KURS NË SHQIP · NGA BAZAT NË NIVEL TË AVANCUAR

Cloud Computing

Cloud Computing — nga themelet te arkitektura dhe operimi i avancuar.

Kurs tekstual në shqip me 40 module, 40 mësime, 80 materiale teori/laborator, 40 kuize modulesh dhe një provim final: 144 pyetje me shpjegime. Përfshin manualin e studentit, aplikacion lokal të ekzekutueshëm, shembuj Docker/Kubernetes/IaC, fletë pune, projekte dhe rubrikë vlerësimi.

Mbulohen modelet cloud, AWS/Azure/Google Cloud, identiteti, rrjetet, compute/storage/databases, serverless, queues, containers/Kubernetes, infrastruktura si kod, CI/CD, observability, SRE, siguria, FinOps, recovery, migrimi, hybrid/multicloud, data dhe AI workloads.

Ngarkesë orientuese 200–280 orë, përfshirë praktikën dhe projektin. Kërkohen përdorimi i terminalit, bazat e rrjeteve dhe skedarëve; shembujt e programimit shpjegohen dhe përdorin Python. Kursi nis me konceptet dhe arrin te kompromiset e sistemeve të shpërndara.

Praktika bazë ekzekutohet lokalisht pa llogari cloud dhe pa krijuar burime me pagesë. Laboratorët e ofruesve janë të drejtuar ose analitikë; zbatimi opsional në sandbox kërkon kontroll të identitetit, rajonit, buxhetit dhe pastrimit. Çmimet, kuotat dhe SLA-të verifikohen në momentin e zbatimit. Emrat teknikë dhe komandat ruhen në origjinal.

Kuizet e moduleve: prag 75% (3/3). Finali: 80% (20/24). Projekti: 80/100 plus kontrollet kritike. Nuk është certifikim i një ofruesi. Materialet importohen si draft; burimet bashkëngjiten manualisht. Një emulator lokal nuk provon sjelljen e një shërbimi managed ose të shumë rajoneve.

Si ta përdorësh kursin

Cloud Computing — Informatika në re trajton koncepte, zbatim dhe operim. Ndiq modulet në rend nëse po fillon. Për një student me përvojë, kuizet ndihmojnë identifikimin e boshllëqeve, por kalimi i tyre nuk zëvendëson laboratorët.

Parakushtet dhe ngarkesa

Duhet të njohësh terminalin, skedarët, bazat IP/DNS/HTTP dhe përdorimin e një kompjuteri. Kursi shpjegon mekanizmat cloud nga fillimi dhe përdor Python për një aplikacion të vogël; nuk kërkon përvojë të avancuar programimi. Nëse IP, subnet, DNS ose TLS janë të panjohura, rishiko kursin e rrjeteve përpara moduleve 7–10.

- Modulet 1–10: themelet, përgjegjësitë, organizimi, IAM dhe rrjeti. 35–45 orë.
- Modulet 11–20: compute, storage, databaza, queues, serverless dhe containers. 45–60 orë.
- Modulet 21–30: Kubernetes, IaC, delivery, platforma, observability dhe siguria. 55–75 orë.

- Modulet 31–38: FinOps, DR, migrim, multicloud, data, AI dhe modele të avancuara. 35–50 orë.
- Modulet 39–40: incidente, projekt, riprovim dhe mbrojtje. 30–50 orë.

Gjithsej 200–280 orë orientuese. Leximi i manualit është vetëm një pjesë; koha përfshin ndërtim, dokumentacion, zgjidhje gabimesh, prova dhe projekt. Nuk është garanci për rol pune ose certifikim.

Praktika lokale

Laboratori kryesor kërkon vetëm Python 3.10+ me sqlite3. Lexo `local-lab/README.sq.md`, pastaj ekzekuto testet. Nuk krijohen burime cloud dhe nuk duhen kredenciale. Aplikacioni ka API, idempotency, transaction, worker, dështim të simuluar, metrics JSON dhe backup/restore. Ai nuk ka autentikim/TLS dhe lidhet si default në localhost.

Docker/Compose dhe Kubernetes janë zgjerime opsionale që duhen instaluar veçmas. Kubernetes kërkon cluster testimi me CNI që zbaton NetworkPolicy për izolimin. IaC lokal përdor `terraform_data`, pa provider cloud dhe pa burime me pagesë. Modeli i kostove përdor numra hipotetikë. Emrat dhe komandat teknike ruhen në origjinal për përputhje me dokumentacionin.

Laboratorët e ofruesve

AWS, Azure dhe Google Cloud përdoren për të lidhur konceptet me produkte konkrete, jo si shërbime identike. Një tabelë ekuivalencash ndihmon orientimin; versioni, quota, rajoni, consistency dhe faturimi kontrollohen në dokumentacionin aktual. Free tier nuk do të thotë që çdo shërbim është pa kosto dhe budget alert nuk është hard cap.

Nëse përdor sandbox ekzistues, verifiko identitetin e llogarisë/subscription/project, rajonin dhe lejet përpara çdo veprimi. Krijo vetëm burimet e laboratorit dhe mbaj inventar/pastrim. Ky kurs nuk kërkon që të hapësh llogari ose të bësh blerje për ta ndjekur. Për detyrat pa mjedis shëno **analizë, pa ekzekutim** dhe plotëso praktikën kur të kesh mjetet.

Evidenca e punës

Për çdo test ruaj: objektivin, mjedisin/versionet, input, pritshmërinë, rezultatin, komandën, kohën dhe shpjegimin. Përfshi teste negative, kthimin pas dështimit dhe kufizimin e simulatorit. Mos dorëzo secrets, tokens, të dhëna personale ose credentials. Një screenshot i konsolës nuk provon vetë që rrjedha e biznesit punon.

Vlerësimi

40 kuize modulesh kanë nga 3 pyetje. Pragu 75% kërkon 3/3; është kontroll i shkurtër i njohurive. Finali ka 24 pyetje, 60 minuta dhe prag 80%, pra 20/24. Projekti praktik kërkon minimum 80/100 dhe kalim të kontrolleve kritike për idempotency, izolim, secrets dhe restore. Skedarët e mësuesit me përgjigje nuk përfshihen në paketën e studentit.

Kufijtë

Një host me dy procese nuk është dy availability zones. SQLite nuk provon motor SQL managed ose exactly-once efekt të jashtëm. Model kostosh nuk është faturë aktuale. YAML i ruajtur nuk provon enforcement në Kubernetes. Projektimi i arkitekturës nuk është deploy i ekzekutuar. Këto dallime duhen ruajtur në çdo raport dhe mbrojtje teknike.

1. Themelet e Cloud Computing

Modelet e shërbimit, elasticiteti dhe përgjegjësia

Teori: Modelet e shërbimit, elasticiteti dhe përgjegjësia

Objektivi: të dallosh çfarë blen nga një shërbim cloud dhe çfarë vazhdon të menaxhosh vetë.

Cloud computing jep burime informatike sipas kërkesës përmes shërbimeve me automatizim, ndarje burimesh dhe matje përdorimi. Një VM e largët është vetëm një nga format; vlera e cloud-it lidhet edhe me API, provisioning, shërbimet managed dhe mundësinë e ndryshimit të kapacitetit. Public cloud përdor infrastrukturë të ofruesit; private cloud operohet për një organizatë; hybrid lidh mjedise me kufij përgjegjësie dhe integrim real. Multicloud do të thotë përdorim i disa ofruesve, jo domosdoshmërisht portabilitet automatik.

IaaS jep kontroll mbi VM, sistemin operativ dhe aplikacionin. PaaS merr përsipër më shumë nga runtime/platforma. SaaS ofron aplikacion të përfunduar me konfigurim dhe detyrime të klientit mbi identitetet, të dhënat dhe përdorimin. Serverless heq menaxhimin e drejtpërdrejtë të serverëve nga përdoruesi, por jo kufizimet e ekzekutimit, kostot dhe operimin e aplikacionit.

Scalability është aftësia për të mbajtur ngarkesë më të madhe; elasticity përshtat kapacitetin ndaj ndryshimit të kërkesës. Horizontal scaling shton instanca; vertical scaling rrit një instancë. Një aplikacion me state lokal ose databazë të mbingarkuar nuk bëhet elastik vetëm duke shtuar VM. Pay-as-you-go zhvendos mënyrën e pagesës, por burime boshe mund të vazhdojnë të faturohen.

Shared responsibility ndryshon sipas shërbimit. Ofruesi mund të menaxhojë hardware dhe patch-et e motorit, ndërsa klienti ende përgjigjet për akseset, skemën, backup policy dhe konfigurimin e rrjetit. Managed nuk do të thotë “pa administrator” dhe SLA e një shërbimi nuk është garanci për të gjithë aplikacionin.

Shembull: zgjedhja midis VM me PostgreSQL dhe databazës managed zhvendos detyrat e patching/failover, por nuk e zgjidh vetë modelin e të dhënave ose një query të keqe.

Laborator: Modelet e shërbimit, elasticiteti dhe përgjegjësia

Koha: 90 minuta. **Mjedisi:** dokumentim lokal; nuk nevojitet llogari cloud.

- Për një faqe të thjeshtë, një sistem pagash dhe një analizë periodike identifiko compute, storage, identitetin dhe përgjegjësitë.
- Krijo tabelë IaaS/PaaS/SaaS/serverless me kontrollin, punën operative dhe kufizimet. Për çdo rresht cakto pronarin e patching, aksesit dhe rikthimit.
- Modelimi: 100 kërkesa/orë gjatë natës dhe 20 000 gjatë ditës. Shpjego cilat pjesë të aplikacionit mund të shkallëzohen dhe cilat kanë state.

- Defekti analitik: “u ble shërbim managed, prandaj nuk duhen backup dhe alarm”. Jep dy kundërshembuj dhe kontrollet përkatëse.
- Shkruaj një vendim të shkurtër arkitekturor për zgjedhjen midis VM dhe shërbimit managed, me një kusht që do ta ndryshonte vendimin.

Dorëzimi: tabela e përgjegjësive dhe vendimi teknik. **Pranimi:** përgjegjësitë lidhen me shërbimin konkret; elasticiteti nuk barazohet me bandwidth; kostoja nuk pretendohet zero kur mungon trafik. **Thellim:** analizo daljen nga një shërbim SaaS: eksporti i të dhënave, identitetet, ndërprerja dhe fshirja e verifikueshme.

2. Virtualizimi dhe izolimi

Hypervisor, containers dhe kufijtë e hostit

Teori: Hypervisor, containers dhe kufijtë e hostit

Objektivi: të dallosh izolimin e VM-së nga ai i container-it dhe të shpjegosh burimet reale të konkurrimit.

Hypervisor-i ndan hardware ndërmjet makinave virtuale me sisteme operative të veçanta. vCPU është kapacitet i planifikuar mbi CPU fizike sipas arkitekturës; nuk garanton gjithmonë një core fizik të dedikuar. CPU steal, memory pressure, NUMA, storage latency dhe kufijtë e rrjetit ndikojnë performancën. Burstable instances përdorin model kreditesh ose kufizimesh sipas ofruesit dhe nuk trajtohen si fuqi e pakufizuar e qëndrueshme.

Container-i zakonisht ndan kernel-in e hostit dhe përdor namespaces/cgroups për ndarje dhe kuota. Image përmban shtresat e aplikacionit; container-i është procesi që ekzekutohet nga ai image. Një container nuk është VM e vogël në çdo aspekt. Rootless, user namespaces dhe sandbox runtimes mund të forcojnë ndarjen, por kërkojnë mbështetje dhe konfigurim.

Disqet ephemeral mund të zhduken kur zëvendësohet instanca; volume persistent ka cikël jete dhe politikë të veçantë. Një snapshot mund të kapë blloqet, por konsistenca e aplikacionit kërkon mekanizëm shtesë. Live migration dhe host maintenance nuk e përjashtojnë nevojën për tolerancë dështimesh në aplikacion.

Noisy neighbor ndodh kur ngarkesa e një tenant-i ndikon një burim të përbashkët. Kuotat dhe izolimi e kufizojnë, por zgjedhja e tipit të instancës, rezervimi dhe matjet mbeten të nevojshme. Për workload me kërkesa të forta izolimi mund të nevojiten VM, dedicated hosts ose kontrolle të tjera sipas kërcënimit.

Shembull: API përdor vetëm 30% CPU, por ka vonesa të larta. Nëse storage latency rritet ose ka throttling, shtimi i vCPU mund të mos ndihmojë. Një metrikë e vetme nuk provon burimin e bottleneck-ut.

Laborator: Hypervisor, containers dhe kufijtë e hostit

Koha: 120 minuta. **Mjedisi:** një kompjuter ose VM; Docker është opsional për pjesën container.

- Regjistro CPU, RAM, disk dhe sistemin operativ. Shëno cilat burime janë fizike dhe cilat të virtualizuara sipas njohurive të mjedisit.
- Në Docker, nëse disponohet, lexo konfigurimin dhe kufijtë e një container-i laboratorik. Mos përdor privileged si zgjidhje për çdo gabim lejeje.
- Krahaso ciklin e jetës së një skedari në shtresën writable të container-it me atë në volume. Përdor vetëm të dhëna prove.
- Defekti: aplikacioni humb skedarët pas zëvendësimit të instancës. Identifiko vendin e ruajtjes dhe propozo persistent storage ose object storage sipas nevojës.

- Në letër projekto izolimin për dy ekipe që nuk i besojnë njëri-tjetrit dhe një worker të brendshëm me besim më të lartë.

Dorëzimi: diagram izolimi dhe raport i persistencës. **Pranimi:** container dhe VM dallohen; kuota RAM/CPU nuk paraqitet si kufi i plotë sigurie; të dhënat testuese ruhen sipas ciklit të zgjedhur. **Thellim:** vlerëso NUMA dhe pasojat e një procesi që kërkon shumë memorie ose pajisje GPU.

3. Sistemet e shpërndara

Konsistenca, dështimet e pjesshme dhe retry

Teori: Konsistenca, dështimet e pjesshme dhe retry

Objektivi: të shpjegosh pse një timeout nuk provon se operacioni dështoi dhe të projektosh riprovim korrekt.

Në sistem të shpërndarë, komponentët komunikojnë me rrjet dhe mund të dështojnë veçmas. Një klient mund të mos marrë përgjigjen edhe kur serveri e ka kryer ndryshimin. Network partition, vonesa, dublime dhe renditje të ndryshme janë pjesë e modelit të projektimit. Ora e murit nuk është rend absolut i të gjitha ngjarjeve; përdor identifikues, versione dhe protokolle përkatëse kur rendi ka rëndësi.

Strong consistency dhe eventual consistency përshkruajnë garanci të ndryshme që duhen lexuar për operacionin dhe shërbimin konkret. CAP flet për kompromisin mes consistency dhe availability gjatë një partition sipas përkufizimeve të modelit, jo për një menu ku çdo sistem zgjedh dy shkronja përgjithmonë. Replication lag mund t'i bëjë lexuesit të shohin gjendje të vjetër.

Idempotency do të thotë që përsëritja me të njëjtin identitet operacioni nuk krijon efekt të ri të paqëllimshëm. Një çelës idempotence lidhet me payload-in dhe ruhet bashkë me rezultatin. Nëse i njëjti çelës vjen me payload tjetër, sistemi duhet të trajtojë konfliktin. Afati i ruajtjes dhe scope-i i çelësit janë pjesë e kontratës.

Retries duan timeout, exponential backoff, jitter dhe kufi përpjekjesh. Circuit breaker kufizon presionin mbi varësi të prishura; bulkhead izolon burimet. Riprovimi i çdo gabimi pa dallim mund të shumëfishojë ngarkesën në incident. Exactly-once transport nuk garanton vetë exactly-once efekt biznesi kur ky efekt ndodh në një sistem tjetër.

Shembull: klienti dërgon porosi, serveri e regjistron, përgjigjja humbet. Retry me të njëjtin çelës duhet të kthejë të njëjtën porosi, jo ta tarifojë klientin dy herë.

Laborator: Konsistenca, dështimet e pjesshme dhe retry

Koha: 150 minuta. **Mjedisi:** Python 3 dhe `local-lab/cloud_lab.py`; ndiq README-n e laboratorit.

- Nis API lokale dhe krijo një porosi me header `Idempotency-Key`. Ruaj identifikuesin dhe përgjigjen.
- Përsërit kërkesën me të njëjtin çelës dhe payload. Pastaj përdor të njëjtin çelës me vlerë tjetër; krahaso statuset dhe numrin e rreshtave.
- Ekzekuto testet e përfshira për kërkesa konkurruese me të njëjtin çelës. Shpjego pse check-then-insert pa constraint/transaction është i pamjaftueshëm.
- Defekti analitik: klienti merr timeout pas commit. Vizatoni tri renditje të mundshme mes commit-it, përgjigjes dhe retry-t.
- Përcakto cilat gabime duhet të riprovohen dhe cilat kërkojnë korrigjim të kërkesës.

Dorëzimi: tri përgjigje HTTP, numri i porosive dhe diagrami kohor. **Pranimi:** vetëm një efekt për të njëjtin payload/çelës; payload i ndryshëm refuzohet; kufijtë e simulatorit shpjegohen. **Thellim:** zgjero modelin për pagesë te sistem i jashtëm dhe identifiko pse transaction lokale SQLite nuk e mbulon atë sistem.

4. Rajonet dhe disponueshmëria

Zona, failure domains, latenca dhe kuotat

Teori: Zona, failure domains, latenca dhe kuotat

Objektivi: të zgjedhësh vendosjen sipas kërkesave dhe të dallosh high availability nga disaster recovery.

Rajoni është grupim gjeografik/logjik sipas ofruesit; zona është kufi vendosjeje me karakteristika izolimi që duhen lexuar për shërbimin konkret. Një shërbim regional mund të përdorë disa zona pa ekspozuar kontroll të drejtpërdrejtë. Emrat dhe map-imi i zonave nuk duhet të krahasohen verbërisht ndërmjet llogarive dhe ofruesve.

Dy instanca në të njëjtën zonë tolerojnë disa dështime hosti, por jo humbjen e gjithë zonës. Multi-zone ul disa rreziqe, duke shtuar trafik ndërzona, vonesë dhe kosto. Multi-region kërkon replika, strategji shkrimi, failover, identitet dhe varësi të afta të mbijetojnë; nuk bëhet vetëm duke kopjuar frontend-in.

Disponueshmëria e përbërë varet nga varësitë dhe supozimet. Nëse dy komponentë të domosdoshëm kanë nga 99.9% dhe dështimet janë të pavarura, produkti është afërsisht 99.8001%. Pavarësia shpesh nuk vlen plotësisht kur ndajnë DNS, identitet, kontroll ose rrjet. SLO i aplikacionit matet nga përdoruesi, jo vetëm nga statusi i VM-ve.

Kuotat, rezervat e kapacitetit dhe kufijtë e API-ve ndikojnë provisioning dhe failover. Edhe kur buxheti ekziston, tipi i instancës mund të mos jetë i disponueshëm në vendin e kërkuar. RTO dhe RPO përcaktojnë strategjinë e rikuperimit dhe nivelin e replikimit. Latenca duhet matur nga klientët dhe ndërmjet varësive.

Shembull: aplikacioni ka dy API në zona të ndryshme por vetëm një databazë zonale. Ai ende ka failure domain të përbashkët për të dhënat. Health check duhet të pasqyrojë gatishmërinë reale dhe të mos krijojë failover të panevojshëm.

Laborator: Zona, failure domains, latenca dhe kuotat

Koha: 120 minuta. **Mjedisi:** projektim dhe laborator lokal; nuk nevojiten rajone cloud reale.

- Vizato API, databazë, queue, DNS dhe identitet në dy zona. Shëno scope-in zonal, regional ose global që supozon për secilin.
- Llogarit downtime buxhetin për 99.9% mbi 30 ditë dhe disponueshmërinë e dy varësive të domosdoshme me pavarësi të supozuar.
- Plotëso tabelën për humbje hosti, zone, rajoni dhe shërbimi identiteti. Për secilin cakto ndikimin dhe rrugën e rikuperimit.
- Defekti: zona rezervë nuk ka kuotë/kapacitet për ngarkesën e plotë. Rishiko planin e failover-it dhe rezervën.

- Përdor laboratorin lokal për të provuar ndalimin e një procesi API; deklaro qartë që kjo simulon proces, jo humbje zone reale.

Dorëzimi: diagram failure domains dhe llogaritjet. **Pranimi:** 99.9% mbi 30 ditë lejon 43.2 minuta të padisponueshme sipas modelit; pavarësia deklarohet; multi-zone nuk quhet backup. **Thellim:** projekto failover rajonal me DNS dhe shpjego ndikimin e cache-it dhe sesioneve ekzistuese.

5. AWS, Azure dhe Google Cloud

Familjet e shërbimeve dhe zgjedhja e platformës

Teori: Familjet e shërbimeve dhe zgjedhja e platformës

Objektivi: të krahasosh aftësi dhe përgjegjësi pa trajtuar emrat tregtarë si ekuivalenca të plota.

AWS, Azure dhe Google Cloud ofrojnë compute, storage, rrjet, databaza, identitet dhe shërbime aplikacionesh. EC2, Azure Virtual Machines dhe Compute Engine i përkasin familjes VM; S3, Blob Storage dhe Cloud Storage familjes object storage; EKS, AKS dhe GKE ofrojnë Kubernetes managed me modele operative që ndryshojnë. Emri i përbashkët i kategorisë nuk garanton të njëjtat SLA, API, quota ose karakteristika.

AWS përdor accounts dhe mekanizma organizimi; Azure përdor tenant, management groups, subscriptions dhe resource groups; Google Cloud përdor organization, folders dhe projects sipas modelit të tij. Këto kufij ndikojnë IAM, faturimin, quota dhe blast radius. Tenant identiteti nuk duhet barazuar me çdo kufi faturimi ose rrjeti.

Përzgjedhja duhet të nisë nga vendndodhja e të dhënave, integrimi i identitetit, kompetencat e ekipit, nevojat teknike dhe kostoja totale. Një shërbim managed që ul punën operative mund të ketë kosto për kërkesë, për kapacitet ose për trafik. Portabiliteti ka disa nivele: kodi, formati i të dhënave, infrastruktura dhe procesi i operimit.

Well-architected review shqyrton sigurinë, besueshmërinë, performancën, operimin dhe kostot; framework-et e ofruesve kanë emërtime dhe organizim specifik. Vlerëso arkitekturën e workload-it, jo numrin e shërbimeve që përdor. Managed alternative mund të jetë më e përshtatshme se një cluster i administruar vetë edhe kur cluster-i duket më i transferueshëm.

Shembull: një ekip me identitet të integruar në një ekosistem mund të përfitojë nga integrimi, por duhet të provojë kërkesat e aplikacionit dhe mundësinë e eksportit përpara vendimit.

Laborator: Familjet e shërbimeve dhe zgjedhja e platformës

Koha: 120 minuta. **Mjedisi:** dokumentacionet zyrtare të tre ofruesve dhe tabela e shërbimeve në burimet e kursit.

- Për VM, object storage, SQL managed, function dhe Kubernetes gjej shërbimin përkatës dhe lexo kufirin e përgjegjësisë.
- Për një aplikacion porosish krahaso dy alternativa nga dy ofrues: vendosje, identitet, persistencë, backup dhe matje kostosh.
- Shëno cilat detaje kërkojnë verifikim aktual: rajoni, kuotat, versioni i motorit, kostoja e trafikut dhe mbështetja.

- Defekti i analizës: një spreadsheet i quan të njëjta shërbimet sepse kanë të njëjtën kategori. Jep tri dallime që mund ta ndryshojnë migrimin.
- Shkruaj vendimin teknik dhe arsyen e moszgjedhjes së alternativës tjetër pa pretendime universale për cilësi.

Dorëzimi: tabelë me lidhje zyrtare dhe datën e kontrollit. **Pranimi:** emrat lidhen me kategori, jo identitet funksional; çmimet e vjetra nuk paraqiten si aktuale; vendimi mbështetet në kërkesa. **Thellim:** analizo lock-in e API-ve, formatit të të dhënave dhe proceseve operative veçmas.

6. Organizimi dhe landing zones

Llogaritë, kufijtë e prodhimit dhe guardrails

Teori: Llogaritë, kufijtë e prodhimit dhe guardrails

Objektivi: të përgatitësh një mjedis ku ekipet mund të vendosin shërbime pa zgjeruar pakontrolluar rrezikun.

Landing zone është themeli i organizimit, identitetit, lidhjeve, log-eve, sigurisë dhe faturimit. Nuk është vetëm një subnet fillestar. Ndaj prodhimin nga zhvillimi dhe sandbox-et sipas failure domain, lejeve dhe kostove. Kufiri i llogarisë/subscription/project shpesh është më i fortë operacionalisht se një emër resource group brenda të njëjtit kontekst, por semantika varet nga ofruesi.

Guardrails mund të ndalojnë konfigurime, të zbulojnë devijime ose të korrigjojnë automatikisht raste të kufizuara. Një organizational policy që kufizon veprime nuk është gjithmonë grant lejesh. Kontrolli preventive, detective dhe corrective duhet të kenë pronar, përjashtime dhe prova. Një deny i përgjithshëm pa proces mund të bllokojë recovery ose operacione legjitime.

Centralizimi i logs, identity federation dhe metadata e pronarit ndihmojnë auditimin. Tags/labels si owner, environment, cost_center dhe data_classification mundësojnë kërkim dhe chargeback, por nuk janë vetë kufij sigurie kur përdoruesit mund t'i ndryshojnë pa kontroll. Politika për rajonet dhe shërbimet duhet të lidhet me kërkesat reale, jo me supozim se çdo shërbim është regional.

Bootstrap dhe break-glass kërkojnë kujdes: identiteti që krijon bazën nuk duhet të mbetet i vetmi mekanizëm i përditshëm. Testo humbjen e IdP-së dhe rikuperimin. Llogaritë e mbyllura mund të lënë backups, identitete ose kontrata që duhen trajtuar sipas modelit të ofruesit.

Shembull: zhvilluesi ka të drejtë deploy në dev por jo të ndryshojë destinacionin qendror të audit log. Kjo ndarje realizohet me role dhe kufij, jo vetëm me një udhëzim në wiki.

Laborator: Llogaritë, kufijtë e prodhimit dhe guardrails

Koha: 150 minuta. **Mjedisi:** projektim lokal; zbatim vetëm në sandbox ekzistues të autorizuar nëse disponohet.

- Vizato organizimin për security, shared services, dev, test dhe prod. Cakto pronarët dhe kufijtë e faturimit.
- Shkruaj pesë guardrails: një për ekspozimin publik, një për log-et, një për identitetet, një për rajonet dhe një për metadata/kostot.
- Për secilin cakto parandalim, zbulim ose korrigjim dhe jep testin e pritur të refuzimit.
- Defekti: policy rajonesh ndalon edhe veprime të një shërbimi global. Identifiko nevojën për scope dhe përjashtim të kontrolluar.

- Përgatit procedurë për krijimin dhe mbylljen e sandbox-it me inventar të mbetjeve dhe pronar për çdo përjashtim.

Dorëzimi: organizimi, guardrails dhe rrjedha e përjashtimeve. **Pranimi:** prod/dev kanë kufij të qartë; grant dhe kufizim dallohen; logs nuk mund të ndryshohen nga çdo ekip aplikacioni. **Thellim:** projekto bootstrap për IaC kur backend-i i state dhe identiteti i pipeline-it ende nuk ekzistojnë.

7. Identiteti dhe IAM

Federation, role, policy dhe privilegji minimal

Teori: Federation, role, policy dhe privilegji minimal

Objektivi: të shpjegosh pse një kërkesë lejohet ose refuzohet dhe të ndërtosh akses të kufizuar.

Authentication vërteton identitetin; authorization vendos veprimin; audit regjistron përdorimin. Federation lidh identitetin e organizatës me role cloud dhe kredenciale afatshkurtra. MFA forcon hyrjen njerëzore, por workload-i kërkon model identiteti të përshtatshëm për automatizim. Një access key i ruajtur në repository ka cikël jete dhe rrezik tjetër nga token i shkurtër me scope të kufizuar.

RBAC jep leje sipas roleve; ABAC përdor atribute dhe kushte. Modeli i saktë i evaluation ndryshon sipas ofruesit: identity policy, resource policy, organization guardrails, permissions boundaries dhe deny mund të kombinohen. Në AWS explicit deny ka rol të fortë në vlerësim, por nuk duhet përgjithësuar i gjithë algoritmi për çdo platformë. Akses cross-account kërkon edhe marrëdhënie besimi dhe scope të saktë.

Least privilege nis nga detyrat dhe rrjedhat konkrete. Lejo vetëm veprimet dhe burimet e nevojshme, kufizo kushtet dhe rishiko përdorimin. Wildcard për çdo resource ose action e thjeshton provën fillestare por zgjeron blast radius. Vetëm observimi i aktivitetit të fundit nuk identifikon domosdoshmërisht lejet e nevojshme për recovery të rrallë.

Ndaj rolet e leximit, deploy, administrimit të identitetit dhe menaxhimit të secrets. Akses emergjent duhet të monitorohet dhe të provohet. Shërbimi mund të refuzojë për IAM, policy të resursit, encryption key, rrjet ose quota; “Access denied” duhet të lidhet me identitetin dhe kërkesën reale.

Shembull: worker-i ka leje të lexojë mesazhe dhe të shkruajë rezultatet në një destinacion, por jo të krijojë përdorues ose të ndryshojë policy të organizatës.

Laborator: Federation, role, policy dhe privilegji minimal

Koha: 150 minuta. **Mjedisi:** analizë policy me identitete fiktive; sandbox cloud opsional për prova pa ndryshime prodhimi.

- Cakto role vëzhgues, deployer dhe worker për aplikacionin e porosive. Për secilin listo veprimet dhe burimet e nevojshme.
- Lexo shembullin `iam-policy-example.json` dhe gjej kufijtë e tij. Është shembull AWS për një bucket fiktiv, jo policy universale.
- Ndërto teste pozitive dhe negative: lexim objekti të lejuar, shkrim i paautorizuar dhe akses jashtë prefiksit.

- Defekti analitik: policy e identitetit lejon, por një deny në nivel tjetër refuzon. Ndiq shtresat e vlerësimit dhe mos shto menjëherë Administrator.
- Dokumento çfarë prove duhet nga audit log për identitetin, session dhe resource të kërkesës.

Dorëzimi: matrica e roleve dhe arsyetimi i katër vendimeve të aksesit. **Pranimi:** privilegji minimal lidhet me detyrën; identitetet njerëzore dhe të aplikacionit dallohen; secrets nuk futen në materialet e dorëzuara. **Thellim:** projekto akses të përkohshëm për kontraktor me skadim, kufi burimesh dhe auditim të përdorimit.

8. Identiteti i workloads dhe secrets

OIDC, token-et, metadata dhe rrotullimi

Teori: OIDC, token-et, metadata dhe rrotullimi

Objektivi: të heqësh kredencialet afatgjata nga aplikacioni dhe të kuptosh zinxhirin e besimit.

Workload identity lidh procesin ose platformën me leje cloud pa shpërndarë access keys statike. VM metadata, service account federation dhe OIDC në CI kanë mekanizma të ndryshëm. OIDC token ka issuer, audience, subject dhe afate; trust policy duhet t'i kufizojë sipas repository, branch, environment ose identitetit përkatës. “Token i vlefshëm” nuk do të thotë “token nga workload-i i duhur”.

Metadata endpoint është burim kredencialesh ose konfigurimi sipas platformës. SSRF mund të shfrytëzojë aksesin e papërshtatshëm drejt tij; mbrojtjet e endpoint-it dhe të aplikacionit plotësojnë njëra-tjetrën. Mos përdor kontrollin e rrjetit si zëvendësim të scope-it të identitetit. Container-i mund të trashëgojë akses të hostit nëse kufijtë janë të hapur.

Secrets manager ruan vlera me kontroll aksesi dhe audit; KMS menaxhon çelësa kriptografikë sipas funksionit të shërbimit. Secret është vlera që përdoret, key është mjet kriptografik dhe identity është kush kërkon. Environment variables janë të përshtatshme për disa konfigurime, por mund të rrjedhin në logs, crash reports ose inspektim procesi.

Rotation kërkon që klientët të rifreskojnë vlerën dhe të pranojnë tranzicionin sipas modelit. Nëse databaza ndryshon password por aplikacioni mban lidhje të vjetra dhe nuk rifreskon cache, efekti nuk është i njëjtë si në një kërkesë të re. Revokimi i token-eve të lëshuara mund të mos jetë i menjëhershëm; afati dhe kontrolli në resource kanë rëndësi.

Shembull: pipeline i një pull request nga burim i jashtëm nuk duhet të marrë automatikisht rolin e prodhimit edhe nëse përdor të njëjtin workflow YAML.

Laborator: OIDC, token-et, metadata dhe rrotullimi

Koha: 120 minuta. **Mjedisi:** analizë lokale e claims fiktive dhe konfigurimit; nuk përdoren token-e reale.

- Krijo tri claims OIDC për prod branch, dev branch dhe repository tjetër. Përcakto cilat përputhen me trust-in e rolin prod.
- Ndaj konfigurimin publik, secret-in dhe referencën e çelësit. Jep mënyrën e shpërndarjes për API dhe worker.
- Shkruaj procedurë rotation me vlerë të vjetër/të re, cache, monitorim dhe rollback. Identifiko çfarë ndodh me instancat që nuk rifillojnë.

- Defekti: aplikacioni ka leje të lexojë secret por nuk ka përdorimin e nevojshëm të encryption key. Ndaj verifikimin IAM dhe atë të çelësit.
- Kontrollo laboratorin lokal për secrets të panevojshme; ai nuk ka nevojë për çelësa cloud dhe nuk duhet t'i kërkojë.

Dorëzimi: tabelë claims/trust dhe plan rrotullimi. **Pranimi:** issuer/audience/subject kontrollohen; secrets nuk regjistrohen; një repository i jashtëm nuk fiton rol prod. **Thellim:** projekto përdorimin e identitetit për një worker që jeton vetëm disa minuta pa shpërndarje manuale kredencialesh.

9. Rrjetet cloud

VPC/VNet, subnet, routing dhe private access

Teori: VPC/VNet, subnet, routing dhe private access

Objektivi: të gjurmosh një rrjedhë nga klienti te shërbimi privat dhe të dallosh IAM nga lidhshmëria.

Rrjeti cloud përfshin VPC/VNet, subnet, route tables, gateways dhe kontrolle filtrimi me semantikë të ofruesit. Disa objekte janë regionale, disa zonale ose globale. Broadcast dhe kontrolli L2 nuk janë gjithmonë të disponueshëm si në një switch fizik. Adresat e rezervuara dhe madhësia e subnet-it kontrollohen në dokumentacionin e platformës.

Një subnet nuk është publik vetëm nga emri. Rruga, adresa publike dhe politika përcaktojnë ekspozimin. NAT për dalje nuk krijon automatikisht hyrje të lejuar. Private endpoint lidh shërbim përmes modelit të ofruesit dhe shpesh kërkon DNS privat, policy dhe autorizim. Një shërbim mund të jetë i arritshëm në rrjet por të refuzojë identitetin.

Peering dhe transit hub kanë kufij routing-u dhe kosto të ndryshme; transitiviteti nuk supozohet. Prefikset duhet të mos mbivendosen mes zyrave, cloud-it, containers dhe partnerëve. VPN dhe lidhje të dedikuara përdorin rrugë në të dy drejtimet; BGP up nuk garanton policy të saktë për çdo subnet.

Security groups, firewall rules dhe subnet ACL mund të jenë stateful/stateless sipas produktit. Ndiq adresat para/pas NAT dhe rrugën e kthimit. Flow logs japin metadata me kufizime, jo payload të plotë. DNS conditional forwarding, MTU dhe egress janë shpesh burim problemesh në lidhje hibride.

Shembull: një databazë private zgjidhet me IP publike nga zyra sepse mungon forwarding i zonës private. Shtimi i një route nuk e ndryshon vetë përgjigjen DNS.

Laborator: VPC/VNet, subnet, routing dhe private access

Koha: 150 minuta. **Mjedisi:** projektim lokal dhe dokumentacion i një ofruesi të zgjedhur; cloud opsional.

- Ndaj 10.80.0.0/16 për aplikacion, data, menaxhim dhe zgjerim; planifiko dy zona dhe shkruaj route tables.
- Vizato trafikun Internet→frontend, API→databazë dhe admin→menaxhim. Shëno IP, DNS, NAT, policy dhe identitetin në çdo hop.
- Për private endpoint shkruaj përgjigjen DNS të pritur brenda cloud-it dhe nga zyra.
- Defekti: A lidhet me B, B me C, por A nuk arrin C. Kontrollo mbështetjen për transit dhe rrugët, jo vetëm peering status.
- Planifiko prova për një trafik të lejuar dhe dy të ndaluar; asnjë databazë nuk ekspozohet publikisht vetëm për ta bërë testin më të lehtë.

Dorëzimi: diagram dhe matrica e flukseve. **Pranimi:** subnet-et nuk mbivendosen; kthimi dhe DNS përfshihen; policy e rrjetit nuk zëvendëson IAM. **Thellim:** analizo egress të centralizuar kundrejt daljes lokale për kosto, failure domain dhe observability.

10. DNS, CDN dhe load balancing

Shpërndarja e trafikut dhe kufijtë e cache-it

Teori: Shpërndarja e trafikut dhe kufijtë e cache-it

Objektivi: të dallosh vendimin DNS, shpërndarjen e lidhjeve dhe ruajtjen e përmbajtjes në cache.

DNS kthen të dhëna që mund të ruhen deri në afatin e tyre; nuk kontrollon çdo kërkesë aplikacioni në kohë reale. Load balancer L4 shpërndan lidhje, ndërsa L7 mund të interpretojë HTTP, paths dhe headers. Health check duhet të përfaqësojë gatishmërinë pa rrëzuar të gjitha instancat për një varësi jo kritike. Liveness dhe readiness kanë kuptime të ndryshme.

CDN ruan përmbajtje afër klientëve sipas cache policy. Cache key mund të përfshijë hostname, path, query dhe headers; një gabim mund të përziejë përgjigje private mes përdoruesve. Cache-Control, invalidimi dhe versionimi i objekteve duhet të projektohen. Mos cache-o përgjigje të personalizuara vetëm sepse janë HTTP 200.

TLS mund të përfundojë në edge dhe të krijohet një lidhje tjetër drejt origin. Verifiko identitetin në të dy segmentet sipas modelit. WAF shqyrton trafik në kontekstin e aplikacionit; nuk korrigjon automatikisht autorizimin e dobët brenda aplikacionit. Rate limiting duhet të ketë identitet dhe scope të përshtatshëm; IP e vetme mund të përfaqësojë shumë klientë pas NAT.

Session affinity mund të ndihmojë aplikacione me state, por një instancë e humbur ende kërkon strategji rikuperimi. Client IP headers duhen besuar vetëm nga proxy-t e kontrolluar. Draining lejon mbyllje të kontrolluar të lidhjeve gjatë deploy. DNS failover ka cache dhe vonesë, ndaj nuk është zëvendësim i plotë i routing-ut të aplikacionit.

Shembull: health check kthen 200, por endpoint-i i porosive dështon. Nëse kontrolli provon vetëm procesin dhe jo gatishmërinë e varësisë kritike, trafiku mund të vazhdojë drejt instancës së papërshtatshme.

Laborator: Shpërndarja e trafikut dhe kufijtë e cache-it

Koha: 150 minuta. **Mjedisi:** API Python lokale; Docker Compose me Nginx opsional nga burimet.

- Provo `/healthz` dhe `/readyz` dhe identifiko çfarë kontrollon secili. Lexo kodin e laboratorit përpara interpretimit.
- Në Compose, nëse disponohet, dërgo disa kërkesa përmes gateway-t dhe identifiko instancën API nga përgjigjja.
- Analizo tri përgjigje: objekt statik publik, profil përdoruesi dhe rezultat pagese. Cakto cache policy dhe cache key për secilën.

- Defekti: një query parameter ndryshon rezultatin, por nuk përfshihet në cache key. Trego dy kërkesa që ilustrojnë përgjigjen e gabuar.
- Projekto draining për një deploy dhe shpjego çfarë ndodh me kërkesat në fluturim.

Dorëzimi: provat e health/readiness dhe tabela e cache-it. **Pranimi:** shërbimi privat nuk cache-ohet pa kontroll; DNS TTL merret parasysh; gateway lokal nuk quhet CDN reale. **Thellim:** projekto mbrojtjen e origin-it që klientët të mos anashkalojnë edge/WAF me adresë direkte.

11. Compute dhe makinat virtuale

Images, bootstrap, patching dhe cikli i instancës

Teori: Images, bootstrap, patching dhe cikli i instancës

Objektivi: të krijosh një model instance që mund të zëvendësohet pa humbur funksionin ose të dhënat.

Një VM cloud ka image, tip compute, disqe, rrjet, identitet dhe metadata. Familjet general-purpose, memory/compute optimized dhe GPU kanë profile të ndryshme; vCPU nuk mjafton për zgjedhje. Kontrollo RAM, storage/network throughput, arkitekturën CPU dhe kufijtë e instancës. Software i ndërtuar vetëm për x86 mund të mos funksionojë pa përshtatje në ARM.

Golden image paketizon bazën e provuar; bootstrap përfundon konfigurimin në nisje. Shmang bootstrap që shkarkon pa kontroll versionin më të fundit dhe secrets të shkruara në user-data. Një script nisjeje duhet të ketë dështim të dukshëm, retry të kontrolluar dhe identitet minimal. Metadata nuk është vend për kredenciale statike të përhershme.

Immutable replacement zëvendëson instancën nga image/konfigurimi i dëshiruar në vend të ndryshimeve manuale pa gjurmë. Patching kërkon testim, rollout dhe rikthim. State që duhet të mbijetojë ruhet jashtë instancës ose në volume me lifecycle të kontrolluar. Stop, reboot, terminate dhe deallocate nuk kanë të njëjtat pasoja për pagesat dhe disqet në çdo ofrues.

Spot/preemptible capacity ul koston në këmbim të mundësisë së ndërprerjes dhe kufizimeve të kapacitetit. Përdore për punë që riprovohen, checkpoint-e dhe fleksibilitet, jo si garanci për çdo shërbim kritik. Dedicated tenancy mund të ndihmojë kërkesa izolimi/licencimi, por duhet vlerësuar sipas kushteve konkrete.

Shembull: një API që ruan uploads në diskun ephemeral humb të dhëna gjatë zëvendësimit. Kalimi në object storage dhe konfigurim të riprodhueshëm ndan ciklin e aplikacionit nga ai i hostit.

Laborator: Images, bootstrap, patching dhe cikli i instancës

Koha: 120 minuta. **Mjedisi:** projektim dhe VM lokale opsionale; nuk kërkohet instancë me pagesë.

- Përgatit manifest me image/version, CPU/RAM, identitet, porta, disqe dhe health check për API-n e kursit.
- Ndaj çfarë hyn në image dhe çfarë merret gjatë bootstrap. Identifiko si verifikohet integriteti dhe versioni i artefaktit.
- Ndalo dhe rindërto procesin ose container-in lokal; provo që porositë mbeten në storage të zgjedhur.
- Defekti: bootstrap kthen sukses megjithëse shërbimi nuk është gati. Shto kontroll operacional të dallueshëm nga exit code i script-it.

- Shkruaj tabelën reboot/stop/terminate për një ofrues të zgjedhur dhe ruaj lidhjen zyrtare që verifikon pasojat e diskut/faturimit.

Dorëzimi: manifesti, ndarja image/bootstrap dhe provë zëvendësimi. **Pranimi:** secrets nuk futen në image; lifecycle i storage dokumentohet; veprimet e ofruesit nuk përgjithësohen pa burim. **Thellim:** projekto worker spot me checkpoint dhe vlerëso humbjen maksimale të punës pas ndërprerjes.

12. Shkallëzimi dhe performanca

Autoscaling, concurrency dhe testimi i ngarkesës

Teori: Autoscaling, concurrency dhe testimi i ngarkesës

Objektivi: të zgjedhësh sinjalin e duhur për shkallëzim dhe të gjesh bottleneck-un real.

Autoscaling ka sinjal, prag/target, minimum, maksimum dhe kohë reagimi. CPU është e dobishme për disa API, por queue age ose backlog për worker-at mund të përfaqësojë më mirë nevojën. Instancat e reja kanë startup time; cooldown/stabilization shmang luhatjet. Nëse varësia është databaza, shtimi i API-ve mund ta mbingarkojë më shumë.

Horizontal scaling kërkon ndarje të state dhe balancim. Vertical scaling mund të ndihmojë shpejt por ka kufij dhe ndërprerje sipas shërbimit. Concurrency përcakton sa punë ndodh njëherësh; parallelism sa punë kryhet fizikisht në të njëjtën kohë. Connection pools, threads dhe rate limits duhet të planifikohen bashkë.

Little's Law, në gjendje të qëndrueshme dhe me supozimet përkatëse, lidh $L = \lambda W$: numri mesatar në sistem barazohet me normën e mbërritjes shumëzuar me kohën mesatare. Nëse 100 kërkesa/s kalojnë mesatarisht 0.2 s, ka rreth 20 kërkesa në sistem mesatarisht. Kjo nuk zëvendëson kapacitetin për p99 ose burst-et.

Testi i ngarkesës ndan warm-up, steady state, burst dhe soak. Mato p50/p95/p99, gabimet, resource saturation dhe koston për operacion të suksesshëm. Mos gjenero ngarkesë kundër shërbimeve të huaja. Një test ku klienti është bottleneck nuk provon limitin e serverit.

Shembull: worker përpunon 5 mesazhe/s dhe mbërrijnë 12/s. Backlog rritet 7/s; dy worker-a të tillë ende nuk mjaftojnë në kushtet e dhëna. Retry dhe ndarja e burimeve mund ta ndryshojnë normën reale.

Laborator: Autoscaling, concurrency dhe testimi i ngarkesës

Koha: 150 minuta. **Mjedisi:** API lokale dhe Python; bëj kërkesa vetëm në localhost.

- Mato 100 kërkesa seriale dhe pastaj me concurrency të kufizuar. Ruaj kohën, statuset dhe normën e suksesit.
- Llogarit L për 100 kërkesa/s dhe 200 ms. Pastaj cakto rezervë kapaciteti dhe shpjego pse mesatarja nuk mbron p99.
- Për queue me 120 mesazhe/s dhe worker 30/s llogarit numrin minimal teorik; shto rezervë dhe kohën për shuarjen e backlog-ut.
- Defekti: shtimi i API-ve rrit connections te databaza dhe gabimet. Ndrysho hipotezën nga “mungon CPU” në kufirin e varësisë dhe identifiko provën.
- Projekto autoscaling me maximum, sinjal, stabilization dhe alarm për shkallëzim të bllokuar nga quota.

Dorëzimi: rezultatet, llogaritjet dhe policy autoscaling. **Pranimi:** matja ka njësi dhe kohë; kufijtë e klientit pranohen; throughput nuk ngatërrohet me latency. **Thellim:** vlerëso si request cancellation, retries dhe long-running tasks e ndryshojnë kapacitetin e nevojshëm.

13. Object storage

Buckets, versioning, lifecycle dhe akses i kontrolluar

Teori: Buckets, versioning, lifecycle dhe akses i kontrolluar

Objektivi: të zgjedhësh object storage për të dhëna dhe të kontrollosh ekspozimin e tyre.

Object storage mban objekte me key, përmbajtje dhe metadata përmes API-ve. Prefikset nuk janë domosdoshmërisht dosje me të njëjtën semantikë POSIX. Nuk ofron automatikisht file locking, append dhe rename atomik si një filesystem. Përdoret për uploads, backups, logs, data lakes dhe përmbajtje statike sipas kufizimeve të shërbimit.

Durability dhe availability janë koncepte të ndryshme. Replikimi brenda shërbimit mbron nga disa dështime fizike, por një fshirje e autorizuar mund të përhapet. Versioning mban versione sipas konfigurimit; retention/object lock dhe legal hold kanë funksione specifike. Lifecycle kalon ose fshin objekte sipas rregullave që duhet të provohen me të dhëna testimi.

Klasat hot/cool/archive ndryshojnë kosto ruajtjeje, kërkesash, minimumi kohe dhe rikthimi. Retrieval mund të ketë vonesë dhe tarifë; “më lirë për GB” nuk jep koston totale. Multipart uploads të papërfunduar dhe versionet e vjetra mund të mbeten burim shpenzimi.

Aksesi publik kërkon vendim të shprehur. Signed URLs kufizojnë metodën, objektin dhe kohën sipas modelit, por kush i zotëron mund t'i përdorë brenda afatit. Emri i objektit nuk duhet të jetë autorizim. Encryption at rest nuk ndalon klientin e autorizuar gabim; IAM dhe key policy plotësojnë njëra-tjetrën. Consistency dhe event notifications lexohen sipas shërbimit, jo nga supozime të vjetra për gjithë object storage.

Shembull: uploads private nuk duhet të bëhen publikë për të lehtësuar frontend-in. API autorizon përdoruesin dhe jep URL të kufizuar vetëm për objektin që i përket.

Laborator: Buckets, versioning, lifecycle dhe akses i kontrolluar

Koha: 120 minuta. **Mjedisi:** projektim me skedarë testimi; sandbox object storage është opsional.

- Krijo plan për uploads private, assets publike dhe backup. Cakto bucket/container, key pattern, pronar dhe retention.
- Ndërto matricë lejesh për klientin, API-n, worker-in dhe operatorin. Shqyrto shembullin IAM të kursit pa e aplikuar në llogari reale.
- Për një objekt të ndryshuar tri herë cakto çfarë mban versioning dhe çfarë fshin lifecycle pas afatit.
- Defekti: fshirja logjike replikohet dhe kopja “rezervë” zhduket. Propozo backup/retention me identitet dhe kufi të pavarur sipas kërkesës.
- Llogarit koston hipotetike duke përfshirë GB-muaj, kërkesat dhe retrieval, jo vetëm ruajtjen.

Dorëzimi: policy të dhënash dhe tabela lifecycle. **Pranimi:** object storage nuk trajtohet si disk lokal; URL e nënshkruar konsiderohet akses i ndashëm; versioning nuk shpallet backup i plotë. **Thellim:** projekto integrity check, multipart abort policy dhe ngjarje dublikatë pas upload-it.

14. Block storage dhe file storage

IOPS, throughput, snapshots dhe persistenca

Teori: IOPS, throughput, snapshots dhe persistenca

Objektivi: të zgjedhësh llojin e ruajtjes sipas semantikës dhe performancës.

Block storage i jep hostit blloqe mbi të cilat vendos filesystem ose databazë. File storage jep hierarki dhe protokolle si NFS/SMB me semantikë të shërbimit. Object storage shërben objekte përmes API. Këto nuk zëvendësohen vetëm duke ndryshuar endpoint-in. Multi-attach nuk garanton që filesystem-i i zakonshëm mund të shkruhet në mënyrë të sigurt nga disa hoste.

IOPS mat operacione/s; throughput byte/s; latency kohën e operacionit. Me 3000 IOPS dhe 16 KiB për operacion, kufiri teorik i të dhënave është rreth 46.875 MiB/s përpara kufizimeve të tjera. Queue depth, cache, tipi i leximit/shkrimit dhe madhësia e punës ndikojnë rezultatet. Instanca ka kufijtë e saj për bandwidth të storage, përveç atyre të volumit.

Snapshot zakonisht kap gjendje blloqesh sipas mekanizmit të ofruesit. Crash-consistent nuk është domosdoshmërisht application-consistent. Databaza mund të kërkojë flushing, backup API ose log replay. Restore duhet të provohet, duke përfshirë fillimin e aplikacionit dhe verifikimin e të dhënave. Snapshot incremental nuk do të thotë se operatori duhet të rikthejë manualisht çdo diferencë.

Encryption, backup retention, zone affinity dhe delete-on-termination ndikojnë ciklin e jetës. File shares mund të kenë throughput të lidhur me kapacitetin ose të provisionuar veçmas. Storage për containers lidhet edhe me StorageClass, access modes dhe reclaim policy.

Shembull: API scale-out me një disk të përbashkët SQLite në filesystem rrjeti nuk është automatikisht arkitekturë e sigurt. Laboratori i kursit përdor SQLite vetëm në një host/volume lokal për mësim dhe nuk demonstron distributed database.

Laborator: IOPS, throughput, snapshots dhe persistenca

Koha: 120 minuta. **Mjedisi:** laborator SQLite lokal dhe analizë storage; nuk bëhen benchmark-e në disk prodhimi.

- Klasifiko sistemin e porosive, uploads, logs dhe dokumentet e përbashkëta sipas block/file/object.
- Llogarit throughput nga 3000 IOPS × 16 KiB dhe krahasoje me një kufi hipotetik instance prej 40 MiB/s.
- Krijo backup me komandën e laboratorit që përdor SQLite backup API. Riktheje në skedar të ri dhe provo të dhënat.
- Defekti: kopjohet vetëm skedari kryesor gjatë shkrimeve aktive në WAL mode. Shpjego pse backup API është më i përshtatshëm se kopjim i verbër.

- Dokumento volume lifecycle kur instanca/container-i hiqet dhe si shmanget fshirja e paqëllimshme.

Dorëzimi: llogaritjet dhe prova restore. **Pranimi:** IOPS dhe throughput dallohen; backup hapet dhe lexohet; SQLite i përbashkët nuk paraqitet si HA. **Thellim:** projekto migrim të volumit ndërzona me kohë kopjimi, konsistencë aplikacioni dhe kriter për kalimin final.

15. Databazat relacionale managed

Transaksionet, replika, failover dhe migrimi i skemës

Teori: Transaksionet, replika, failover dhe migrimi i skemës

Objektivi: të zgjedhësh garancitë e databazës dhe të dallosh replikën e leximit nga zgjidhja HA.

SQL managed zvogëlon administrimin e hardware/motorit sipas shërbimit, por kërkon skemë, indekse, query tuning, leje dhe plan rikthimi. Transaksioni grupon ndryshime me garanci ACID sipas motorit dhe isolation level. Unique constraints dhe foreign keys mund të mbrojnë invariants më mirë se kontrolle të ndara në aplikacion.

Read replica ndihmon leximin por mund të ketë lag dhe semantikë promovimi të ndryshme. Standby HA dhe read replica nuk janë sinonime universale. Failover mund të ndryshojë endpoint, lidhjet dhe transaksionet në fluturim. Klienti duhet të trajtojë reconnect dhe retry pa dyfishuar efektin. Connection pool i çdo API-je shumëzohet me numrin e instancave.

Isolation levels ndikojnë anomalitë dhe konkurrimin; “transaction ekziston” nuk do të thotë se çdo model check-then-write është i sigurt. Optimistic concurrency përdor version/check; pessimistic locking rezervon gjendjen sipas motorit. Deadlocks duan retry të kontrolluar dhe rend konsistent veprimesh.

Backup, log-based point-in-time recovery dhe retention kanë kosto dhe kufij. Replikimi shpërndan edhe gabime logjike; backup mbron kërkesë tjetër. Schema migration duhet të jetë e përputhshme me versionin e vjetër dhe të ri gjatë rollout-it. Expand-contract shton fushën/funksionin, migron përdorimin dhe vetëm pastaj heq të vjetrën.

Shembull: 20 API me pool 50 krijojnë potencial 1000 lidhje. Nëse databaza lejon më pak, autoscaling mund të ulë disponueshmërinë në vend që ta rrisë. Kërko pooling, kufi dhe matje.

Laborator: Transaksionet, replika, failover dhe migrimi i skemës

Koha: 150 minuta. **Mjedisi:** SQLite lokal për constraints/transaksione; sjellja e motorit managed analizohet veçmas.

- Lexo skemën e porosive dhe unique key. Krijo kërkesa konkurruese dhe verifiko që nuk ka dy rreshta për të njëjtin çelës.
- Llogarit connections për 12 API me pool 30 dhe dy worker-a me pool 10. Cakto rezervën për administrim dhe failover.
- Shkruaj plan expand-contract për shtimin e fushës currency pa e prishur klientin e vjetër.
- Defekti: lexim i menjëhershëm nga replika nuk sheh porosinë e krijuar. Dallo replication lag nga humbja e të dhënave dhe jep strategji read-your-writes.

- Për një shërbim SQL managed lexo nga dokumentacioni mënyrën e failover, backup dhe PITR; mos e nxirr nga rezultati SQLite.

Dorëzimi: constraint i provuar, llogaritja e pool-eve dhe plani i migrimit. **Pranimi:** kufijtë e motorit lokal deklarohen; HA/replica/backup dallohen; retry mbetet idempotent. **Thellim:** analizo si ndryshon sistemi kur shkrimi kërkon transaction mes dy databazave të pavarura.

16. NoSQL dhe shpërndarja e të dhënave

Partition keys, indekset dhe hot partitions

Teori: Partition keys, indekset dhe hot partitions

Objektivi: të modelosh të dhënat nga access patterns dhe të parashikosh pikat e nxehta.

NoSQL përfshin modele key-value, document, wide-column dhe graph me garanci të ndryshme. Zgjedhja fillon nga pyetjet, volumi, latenca, konsistenca dhe ndryshimet e skemës. Document nuk do të thotë mungesë skeme; kontrata dhe validimi mbeten të nevojshme. Graph favorizon marrëdhënie dhe traversime, por nuk është zgjidhje universale për çdo query.

Partition key përcakton shpërndarjen sipas shërbimit. Një key si data e sotme mund të grumbullojë gjithë shkrimet në një partition. Cardinality e lartë ndihmon por shpërndarja e frekuencës ka rëndësi: një tenant shumë i madh mund të dominojë edhe mes miliona key-ve. Sharding/salting shpërndan ngarkesën por ndërlikon leximin dhe agregimin.

Indekset sekondare krijojnë rrugë të tjera kërkimi me kosto ruajtjeje dhe shkrimi. Query që bën scan të plotë mund të kushtojë shumë edhe kur kthen pak rezultate. TTL mund të ndihmojë skadimin, por koha e fshirjes fizike dhe garancitë duhen lexuar; nuk është automatikisht mekanizëm i menjëhershëm sigurie.

Konsistenca, transactions dhe multi-region writes ndryshojnë midis produkteve. Konfliktet mund të zgjidhen me last-write-wins ose mekanizma të tjerë që kërkojnë kuptim të orës dhe invariants. Denormalization ul joins por shumëfishon kopjet që duhen rifreskuar.

Shembull: për porosi të një klienti, key mund të lidhë tenant-in me një bucket kohor dhe sort key me kohën/ID-në. Nëse një tenant ka shumicën e trafikut, nevojitet ndarje shtesë pa humbur kërkimin e synuar.

Laborator: Partition keys, indekset dhe hot partitions

Koha: 150 minuta. **Mjedisi:** letër/JSON dhe Python opsional; nuk kërkohet databazë cloud.

- Shkruaj pesë access patterns: porosi me ID, porosi për klient, të fundit, sipas statusit dhe raport ditor.
- Projekto key/index për secilin dhe shëno cilat lexime kërkojnë scatter-gather ose scan.
- Për 1000 shkrime/s me 80% nga një tenant llogarit ngarkesën e atij tenant-i. Propozo sharding dhe shpjego koston e bashkimit të leximeve.
- Defekti: key është vetëm data dhe gjithë ngarkesa bie sot. Ndrysho modelin pa humbur kërkesën e raportimit ditor.
- Jep dy raste ku SQL do të ishte zgjedhje më e thjeshtë dhe dy ku një model NoSQL ka avantazh të arsyetuar.

Dorëzimi: access patterns, keys dhe plan migrimi. **Pranimi:** nuk përdoret emri NoSQL si garanci shpejtësie; hot partition shpjegohet me shpërndarjen reale; indekset kanë kosto. **Thellim:** analizo konfliktin e dy shkrimeve rajonale për të njëjtin inventory count dhe mbro invariant-in e stokut.

17. Cache dhe performanca e leximit

Cache-aside, invalidimi dhe thundering herd

Teori: Cache-aside, invalidimi dhe thundering herd

Objektivi: të përdorësh cache pa e bërë burim të fshehtë të të dhënave të gabuara.

Cache mban kopje për të ulur latency ose ngarkesën mbi burimin. Cache-aside lexon fillimisht cache dhe e mbush pas miss; write-through përfshin shkrimin e cache-it në rrjedhën e shkrimit sipas arkitekturës. TTL, eviction dhe invalidation janë vendime korrektësie. Një vlerë e skaduar nuk është gjithmonë e papranueshme, por kufiri i staleness duhet të jetë i shprehur.

Hit ratio vetëm nuk mjafton. Mato latency, madhësinë, eviction rate, memory pressure dhe ngarkesën e origin pas miss-eve. Cache key duhet të përfshijë tenant, identitet ose version kur përgjigjja ndryshon sipas tyre. Rrjedhja e të dhënave mes tenant-ëve është më serioze se një ulje performance.

Thundering herd ndodh kur shumë klientë rifreskojnë të njëjtën vlerë pas skadimit. Jitter në TTL, request coalescing, locks të kufizuara dhe stale-while-revalidate mund ta reduktojnë sipas kërkesës. Negative caching ul pyetjet për mungesa, por mund të fshehë krijime të reja derisa të skadojë.

Cache i shpërndarë mund të humbasë lidhje ose data gjatë failover. Nëse aplikacioni nuk mund të funksionojë pa të, cache është bërë varësi kritike dhe ka nevojë për objektiva të veçantë. Distributed locks kërkojnë analizë lease, clock dhe fencing; një lock i thjeshtë nuk është garanci universale që dy workers nuk kryejnë efekt të jashtëm.

Shembull: katalogu mund të pranojë vlerë pak të vjetër; bilanci për pagesë mund të kërkojë kontroll autoritativ. Një policy e vetme cache për të dyja do të ishte kompromis i fshehur.

Laborator: Cache-aside, invalidimi dhe thundering herd

Koha: 120 minuta. **Mjedisi:** projektim dhe kod i thjeshtë lokal sipas nevojës.

- Klasifiko katalog, bilanc, profil dhe status porosie sipas tolerancës ndaj staleness. Cakto TTL dhe invalidim.
- Krijo dy tenantë me të njëjtin ID objekti dhe trego pse key vetëm me ID është i pamjaftueshëm.
- Modelimi: 500 kërkesa njëherësh kur skadon një key. Krahaso 500 lexime origin me një rifreskim të bashkuar.
- Defekti: cache bie dhe databaza nuk përballon trafikun e plotë. Projekto rate limit, degradim dhe rezervë kapaciteti.
- Përgatit test që dallon një përgjigje të vjetër të lejuar nga një shkelje e kontratës së saktësisë.

Dorëzimi: politika për katër kategori dhe simulimi i cache miss. **Pranimi:** keys përfshijnë scope-in; hit ratio nuk është prova e vetme; bilanci nuk lexohet verbërisht nga cache e vjetër. **Thellim:** analizo fencing token për një worker me lease të skaduar që vazhdon të ekzekutohet.

18. Mesazhet dhe arkitektura me ngjarje

Queue, pub/sub, retry dhe dead-letter

Teori: Queue, pub/sub, retry dhe dead-letter

Objektivi: të shkëputësh prodhuesin nga konsumatori duke ruajtur saktësinë e efektit biznesi.

Queue shpërndan punë te konsumatorët sipas modelit; pub/sub shpërndan ngjarje te abonentë të ndryshëm; event stream mund të ruajë log për replay dhe konsum sipas offsets. Këto kanë kontrata të ndryshme renditjeje, retention dhe replay. Një queue nuk është databazë raportimi vetëm sepse mban mesazhe përkohësisht.

At-least-once mund të sjellë dublikime, sidomos kur konsumatori kryen punën por ACK humbet. Visibility timeout/lease rezervon punën për një interval sipas produktit; kur skadon mund të dorëzohet përsëri. Konsumatori duhet idempotent, ose efekti dhe shënimi i përfundimit të jenë të koordinuar. Exactly-once brenda broker-it nuk përfshin automatikisht email, pagesë ose databazë të jashtme.

Retry policy dallon gabime transient nga payload i pavlefshëm. Backoff, jitter, maksimum përpjekjesh dhe dead-letter queue kufizojnë ciklin. DLQ kërkon pronar, alarm dhe proces replay me kontroll; thjesht grumbullimi i mesazheve nuk është zgjidhje. Monitoro age of oldest message dhe throughput, jo vetëm gjatësinë e radhës.

Transactional outbox shkruan ndryshimin dhe ngjarjen në të njëjtin transaction lokal; publikuesi i mëvonshëm ende mund të dërgojë dy herë, ndaj konsumatori ruan deduplikim. Saga koordinon hapa me compensations; compensation nuk është gjithmonë rollback që e kthen botën në gjendje identike.

Shembull: krijimi i porosisë dhe detyra për faturim ruhen bashkë. Worker-i i laboratorit demonstron transaksion lokal dhe retry, por nuk provon exactly-once pagesë te një ofrues i jashtëm.

Laborator: Queue, pub/sub, retry dhe dead-letter

Koha: 150 minuta. **Mjedisi:** API dhe worker nga `local-lab`.

- Krijo porosi dhe lexo statusin pending përpara nisjes së worker-it. Ekzekuto worker `--once` dhe kontrollo ndryshimin.
- Nis worker me dështim të simuluar në përpjekjen e parë sipas README-së. Kontrollo attempts dhe retry përpara përfundimit.
- Provo një rishfaqje të të njëjtit identitet porosie dhe verifiko se nuk krijohet efekt i dyfishtë në modelin lokal.
- Defekti analitik: efekti i jashtëm kryhet, pastaj procesi vdes para ACK. Shpjego pse transaction lokale nuk mjafton dhe cakto çelësin për sistemin e jashtëm.

- Projekto DLQ me afat, pronar dhe replay të kontrolluar; përcakto alarm për moshën e mesazhit më të vjetër.

Dorëzimi: statuset dhe timeline prodhues/worker. **Pranimi:** retry dhe dublikimi trajtohen; kufiri i transaction shënohet; renditja nuk supozohet globale. **Thellim:** projekto saga rezervim stoku→pagesë→dërgesë dhe shpjego kompensimin kur pagesa refuzohet pas rezervimit.

19. Serverless dhe funksionet

Event triggers, concurrency dhe kufijtë e runtime

Teori: Event triggers, concurrency dhe kufijtë e runtime

Objektivi: të zgjedhësh funksione për punë të përshtatshme dhe të ruash kontrollin e varësive.

Function-as-a-Service ekzekuton kod sipas ngjarjeve pa menaxhuar serverin drejtpërdrejt. Runtime, memory/CPU, timeout, concurrency, payload dhe trigger policy janë pjesë e konfigurimit. Cold start është inicializimi që shfaqet në disa ekzekutime; ndikimi ndryshon me runtime, paketën, rrjetin dhe konfigurimin. Serverless containers dhe workflows kanë kufij të tjerë nga funksionet e shkurtra.

Funksioni duhet të trajtojë dublikime dhe retries sipas burimit. Sinkron, queue trigger dhe event bus mund të kenë politika të ndryshme. Concurrency e pakufizuar në dukje mund të mbushë connection pool të databazës ose të arrijë quota. Reserved/provisioned capacity ose mekanizma analogë ndihmojnë në raste të caktuara, me kosto dhe kufizime që verifikohen.

State në memorie ose disk të përkohshëm mund të mbijetojë për disa thirrje por nuk është garanci persistente. Paketimi, dependencies dhe secrets duhen kontrolluar si në çdo aplikacion. Kohëzgjatja e pritjes për I/O mund të kushtojë sipas modelit të faturimit; nuk krahasohet vetëm numri i kërkesave.

Workflow orchestrator mban hapat, retries dhe afatet e një procesi, por kërkon idempotency të aktiviteteve. Long-running jobs me burime të specializuara mund të përshtaten më mirë në batch/containers. Zgjedhja varet nga modeli i ngarkesës, jo nga moda e emrit.

Shembull: upload i një objekti ndez një funksion për thumbnail. Nëse funksioni shkruan përsëri në të njëjtin prefiks trigger pa filtër, mund të krijojë cikël ngjarjesh dhe kosto të panevojshme.

Laborator: Event triggers, concurrency dhe kufijtë e runtime

Koha: 120 minuta. **Mjedisi:** analizë lokale e event handler; zbatimi cloud është opsional.

- Projekto handler për object-upload me identifikues ngjarjeje, version objekti dhe output prefix të ndarë.
- Modelimi: mesatarisht 200 kërkesa/s me kohë 0.5 s. Llogarit concurrency mesatare teorike dhe cakto kufi për databazën.
- Shkruaj policy retry për timeout dhe payload të pavlefshëm. Specifiko destinacionin e dështimeve dhe alarmet.
- Defekti: output-i aktivizon të njëjtin trigger. Shto filtrimin dhe test me event të output-it që nuk duhet të krijojë punë të re.
- Krahaso function me worker container për punë 30-minutëshe, duke kontrolluar kufirin real të runtime në ofruesin e zgjedhur.

Dorëzimi: kontrata e event-it dhe vendimi function/container. **Pranimi:** concurrency teorike është 100 sipas supozimeve; retries janë të kufizuara; storage i përkohshëm nuk përdoret si burim i vetëm i të dhënave. **Thellim:** projekto workflow me callback të jashtëm, afat dhe deduplikim të callback-eve.

20. Containers dhe paketimi

Images, Docker Compose dhe supply chain

Teori: Images, Docker Compose dhe supply chain

Objektivi: të ndërtosh artefakt të riprodhueshëm që mund të nisë me konfigurim të jashtëm.

Image është artefakt me shtresa; tag është referencë që mund të ndryshojë; digest identifikon përmbajtjen përkatëse. Dockerfile duhet të kontrollojë bazën, dependencies dhe përdoruesin. Multi-stage build ndan mjetet e ndërtimit nga runtime. Mos ruaj tokens në layers edhe nëse fshihen në një hap të mëvonshëm: historiku mund t'i ruajë.

Procesi në container duhet të trajtojë sinjalet dhe mbylljen, të shkruajë logs të dobishme dhe të ekspozojë health/readiness sipas platformës. Container-i running nuk është domosdoshmërisht ready. Në Compose, `depends_on` me service started nuk garanton gatishmëri; health checks dhe `service_healthy` adresojnë nisjen sipas mbështetjes. Aplikacioni ende ka nevojë për trajtim dështimesh pas nisjes.

Volumes ruajnë state jashtë shtresës së container-it. Port publishing mund të lidhet vetëm me localhost për laborator. Network i brendshëm nuk është identitet aplikacioni. Një socket Docker i montuar brenda container-it mund të japë kontroll të gjerë mbi hostin; shmange si zgjidhje të thjeshtë për CI pa analizë.

Supply chain përfshin burimin, dependencies, ndërtimin, registry dhe deploy. SBOM liston komponentët; skanimi zbulon probleme të njohura; nënshkrimi/provenance ndihmon origjinën dhe integritetin. Asnjëri nuk është provë se aplikacioni është pa dobësi. Pinning i një versioni kërkon proces përditësimi, jo ngrirje të përhershme.

Shembull: dy API replicas në Compose ndajnë një SQLite volume lokal për demonstrim. Ky model nuk provon disponueshmëri databaze kur hosti bie dhe nuk rekomandohet si cluster storage i shpërndarë.

Laborator: Images, Docker Compose dhe supply chain

Koha: 150 minuta. **Mjedisi:** Docker Engine/Desktop dhe Compose, nëse të instaluara; alternativa është analizë e skedarëve.

- Lexo Dockerfile dhe Compose të laboratorit. Identifiko përdoruesin, portat, volume dhe komandat API/worker.
- Ndërto dhe nis projektin vetëm me emrin e përcaktuar; testi lidhet në 127.0.0.1. Provo health, krijim porosie dhe worker.
- Lexo logs dhe identitetin e instancës në përgjigje për disa kërkesa përmes Nginx.
- Defekti: një container është running por `/readyz` dështon. Krahaso statusin e procesit me kontrollin e varësisë.

- Ndale projektin me ruajtje volume, pastaj provo rinisjen. Fshirjen e volume bëje vetëm nëse dëshiron të heqësh të dhënat e këtij laboratori.

Dorëzimi: manifesti i lexuar, rezultatet dhe lifecycle i volume. **Pranimi:** publikohet vetëm porta lokale; nuk ka secrets në image; multi-container nuk quhet multi-zone. **Thellim:** përgatit pipeline që promovon të njëjtin digest nga test në prod dhe refuzon artefakte pa provenance të kërkuar.

21. Kubernetes dhe orkestrimi

Pods, controllers, Services dhe konfigurimi

Teori: Pods, controllers, Services dhe konfigurimi

Objektivi: të ndjekësh gjendjen e dëshiruar nga manifesti te pod-i që i shërben klientit.

Kubernetes përdor API dhe controllers që pajtojnë gjendjen reale me atë të dëshiruar. Pod është njësi ekzekutimi me containers që ndajnë disa burime. Deployment menaxhon replicas dhe rollout për aplikacione të përshtatshme; StatefulSet jep identitet të qëndrueshëm dhe lidhje me storage, por nuk krijon vetë protokoll replikimi të databazës. Job/CronJob përshtaten për detyra të përfunduara dhe planifikim me kufizimet e tyre.

Service zgjedh endpoints përmes labels dhe selectors. ClusterIP, headless Service dhe mënyrat e ekspozimit kanë role të ndryshme. Ingress/Gateway API kërkojnë controller përkatës. CNI implementon lidhjen e rrjetit dhe CSI integrimin storage. Cluster managed zvogëlon disa detyra të control plane, por node pools, add-ons, workload security dhe recovery mbeten për t'u menaxhuar sipas shërbimit.

Requests ndihmojnë scheduling; limits kufizojnë përdorimin sipas burimit dhe mekanizmit. Limit RAM mund të sjellë OOM kill; CPU limit mund të shkaktojë throttling. Probes dallojnë nisjen, gjallërinë dhe gatishmërinë. Liveness që varet nga databaza e përbashkët mund t'i rinisë të gjithë pods gjatë një incidenti të databazës dhe ta përkeqësojë ngarkesën.

ConfigMap jep konfigurim jo sekret; Secret ka kontrolle aksesi dhe nevojë për mbrojtje. Base64 në manifest nuk është kriptim. Service account dhe RBAC kufizojnë aksesin në API; workload identity cloud është integrim tjetër që duhet konfiguruar.

Shembull: Deployment ka dy pods Ready por Service nuk ka endpoints sepse selector është gabim. DNS mund të zgjidhë emrin e Service-it edhe kur nuk ekziston backend i përputhshëm.

Laborator: Pods, controllers, Services dhe konfigurimi

Koha: 180 minuta. **Mjedisi:** cluster lokal testimi me CNI; shih `kubernetes/README.sq.md`.

- Lexo namespace, Deployment dhe Service të kursit. Shembulli është HTTP stateless për mekanizmat Kubernetes, jo databaza e plotë e porosive.
- Apliko manifestin vetëm në namespace `cloud-course`. Kontrollo pods, events, Service dhe EndpointSlices.
- Përdor port-forward për akses lokal; mos krijo LoadBalancer publik si kusht të ushtrimit.
- Defekti: ndrysho selector në një kopje laboratorike dhe vëzhgo endpoints bosh. Riktheje dhe provo kërkesën.

- Ndrysho replicas dhe lexo requests/limits. Shpjego pse dy replicas nuk ndahen domosdoshmërisht në hoste të ndryshme pa kontroll scheduling.

Dorëzimi: manifesti dhe prova e rrugës Service→endpoint. **Pranimi:** namespace verifikohet; probe ka qëllim të qartë; Secret nuk quhet i kriptuar vetëm nga base64. **Thellim:** projekto rollout me startupProbe për aplikacion që inicializohet ngadalë dhe shpjego ndikimin në disponueshmëri.

22. Kubernetes në prodhim

Izolimi, storage, upgrades dhe politika

Teori: Izolimi, storage, upgrades dhe politika

Objektivi: të dallosh një deploy që punon nga një platformë që mund të operohet me kufij të fortë.

Namespace organizon objekte dhe leje por nuk është vetë izolim i plotë për tenantë që nuk besojnë njëri-tjetrin. RBAC, quotas, NetworkPolicy, pod security, node isolation dhe shërbimet e përbashkëta duhen analizuar së bashku. Cluster-scoped resources, controllers dhe webhooks mund të ndikojnë më gjerë se një namespace. Kërkesa e hard multitenancy mund të çojë te cluster-a ose plane kontrolli të ndara.

NetworkPolicy kërkon CNI me enforcement. Lejet kombinohen sipas modelit Kubernetes, jo rendit të file-ve si first-match ACL. DNS egress dhe trafiku i nevojshëm për platformën duhen lejuar kur aktivizohet default deny. Pod Security Admission dhe securityContext kufizojnë sjelljen e containers sipas profileve dhe nevojave.

PodDisruptionBudget ndihmon disa ndërprerje vullnetare dhe nuk garanton mbijetesë ndaj çdo dështimi. Topology spread/anti-affinity lidh vendosjen me failure domains reale të node-ve. HPA ka nevojë për metrika dhe kërkesa të sakta; autoscaling i pods nuk shton automatikisht node capacity në çdo cluster.

StorageClass dhe reclaim policy përcaktojnë lifecycle; access modes nuk janë protokoll aplikacioni për replikim. Upgrade kërkon kontroll version skew, API të hequra, add-ons dhe provë restore. Backup i manifest-eve pa persistent data, secrets dhe varësitë e jashtme nuk rikthen të gjithë shërbimin.

Shembull: dy replicas ndodhen në të njëjtin node dhe PDB thotë minAvailable 1. Humbja e node-it mund t'i humbasë të dyja njëherësh; PDB nuk e pengon dështimin fizik.

Laborator: Izolimi, storage, upgrades dhe politika

Koha: 180 minuta. **Mjedisi:** cluster lokal dhe politika shembull; praktikë vetëm në namespace të kursit.

- Apliko policy default deny dhe lejen e klientit sipas README-së. Kontrollo një pod me label të lejuar dhe një pa të.
- Verifiko nëse CNI e zbaton politikën. Nëse të dy klientët kalojnë, mos e regjistro testin si sukses.
- Shkruaj role read-only dhe krahaso leximin e pods me tentativën e ndryshimit të Deployment-it.
- Defekti: pas default deny pod-i nuk zgjidh DNS. Identifiko trafikun e nevojshëm dhe semantikën e DNS të cluster-it përpara lejes së përgjithshme.
- Projekto upgrade me pilot node pool, kontroll API compatibility dhe rollback/restore të workload-it.

Dorëzimi: provat positive/negative dhe plani upgrade. **Pranimi:** enforcement demonstrohet; PDB nuk mbivlerësohet; namespace nuk paraqitet si kufi absolut. **Thellim:** projekto multi-tenancy për dy klientë të jashtëm me rrezik të ndryshëm dhe justifiko ndarjen e cluster-it ose të planeve të kontrollit.

23. Infrastructure as Code

Terraform/OpenTofu, state, plan dhe drift

Teori: Terraform/OpenTofu, state, plan dhe drift

Objektivi: të menaxhosh gjendjen e infrastrukturës me ndryshime të rishikueshme dhe të kontrolluara.

IaC përshkruan burimet dhe marrëdhëniet me kod/deklarata. Terraform/OpenTofu përdorin providers, dependency graph dhe state për lidhjen me burimet reale. Plan përllogarit ndryshimet në kontekstin e konfigurimit dhe gjendjes së lexuar; nuk garanton që kapaciteti ose lejet do të mbeten të njëjta në apply. Ndryshimi i identitetit ose rajonit mes plan dhe apply mund të jetë veçanërisht i rrezikshëm.

State përmban metadata dhe mund të përmbajë secrets. Shënimi sensitive fsheh disa paraqitje, por nuk është vetë kriptim i storage-it. Backend-i duhet të kontrollojë aksesin, versionimin, encryption dhe locking sipas aftësive. Lock nuk mbron dy state të ndryshme që menaxhojnë të njëjtin resource. Ndaj pronësinë e burimeve dhe kufijtë e mjediseve.

Modules krijojnë ndërtim të përsëritshëm kur kontrata, versionet dhe defaults janë të qarta. Workspaces nuk janë gjithmonë kufi i mjaftueshëm sigurie për prod/dev. Drift vjen nga ndryshime jashtë kodit ose përditësime të shërbimit; duhet të zbulohet dhe të trajtohet, jo të pranohet verbërisht.

Import lidh burim ekzistues me state sipas workflow; nuk garanton se konfigurimi i gjeneruar mbulon gjithë politikën e dëshiruar. Destroy kërkon shqyrtim të scope-it, backups dhe retention. Pipeline duhet të dallojë plan lokal pa provider nga apply që krijon burime me pagesë.

Shembull: shembulli IaC i kursit përdor vetëm `terraform_data` dhe validim lokal, pa cloud provider. Ai demonstron workflow/state, jo krijimin e një VPC reale.

Laborator: Terraform/OpenTofu, state, plan dhe drift

Koha: 150 minuta. **Mjedisi:** OpenTofu ose Terraform i përshtatshëm; shembulli lokal është te `iac/`.

- Lexo `main.tf` dhe konfirmo mungesën e provider-it cloud dhe kredencialeve. Zgjidh një mjet për të gjithë ciklin.
- Ekzekuto fmt, init, validate dhe plan në dosjen e laboratorit. Lexo resource-in lokal që do të ndryshojë para apply.
- Ndrysho environment nga dev në një vlerë të palejuar; verifiko refuzimin nga validation. Riktheje dhe provo ndryshim të tags.
- Defekti analitik: dy pipeline bëjnë apply mbi të njëjtën gjendje. Projekto locking, serializim dhe kontroll të identitetit/mjedisit.

- Shpjego pse state nuk duhet dorëzuar publikisht për një shembull real me secrets, edhe kur output është shënuar sensitive.

Dorëzimi: plan i redaktuar dhe rezultat validimi. **Pranimi:** nuk krijohen burime cloud; dallohet plan nga apply; scope i destroy lokal është i kuptuar. **Thellim:** projekto ndarjen e state për network, platform dhe workload me dependencies dhe proces recovery kur backend nuk arrihet.

24. CI/CD dhe GitOps

Promovimi i artefaktit, rollout dhe rollback

Teori: Promovimi i artefaktit, rollout dhe rollback

Objektivi: të shpërndash një version me prova dhe mundësi ndalimi përpara se të ndikojë gjithë përdoruesit.

CI kontrollon ndryshimet me build, teste dhe analiza. Continuous delivery përgatit publikim të kontrolluar; deployment e zbaton sipas procesit. Artefakti duhet të ndërtohet një herë dhe të promovohet me identifikues të qëndrueshëm. Rindërtimi në prod mund të marrë dependencies të tjera edhe kur kodi është i njëjtë.

Rolling rollout zëvendëson gradualisht instancat; blue/green mban dy mjedise; canary provon një pjesë trafiku ose përdoruesish. Criteret e suksesit përfshijnë gabimet, latency dhe rezultatet e biznesit. Rollback i image nuk e kthen automatikisht schema migration ose efektet e jashtme. Expand-contract dhe feature flags ndihmojnë përputhshmërinë.

GitOps përdor gjendjen e deklaruar dhe controller që pajton mjedisin. Repository bëhet burim i dëshiruar, por secrets, akseset dhe emergjencat duhen trajtuar veçmas. Ndryshimi manual mund të rikthehet nga controller; operacionet emergjente duhet të përputhen me modelin. Git history nuk është vetë backup i databazës.

Pipeline identity duhet të ketë scope mjedisi dhe kredenciale të shkurtra. Input nga pull requests të jashtme nuk duhet të marrë secrets prodhimi. Skanimi, nënshkrimi, approval i mjedisit dhe post-deploy checks plotësojnë njëra-tjetrën. Vetëm statusi i komandës deploy nuk provon shëndetin e aplikacionit.

Shembull: canary ka error rate 10% kundrejt 0.2% baseline. Pipeline duhet të ndalë promovimin dhe të ruajë provat, edhe nëse pods janë Ready dhe liveness kalon.

Laborator: Promovimi i artefaktit, rollout dhe rollback

Koha: 150 minuta. **Mjedisi:** projektim pipeline dhe aplikacioni lokal; CI i jashtëm nuk kërkohet.

- Ndërto rrjedhën test→build→scan→publish→deploy test→post-check→promote. Për çdo hap cakto input/output dhe identitetin.
- Përdor testet lokale të aplikacionit si gate të vërtetë; ndrysho payload contract në një kopje për të demonstruar refuzimin e regresionit.
- Projekto canary 5% me prag gabimesh dhe interval vëzhgimi. Shpjego cilat metrika kanë kampion të mjaftueshëm.
- Defekti: image rollback punon, por skema e re është e papërputhshme. Shkruaj plan expand-contract dhe forward fix.
- Për GitOps përcakto kush ndryshon repository, kush e zbaton dhe si regjistrohet ndërhyrja emergjente.

Dorëzimi: pipeline, gates dhe procedurë rollback. **Pranimi:** i njëjti artefakt promovohet; secrets nuk u jepen PR-ve të jashtme; aplikacioni testohet pas deploy. **Thellim:** krahaso canary sipas përqindjes së trafikut me canary sipas tenant-it për sistem me ngarkesë shumë të pabarabartë.

25. Platform engineering

Golden paths, self-service dhe kontratat e platformës

Teori: Golden paths, self-service dhe kontratat e platformës

Objektivi: të krijosh një platformë që ul punën e përsëritur pa fshehur përgjegjësitë kritike.

Platform engineering ofron aftësi të brendshme si produkte për ekipet e aplikacioneve: templates, provisioning, deploy, observability dhe politika. Golden path është rruga e mbështetur që mbulon rastet e zakonshme me defaults të provuara. Nuk duhet të bëhet rrugë e vetme që pengon çdo kërkesë tjetër; nevojiten përjashtime dhe versionim i kontratës.

Self-service nuk do të thotë leje të pakufizuar. Një kërkesë mund të zgjedhë madhësi, rajon dhe klasë të dhënash brenda katalogut të lejuar. Platforma zbaton quotas, identitet, naming, logs dhe cost ownership. Template që hap akses publik si default për të kursyer hapa i zhvendos problemet te çdo ekip.

Përcakto kush zotëron aplikacionin, runtime, cluster-in, bazën e të dhënave dhe incidentet. SLO e platformës dhe SLO e workload-it janë të lidhura por jo identike. Developer experience matet me kohën deri te deploy i besueshëm, numrin e hapave manualë dhe incidentet, jo vetëm numrin e butonave në portal.

Katalogu i shërbimeve mban pronar, repository, tier, dependencies dhe runbooks. Lifecycle përfshin deprecation, migrim dhe pastrim të burimeve që nuk kanë më pronar. API e platformës duhet të jetë idempotente dhe të raportojë gjendje të pjesshme, sepse provisioning mund të zgjasë dhe të dështojë në mes.

Shembull: një ekip kërkon API me queue dhe databazë. Platforma jep identitete të ndara, dashboard, backup policy dhe pipeline; ekipi ende përgjigjet për invariants e porosisë dhe kodin.

Laborator: Golden paths, self-service dhe kontratat e platformës

Koha: 120 minuta. **Mjedisi:** dokumentim lokal dhe skemat shembull të kursit.

- Projekto një ofertë self-service “API me worker”. Përcakto fushat e kërkesës dhe cilat vlera lejohen.
- Shkruaj kontratën: input, burimet e krijuara, identitetet, monitorimi, kostoja dhe procesi i heqjes.
- Përgatit RACI të thjeshtë për aplikacion, platform, security dhe financë. Cakto pronarin e incidentit jashtë orarit.
- Defekti: provisioning krijon databazën por dështon te DNS. Jep gjendjen që sheh përdoruesi dhe si riprovohet pa krijuar databazë tjetër.
- Identifiko tri matje të përvojës së zhvilluesit dhe një mënyrë për të marrë feedback.

Dorëzimi: kontrata dhe RACI. **Pranimi:** defaults janë të kufizuara; gjendja e pjesshme përfaqësohet; përdoruesi nuk ka nevojë të njohë çdo hollësi të provider-it për rastin e zakonshëm. **Thellim:** projekto upgrade të versionit të golden path pa ndryshuar të gjitha aplikacionet njëherësh.

26. Observability

Metrics, logs, traces dhe konteksti i kërkesës

Teori: Metrics, logs, traces dhe konteksti i kërkesës

Objektivi: të shpjegosh sjelljen e sistemit nga sinjalet dhe të ndjekësh një porosi përmes komponentëve.

Metrics japin vlera të agreguara në kohë; logs ngjarje me kontekst; traces lidhje shkakësore ndërmjet spans sipas instrumentimit. Një request ID mund të lidhë logs pa qenë vetë trace i plotë. OpenTelemetry jep standarde dhe mjete instrumentimi/eksporti; backend-i i ruajtjes dhe analizës është zgjedhje tjetër.

Për API ndiq rate, errors dhe duration; për burimet utilization, saturation dhe errors. Për queue ndiq backlog, oldest age, throughput dhe failures. Histograms ndihmojnë percentilet, por buckets, aggregation dhe sampling ndryshojnë interpretimin. Nuk nxirret p99 i përgjithshëm duke mesatarizuar verbërisht p99 e instancave.

High-cardinality labels si order_id në çdo metric mund të shpërthejnë numrin e series dhe koston. Përdor identifikues të veçantë në logs/traces dhe labels të kontrolluara në metrics. Mos regjistro secrets, tokens ose payload me të dhëna personale pa nevojë e kontroll. Retention, sampling dhe redaction janë pjesë e projektimit.

Alarmi duhet të ketë ndikim, pronar, veprim dhe kriter mbylljeje. Një dashboard i gjelbër që mat vetëm CPU nuk e provon rrjedhën e pagesës. Synthetic checks provojnë rrugë të njohura; sinjalet e përdoruesit japin shtrirjen reale. Monitoro edhe humbjen e vetë telemetrisë.

Shembull: API kthen 202, por worker-i nuk përfundon asgjë. Vetëm success rate HTTP e fsheh incidentin; mosha e porosive pending dhe output rate tregojnë problemin e biznesit.

Laborator: Metrics, logs, traces dhe konteksti i kërkesës

Koha: 150 minuta. **Mjedisi:** API/worker lokal dhe endpoint `/metrics`.

- Krijo tri porosi dhe lexo metrics për statuset. Ndalo worker-in dhe shiko pse suksesi API nuk tregon përfundim pune.
- Lidh një order ID me përgjigjen dhe statusin në databazë. Përcakto cilat fusha duhen në log strukturor dhe cilat nuk duhen.
- Projekto një histogram latency dhe shpjego pse mesatarja nuk mjafton për bishtin e vonesës.
- Defekti: metric label përdor çdo request ID. Vlerëso rritjen e series dhe zhvendose identifikuesin te logs/traces.
- Përgatit alarm për porosi pending të vjetra dhe alarm për mungesë të metrics, me pronar dhe runbook.

Dorëzimi: dashboard i skicuar, metrics reale dhe alarmet. **Pranimi:** përfshihet rrjedha e biznesit; cardinality kontrollohet; mungesa e sinjalit nuk interpretohet si zero gabime. **Thellim:** projekto trace propagation për

API→queue→worker, duke shpjeguar spans asinkrone dhe sampling.

27. SRE dhe besueshmëria

SLI, SLO, error budget dhe incidente

Teori: SLI, SLO, error budget dhe incidente

Objektivi: të lidhësh objektivat e shërbimit me matjet dhe vendimet e publikimit.

SLI është matje, SLO objektiv dhe SLA marrëveshje sipas kushteve përkatëse. Për API mund të matet përqindja e kërkesave të vlefshme që përfundojnë brenda latency të caktuar. Për queue matja duhet të përfshijë përfundimin e punës brenda afatit, jo vetëm pranimin e mesazhit. Përkufizo emëruesin, përjashtimet dhe dritaren para incidentit.

Error budget është pjesa e lejuar e dështimeve sipas SLO. Burn rate krahason përdorimin e buxhetit me normën e pritshme dhe mund të alarmojë në dritare të ndryshme. Pragu duhet të shmangë reagim të tepruar ndaj pak kërkesave, por të mos fshehë incident të madh. SLO 100% zakonisht nuk jep buxhet pune realist për ndryshime dhe dështime.

Toil është punë operative e përsëritur që mund të reduktohet; automatizimi duhet të heqë mekanizmin e përsëritjes, jo të fshehë çdo gabim. Runbooks dhe on-call kanë nevojë për shtrirje të qartë dhe eskalim. Incident commander, komunikimi dhe mitigimi ndahen kur incidenti është i madh.

Postmortem shpjegon shkakun teknik, kushtet që e mundësuan, ndikimin dhe veprimet parandaluese me pronar. Chaos engineering nis nga hipotezë e kontrolluar, blast radius dhe stop conditions; nuk është prishje e rastësishme në prodhim. Testo fillimisht në laborator dhe mos krijo prova më të rrezikshme se aftësia e rikthimit.

Shembull: nëse buxheti i gabimeve po shterohet, ekipi mund të ndalë funksione të reja dhe të adresojë retry storms ose alarmet e dobëta. Vendimi lidhet me objektivin e përdoruesit.

Laborator: SLI, SLO, error budget dhe incidente

Koha: 150 minuta. **Mjedisi:** matje laboratorike dhe të dhëna hipotetike.

- Për 100 000 kërkesa me SLO 99.9% llogarit numrin e gabimeve të lejuara. Cakto cilat status codes numërohen sipas kontratës.
- Për porositë cakto SLO “përfundim brenda 60 s” dhe përkufizo emëruesin në prani të kërkesave të pavlefshme.
- Ndalo worker-in në laborator, mat simptomën dhe përgatit runbook për rikthim. Mos ndrysho disa komponentë njëherësh.
- Defekti: të gjitha alarmet bazohen në CPU, por porositë mbeten pending. Zëvendëso me sinjal të lidhur me përdoruesin.

- Shkruaj postmortem të shkurtër me tri veprime, pronarë dhe kriter verifikimi.

Dorëzimi: SLI/SLO, buxheti dhe raporti i incidentit. **Pranimi:** 99.9% lejon 100 gabime në 100 000 kërkesa sipas modelit; matja nuk ndryshohet pas incidentit për ta fshehur; veprimet janë të testueshme. **Thellim:** projekto alarm burn-rate me dritare të shkurtër/gjatë dhe shpjego kompromisin e shpejtësisë me false positives.

28. Siguria e infrastrukturës cloud

Threat modeling, ekspozimi dhe reagimi

Teori: Threat modeling, ekspozimi dhe reagimi

Objektivi: të lidhësh kontrollet me rrugë sulmi konkrete dhe të projektosh reagim pa humbur provat.

Threat model identifikon asetet, identitetet, trust boundaries, hyrjet dhe keqpërdorimet e mundshme. Kontrollet duhet të mbulojnë identitetin, rrjetin, workload-in, të dhënat dhe supply chain. CSPM zbulon konfigurime, ndërsa mbrojtja runtime zbulon sjellje; produkti i vetëm nuk zëvendëson aplikacionin me autorizim korrekt.

Ekspozimi publik analizohet sipas rrugës reale: adresë, route, firewall, endpoint dhe autentikim. Portë e mbyllur nuk mbron një secret të rrjedhur që mund të përdoret përmes API publike të ofruesit. SSRF, metadata access dhe identity scope mund të lidhen në një zinxhir. Least privilege dhe segmentimi e kufizojnë blast radius kur një komponent kompromentohet.

Logging duhet të mbulojë ndryshimet e kontrollit dhe aktivitetin e të dhënave sipas kërkesës; jo çdo audit category është aktive si default. Destinacioni i log-eve kërkon mbrojtje nga ndryshimi/fshirja prej identiteteve të aplikacionit. Ruaj kohën, identitetin, resource dhe correlation ID për hetim.

Incident response përfshin zbulimin, triage, izolimin, ruajtjen e provave, revokimin/rotation dhe rikuperimin nga gjendje e besuar. Fshirja e menjëhershme e VM-së mund të humbasë prova. Për disa raste containment përmes IAM ose rrjetit është më i përshtatshëm përpara imaging sipas procedurës. Veprimet në prodhim kërkojnë ekipin dhe procesin përkatës; laboratori përdor vetëm të dhëna fiktive.

Shembull: një token pipeline rrjedh. Ndryshimi i password të përdoruesit nuk e revokon domosdoshmërisht çdo session të lëshuar; kërko identifikimin e token-it, afatin dhe kontrollin e aksesit te resource.

Laborator: Threat modeling, ekspozimi dhe reagimi

Koha: 150 minuta. **Mjedisi:** tabletop dhe audit log fiktiv; nuk përdoren llogari të kompromentuara reale.

- Vizato trust boundaries për klient→API→queue→worker→database dhe pipeline→deployment.
- Përzgjidh katër kërcënime: akses objekti të huaj, secret në image, token me scope të gjerë dhe databazë publike. Cakto parandalim dhe zbulim.
- Për një token të rrjedhur krijo timeline containment→revokim→hetim→recovery; identifiko çfarë prove ruhet.
- Defekti: log-et e auditit mund të fshihen nga i njëjti rol që ndryshon infrastrukturën. Rishiko ndarjen e detyrave dhe destinacionin.
- Përgatit test për një kontroll dhe një mënyrë për të provuar që alarmi arrin te pronari.

Dorëzimi: threat model dhe runbook incidenti. **Pranimi:** kontrollet lidhen me kërcënime; secrets nuk dorëzohen; ruajtja e provave nuk harrohet. **Thellim:** projekto identifikimin e lateral movement mes dy llogarive dhe kufizo besimin cross-account pa bllokuar shërbimin legjitim.

29. Kriptimi dhe mbrojtja e të dhënave

KMS, envelope encryption dhe cikli i çelësave

Teori: KMS, envelope encryption dhe cikli i çelësave

Objektivi: të dallosh kontrollin e aksesit nga kriptimi dhe të projektosh lifecycle të të dhënave/çelësave.

Encryption in transit mbron komunikimin sipas TLS ose mekanizmit përkatës; at rest mbron të dhënat e ruajtura sipas shërbimit; in use kërkon teknologji dhe model kërcënimi të veçantë. Asnjëri nuk e korrigjon autorizimin e gabuar të aplikacionit. Një përdorues i lejuar gabimisht mund të marrë plaintext përmes një shërbimi që dekripton ligjërisht për të.

Envelope encryption përdor data key për të dhënat dhe key-encryption key për mbrojtjen e data key, sipas implementimit. KMS/HSM menaxhon operacione dhe politika me aftësi të ndryshme. Çelësi i menaxhuar nga klienti jep kontroll shtesë por sjell rrezik operacional: çaktivizimi ose fshirja mund të bllokojë backup dhe shërbime që varen prej tij.

Rotation mund të ndryshojë materialin për operacione të reja pa rishkruar automatikisht gjithë të dhënat e vjetra. Re-encryption dhe rewrapping janë procese me kuptim tjetër. Key policy, IAM, audit dhe identiteti i shërbimit duhet të përputhen. Mos supozo se kopjimi ndërrajonal sjell vetë të njëjtin çelës dhe leje.

Data classification përcakton nivelin, pronarin, retention dhe vendndodhjen e kërkuar. Tokenization, masking dhe minimization ulin ekspozimin sipas rastit. Backup, replica, logs dhe analytics krijojnë kopje që hyjnë në lifecycle. Fshirja e rekordit primar nuk garanton heqjen e çdo kopjeje; procesi dokumentohet sipas kërkesave dhe sistemit.

Shembull: disaster recovery ka kopjen e databazës në rajon tjetër, por çelësi ose lejet nuk janë të përdorshme atje. Të dhënat ekzistojnë por shërbimi nuk rikthehet.

Laborator: KMS, envelope encryption dhe cikli i çelësave

Koha: 120 minuta. **Mjedisi:** projektim me të dhëna fiktive; nuk krijohen çelësa që mbrojnë prodhim.

- Klasifiko email, ID porosie, shuma dhe logs. Cakto pronarin, nevojën e ruajtjes dhe redaction.
- Vizato envelope encryption dhe kush lejohet të kërkojë decrypt për API, worker dhe operator.
- Krijo plan për backup ndërrajonal që përfshin çelësin, identitetin dhe provën restore.
- Defekti: key çaktivizohet gjatë “pastrimit”, duke prishur backup. Cakto kontrolle para fshirjes dhe alarm për varësi.
- Shpjego dallimin rotation/re-encryption dhe si verifikohet që kopjet e vjetra mbeten të lexueshme gjatë tranzicionit.

Dorëzimi: diagram besimi dhe matricë lifecycle. **Pranimi:** kriptimi nuk quhet autorizim; të gjitha kopjet e të dhënave merren parasysh; çelësat kanë recovery plan. **Thellim:** vlerëso confidential computing për një model kërcënimi konkret dhe shpjego çfarë nuk mbulon ndaj aplikacionit të kompromet

30. Governance dhe kontrolli i organizatës

Policy as code, evidenca dhe përjashtimet

Teori: Policy as code, evidenca dhe përjashtimet

Objektivi: të bësh rregullat të testueshme dhe të ruash evidencë të sjelljes reale.

Governance përcakton kush vendos rregullat, si zbatohen dhe si rishikohen. Policy as code mund të kontrollojë IaC, API ose gjendjen e zbuluar. Një kontroll në pull request nuk sheh domosdoshmërisht ndryshimet manuale pas deploy; kombinimi preventive/detective jep pamje më të plotë. Kontrolli duhet të jetë i lidhur me kërkesë dhe pronar.

Configuration inventory mban asetet, pronarët, klasifikimin dhe lidhjet. Tags/labels ndihmojnë atribuimin, por kërkojnë enforcement dhe proces kur mungojnë. Audit evidence përfshin burimin, kohën dhe versionin e policy. Screenshot pa kontekst mund të mos provojë se kontrolli ishte aktiv gjatë intervalit të kërkuar.

Përjashtimi duhet të ketë arsye, scope, afat dhe kompensim. Një allowlist e përhershme pa pronar bëhet rrugë e pakontrolluar. Remediation automatike duhet të kufizohet: fshirja e një resource vetëm sepse mungon tag mund të shkaktojë ndërprerje më të madhe se problemi fillestar. Dry-run dhe klasifikimi sipas rrezikut ndihmojnë.

Data residency, sovereignty dhe kërkesat rregullatore nuk zgjidhen vetëm duke zgjedhur rajonin e VM-së; logs, backups, support dhe shërbimet globale mund të kenë modele të tjera. Në këtë kurs ndërtohet hartë teknike dhe evidencë; interpretimi detyrues i rregullave kërkon pronarët ligjorë/compliance të organizatës dhe burime aktuale.

Shembull: policy “storage privat” kontrollon konfigurimin e bucket-it dhe rrugët e aksesit. Një përjashtim për assets publike mund të jetë i arsyetuar kur scope-i është i veçantë dhe pa të dhëna personale.

Laborator: Policy as code, evidenca dhe përjashtimet

Koha: 120 minuta. **Mjedisi:** validator lokal i arkitekturës nga `automation/` dhe dokumentim.

- Ekzekuto validator-in mbi arkitekturën shembull. Lexo cilat fusha kontrollon dhe cilat nuk i njeh.
- Hiq owner dhe aktivizo storage public në një kopje me klasë private. Verifiko refuzimet veçmas dhe korrigjoji.
- Shkruaj një përjashtim me afat për asset publik dhe përcakto evidencën që e kufizon vetëm aty.
- Defekti: një policy e pipeline-it kalon, por operatori ndryshon konfigurimin manualisht. Projekto zbulim drift dhe eskalim.
- Krijo regjistër evidence me kontroll, datë, version, rezultat dhe skedar prove.

Dorëzimi: arkitekturë e vlefshme, dy refuzime dhe një përjashtim. **Pranimi:** validator-i nuk pretendohet si certifikim compliance; afatet kanë pronar; kontrollet merren edhe pas deploy. **Thellim:** projekto remediation me fazë vëzhgimi dhe kufi blast radius për një organizatë me qindra projekte.

31. FinOps dhe ekonomia e cloud-it

Kostot, unit economics, buxhetet dhe optimizimi

Teori: Kostot, unit economics, buxhetet dhe optimizimi

Objektivi: të llogaritësh koston e workload-it dhe të dallosh kursimin real nga zhvendosja e shpenzimit.

Fatura mund të përfshijë compute, storage, kërkesa, databaza, licenca, logs, backups, NAT, IP dhe transferim. Një VM e fikur mund të lërë volume ose adresa të faturueshme sipas platformës. Kostoja e ekipit dhe operimit hyn në total cost of ownership, edhe kur nuk shfaqet në faturën e ofruesit.

FinOps lidh inxhinierinë, financën dhe biznesin. Allocation përdor llogari, tags dhe modele për ndarjen e kostove. Showback i tregon ekipit përdorimin; chargeback ia atribuon financiarisht sipas modelit. Unit economics, si kosto për 1000 porosi të përfunduara, mund të jetë më e dobishme se vetëm totali mujor kur ndryshon volumi.

Rightsizing bazohet në matje të CPU, memory, I/O dhe kulmeve; mesatarja nuk mjafton. Scheduling për dev, storage lifecycle, ulja e egress dhe kontrolli i logs shpesh japin kursime. Commitments/reservations lidhen me kushte afati, scope dhe fleksibiliteti; blerja për ngarkesë të paqëndrueshme mund të krijojë shpenzim të papërdorur. Spot kërkon tolerancë ndërprerjesh.

Budget alert nuk është domosdoshmërisht hard spending cap. Vonesa e faturimit dhe njoftimit mund të lejojë shpenzim shtesë; kontrollet teknike dhe quotas kanë rol tjetër. Anomaly detection kërkon pronar dhe proces reagimi. Kursi përdor tarifa hipotetike me qëllim të shprehur; ato nuk janë oferta ose çmime aktuale.

Shembull: ulja e kostos së compute me 20% nuk është fitore nëse egress dhe incidents rriten më shumë. Krahaso të njëjtin nivel shërbimi dhe të njëjtin volum të suksesshëm.

Laborator: Kostot, unit economics, buxhetet dhe optimizimi

Koha: 150 minuta. **Mjedisi:** `automation/cost_model.py` dhe `cost-example.json`, pa llogari cloud.

- Ekzekuto modelin dhe lexo përbërësit e totalit. Verifiko compute, storage, egress dhe requests me llogaritje manuale.
- Ndrysho numrin e instancave, orët dhe volumin e porosive. Krahaso totalin me koston për 1000 porosi të përfunduara.
- Krijo skenar baseline, rritje dyfish dhe disaster recovery. Shëno çfarë nuk mbulon modeli: taksa, discounts, logs dhe tarifa të veçanta.
- Defekti: budget njofton pas vonese, por deploy vazhdon të krijojë burime. Shto quota, maksimum autoscaling dhe kontroll lifecycle.

- Për zbatim real zëvendëso tarifat me vlera nga kalkulatori zyrtar dhe regjistro datën, rajonin dhe monedhën.

Dorëzimi: tre skenarë dhe plan optimizimi. **Pranimi:** numrat hipotetikë etiketohen; buxheti nuk quhet hard cap; cilësia e shërbimit ruhet në krahasim. **Thellim:** vlerëso break-even të një commitment kur shfrytëzimi është 40%, 70% dhe 95%, duke përdorur kushtet konkrete të produktit.

32. Backup dhe disaster recovery

RTO, RPO, restore dhe strategjitë rajonale

Teori: RTO, RPO, restore dhe strategjitë rajonale

Objektivi: të provosh se të dhënat dhe shërbimi mund të rikthehen brenda objektivit të zgjedhur.

High availability synon vazhdimësi për dështime të caktuara; disaster recovery rikthen pas skenarëve më të rëndë; backup krijon kopje të rikthyeshme sipas politikës. Replica mund të shpërndajë fshirje ose korruptim logjik, ndaj nuk është zëvendësim automatik për backup. RPO përcakton humbjen e pranueshme të të dhënave; RTO kohën e synuar të rikuperimit.

Backup/restore, pilot light, warm standby dhe active-active kanë kosto dhe kompleksitet të ndryshëm. Active-active për shkrime kërkon strategji konfliktesh dhe invariants të përshtatshme. Një sistem mund të jetë active-active në frontend dhe single-writer për data; emërtimi duhet të jetë i saktë për shtresën.

Plani përfshin të dhënat, kodin, infrastrukturën, identitetet, çelësat, DNS, secrets, certifikatat dhe kapacitetin rezervë. Restore pa çelës ose pa leje të sakta nuk e rikthen shërbimin. Failback mund të kërkojë sinkronizim të gjendjes së krijuar në site rezervë; nuk është thjesht kthim DNS.

Testet e rikthimit përdorin të dhëna të verifikueshme dhe masin kohën nga vendimi deri te funksioni i pranuar. Integrity e skedarit nuk garanton konsistencë të aplikacionit. Retention, kopje në kufi tjetër identiteti dhe mbrojtja nga fshirja përzgjidhen sipas modelit të kërcënimit. Dokumento varësitë që mungojnë në simulimin lokal.

Shembull: backup çdo orë nuk garanton RPO një orë nëse job-et kanë dështuar gjithë natën. Duhet monitorim i suksesit dhe prova restore, jo vetëm konfigurim scheduler-i.

Laborator: RTO, RPO, restore dhe strategjitë rajonale

Koha: 180 minuta. **Mjedisi:** laborator Python lokal me të dhëna prove dhe direktori backup të veçantë.

- Krijo disa porosi dhe kryej backup me komandën e ofruar. Regjistro kohën dhe ID-të që duhet të ekzistojnë në kopje.
- Krijo porosi të re pas backup-it. Rikthe backup-in në skedar tjetër, pa mbishkruar databazën aktive.
- Nis API kundrejt kopjes në portë tjetër dhe provo gjendjen. Krahaso cilat porosi janë rikthyer dhe cilat bien jashtë pikës së backup-it.
- Defekti: kopja ekziston por nuk lexohet për shkak të lejeve ose key access në modelin cloud. Zgjero checklist për të gjitha varësitë.
- Shkruaj plan failback dhe kriteret e pranimit pas rikthimit.

Dorëzimi: backup, rezultat restore dhe RTO/RPO të matur të laboratorit. **Pranimi:** origjinali nuk mbishkruhet; të dhënat verifikohen përmes aplikacionit; matja lokale nuk shpallet garanci rajonale. **Thellim:** projekto rikuperim nga fshirje keqdashëse kur identiteti prod ka kompromet uar edhe replikën online.

33. Migrimi në cloud

Inventari, varësitë, cutover dhe modernizimi

Teori: Inventari, varësitë, cutover dhe modernizimi

Objektivi: të zgjedhësh strategjinë për workload dhe të planifikosh kalimin me prova.

Migrimi nis me inventar të aplikacioneve, të dhënave, integrimeve, licencave dhe pronarëve. Discover nuk është vetëm listë VM-sh; kërkon dependencies, jobs periodike, identitete dhe flukse. Rehost, replatform, refactor, repurchase, retire dhe retain përshkruajnë vendime të ndryshme; taksonomitë mund të shtojnë kategori sipas framework-ut.

Rehost ul ndryshimin e kodit por mund të ruajë inefficiency dhe punë operative. Replatform ndryshon disa shtresa si databaza managed; refactor ndryshon arkitekturën dhe sjell rrezik/dobi më të madhe. Mos modernizo çdo komponent njëkohësisht pa mundësi lokalizimi të regresionit.

Data migration mund të përdorë kopje fillestare dhe change data capture për të ulur dritaren e ndërprerjes. Kontrollo schema compatibility, encoding, timezone, sequences, indexes dhe kontratat e aplikacionit. Row counts nuk provojnë të vetme korrektësinë: duhen checksums ose kontroll biznesi sipas rastit.

Cutover ka freeze/dual-write policy, plan DNS, routing, final sync, smoke tests dhe rollback deadline. Dual writes pa koordinim krijojnë divergjencë. Pas kalimit mbaj monitorim të krahasueshëm dhe mos fshi burimin para verifikimit dhe afatit të rënë dakord. Rollback pas shkrimeve në destinacion kërkon plan për të dhënat e reja.

Shembull: API zhvendoset por job-i i natës ende shkruan në databazën e vjetër. Sistemi duket mirë ditën e migrimit dhe ndahet në dy burime të së vërtetës më vonë.

Laborator: Inventari, varësitë, cutover dhe modernizimi

Koha: 150 minuta. **Mjedisi:** dokumentim dhe dy kopje lokale të të dhënave testuese.

- Inventarizo aplikacionin e porosive: API, worker, databazë, konfigurim, identitete dhe backup. Shto një job periodik hipotetik.
- Zgjidh rehost/replatform/refactor për secilin komponent dhe jep kriterin që e justifikon.
- Përgatit runbook cutover me final sync, checks dhe pikën e vendimit go/no-go.
- Defekti: pas cutover ka shkrime në të dy databazat. Përcakto autoritetin dhe planin e reconciliimit para kthimit mbrapsht.
- Provo migrim të kopjes SQLite vetëm si demonstrim i transferimit dhe verifikimit; mos e quaj provë CDC të një motori tjetër.

Dorëzimi: inventari, strategjia dhe runbook. **Pranimi:** varësitë përfshijnë jobs/identitet; rollback i të dhënave ka plan; orari i matjes dhe deadline janë të qarta. **Thellim:** projekto migrim të një monoliti me strangler pattern dhe ndarje graduale të një funksioni pa ndryshuar çdo klient.

34. Hybrid dhe multicloud

Lidhshmëria, identiteti, portabiliteti dhe kompromiset

Teori: Lidhshmëria, identiteti, portabiliteti dhe kompromiset

Objektivi: të përcaktosh çfarë duhet të ndahet mes mjediseve dhe çfarë duhet të mbetet lokal.

Hybrid integron datacenter/on-premises me cloud; multicloud përdor disa ofrues. Kjo mund të vijë nga kërkesa biznesi, blerje kompanish ose specializim shërbimesh. Nuk jep automatikisht disponueshmëri më të lartë: varësitë ndërcloud mund të shtojnë latency, egress, kontroll aksesi dhe punë operative.

Lidhja mund të përdorë VPN, shërbime të dedikuara ose Internet me TLS sipas fluksit. Prefikset, DNS, certifikatat, MTU dhe routing-u i kthimit kërkojnë plan të përbashkët. Identiteti mund të federohet, por trust dhe audit duhet të mbeten të kufizuara. Humbja e IdP qendrore mund të prekë disa mjedise njëkohësisht.

Portabiliteti i image nuk nënkupton portabilitet të databazës, IAM, managed events ose observability. Kubernetes standardizon disa API aplikacionesh, ndërsa load balancer, storage dhe workload identity mbeten specifike. Abstraction layer shumë i gjerë mund të kufizohet te karakteristikat më të pakta të përbashkëta dhe të humbasë vlerën e shërbimeve managed.

Data gravity i bën transferimet e mëdha të kushtueshme dhe të ngadalta. Zgjidh vendosjen afër të dhënave kur është e arsyeshme. Recovery në ofrues tjetër kërkon prova të eksportit, importit, kapacitetit dhe identitetit; një diagram simetrik nuk e krijon këtë aftësi.

Shembull: frontend te një ofrues dhe databazë te tjetri shtojnë RTT për çdo query. Nëse kërkesa bën 20 query seriale, latency e rrjetit shumëzohet në rrjedhën e përdoruesit.

Laborator: Lidhshmëria, identiteti, portabiliteti dhe kompromiset

Koha: 150 minuta. **Mjedisi:** projektim me dy platforma fiktive dhe lidhje të parametrizuara.

- Për një workload ndaj compute, data, identitet dhe kontrolle. Shëno cilat duhen të njëjta dhe cilat mund të ndryshojnë.
- Modelimi: RTT ndërmjedis 30 ms dhe 20 query seriale. Llogarit komponentin e shtuar dhe propozo reduktim të calls ose vendosje tjetër.
- Krijo matricë portabiliteti për image, SQL schema, object data, event API dhe IAM.
- Defekti: DNS privat funksionon vetëm nga cloud A. Projekto forwarding dhe testin nga zyra/cloud B.
- Shkruaj plan daljeje nga një ofrues me afat, volum transferimi dhe kosto hipotetike të egress.

Dorëzimi: diagram dhe matricë kompromisesh. **Pranimi:** përfitimi lidhet me kërkesë; 20×30 ms jep 600 ms komponent serial sipas supozimit; varësitë globale identifikohen. **Thellim:** krahaso active-active ndërcloud me warm recovery në të dytin për një shërbim me invariants të forta të të dhënave.

35. Data engineering dhe analytics

Lake, warehouse, streaming dhe cilësia

Teori: Lake, warehouse, streaming dhe cilësia

Objektivi: të projektohet rrjedhë të dhënash me cilësi, lineage dhe kosto të kontrolluar.

Data lake ruan formate të ndryshme në storage të përshtatshëm; warehouse optimizon pyetjet analitike me modelin e vet; lakehouse kombinon aftësi mbi tabela/formate sipas teknologjisë. Emërtimi nuk garanton transaksione ose menaxhim skeme identik. OLTP për porositë dhe OLAP për raportet kanë modele ngarkese të ndryshme.

Batch përpunon grupe; streaming përpunon ngjarje në vazhdim. Event time dhe processing time ndryshojnë kur mesazhet vonohen. Watermark dhe allowed lateness përcaktojnë trajtimin e ngjarjeve të vona sipas motorit. Dublikimet dhe replay kërkojnë deduplikim dhe kontratë të qartë të rezultatit.

ETL transformon para ngarkimit; ELT transformon në platformën e destinacionit. Partitioning dhe clustering mund të ulin scan, por shumë partitions ose small files rrisin overhead. Columnar formats dhe compression ndihmojnë disa workload-e. Schema evolution duhet të mbrojë konsumatorët; ndryshimi i kuptimit të një fushe mund të jetë më i rrezikshëm se ndryshimi i tipit.

Data quality përfshin completeness, uniqueness, freshness dhe validity sipas përdorimit. Catalog dhe lineage tregojnë burimin dhe transformimet. Akseset duhet të ndjekin klasifikimin edhe në kopjet analitike; eksporti nga prod nuk e heq ndjeshmërinë. Kostoja e queries mund të varet nga bytes scanned ose compute i rezervuar.

Shembull: raporti ditor numëron pagesa dublikatë pas replay të stream-it. Kontrolli i transportit nuk mjafton; modeli i agregimit duhet të përdorë identitetin e event-it dhe semantikën e biznesit.

Laborator: Lake, warehouse, streaming dhe cilësia

Koha: 150 minuta. **Mjedisi:** CSV/JSON lokal me të dhëna fiktive dhe Python opsional.

- Krijo dhjetë events porosish me ID, event_time dhe amount; shto një dublikatë dhe një ngjarje të vonuar.
- Llogarit raportin naiv dhe atë me deduplikim. Shëno kontratën për updates/corrections.
- Projekto raw→validated→analytics me kontroll të freshness, schema dhe ownership.
- Defekti: query skanon gjithë historikun sepse nuk përdor partition filter. Jep plan partition dhe test që provon kufizimin e skanimit.
- Dokumento kush mund të shohë identifikuesit personalë në raw dhe çfarë maskohet në raport.

Dorëzimi: dataset fiktiv, dy rezultate dhe plan cilësie. **Pranimi:** event time dallohet nga processing time; dublikimi nuk rrit totalin e saktë; kopjet ruajnë klasifikimin. **Thellim:** projekto përditësim skeme kur konsumatorët e vjetër dhe të rinj ekzekutohen paralelisht.

36. AI, ML dhe workloads GPU

Training, inference, MLOps dhe kufijtë e të dhënave

Teori: Training, inference, MLOps dhe kufijtë e të dhënave

Objektivi: të dallosh kërkesat infrastrukturore të training dhe inference pa e reduktuar AI në zgjedhjen e një GPU-je.

Training përpunon të dhëna për të ndryshuar parametrat e modelit; inference përdor modelin për parashikim/gjenerim. Kërkesat për memorie, compute, throughput dhe latency ndryshojnë. GPU memory, interconnect, checkpointing dhe data loading mund të bëhen bottleneck; përshpejtuesi më i shtrenjtë nuk garanton shfrytëzim më të mirë.

MLOps përfshin versionim të të dhënave/modelit, eksperimente, vlerësim, regjistër artefaktesh, deploy dhe monitorim. Model drift dhe data drift kanë kuptime të ndryshme nga CPU saturation. Rollback duhet të lidhë kodin, modelin, tokenizer-in dhe transformimet. Provenance e të dhënave dhe lejet për përdorim nuk zgjidhen nga platforma compute.

Inference batch dhe online kanë objektiva të ndryshëm. Batching rrit efikasitetin por mund të shtojë pritje. Për LLM, context length, output tokens, concurrency dhe KV cache ndikojnë memorien/latency; model size nuk është faktori i vetëm. RAG shton retrieval dhe indeks vektorial; nuk garanton saktësi ose autorizim të dokumenteve. Filtrat e aksesit duhet të ruhen në retrieval dhe output.

Model APIs managed ndryshojnë përgjegjësitë, kuotat, rezidencën dhe faturimin. Mos dërgo të dhëna prodhimi pa model të miratuar përdorimi. Prompt injection është rrezik në trajtimin e përmbajtjes së jashtme; përgjigjja e modelit nuk duhet të fitojë automatikisht autorizim për veprime cloud.

Shembull: agjenti mund të sugjerojë plan infrastrukture, por identiteti dhe workflow vendosin nëse veprimi lejohet. Teksti i marrë nga një dokument nuk është urdhër i besuar për pipeline.

Laborator: Training, inference, MLOps dhe kufijtë e të dhënave

Koha: 150 minuta. **Mjedisi:** projektim me të dhëna fiktive; nuk kërkohet GPU ose API me pagesë.

- Ndaj një aplikacion inference në model serving, retrieval, identitet, logs dhe storage. Cakto SLO latency dhe kufijtë e të dhënave.
- Krijo plan matjeje për input size, tokens/output, concurrency, throughput dhe kosto për përgjigje të vlefshme.
- Projekto evaluation dataset me raste normale, dokument të paautorizuar dhe përmbajtje që kërkon veprim jashtë scope-it.

- Defekti: retrieval kthen dokument të tenant-it tjetër. Vendos filter identiteti dhe test negativ përpara vlerësimit të cilësisë së modelit.
- Për training projekto checkpoint, versionet dhe retry në ndërprerje të kapacitetit spot.

Dorëzimi: arkitekturë AI, kontratë të dhënash dhe plan evaluation. **Pranimi:** nuk kërkohet pagesë; akseset nuk delegohen te teksti i modelit; performance dhe korrektësi maten veçmas. **Thellim:** krahaso model API me self-hosted për traffic të qëndrueshëm duke përfshirë GPU utilization, staf, failover dhe përditësime.

37. Edge, IoT dhe puna offline

Pajisjet, sinkronizimi dhe lidhjet e pasigurta

Teori: Pajisjet, sinkronizimi dhe lidhjet e pasigurta

Objektivi: të projektosh vazhdimësi kur pajisja nuk ka lidhje të qëndrueshme me cloud-in.

Edge vendos përpunimin afër burimit për latency, bandwidth ose vazhdimësi lokale. IoT përfshin pajisje me identitet, firmware, telemetri dhe lifecycle fizik. Cloud control plane mund të menaxhojë fleet, por funksioni kritik mund të ketë nevojë të vazhdojë offline. Modeli i besimit ndryshon kur pajisja mund të preket fizikisht.

Protokolle si MQTT përdorin broker dhe QoS me garanci të përcaktuara në kufirin përkatës; ato nuk provojnë automatikisht exactly-once efekt biznesi në aplikacion. Store-and-forward ruan mesazhe lokalisht dhe i dërgon pas rikthimit. Duhet kapacitet disk, retention, identitet mesazhi dhe trajtim dublikimesh.

Clock drift dhe event ordering ndikojnë analizën kur pajisja qëndron offline. Përdor sequence/version dhe event time me kuptim të shprehur. Conflict resolution për konfigurim nuk duhet të lejojë që një version i vjetër të mbishkruajë pa kontroll një politikë më të re. Firmware update kërkon nënshkrim, rollout gradual dhe rikthim sipas aftësive hardware.

Certificate provisioning, rotation dhe revocation duhet të funksionojnë edhe me humbje pajisjesh. Shared secret për gjithë flotën zgjeron blast radius. Monitorimi duhet të dallojë pajisje offline, të fikur dhe të kompromet uar kur provat e lejojnë; mungesa e telemetrisë vetëm nuk e identifikon shkakun.

Shembull: 500 sensorë mbeten offline për një orë dhe dërgojnë gjithçka njëherësh pas rikthimit. Pa jitter, kufi dhe backlog management, sistemi cloud mund të përjetojë retry storm.

Laborator: Pajisjet, sinkronizimi dhe lidhjet e pasigurta

Koha: 120 minuta. **Mjedisi:** simulim lokal me skedarë/ngjarje fiktive; nuk lidhen pajisje industriale reale.

- Llogarit storage lokal për 10 mesazhe/minutë nga 100 pajisje me 1 KB për mesazh gjatë 24 orëve offline.
- Projekto identitet unik për pajisje dhe plan zëvendësimi/revokimi kur humbet njëra.
- Krijo rend ngjarjesh me timestamps jo të renditura dhe sequence numbers. Shpjego si trajtohet replay pas lidhjes.
- Defekti: e gjithë flota rilidhet njëherësh. Shto jitter, batching dhe rate limit që ruan freskinë e ngjarjeve kritike.
- Shkruaj rollout firmware me grup pilot dhe stop condition, duke mbajtur versionet e provuara.

Dorëzimi: modeli offline dhe plani i identitetit. **Pranimi:** dublikimet trajtohen; disku ka kufi/retention; një secret i vetëm për gjithë flotën nuk është zgjedhje default. **Thellim:** projekto reconciliim të konfigurimit kur

edge ndryshon lokalisht dhe qendra ka lëshuar version tjetër gjatë ndërprerjes.

38. Modele të avancuara arkitekturore

CQRS, saga, outbox dhe kontrolli i mbingarkesës

Teori: CQRS, saga, outbox dhe kontrolli i mbingarkesës

Objektivi: të zgjedhësh modelin vetëm kur zgjidh një problem të identifikuar dhe të ruash invariants e biznesit.

CQRS ndan modelin e shkrimit nga ai i leximit; kjo mund të optimizojë queries por krijon sinkronizim dhe staleness. Event sourcing ruan ngjarjet si bazë për rindërtimin e gjendjes; nuk është sinonim me dërgimin e events te broker-i. Replay kërkon versionim të skemës dhe kontroll mbi efektet e jashtme që nuk duhet të përsëriten.

Saga koordinon transaksione lokale me hapa dhe kompensime. Pagesa e kthyer nuk e bën të paqenë çdo veprim fizik si dërgimi i mallit. Transactional outbox ul hendekun mes commit-it lokal dhe publikimit, por konsumatorët ende duhet të trajtojnë dublikime. Inbox/dedup store mund të ruajë identitetin e ngjarjeve të përpunuara sipas modelit.

Circuit breaker ndal përkohësisht thirrjet ndaj varësisë problematike; bulkhead ndan pools/resources; timeout kufizon pritjen; retry kufizon përpjekjet; rate limiting kontrollon hyrjen; backpressure u tregon prodhuesve se kapaciteti është i kufizuar. Këto duhen bashkërenduar: tri shtresa me nga tri përpjekje mund të krijojnë deri në 27 tentativa te varësia sipas rrjedhës.

Multi-region writes kërkojnë garanci për konfliktet dhe koordinimin. Për disa invariants, një writer autoritativ ose partition ownership është më i thjeshtë se shkrimi kudo. Zgjedhjet dokumentohen si architecture decision records me kushtet dhe pasojat, jo si listë termash të avancuar.

Shembull: stoku nuk duhet të bëhet negativ. Një read model eventual përdoret për shfaqje, por rezervimi kërkon veprim autoritativ me constraint/koordinim të përshtatshëm.

Laborator: CQRS, saga, outbox dhe kontrolli i mbingarkesës

Koha: 180 minuta. **Mjedisi:** aplikacioni lokal dhe diagramim i rrjedhës së biznesit.

- Vizato porosi→rezervim→pagesë→dërgesë me gjendjet dhe kompensimet. Cakto invariants para zgjedhjes së teknologjive.
- Identifiko transaction boundary të laboratorit dhe çfarë do të ndryshonte me broker/db të jashtëm.
- Llogarit retry amplification për tri shtresa me tri përpjekje dhe cakto një buxhet të përbashkët retry.
- Defekti: replay i event-it dërgon email përsëri. Ndaj rindërtimin e state nga side effects dhe cakto dedup scope.

- Shkruaj një ADR që refuzon event sourcing për një rast të thjeshtë dhe një tjetër që e justifikon kur kërkohet historik i rindërtueshëm.

Dorëzimi: state machine, ADR dhe analiza retry. **Pranimi:** invariants ruhen; termat nuk përdoren si zëvendësim i provës; compensation ka kufij të deklaruar. **Thellim:** projekto shkrime rajonale për inventory duke krahasuar partition ownership me koordinim konsistent ndërrajonal.

39. Operimi dhe diagnostikimi i avancuar

Kontrolli i varësive, degradimi dhe game days

Teori: Kontrolli i varësive, degradimi dhe game days

Objektivi: të lokalizosh incidentin pa ndryshime të rastësishme dhe të provosh rikthimin e funksionit të përdoruesit.

Diagnostikimi nis me shtrirjen: kush preket, kur filloi, cilat rajone/versione dhe çfarë ndryshoi. Ndaj control plane nga data plane; një API e provisioning mund të dështojë ndërsa instancat ekzistuese vazhdojnë. Status page e ofruesit është sinjal, por nuk zëvendëson matjen e workload-it tënd.

Përdor hipoteza të falsifikueshme: DNS i gabuar, identitet i skaduar, quota, throttling, varësi data ose deploy i ri. Krahaso flow të suksesshëm me të dështuar dhe ndrysho një faktor. Request IDs, audit logs, traces dhe counters lidhin provën. Timeout nuk identifikon automatikisht komponentin fajtor.

Graceful degradation ruan funksione kryesore kur të tjerat mungojnë. Për shembull katalogu mund të përdorë cache të vjetër me shënim, por pagesa nuk duhet të raportohet e kryer pa konfirmim. Load shedding refuzon punë të kontrolluar në vend të kolapsit të gjithë sistemit. Runbook duhet të tregojë kush mund të aktivizojë mitigimin dhe kur rikthehet normaliteti.

Game day teston hipoteza me failure injection, blast radius, monitorim, stop conditions dhe rollback. Një ndalim worker-i në laptop është provë e aplikacionit lokal, jo certifikim i DR rajonal. Postmortem duhet të përditësojë runbook, testet dhe modelin e arkitekturës kur zbulohen varësi të fshehura.

Shembull: API e porosive kthen 202, por klientët nuk marrin përfundim. Krahaso backlog, worker attempts, database errors dhe versionin e deploy përpara rritjes së replicas pa kufi.

Laborator: Kontrolli i varësive, degradimi dhe game days

Koha: 180 minuta. **Mjedisi:** API/worker lokale, testet dhe fleta e incidentit.

- Regjistro baseline: health, readiness, statuset e porosive dhe koha e përfundimit. Mbaj kopje backup para provës.
- Fut tre defekte në seanca të ndara: worker i ndalur, databazë jo e arritshme për readiness dhe payload konfliktual me të njëjtin çelës.
- Për secilin shkruaj dy hipoteza dhe provën që i ndan. Ruaj komandat dhe kohën e korrigjimit.
- Defekti i procesit: alarmi mbyllet sapo API është up, por backlog ende nuk përpunohet. Cakto kriter mbylljeje sipas funksionit të përdoruesit.
- Kryej një tabletop për humbje rajoni me identitet, key dhe quota si varësi të detyrueshme.

Dorëzimi: tre raporte dhe runbook i përditësuar. **Pranimi:** nuk mbishkruhet databaza aktive gjatë restore; përfundimi i porosive testohet; ndryshimet kanë rollback. **Thellim:** projekto incident me gabim të pjesshëm vetëm për një tenant dhe provo si ngushtohet hetimi pa regjistruar të dhëna private.

40. Projekti final dhe mbrojtja teknike

Platforma e porosive me prova funksionale dhe operative

Teori: Platforma e porosive me prova funksionale dhe operative

Objektivi: të dorëzosh një arkitekturë cloud të arsyetuar dhe një zbatim lokal të riprodhueshëm me kufij të deklaruar.

Organizata ka një shërbim porosish me API, përpunim asinkron, databazë, uploads private dhe raporte. Ngarkesa bazë është 100 kërkesa/s me kulm të projektuar 1000/s; këto janë kërkesa projektimi, jo pretendim për kapacitetin e laboratorit SQLite. Objektivi i API-së është 99.9% sipas SLI të përkufizuar; porositë e vlefshme duhet të përfundojnë brenda afatit të zgjedhur dhe të justifikuar. RTO/RPO caktohen në dokument para testimit.

Pjesa e zbatuar lokalisht provon API, validation, idempotency, konkurrim, queue/worker, dështim të simuluar dhe backup/restore. Pjesa cloud projekton rajone/zona, IAM, rrjet privat, managed data, IaC, monitorim dhe kosto. Për funksionet e ekzekutuara në sandbox paraqit prova reale; për pjesët vetëm në letër përdor etiketën “projektim”. Asnjë test lokal nuk shpallet provë e sjelljes managed ose multi-region.

Zgjedhja e shërbimeve duhet të shpjegohet me kërkesat dhe kompromiset. Nuk është e detyrueshme të përdoret Kubernetes nëse shërbim managed më i thjeshtë përmbush nevojën. Kërkohet një zgjerim i avancuar: politika Kubernetes, pipeline me promovim artefakti, model data pipeline, recovery i zgjeruar ose threat model/workload identity i testuar.

Dorëzimi përfshin source, versionet, HLD/LLD, matricën e flukseve, rolet, testet, modelin e kostove dhe runbooks. Mbrojtja pyet pse u zgjodh kjo databazë, çfarë ndodh pas timeout, si priten shpenzimet dhe si provohet rikthimi. Kompleksiteti pa prova nuk merr pikë shtesë.

Përgatitja: ndiq `projektet.sq.md` dhe përdor fletën e provave për çdo test. Mbaj çelësat dhe të dhënat reale jashtë paketës së dorëzuar.

Laborator: Platforma e porosive me prova funksionale dhe operative

Koha: 1200 minuta në disa seanca. **Mjedisi:** Python lokal; Docker/Kubernetes ose sandbox janë zgjerime sipas disponueshmërisë.

- Dorëzo kërkesat, SLI/SLO, RTO/RPO dhe arkitekturën për review para zbatimit. Cakto scope-in lokal/cloud dhe kufizimet.
- Zbato API/worker dhe ekzekuto të gjitha testet e ofruara. Shto provat e tua për versionin dhe mjedisin.
- Kryej të paktën 16 testet te projekti: përfshi kërkesat e pavlefshme, dublikimin, konfliktin e payload-it, konkurrimin dhe restore në kopje të re.

- Diagnostiko katër defekte të futura nga mësuesi, duke ruajtur hipotezat, provën dhe korrigjimin minimal.
- Plotëso modelin e kostove me skenarët baseline/kulm/recovery dhe prezanto planin e pastrimit të burimeve opsionale.

Dorëzimi: source, diagramet, konfigurimet pa secrets, provat, incidentet dhe mbrojtje 20-minutëshe. **Pranimi:** minimumi 80/100 plus kontrollet kritike: idempotency, ndarja e tenantëve në projektim, mbrojtja e secrets, testet negative dhe restore pa humbje të origjinalit. **Thellim:** demonstro një zgjerim të avancuar dhe shpjego çfarë prove shtesë kërkohet për ta quajtur gati për prodhim.

Projektet dhe burimet e studentit

Projektet e kursit

Projekti 1 — Zgjedhja e modelit dhe platformës

Pas modulit 10, projekto një aplikacion për dokumente private. Kërkohet autentikim, upload, shkarkim i autorizuar dhe audit. Krahaso VM/PaaS/serverless, map-o familjet e shërbimeve për dy ofrues dhe justifiko zgjedhjen. Dorëzo HLD, rolet, flukset dhe një threat model të shkurtër. Nuk krijohen burime me pagesë. Koha orientuese 4–6 orë.

Projekti 2 — Porosi pa dublikime

Pas modulit 20, ekzekuto API/worker lokale, testet e përfshira dhe tri prova manuale me të njëjtin çelës. Demonstro se payload i ndryshëm jep konflikt. Kryej dështim të përpjekjes së parë dhe përfundim në të dytën. Vizato transaction boundary dhe shpjego çfarë do të ndryshonte nëse efekti do të ishte pagesë e jashtme. Koha 6–8 orë.

Projekti 3 — Deploy dhe operim

Pas modulit 30, zgjidh Docker Compose, Kubernetes ose një pipeline të projektuar në hollësi. Kërkohen health/readiness, konfigurim i jashtëm, test negativ dhe plan rollback. Në Kubernetes provo NetworkPolicy me klient të lejuar dhe të ndaluar; në pipeline provo gate që refuzon regresionin. Dorëzo versionet dhe shëno pjesët që nuk u ekzekutuan. Koha 8–12 orë.

Projekti final — Shërbimi cloud i porosive

Organizata trajton porosi dhe uploads private për disa tenantë. Ngarkesa e projektimit është 100 kërkesa/s bazë dhe 1000/s kulm. Numrat shërbejnë për kapacitet dhe kosto arkitekturore, jo benchmark të garantuar për laboratorin. API synon SLO 99.9%; procesi asinkron duhet të ketë SLI për përfundimin brenda afatit të përcaktuar nga studenti. Kërkohen RTO/RPO, threat model dhe plan FinOps.

Dy shtresat e dorëzimit

Zbatim lokal: Python API/worker, validation, idempotency, konkurrim, metrics dhe backup/restore. Nuk kërkohet cloud account. Në këtë laborator mungon autorizimi tenant; ai duhet të projektohet dhe të shënohet si funksion i paimplementuar, jo të paraqitet si kontroll i kaluar.

Projektim cloud: organizimi, IAM/workload identity, dy zona, rrjet privat për data, zgjedhja e compute dhe shërbimeve managed, backup/keys, observability, IaC dhe model kostoje. Nëse e ekzekuton në sandbox, bashkëngjit prova dhe pastrim; nëse jo, etiketoje si projektim. Pikët praktike jepen vetëm për pjesët e

ekzekutuara realisht.

Testet minimale

ID	Testi	Pritshmëria
T01	Health lokal	200 kur procesi punon
T02	Readiness me DB të disponueshme	200
T03	Readiness me skemë/varësi të munguar	503, ndërsa liveness mbetet 200
T04	Krijim porosie të vlefshme	201 dhe ID
T05	Retry i të njëjtit key/payload	200 dhe e njëjta ID
T06	I njëjti key, amount tjetër	409 dhe pa porosi shtesë
T07	Key i munguar dhe amount i pavlefshëm	400 pa shkrim
T08	16 kërkesa konkurruese me të njëjtin key	Vetëm një krijim
T09	Worker pa dështim	Pending→done
T10	Worker fail-first	Pending në përpjekjen 1, done në 2
T11	Workers konkurrues	Çdo efekt lokal kryhet vetëm një herë
T12	Metrics gjatë backlog	Pending/age pasqyrojnë punën
T13	Backup gjatë gjendjes së vlefshme	Krijohet kopje e lexueshme
T14	Restore në skedar të ri	Origjinali ruhet; kopja ka pikën e pritur
T15	Tentativë mbishkrimi backup/restore	Refuzohet
T16	Plan kostosh baseline/kulm/recovery	Njësi dhe supozime të qarta

Për çdo test ruaj kohën, komandën, input/output dhe shpjegimin. Testet automatike ndihmojnë por nuk e zëvendësojnë shpjegimin e mekanizmit. Mos manipulo databazë reale për testin e dështimit. Përdor direktori prove dhe rikthim në emër të ri.

Zgjerimi i avancuar — zgjidh një

- Kubernetes me image të ngarkuar, probes dhe NetworkPolicy të verifikuar nga dy pod-e.
- CI/CD me promovim të të njëjtit artefakt, gate regresioni dhe plan schema compatibility.
- Arkitekturë me outbox dhe konsumator idempotent të implementuar për një kufi të veçantë transaction.
- Restore drill me verifikim biznesi, matje kohe dhe provë se origjinali nuk u dëmtua.
- Data pipeline me event-e të dyfishta/të vona, schema versioning dhe quality gates.

Çdo zgjerim kërkon dy prova normale, dy negative dhe një dështim me rikthim. Nëse zgjidhet vetëm projektim pa ekzekutim, kjo vlerësohet për analizë dhe nuk plotëson vetë pjesën praktike të zgjerimit.

Dokumentet dhe mbrojtja

Dorëzo README me mjetet/versionet, source dhe konfigurimet pa secrets, HLD/LLD, IAM matrix, flukset, tests T01–T16, SLO, modelin e kostove dhe katër incidente. Prezanto për 20 minuta dhe mbro vendimet për databazën, retry, sigurinë, kapacitetin dhe recovery. Shpjego çfarë prove do të duhej për ta kaluar nga laboratori në prodhim.

Pragu është 80/100 dhe duhen kaluar kontrollet kritike të rubrikës. Nëse ke sandbox cloud, inventari dhe pastrimi janë pjesë e provës; fatura zero nuk supozohet nga mungesa e trafikut.

Fletët e punës

Regjistri i laboratorit

Moduli: ___ Data/ora/zona: ___ Mjedisi/versionet: ___ Praktikë apo analizë: ___

Qëllimi: ___ Identiteti/scope i përdorur pa secrets: ___ Gjendja fillestare: ___

Hapi	Input/komanda	Rezultati i pritur	Rezultati real	Prova	Kufizimi
1					
2					

Çfarë u ndryshua: ___ Rikthimi: ___ Retestimi i aplikacionit: ___

Llogaritje dhe analiza

1. Dy varësi të domosdoshme kanë nga 99.9%. Llogarit disponueshmërinë me supozimin e pavarësisë dhe shpjego pse ai mund të mos vlejë.

2. Llogarit buxhetin kohor për 99.9% mbi 30 ditë dhe numrin e gabimeve për 100 000 kërkesa me të njëjtin objektiv.

3. 100 kërkesa/s dhe kohë mesatare 0.2 s: llogarit numrin mesatar në sistem. Pastaj 200/s me 0.5 s.

4. Queue merr 120 mesazhe/s; worker trajton 30/s. Sa duhen teorikisht pa rezervë? Çfarë ndryshon kur 20% e përpjekjeve duhet riprovuar?

5. 3000 IOPS me 16 KiB japin sa MiB/s? Çfarë kufiri mbizotëron nëse instanca lejon vetëm 40 MiB/s?

6. 12 API me pool 30 dhe 2 workers me pool 10 japin sa connections potenciale?

7. Një tenant shkruan 80% të 1000 shkrimeve/s. Sa shkrime/s duhen shpërndarë për të dhe si ndikon sharding te leximi?

8. Me 20 query seriale dhe RTT ndërcloud 30 ms, sa shtohet në rrugën kritike nga ky komponent sipas modelit?

9. 100 pajisje me 10 mesazhe/min, 1 KB secili, 24 h offline: sa KB ruajtje nominale duhen përpara overhead/rezervës?

10. Për modelin e kostove llogarit compute, storage, egress, requests dhe koston për 1000 porosi. Numrat janë hipotetikë.

Matrica e përgjegjësive

Funksioni	IaaS	Managed data/PaaS	SaaS	Prova e përgjegjësisë
Patching				
Identiteti				
Backup/restore				
Të dhënat				

Regjistri i vendimeve arkitekturore

Kërkesa: ___ Alternativa A/B: ___ Vendimi: ___

Arsyeja dhe kompromiset: ___ Çfarë do ta ndryshonte vendimin: ___

Prova që kërkohet: ___ Kostoja dhe pronari: ___

Regjistri i incidentit

Ndikimi dhe shtrirja: ___ Ndryshimi i fundit: ___ Ora e nisjes: ___

Hipoteza H1/prova refuzuese: ___ H2/prova refuzuese: ___

Prova e mbledhur: ___ Korrigjimi minimal: ___ Rollback: ___

Rikthimi i funksionit të përdoruesit: ___ Backlog i mbetur: ___

Veprimi parandalues, pronari dhe afati: ___

Kontrolli para dorëzimit

- Praktika e ekzekutuar dallohet nga projektimi.
- Komandat dhe mjediset përputhen me diagramin.
- Kërkesat e dyfishta dhe testet negative janë provuar.
- Backup u rikthye në kopje të re dhe origjinali ruhet.
- Kostoja përfshin supozime, njësi dhe kufizime.
- Nuk dorëzohen state sensitive, keys, tokens ose të dhëna reale.

Fjalor teknik

Termi	Kuptimi në kurs

ABAC	Autorizim sipas atributeve dhe kushteve
ACID	Familje garancish transaksioni sipas motorit
ADR	Regjistër vendimi arkitekturor dhe kompromisesh
API	Kontratë komunikimi me funksionet e një sistemi
Artifact	Produkt i ndërtimit që promovohet/deploy-ohet
Autoscaling	Përshtatje kapaciteti sipas sinjalit dhe politikës
Availability	Aftësia e shërbimit për të qenë i përdorshëm sipas matjes
Availability zone	Kufizim vendosjeje/failure domain sipas ofruesit
Backoff/jitter	Vonesë riprovimi dhe variacion për shpërndarje në kohë
Backpressure	Sinjalizim/kontroll i prodhimit kur konsumatori është i mbingarkuar
Backup	Kopje me politikë rikthimi, retention dhe prova
Batch	Përpunim i grupuar i punës/të dhënave
Blast radius	Shtrirja e mundshme e ndikimit të një dështimi ose kompromet imi
Blue/green	Kalim mes dy mjediseve të përgatitura
Budget	Objektiv/monitorim shpenzimi, jo domosdoshmërisht hard cap
Bulkhead	Izolim pools/resources që një dështim të mos konsumojë gjithçka
Cache	Kopje për ulje latency/ngarkese me kontratë staleness
Canary	Provë graduale versioni mbi pjesë trafiku/përdoruesish
Cardinality	Numri i vlerave/kombinimeve, p.sh. labels të metrics
CDN	Shpërndarje/cache përmbajtjeje afër përdoruesit
CDC	Kapje e ndryshimeve të të dhënave për replikim/migrim/përpunim
CI/CD	Integrim dhe delivery/deployment i kontrolluar
Circuit breaker	Kufizim i thirrjeve ndaj një varësie që po dështon
CNI/CSI	Integrime Kubernetes për rrjet/storage
Cold start	Inicializim runtime/instance që shton latency në disa thirrje
Commit	Përfundim i ndryshimit sipas kontratës së transaksionit
Concurrency	Punë në zhvillim njëkohësisht, jo domosdoshmërisht paralelizëm fizik
Container	Proces i paketuar me izolim sipas runtime/kernel-it
Control plane	API/kontrolli që menaxhon gjendjen e sistemit
CQRS	Ndarje e modelit të komandave/shkrimit nga leximi
Data gravity	Kosto/vështirësi lëvizjeje për të dhëna të mëdha dhe varësitë e tyre
Data plane	Pjesa që kryen trafikun ose funksionin e të dhënave
Deduplication	Njohje dhe trajtim i përsëritjes së të njëjtit identitet operacioni
Deployment	Burim Kubernetes për rollout/replicas ose proces publikimi sipas kontekstit

Digest	Identifikues i përmbajtjes së artefaktit sipas algoritmit
DLQ	Radhë/destinacion për punë të dështuar që kërkon hetim
Drift	Devijim i gjendjes reale nga ajo e deklaruar
Durability	Qëndrueshmëria e të dhënave kundrejt humbjes sipas modelit
Egress	Trafik dalës sipas kufirit dhe faturimit të shërbimit
Elasticity	Përshtatje kapaciteti me ndryshimin e kërkesës
Envelope encryption	Mbrojtje e data key përmes një çelësi tjetër sipas mekanizmit
Error budget	Dështime të lejuara sipas objektivit të shërbimit
Event sourcing	Rindërtim state nga historia e ngjarjeve sipas modelit
Eventual consistency	Model ku kopjet mund të konvergojnë pasi ndalojnë ndryshimet sipas garancive
Federation	Lidhje besimi mes sistemeve të identitetit
FinOps	Praktikë bashkëpunimi për vlerën dhe menaxhimin e kostos cloud
Golden image/path	Bazë instance e provuar/rrugë platforme e mbështetur
Guardrail	Kontroll që kufizon, zbulon ose korrigjon konfigurime/veprime
HLD/LLD	Projekt arkitekturor/projekt i hollësishëm zbatimi
HPA	Shkallëzim horizontal pods sipas metrikave të konfiguruara
HSM	Modul hardware me funksione mbrojtjeje çelësash/operacionesh
Hybrid cloud	Integrim i mjediseve lokale/private me cloud sipas arkitekturës
IaaS/PaaS/SaaS	Infrastrukturë/platformë/aplikacion si shërbim
IAM	Menaxhim identitetesh dhe aksesesh
IaC	Infrastrukturë e përshkruar dhe menaxhuar me kod
Idempotency	Përsëritje me të njëjtin identitet pa efekt shtesë të paqëllimshëm
Image	Artefakt baze VM ose container sipas kontekstit
Immutable	Model zëvendësimi nga gjendje e deklaruar, me ndryshime të kontrolluara
IOPS	Operacione hyrje/dalje për sekondë
KMS	Shërbim menaxhimi çelësash me politika dhe operacione përkatëse
Landing zone	Themeli organizativ, i identitetit, rrjetit dhe kontrollit cloud
Lease	E drejtë rezervimi me afat sipas protokollit
Lifecycle	Gjendjet, afatet dhe veprimet gjatë jetës së resource/të dhënave
Liveness/readiness	Gjallëri procesi/gatishmëri për trafik
Managed service	Shërbim ku një pjesë e administrimit merret nga ofruesi
Multicloud	Përdorim i disa ofruesve cloud me integrim sipas nevojës
Namespace	Kontekst objektesh Kubernetes, jo izolim i plotë vetë

Noisy neighbor	Ndikim nga konkurrimi mbi burime të përbashkëta
Object storage	Ruajtje objektesh me key/metadata/API
OIDC	Shtresë identiteti me claims dhe marrëdhënie besimi
Outbox	Regjistrim lokal i event-it bashkë me ndryshimin për publikim të mëvonshëm
Partition key	Fushë që ndikon ndarjen fizike/logjike të të dhënave
PDB	Kufizim për disa ndërprerje vullnetare pods
PITR	Rikthim i të dhënave në pikë kohe brenda aftësive/retention
Policy as code	Rregulla të kontrollueshme automatikisht mbi modele ose gjendje
Principal	Identitet/entitet që kërkon veprim sipas IAM
Quota	Kufizim përdorimi/burimesh sipas shërbimit dhe kontekstit
RBAC	Autorizim sipas roleve
Region	Vendndodhje/grupim shërbimesh sipas ofruesit
Replica	Kopje me funksion dhe garanci të përcaktuara
Retention	Afati/politika e mbajtjes së të dhënave
Rollback	Rikthim i kontrolluar i një ndryshimi sipas planit
RPO/RTO	Objektiv humbjeje të dhënash/kohe rikuperimi
Saga	Koordinim hapash lokalë me kompensime biznesi
SBOM	Inventar komponentësh software në artefakt
Serverless	Ekzekutim ku përdoruesi nuk administron drejtpërdrejt serverët
Shared responsibility	Ndarje detyrash mes ofruesit dhe klientit sipas shërbimit
SLI/SLO/SLA	Matje/objektiv/marrëveshje e nivelit të shërbimit
Spot	Kapacitet me kushte ndërprerjeje dhe model çmimi përkatës
State	Gjendje sistemi; në IaC lidhje mes konfigurimit dhe burimeve
Telemetry	Sinjale operative të emetuara nga sistemi
Tenant	Kufiri i klientit/ekipit sipas modelit të sistemit
Throttling	Kufizim i normës ose përdorimit nga mekanizmi përkatës
Toil	Punë operative e përsëritur që mund të reduktohet
Trace/span	Gjurmë e rrjedhës dhe segment i instrumentuar i saj
Unit economics	Kosto/vlerë për njësi rezultati biznesi
Visibility timeout	Afat rezervimi të mesazhit sipas broker-it
Workload identity	Identitet i procesit/platformës për marrjen e lejeve

Harta e mbulimit dhe specializimi

Modulet	Përmbajtja	Forma
1–4	Modele, izolim, sisteme të shpërndara, rajone	Teori, llogaritje dhe laborator idempotency
5–10	Ofruesit, landing zone, IAM, secrets, rrjet, edge	Projektim me dokumentacion zyrtar dhe prova lokale probes
11–17	VM, autoscaling, object/block/file, SQL/NoSQL/cache	Laborator lokal, modele kapaciteti dhe projektim managed
18–20	Queue, retries, serverless, containers	Worker i ekzekutueshëm dhe shembull Compose
21–22	Kubernetes, probes, policy, operim	Manifest stateless dhe testim CNI në cluster testimi
23–25	IaC, CI/CD, platform engineering	IaC lokal pa provider dhe pipeline/kontrata
26–30	Observability, SRE, security, keys, governance	Metrics lokale, incidente, validator dhe projektim
31–34	FinOps, DR, migrim, hybrid/multicloud	Model kostosh, restore i provuar dhe runbooks
35–38	Data, AI/GPU, IoT/edge, modele të avancuara	Analiza dhe projekte specializimi
39–40	Diagnostikim dhe projekt final	Prova integruese dhe mbrojtje teknike

Orientim AWS/Azure/Google Cloud

Aftësia	AWS	Azure	Google Cloud
VM	EC2	Azure Virtual Machines	Compute Engine
Object storage	S3	Blob Storage	Cloud Storage
Kubernetes managed	EKS	AKS	GKE
Organizimi bazë i workloads	Account	Subscription/resource groups sipas rolit	Project

Kjo është hartë orientimi, jo ekuivalencë e plotë funksionale. Lexo dokumentacionin e produktit për të kuptuar scope, SLA, identitet, pagesa dhe kufizime. Shërbimet serverless, databazat dhe queues krahasohen sipas kontratës reale, jo duke zëvendësuar emrat në diagram.

Çfarë është praktikë e ekzekutueshme

Python API/worker, testet, backup/restore, model kostosh dhe validator arkitekture mund të ekzekutohen lokalisht. Compose, Kubernetes dhe IaC kanë skedarë dhe hapa por duhen mjetet/runtime përkatëse. Laboratorët me konsola cloud janë projektim ose zbatim opsional në sandbox; nuk pretendohet deploy real vetëm nga ekzistenca e skedarit.

Specializimi pas kursit

Cloud architecture kërkon thellim në një ofrues, limits dhe operim real. Cloud security kërkon praktikë IAM, incident response dhe audit. Platform/SRE kërkon Kubernetes, IaC, release dhe observability në ngarkesë. Data engineering kërkon motorë stream/batch dhe governance të të dhënave. MLOps kërkon model evaluation, GPU serving dhe data/model lifecycle.

Nuk trajtohen si specializime të plota të gjitha komandat e çdo ofruesi, implementimi i hypervisor-it, kriptografia formale, interpretimi juridik i çdo rregulli, të gjithë motorët e databazave ose trajnimi matematikor i modeleve AI. Qëllimi është bazë e gjerë me mekanizma dhe prova të avancuara, jo pretendim se çdo produkt ekzistues është mbuluar.

Burimet zyrtare dhe mënyra e përdorimit

Përgatitur më 16 nëntor 2026. Referencat ndihmojnë verifikimin e mekanizmave dhe zbatimit. Dokumentacioni i shërbimeve ndryshon: ruaj versionin/datën për laboratorin. Teksti mësimor është në shqip; burimet zyrtare janë kryesisht në anglisht dhe nuk janë kopjuar si kapituj të përkthyer.

Themelet dhe arkitektura

- Moduli 1: NIST SP 800-145, përkufizimi i cloud computing — https://csrc.nist.gov/pubs/sp/800/145/final
- Modulet 4–6, 39–40: AWS Well-Architected Framework — https://wa.aws.amazon.com/index.en.html
- Modulet 3, 17–19, 33, 38: Azure Architecture Center, cloud design patterns — https://learn.microsoft.com/en-us/azure/architecture/patterns/
- Modulet 4–6, 34–36: Google Cloud Well-Architected Framework — https://docs.cloud.google.com/architecture/framework

Identiteti, storage dhe infrastruktura

- Modulet 7–8: AWS IAM security best practices — https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html
- Moduli 13: Amazon S3, veçoritë dhe modeli i konsistencës — https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html
- Moduli 23: OpenTofu, sensitive data in state — https://opentofu.org/docs/language/state/sensitive-data/
- Modulet 14, 15, 32: SQLite Online Backup API — https://www.sqlite.org/backup.html

SQLite është burim për motorin e laboratorit. Nuk përdoret si referencë për të pretenduar garanci identike te PostgreSQL, SQL Server ose motorët managed cloud. Për to shto dokumentacionin e motorit dhe shërbimit që zgjedh.

Containers, Kubernetes dhe operimi

- Moduli 20: Docker Compose startup order dhe readiness — https://docs.docker.com/compose/how-tos/startup-order/
- Moduli 21: Kubernetes probes — https://kubernetes.io/docs/concepts/workloads/pods/probes/
- Moduli 22: Kubernetes Network Policies — https://kubernetes.io/docs/concepts/services-networking/network-policies/
- Moduli 22: Kubernetes multi-tenancy — https://kubernetes.io/docs/concepts/security/multi-tenancy/
- Moduli 26: OpenTelemetry signals — https://opentelemetry.io/docs/concepts/signals/
- Moduli 27: Google SRE Workbook, Implementing SLOs — https://sre.google/workbook/implementing-slos/

Çfarë duhet verifikuar për çdo shërbim real

Përdor dokumentacionin dhe kalkulatorin zyrtar të ofruesit për rajonet, versionet, limits, quotas, çmimet, SLA dhe kushtet e planit. Ruaj rajonin, monedhën, datën, modelin e trafikut dhe discounts e supozuara. Mos e trajto një shembull kostosh në këtë kurs si ofertë aktuale.

Për IAM regjistro policy evaluation dhe scope të burimit; për databazë regjistro consistency, failover dhe PITR; për queues renditjen, retry dhe retention; për serverless timeout dhe concurrency. Lexo edhe kufizimet para kopjimit të një komande. Një funksion i disponueshëm në një rajon ose variant produkti mund të mungojë në tjetrin.

Për kërkesa rregullatore përgatit hartën e të dhënave dhe pyetjet për pronarët përgjegjës. Ky material jep arkitekturë dhe praktikë teknike, jo interpretim juridik universal. Për AI/model APIs kontrollo politikat e të dhënave dhe kushtet e përdorimit të shërbimit të zgjedhur përpara ngarkimit të të dhënave reale.