	CloudWatch, log, incident, runbook	Alarm ດັ່ງ + post-mortem ຈາກ game day	—
ອາທິດທີ 25–28	Security ops; ການຄຸ້ມຄອງ ຕົ້ນທຶນ	ລາຍງານກວດສອບຕົ້ນເອງ; ການທົດສອບ restore	—
ເດືອນທີ 8–10	ໂຄງການ Portfolio 1–3	ສາມ repo ມີເອກະສານ	SAA-C03 (ເດືອນທີ 8–9)
ເດືອນທີ 10–11	ການກຽມ Terraform Associate	—	Terraform Associate
ເດືອນທີ 11–12	Résumé, ການສຳພາດ, ການສະໝັກ	ສືບຄຳຕອບ, ອອກສຽງ	ຂໍສະເໜີວຽກທຳອິດ

ຄຳສັ່ງທ້າຍຈາກຄູຂອງທ່ານ. ສືບສອງເດືອນສັ້ນສຳລັບການປ່ຽນອາຊີບ ແລະ ຍາວສຳລັບນິໄສປະຈຳວັນ, ສະນັ້ນນີ້ຄືຂໍ້ຕົກລົງ ແບບຊື່ສັດ: ຄົນທີ່ຮຽນຈົບຫຼັກສູດນີ້ ບໍ່ແມ່ນຄົນສະຫຼາດທີ່ສຸດ — ແມ່ນຄົນທີ່ພົບຄຳສັ່ງໃນມື້ທີ່ອິດເມື່ອຍນຳ. ທຸກຂໍ້ຄວາມ error ທີ່ທ່ານພົບ ຄືຫຼັກສູດກຳລັງເຮັດວຽກ, ບໍ່ແມ່ນລົມເຫຼວ; ວິສະວະກອນທີ່ໄດ້ທຳລາຍ ແລະ ແກ້ໄຂສິ່ງນ້ອຍໆມາຮ້ອຍອັນ ຄືສິ່ງທີ່ hiring manager ໝາຍເຖິງແທ້ໆດ້ວຍຄຳວ່າ “ປະສົບການ.” ຮັກສາປຶ້ມບັນທຶກ, ຮືຖອນສິ່ງທີ່ທ່ານສ້າງ, ຢ່າອ້າງໃນການສຳພາດ ສິ່ງທີ່ repo ຂອງທ່ານຍືນຍັນບໍ່ໄດ້ — ແລ້ວອີກໜຶ່ງປີຈາກນີ້, ເມື່ອ pager ດັ່ງຕອນຕີສາມ, ທ່ານຈະຮູ້ສຶກສິ່ງທີ່ບໍ່ຄາດຄິດຢູ່ໄ້ adrenaline: ຄວາມສາມາດ. ໄປສ້າງເລີຍ.

Sources

ການຄົ້ນຄວ້າປະກາດຮັບສະໝັກວຽກ (ສິງຫາ 2026): [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](#). ເສັ້ນທາງ certification ແລະ ເວລາກຽມ: [StudyTech — AWS Certification Roadmap 2026](#) · [Cloud Evolvers — Cloud Engineer Roadmap 2026](#) · [HashiCorp — Terraform Associate 003 prep](#). ບໍລິບົດເງິນເດືອນ (ຄ່າກາງສະຫະລັດ ≈ \$104K, ຊ່ວງ \$85K–\$140K): Wiz, ຂ້າງເທິງ. ລິງ YouTube ທັງໝົດ ກວດສອບຜ່ານ metadata ຂອງ YouTube ໃນເວລາຂຽນ.

ຫຼັກສູດ B4LCILC — ເຫຼັ້ມຄູ່ຂອງ “The Cloud Leader Course.”

ບົດຮຽນໃໝ່ — ສະບັບເດືອນສິງຫາ 2026

ໃນເດືອນສິງຫາ 2026, ບົດຮຽນໃໝ່ໄດ້ຖືກເພີ່ມເຂົ້າໃນສາມຫຼັກສູດແບບໂຕ້ຕອບທີ່ b4lcilc.org — ຫຼັກສູດຜູ້ນຳດ້ານ Cloud (Cloud Leader), ວິສະວະກອນ Cloud (Cloud Engineer), ແລະ ສະຖາປະນິກ Cloud (Cloud Architect). ເອກະສານເພີ່ມເຕີມສະບັບນີ້ ນຳສະບັບອ່ານໃຫ້ທັນສະໄໝ ດ້ວຍບົດສະຫຼຸບຫຍໍ້ທີ່ຊື່ສັດຕໍ່ເນື້ອຫາໃໝ່ນັ້ນ, ໃນນັ້ນສຽງດຽວກັນກັບສ່ວນອື່ນຂອງປຶ້ມເຫຼົ່ານີ້.

ຂໍເວົ້າຈັກຄຳກ່ຽວກັບສິ່ງທີ່ທ່ານຖືຢູ່ໃນມື. ໜ້າເຫຼົ່ານີ້ຄືສະບັບອ່ານ: ແນວຄິດ, ຄຳປຽບທຽບ, ແລະ ປະໂຫຍກທີ່ຖືກຄຳແກ່ການເວົ້າອອກສຽງ. ສະບັບເຕັມແບບໂຕ້ຕອບທາງອອນລາຍ ມີເນື້ອຫາຫຼາຍກວ່ານີ້ຫຼາຍ — ທຸກບົດຮຽນໃໝ່ມາພ້ອມກັບແບບທົດສອບ (quiz) ທີ່ກວດຄວາມເຂົ້າໃຈຂອງທ່ານກ່ອນຈະໃຫ້ກ້າວຕໍ່ໄປ, ຫ້ອງທົດລອງແບບລົງມືເຮັດຈິງ (ລວມທັງ terminal ຝຶກຫັດ ທີ່ເຮັດວຽກຄືກັບຂອງແທ້ ແຕ່ບໍ່ສາມາດເສຍເງິນໄດ້), ວິດີໂອ, ບົດຝຶກຄຳສັບ, ແລະ — ສຳລັບເນື້ອຫາ capstone (ບົດຝຶກລວມຍອດ) — ບົດຝຶກອອກແບບແບບມີຄຳແນະນຳ ພ້ອມລາຍການກວດຄວາມຕ້ອງການ ແລະ ຄຳຕອບຕົວຢ່າງທີ່ເຮັດໃຫ້ເບິ່ງ. ຈົ່ງອ່ານບົດສະຫຼຸບເຫຼົ່ານີ້ ເພື່ອໃຫ້ຮູບຮ່າງຂອງແຕ່ລະແນວຄິດເຂົ້າໄປຢູ່ໃນຫົວຂອງທ່ານ; ຈາກນັ້ນ, ເມື່ອມີໂອກາດ, ໃຫ້ນັ່ງເຮັດສະບັບໂຕ້ຕອບ ເພື່ອໃຫ້ຄວາມຮູ້ຕິດແໜ້ນ.

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

ສຳລັບຫຼັກສູດວິສະວະກອນ Cloud

Serverless ແທ້ຈິງແລ້ວແມ່ນຫຍັງ

ທຸກຢ່າງທີ່ທ່ານໄດ້ເຮັດໃຫ້ເປັນອັດຕະໂນມັດມາຮອດຕອນນີ້ ຍັງສົມມຸດວ່າມີເຄື່ອງໝົງໜ່ວຍ: cron job ໃນຫ້ອງທົດລອງຂອງ module ນີ້ ແລ່ນມືລະສອງນາທີ ເທິງ virtual machine ທີ່ທ່ານຈ່າຍຄ່າຕະຫຼອດເດືອນ, ຕ້ອງ patch, ຕ້ອງເປົ່າຕິດຕາມ, ແລະ ມີໜຶ່ງຈະຕ້ອງ debug ຕອນຕີສາມ. Serverless ຄືຄຳຕອບຂອງ cloud ຕໍ່ຄວາມບໍ່ສອດຄ່ອງນັ້ນ. Server ຍັງມີຢູ່ — ຊິມີແມ່ນເລື່ອງການຕະຫຼາດ — ແຕ່ພວກມັນເປັນບັນຫາຂອງຜູ້ໃຫ້ບໍລິການ. ທ່ານມອບ function ໜຶ່ງໃຫ້, ຜູ້ໃຫ້ບໍລິການແລ່ນມັນເມື່ອມີສິ່ງໃດມາກະຕຸ້ນ (trigger), ແລະ ທ່ານຈ່າຍຕໍ່ການເອີ້ນໃຊ້ (invocation), ຕາມ millisecond. ໂມເດວນີ້ເອີ້ນວ່າ **FaaS — Function as a Service (function ໃນຖານະບໍລິການ)**: Lambda ເທິງ AWS, Azure Functions ເທິງ Azure. ແນວຄິດຖ່າຍທອດຫາກັນໄດ້ຊື່ຕື່ມ.

ໂຄງສ້າງທັງໝົດຈັບໄດ້ໃນມືດຽວ. handler ຄື function ທີ່ platform ເອີ້ນໃຊ້, ແລະ ມັນຮັບ event — ຄຳບັນຍາຍໃນຮູບແບບ JSON (JavaScript Object Notation) ຂອງສິ່ງທີ່ເກີດຂຶ້ນ: ໄຟລໝົງມາຮອດ bucket, HTTP request ມາຮອດ, ຕາຕະລາງ

ເວລາເຖິງກຳນົດ. ຈຶ່ງນັ່ງຊຶ້ງອັນສຸດທ້າຍ: “ຕາຕະລາງເວລາເຖິງກຳນົດ” ຄືສິ່ງທີ່ cron ເຮັດແທ້ໆ — ພຽງແຕ່ Lambda ທີ່ແລ່ນຕາມຕາຕະລາງເວລາ ບໍ່ຕ້ອງມີເຄື່ອງເປີດຄ້າງໄວ້ລະຫວ່າງຮອບ. ການປ່ຽນສະຄຣິບປະຕິບັດງານອັນໜຶ່ງຂອງ module ນີ້ ຄື ໂຄງການ serverless ທຳອິດແບບຄລາສສິກ.

ເສດຖະສາດຄິເຫດຜົນທີ່ໂມເດວນີມີຢູ່, ແລະ ມັນຕັດທັງສອງທາງ. Free tier ຂອງ AWS ລວມການເອີ້ນໃຊ້ Lambda ໜຶ່ງລ້ານຄັ້ງຕໍ່ເດືອນ, ຕະຫຼອດໄປ — function ໃນຫ້ອງທົດລອງຂອງທ່ານ ສາມາດແລ່ນທຸກໆຄັ້ງທີ່ຕະຫຼອດປີ ໂດຍບໍ່ເສຍຄ່າຫຍັງ, ເຮັດໃຫ້ serverless ເປັນບ້ານທີ່ຖືກທີ່ສຸດ ສຳລັບວຽກກາວເຊືອມດ້ານປະຕິບັດງານ. ແຕ່ລາຄາຕໍ່ millisecond ທີ່ເປັນເສດເລກນ້ອຍໆຢູ່ປະລິມານຕ່ຳ ຈະກາຍເປັນເງິນແທ້ຢູ່ປະລິມານສູງ; function ທີ່ແລ່ນຕະຫຼອດເວລາ ມັກຈະຖືກກວ່າ ຖ້າເປັນ container ຫຼື VM. ຫັກສະຂອງມືອາຊີບ ຄືການຮູ້ວ່າ workload ໜຶ່ງຢູ່ຝາກໃດຂອງເສັ້ນນັ້ນ.

ສອງຂັດວນລະວັງທີ່ຊື່ສັດ. ທຳອິດ, **cold start** (ການເລີ່ມເຄື່ອງເຢັນ): function ທີ່ບໍ່ໄດ້ແລ່ນມາໄລຍະໜຶ່ງ ຈະໃຊ້ເວລາສອງສາມຮ້ອຍ millisecond ເພື່ອຕົ້ນ — ບໍ່ສຳຄັນສຳລັບລາຍງານປະຈຳຄືນ, ແຕ່ເປັນຂໍ້ມົນຈາລະນາໃນການອອກແບບ ສຳລັບ API ທີ່ອ່ອນໄຫວຕໍ່ latency. ທີ່ສອງ, function ຍັງຄົງແລ່ນໃນນາມຕົວຕົນໜຶ່ງ: Lambda ຮັບບົດບາດ (role) ຂອງ IAM (Identity and Access Management — ການຄຸ້ມຄອງຕົວຕົນ ແລະ ການເຂົ້າເຖິງ) ຄືກັນກັບ EC2 instance ທຸກປະການ, ແລະ function ທີ່ໄດ້ຮັບສິດເກີນຄວາມຈຳເປັນ ຄືຄວາມຜິດພາດດ້ານຄວາມປອດໄພອັນດຽວກັນ ໃນໜ້າທີ່ນ້ອຍກວ່າ.
