UTS XOps (IF4054 Software System X-Ops)

Materi inti W00–W07, 38 soal pilihan ganda multi-jawaban,
dan 10 essay berjawaban

Bahan UTS dari slide semester 1 2026/2027 (folder *ppt baru*), yaitu catatan 17 sampai 26. Istilah singkat ada di *Glosarium_XOps*. Penjelasan yang tidak ada di slide ditandai *(tambahan)*.

Cakupan dan Peta Outcome

Minggu	Topik	Catatan	Bagian
W00	Course Introduction	17	1.1
W01a	X-Ops Introduction (Fowler, DevOps)	18	1.1
W01b	MLOps dan DataOps	19	1.2
W02	DevOps, cloud native, container	20	1.3
W02	Foundations Architecture (4 pilar XOps, 12-factor, microservices, container, Git/Docker/Compose/MinIO)	21	1.4
W03	CI dan Shift-Left	22	1.5
W04	IaC dan Cluster Security	23	1.6
W05	Release Engineering dan Delivery	24	1.7
W06	DataOps Foundation (ShopStream)	25	1.8
W07	DataOps Security, Quality, Privacy (PayLend)	26	1.9

Catatan 1–16 berasal dari slide versi lama. Sebagian isinya tumpang tindih (W01a sama persis dengan catatan 1), tetapi cakupan UTS tahun ini mengikuti slide baru.

Outcome resmi (W00): (1) memahami prinsip dan praktik X-Ops; (2) hands-on container dan orchestration; (3) mengimplementasikan CI/CD; (4) mengintegrasikan dan mengotomasi tahap software dan data lifecycle; (5) menjalankan praktik X-Ops; (6) merancang sistem software untuk X-Ops.

Format Soal dan Strategi

Pilihan ganda (bisa lebih dari satu benar). Setiap opsi dinilai sendiri, mirip grid True/False. Pola jebakan yang dipakai di bank soal ini:

- **Ditukar:** "mutating bisa inject sidecar" benar, "validating bisa inject sidecar" salah.
- **Arah terbalik:** "shift-left memindahkan pengecekan ke akhir", "at-most-once = proses dulu baru commit".
- **Level/tempat salah:** fitur level 2 MLOps ditulis sebagai level 1; Role disebut cluster-wide.
- **Praktik buruk disajikan sebagai praktik baik:** "merge anyway", "tag `latest`", "kubectl apply untuk hotfix", "simpan state Terraform di laptop".

Kerangka menjawab essay

Soal essay dibentuk dari materi: diberi skenario, lalu diminta merancang, membandingkan, atau menjelaskan alasan.

1. **Identifikasi masalah** di skenario (apa yang salah atau apa yang dibutuhkan).
2. **Sebut konsep/prinsip** dari materi yang relevan (dengan nama istilahnya).
3. **Rancang/terapkan** langkah demi langkah (urutan stage, komponen, konfigurasi).
4. **Alasan dan trade-off** (kenapa urutan itu, apa risikonya, anti-pattern yang dihindari).

Daftar Isi

1 Materi Inti **3**

1.1 X-Ops, Fowler, dan DevOps (W00, W01a) . . . 3
1.2 MLOps dan DataOps (W01b) . . . 3
1.3 DevOps, Cloud Native, Container (W02) . . . 4
1.4 Foundations Architecture (W02) . . . 5
1.5 CI dan Shift-Left (W03) . . . 6
1.6 IaC dan Cluster Security (W04) . . . 7
1.7 Release Engineering dan GitOps (W05) . . . 7
1.8 DataOps Foundation (W06, ShopStream) . . . 8
1.9 DataOps Quality, Security, Privacy (W07, PayLend) . . . 9

2 Soal Pilihan Ganda (Bisa Lebih dari Satu Benar) **10**

3 Soal Essay **20**

3.1 Essay 1 (W01a). Automated dan Frequent . . . 20
3.2 Essay 2 (W01b). Menaikkan Level MLOps . . . 21
3.3 Essay 3 (W02). Migrasi Aplikasi ke Container . . . 21
3.4 Essay 4 (W02). Service yang Melanggar 12-Factor . . . 22
3.5 Essay 5 (W03). Merancang CI Pipeline Build-Time . . . 22
3.6 Essay 6 (W04). Provisioning dan Pengamanan Cluster . . . 23
3.7 Essay 7 (W05). Strategi Rilis Fitur Checkout . . . 24
3.8 Essay 8 (W06). Pipeline Event ShopStream . . . 24
3.9 Essay 9 (W07). Mengamankan dan Memvalidasi Event PayLend . . . 25
3.10 Essay 10 (Integratif). Satu Taksonomi X-Ops . . . 26

4 Ringkasan Inti **26**

1 Materi Inti

1.1 X-Ops, Fowler, dan DevOps (W00, W01a)

- **Operationalize software**: pendekatan sistematis untuk mengotomasi build dan mengorkestrasi operasi agar memenuhi tujuan terukur yang selaras dengan prioritas bisnis.
- **Topik silabus**: intro X-Ops; DevOps/MLOps/DataOps (culture, principles, practices, metrics/KPI, tools); CI/CD; IaC; **observability** (memahami kondisi internal sistem dari log, metric, trace; penjelasan tambahan); security.
- **Kelompok topik**: Best Practices (agility, automation, integrated configuration & change management, monitoring); Tools (container & orchestration, CI/CD, monitoring tools, cloud); Fields (DevOps, DataOps, MLOps).
- **Deployment pipeline (Fowler)**: commit stage (compile, unit test, analysis, build installers) → automated acceptance testing → automated capacity testing → manual testing (showcase, exploratory) → release.
- **Anti-pattern rilis**: deploy manual; production-like environment baru dipakai setelah development selesai; manual configuration management production.
- **Dua kata kunci**: **Automated** (tanpa otomasi proses tidak repeatable) dan **Frequent** (delta kecil → risiko turun, rollback mudah, feedback cepat).
- **Motivasi: feedback**. Setiap perubahan memicu feedback, feedback secepat mungkin, tim menindaklanjutinya.
- **Agile Manifesto** (2001, Snowbird, 17 orang): individuals & interactions, working software, customer collaboration, responding to change, lebih diutamakan daripada process & tools, documentation, contract, plan.
- **8 prinsip delivery**: repeatable & reliable release; automate almost everything; keep everything in version control; if it hurts do it more frequently & bring the pain forward; build quality in; done means released; everybody responsible; continuous improvement.
- **DevOps**: practices + tools + cultural philosophy yang mengotomasi dan mengintegrasikan dev dan IT (Atlassian). Mulai sekitar 2007 karena **silo** dev dan ops.
- **Lifecycle (infinity loop)**: Discover → Plan → Build → Test → Deploy → Operate → Observe → Continuous feedback, dibungkus Collaboration & Communication.

1.2 MLOps dan DataOps (W01b)

- **Karakteristik software ML**: data dependency; **model drift** (perlu monitoring + retraining); eksperimental (model version control & tracking); specialized tools; integrasi dengan tim data science; continuous evaluation dengan feedback loop.
- **MLOps** = DevOps untuk ML engineering; irisan ML, data engineering, DevOps. Flow: data sourcing → analysis → preparation → model management (development ↔ evaluation ↔ versioning) → deployment → serving → health monitoring + governance; retraining kembali ke data analysis; semua dipayungi orchestration.
- **7 langkah model development**: business objectives, data sources, EDA, feature engineering & selection, training & evaluation, validation, reproducibility.
- **Serving**: microservice REST (online), embedded di edge/mobile, batch prediction.

Level (Google)	Ciri
0 Manual	Script-driven, tanpa CI/CD, ML terputus dari ops, rilis jarang, tanpa monitoring aktif; model terlatih sudah masuk **model registry** lalu di-serve
1 ML pipeline automation	Pipeline training otomatis dipicu trigger (continuous training), feature store, ML metadata store
2 CI/CD pipeline automation	Komponen pipeline itu sendiri di-build, test, package, deploy lewat CI/CD

- **DataOps** = practices, procedures, technologies data management; integrated & process-oriented; tujuan quality, speed, collaboration; hasil data products & data apps. Key roles: developers, data engineers, analysts, data scientist, business users. **3V**: volume, velocity, variety.
- **Task DataOps** per fase: Planning (goals, sumber data, standar kualitas), Development (extract, transform, load), Testing (quality checks, validation), Deployment (deploy pipeline ETL, monitoring & logging), Operations (data management, governance, security), Feedback & Improvement.
- **Relasi MLOps–DataOps**: data as a foundation, integrated workflow, shared goals.

1.3 DevOps, Cloud Native, Container (W02)

- **Common practices**: CI (merge sering ke shared repo), CD (software teruji siap ke production), **IaC** (resource infra lewat kode), **CaC** (pengaturan konfigurasi lewat kode + VCS), monitoring & logging, communication & collaboration.
- **Cloud native**: microservice (loosely coupled), containerization, DevOps & automation, dynamic orchestration. **Bukan tujuan kuliah**: provisioning manual, deploy ke physical server/VM, vertical scaling, redundant servers.
- **Physical → VM → Container**: physical berbagi library (konflik); VM = host OS + **hypervisor** + guest OS per VM (berat); container = satu OS + **container runtime**, tiap container hanya libraries/dependencies (ringan).
- **Workflow**: kode → Dockerfile → build, run, unit test → push ke registry → jalankan di cluster. Sample pipeline: Git → Jenkins CI (build, run, test) → push Docker Hub; manifest → Jenkins CD → deploy ke Kubernetes, cluster pull image.
- **Migrasi VM → container**: discovery → requirement assessment → container infrastructure design → containerize → testing → deployment & rollout (lalu ulang).
- **Kriteria app cocok**: stateless; high cohesion (microservice, API-first); frequent updates; kompatibel; dependency & config eksternal; **tidak** dijaga tim khusus / banyak tim; environment homogen.
- **Security**: minimal base image (kurangi attack surface), scan rutin, least privilege (non-root), immutable infrastructure, TLS antar-container, update & patch, network segmentation, monitor & log, digital signature, secure registry.
- **Performance**: base image tepat, lightweight (**multi-stage build**), resource limit, storage efisien, caching (Redis/Memcached), monitor, networking, orchestration, health check.
- **Logging**: centralize (ELK, Fluentd), structured (JSON), level tepat, secure, log rotation & retention, integrasi monitoring.
- **Monitoring goals** (identifikasi failure saat runtime maupun **postmortem**, yaitu analisis setelah failure): failure, degradasi performa, capacity planning, intruder → data aplikasi + infrastruktur; **reaksi user terhadap business offerings → aplikasi saja**. Tantangan: perubahan terus-menerus, bottom-up vs top-down & cloud, microservice, volume log terdistribusi.

1.4 Foundations Architecture (W02)

- **XOps** = menerapkan disiplin yang sama (**automation, versioning, testing, monitoring**) ke setiap artifact: code, data, models. Taksonomi untuk **shared vocabulary**, bukan tembok antar-disiplin.

Pilar	Fokus
DevOps	Build, test, release software reliable & frequent; fondasi pilar lain. Diukur dengan **delivery metrics**: deployment frequency, lead time, change failure rate, time to restore
DataOps	CI/CD untuk data pipeline: ingestion, quality, schema, lineage; quality checks jadi pipeline gate; lineage & provenance menelusuri angka dashboard ke raw record
MLOps	DevOps untuk model lifecycle; model = versioned artifact; concern baru (hyperparameter, versi data, drift, fairness); **Continuous Training (CT)** dipicu data baru atau performance decay
SecOps	Lensa di setiap tahap, bukan ditempel di akhir; **shift-left security** saat build time, bukan audit time; meluas ke data (PII masking) dan model (adversarial risk)

- **One lifecycle, four lenses**: Plan → Build → Verify → Release → Operate (+ feedback), tiap tahap dilihat lewat lensa DevOps, DataOps, MLOps, SecOps.
- **12-Factor App** (Heroku, dari pengamatan ratusan aplikasi; menyasar "jalan di laptop, rusak di production"). Satu ide: pisahkan **yang berubah (code)** dari **yang bervariasi (config, environment)**.

No	Faktor	Inti
I	Codebase	Satu codebase di version control, banyak deploy; jangan fork per environment
II	Dependencies	Deklarasikan dan isolasi dependency secara eksplisit
III	Config	Config (credentials, endpoints, feature flags) di environment variables
IV	Backing services	DB, queue, cache = attached resources, ditukar lewat config
V	Build, release, run	Pisahkan tegas build, release (build + config), run
VI	Processes	Stateless process; data persisten di backing service
VII	Port binding	Self-contained, ekspos service lewat port
VIII	Concurrency	Scale out dengan menambah process
IX	Disposability	Fast startup, graceful shutdown
X	Dev/prod parity	Dev, staging, prod semirip mungkin
XI	Logs	Log sebagai event stream ke stdout
XII	Admin processes	Task admin sebagai one-off process di environment identik

- **12-factor & container**: image hampir by definition 12-factor artifact (tanpa baked-in config); Factor III ↔ `--env` Docker / ConfigMaps & Secrets K8s; Factor IX memungkinkan orchestrator kill & reschedule. **Batas "build once"**: arsitektur ARM vs x86 (multi-arch image), workload bergantung hardware (GPU), backing service tetap berbeda, race condition di skala production.
- **Config separation** mengatasi "works on my machine" dan secret di kode, tetapi **tidak** mengatasi **blast radius**: **Conway's Law** (sistem mencerminkan struktur komunikasi organisasi), tim kusut → **distributed monolith**.
- **Microservices**: deploy independen per business capability; untung independent scaling, deployment, failure isolation; biaya network call menggantikan function call. **Decompose by business capability** (orders, inventory, billing), bukan technical layer. **API Gateway**: single entry point, routing, auth. **State**: Factor VI, service tanpa durable state; **database-per-service** (akses DB service lain merusak encapsulation); konsistensi (saga, eventual consistency) dibahas di DataOps.

- **Containerization theory**: isolasi level OS (**namespaces, cgroups**), bukan virtualisasi hardware; image immutable dan berlapis; container = running instance + writable layer tipis.
- **Container vs VM**: container berbagi kernel host (isolasi proses), MB, start ms–detik, cocok microservice; VM guest OS penuh, isolasi kuat, berat, boot puluhan detik–menit, saat butuh kernel/OS berbeda.
- **Image layers**: base image → dependencies → app code → entrypoint; `COPY requirements.txt` + install **sebelum** `COPY . .` agar layer dependency ter-cache.
- **"Works on my machine"** diselesaikan dengan Dockerfile di version control → image identik dan reproducible di lokal, CI, production.
- **Git**: distributed (tiap clone history penuh), branch = pointer ringan. Branching kuliah: main selalu deployable, `feature/*` short-lived, PR (review + CI), squash merge.
- **Docker Compose**: service + backing service (db, MinIO) dalam satu file sebagai attached resources (**Factor IV**); environment lokal identik untuk tiap anggota tim. **MinIO**: object storage S3-compatible, offline, gratis, reset cepat; pindah ke cloud cukup ganti config (**Factor III**).
- **Takeaways**: one taxonomy, 12 factors one idea, boundaries matter (business capability, data store sendiri), reproducibility first.

1.5 CI dan Shift-Left (W03)

- **CI**: merge ke shared trunk beberapa kali sehari, tiap merge memicu build + test; integrasi jadi non-event.
- **Cost of defects**: IDE/pre-commit **1x**, CI **15x**, production **100x+**. **Shift-left** = pindahkan pengecekan ke tahap awal.
- **Pengecekan per tahap**: Code (IDE lint, pre-commit hooks) → Commit (unit test, SAST on diff) → Build (compile, dependency scan) → Package (container scan, sign artifact) → Deploy (policy checks, last line of defense).
- **4 prinsip CI**: fail fast fail loud (stage murah dulu, pipeline = gate); **immutable artifacts** (build once per commit, promote by content hash, config di-inject saat deploy); fast feedback (menit, paralel, cache); trunk-based & small batches (feature flag, hindari merge hell).
- **Testing pyramid**: unit (banyak, detik, mock/stub) → integration (menit, DB/service nyata, tangkap SQL salah, serialization mismatch, wiring error) → contract (sedikit, consumer vs provider, pact). Anti-pattern **ice cream cone**.
- **Build-time pipeline**: lint → unit test → **SAST** → build → **SCA** → image scan → publish.
  - SAST: injection, hardcoded secrets, insecure deserialization, broken access logic. Fix SQL injection: **parameterized query**.
  - SCA: direct + transitive dependency vs NVD (database CVE, yaitu ID kerentanan publik; definisi CVE tambahan); 80–90% kode aplikasi adalah pihak ketiga.
  - Image scan: setelah build, sebelum push; blok Critical/High, track Medium/Low. Base image minimal (distroless/slim) sudah mengecilkan attack surface.
- **Registry & provenance**: artifact content-addressed (**digest**, bukan `latest`), kontrol push/pull, credential build tidak boleh deploy prod. **SLSA**: membuktikan artifact dibangun proses tepercaya lewat provenance.
- **Tools**: GitLab CI (`.gitlab-ci.yml`, stages, job paralel dalam stage, registry bawaan), GitHub Actions (`.github/workflows`, event, marketplace, matrix), SonarQube (quality gate, new code vs legacy), Trivy (`--exit-code 1 --severity CRITICAL,HIGH`).
- **Pitfalls**: alert fatigue, flaky tests, mutable `latest`, secrets in logs.

1.6 IaC dan Cluster Security (W04)

- **Alur**: scanned image → Terraform provision → Kubernetes orkestrasi → Kyverno policy → workload compliant.
- **IaC**: infra di file teks + version control + review PR; satu source of truth; perluasan Factor I.
- **Imperative (how)** vs **declarative (what)**: declarative mendeskripsikan end state, tool menghitung langkah, aman dijalankan ulang.
- **Pattern**: resource graph, modules, environments as data, state.
- **Orchestrator**: scheduling, scaling, networking, **self-healing**; Kubernetes terus mencocokkan actual dengan desired.
- **Idempotency**: jalan dua kali = jalan sekali; sifat logika reconciliation tool. **Immutable infrastructure**: ganti, bukan patch; hilangkan configuration drift; rollback = deploy ulang definisi known-good.
- **Terraform**: write → **plan** (diff create/change/destroy, tanpa menyentuh apa pun) → review → **apply** (persis plan) → state. Plan membandingkan **state file, infra aktual, desired config**. State di remote storage ter-lock; hilang/korup = insiden umum. Provider menerjemahkan HCL jadi API call.
- **Kubernetes**: control plane (API server = pintu depan, scheduler, etcd, controller manager); worker node dengan kubelet; `kubectl` = HTTP ke API server. Objek: Pod, Deployment (N replica, reconcile), Service (alamat stabil). Kind (node sebagai container, CI, multi-node) vs Minikube (VM/single-node, driver & addon lebih banyak).
- **RBAC** (authorization): Role (satu namespace) + RoleBinding; ClusterRole/ClusterRoleBinding (cluster-wide). Contoh: **ServiceAccount** `ci-deployer` diberi Role "get, list, create pods" di namespace staging.
- **Contoh Terraform**: namespace + **resource quota** (batas total request CPU/memory dalam namespace; penjelasan tambahan).
- **Non-root**: `runAsNonRoot: true`, read-only root fs → blast radius kecil.
- **Admission controller**: setelah authn + authz, sebelum disimpan ke etcd. **Mutating** (ubah request, inject sidecar, default limit) vs **validating** (terima/tolak). **Kyverno**: policy YAML (validate, mutate, generate); `Enforce` menolak, `Audit` mencatat.
- **Defense in depth**: build-time gate (CI scan) → provisioning gate (Terraform plan/review) → access gate (RBAC) → runtime gate (Kyverno).
- **Pitfalls**: manual cluster drift (`kubectl edit`), unlocked shared state, overly broad ClusterRoles, audit-only forever.

1.7 Release Engineering dan GitOps (W05)

- **Release engineering** (GSRE Ch. 8): pemisahan build, test, release; velocity tanpa korbankan stabilitas; release pipeline sebagai produk.
- **Blast radius**: jumlah user/sistem terdampak bila deployment gagal. Tujuan: kecilkan blast radius dan MTTR.

Strategi	Pros	Cons
Big bang	–	Risiko dan blast radius tinggi
Rolling (default K8s)	Tanpa infra tambahan, zero downtime	Dua versi jalan bersamaan; buruk untuk schema DB tidak backward-compatible
Blue-green	Switch instan, rollback instan, green bisa dites dulu	2x kapasitas
Canary	Blast radius minimal, user nyata	Butuh observability + traffic splitting

- **Progressive delivery**: otomasi canary/blue-green berbasis telemetry; **sistem** memutuskan Go/No-Go. Literatur juga membahas **A/B testing** (dua varian ke kelompok user berbeda), **ring deployment** (bertahap per cincin user), **dark launching** (deploy tanpa terlihat user) (penjelasan tambahan).
- **Deployment** (teknis, kode ke production) vs **release** (bisnis, fitur ke user); **feature flag** membuat kode dorman.
- **Rollback**: blameless, frictionless; rollback 30 menit = strategi gagal; tujuan otomatis berbasis SLO error budget.
- **GitOps**: Git single source of truth; 4 prinsip **declarative, versioned & immutable, pulled automatically, continuously reconciled**. Tidak ada `kubectl apply` dari laptop.
- **Push vs pull**: push = CI butuh credential K8s; pull = agent di cluster menarik dari Git. Pull unggul: security, network traversal (outbound), drift detection. Alur pull: build & push image + update manifest di Git → agent poll, render, compare, apply, synced & healthy → drift detection dan self-heal.
- **Helm**: package manager K8s; chart, release (instance chart), repository; `values.yaml` per environment. **ArgoCD**: GitOps controller pull-based (alternatif: Flux); repo server, application controller, reconcile; sync Manual vs Automated (prune & self-heal).
- **DevSecOps**: image signing (Cosign/Notary), manifest scanning (kube-score, Polaris), RBAC ArgoCD per namespace, jangan commit raw secret (External Secrets, Sealed Secrets, Vault), PR approval.
- **Ilustrasi**: Argo Rollouts canary 10%, Analysis Template query Prometheus "error rate > 5%?", gagal → abort otomatis ke v1.
- **Best practice**: immutable artifacts (by digest), Git absolute source of truth (hotfix lewat emergency PR; mencegah konfigurasi shadow IT), metric-driven automated rollback (SLI jelas). **Anti-pattern**: snowflake clusters, push-based "GitOps", big bang Friday, canary tanpa observability.

1.8 DataOps Foundation (W06, ShopStream)

- **Masalah stale CSV**: data berpindah lewat export manual; masalahnya di cara data berpindah, bukan di model.
- **DevOps vs DataOps**: artifact code (stateless) vs data + code (stateful); testing unit/integration/E2E vs data quality/schema/anomaly; failure outage vs garbage in garbage out; Git vs Git + data versioning/schema registry.
- **Data lebih sulit**: berubah saat dunia berubah, gagal diam-diam, input tak terbatas, rollback = tulis ulang histori, producer dan consumer tim berbeda. Kegagalan diam-diam: late/missing data, duplikat, schema drift (struktur berubah di upstream), bad values, data sensitif bocor.
- **Lifecycle**: ingest → store → transform → serve → monitor (feedback ke transform dan ingest). **4 prinsip**: automation, CI/CD for data, observability (pipeline + data health), collaboration. **5 pilar**: automation, quality & testing, observability, incident response, governance & security.

- **Batch (bounded)** vs **streaming (unbounded)**; spektrum nightly → micro-batch → stream. Streaming: latency rendah, kompleks, butuh replayable log, always-on, failure lag/duplikat/poison message.
- **Pub/sub**: producer publish ke topic lewat message broker (Kafka, RabbitMQ), decoupled dalam time, space, rate. **Log** append-only, ordered, immutable, offset → replay (Kafka bukan sekadar queue).
- **Kafka**: topic → partition; key di-hash ke partition; **ordering hanya dalam satu partition**; consumer group: satu partition dibaca tepat satu consumer dalam group, scaling sampai jumlah partition; committed offset; retention; compaction.

Delivery	Urutan	Risiko
At-most-once	Commit lalu proses	Data loss
At-least-once	Proses lalu commit	Duplikat
Exactly-once	Efek sekali	Sulit, terbatas

- **Effectively-once** = at-least-once + consumer idempotent (dedupe `event_id` atau nama object deterministik `topic-partition-startOffset`) + producer `enable.idempotence=true`, `acks=all`.
- **Medallion**: bronze (raw, immutable, jangan diubah), silver (cleaned, masked), gold (features, marts). Lake dipartisi `dt=/hour=` untuk pruning dan backfill; hindari small-file problem.
- **Schema**: schema adalah API. Schema-on-write vs schema-on-read. Format: JSON (tanpa schema), Avro (pipeline, schema resolution), Protobuf (service-to-service, field numbers). **Schema Registry**: producer kirim schema id + payload. **Compatibility**: backward (baru baca lama), forward (lama baca baru), full. Safe: tambah optional + default; breaking: hapus/rename required, ubah tipe (serta tambah required tanpa default, tambahan).
- **Data contract** = schema + semantics + ownership (field, makna, tipe, required/optional, rentang, delivery & freshness, klasifikasi PII), ditegakkan di producer.
- **Drill**: A duplikat → idempotency; B breaking schema → compat check di CI (409 Conflict); C poison message → **DLQ**; D consumer lag → lag monitoring.
- **Pitfalls**: commit sebelum tulis (data loss), lupa key (urutan hilang), satu file per event (small files), tanpa compatibility mode, retry tanpa batas (partition stalled).

1.9 DataOps Quality, Security, Privacy (W07, PayLend)

- **Cost of bad data**: garbage in garbage out diperbesar ML; silent failures; compounding errors; **Reliability = Uptime × Data Quality**.
- **DAMA-DMBOK**: framework (bukan tool), 11 knowledge area dengan Data Governance di tengah. Lensa: Data Quality, Data Security, Metadata Management, diikat Governance.
- **Dimensi DQ**: accuracy, completeness, validity, timeliness, uniqueness, integrity, consistency, reasonability (slide juga menyebut "6 core").

Observasi PayLend	Dimensi
`currency_code = "USDD"`	Validity (bukan kode **ISO 4217**)
amount negatif tanpa `transaction_type`	Completeness
`transaction_id` dua kali	Uniqueness
timestamp 3 hari / masa depan	Timeliness
amount 5.000.000	Reasonability
`user_id` tidak ada di users	Integrity (versi lain slide: Consistency)
negara berbeda antar service	Consistency
beda 0,01 dari ledger	Accuracy

- **DQ lifecycle**: define → measure → analyze root cause → improve → control; perbaiki di sumber.
- **Shift-left & contract**: validasi di edge; lapisan contract structure (Avro), domain (enum/GX), semantics (GX/dbt), SLA (freshness), ownership. Avro enum membuat "USDD" tidak bisa di-serialize.
- **Security vs privacy**: security = CIA; privacy = purpose, consent, minimization, retention. Security tanpa privacy mungkin, privacy tanpa security tidak. PCI-DSS standar security; GDPR / UU PDP undang-undang privacy.
- **PCI** = data kartu pembayaran; **PAN** = nomor kartu lengkap, di-mask menjadi 4 digit terakhir.
- **Metadata menemukan PII**: catalog → glossary → classification tags → policy engine (PII → tokenize, PCI → mask, public → simpan).

Teknik	Utilitas ML	Compliance
Masking	Merusak utilitas	Bagus untuk display
Tokenization (deterministik)	Menjaga referential integrity	Bagus untuk analitik
Encryption	Perlu dekripsi	Terbaik at-rest
Deletion	Tidak ada data	Terbaik untuk minimisasi

- Pseudonymized data tetap data pribadi (GDPR). **Dynamic masking** per role + audit akses. **Attacker's view**: quasi-identifiers, linkage, weak tokens (hash tanpa salt), leaky evidence. Data poisoning = masalah quality dan security.
- **Tools**: Presidio (PII, tokenize dengan HMAC berkunci dari secret store, field-level), Great Expectations (validasi sebelum mendarat, `BOOLEAN_ONLY`), dbt tests (tabel di warehouse, `contract enforced`, `relationships`), schema registry.
- **Pipeline**: Kafka → Presidio → staging → GX → feature store / DLQ → alert. Validasi format SSN **sebelum** tokenize; DLQ sama sensitifnya (enkripsi, batasi, expiry).
- **Saat check gagal**: block (defect tidak aman), quarantine (parsial, ke DLQ), warn (severity rendah), circuit breaker (producer rusak). 40% ke DLQ = incident.
- **Prinsip**: privacy diperbaiki dengan menghapus, quality dengan menolak.

2 Soal Pilihan Ganda (Bisa Lebih dari Satu Benar)

Kerjakan dulu dengan menutup kunci. Setiap opsi dinilai sendiri.

PG 1 (W01a) Anti-pattern rilis menurut Fowler adalah...

A. Deploying software manually

B. Deploying ke production-like environment hanya setelah development selesai

C. Manual configuration management untuk production environment

D. Merilis sesering mungkin dengan delta kecil

E. Menyimpan semua konfigurasi di version control

Kunci: A, B, C. Tiga anti-pattern yang disebut Fowler. D salah: justru yang dianjurkan ("Frequent"). E salah: itu prinsip "keep everything in version control".

PG 2 (W01a) Pernyataan yang sesuai prinsip software delivery Fowler...

A. "Done means released": fitur selesai bila sudah dirilis

B. "If it hurts, do it less frequently"

C. Proses delivery adalah tanggung jawab tim operations

D. "Build quality in": kualitas dibangun sejak awal

E. "Automate almost everything"

Kunci: A, D, E. B salah: yang benar *do it more frequently and bring the pain forward.* C salah: *everybody is responsible.*

PG 3 (W01a) Tentang DevOps...

A. DevOps pada dasarnya adalah sebuah tool otomasi

B. Gerakan DevOps muncul sekitar 2007 karena silo antara developer dan operations

C. Lifecycle DevOps berakhir di tahap Operate

D. Tahap Observe berarti mengidentifikasi dan menyelesaikan issue

E. Collaboration and communication membungkus seluruh lifecycle

Kunci: B, D, E. A salah: practices, tools, **dan** cultural philosophy. C salah: lifecycle adalah infinity loop, berlanjut ke continuous feedback.

PG 4 (W01b) Karakteristik software berbasis ML yang membuat DevOps biasa tidak cukup...

A. Model drift sehingga perlu monitoring dan retraining

B. Perilaku sistem ditentukan sepenuhnya oleh kode

C. Butuh model version control dan tracking karena prosesnya eksperimental

D. Performa model cukup dievaluasi sekali saat training

E. Perlu integrasi dengan tim data science

Kunci: A, C, E. B salah: ada **data dependency**. D salah: perlu **continuous evaluation** dengan feedback loop.

PG 5 (W01b) Tentang MLOps maturity level (Google)...

A. Level 0 bersifat manual dan script-driven, tanpa CI/CD
B. Level 0 sudah punya active performance monitoring
C. Level 1 mengotomasi pipeline training yang dipicu trigger
D. Level 1 sudah mem-build, test, dan deploy komponen pipeline lewat CI/CD
E. Feature store dan ML metadata store muncul di level 1

Kunci: A, C, E. B salah: level 0 *lack of active performance monitoring*. D salah: itu ciri **level 2**.

PG 6 (W01b) Tentang DataOps...

A. Kumpulan practices, procedures, dan technologies data management
B. Bertujuan meningkatkan quality, speed, dan collaboration
C. Data pipeline deployment (ETL ke production) termasuk fase Deployment
D. Data quality checks termasuk fase Planning
E. Three Vs adalah Volume, Velocity, Value

Kunci: A, B, C. D salah: quality checks ada di fase **Testing** (Planning berisi *establish data quality standards*). E salah: V ketiga adalah **Variety**.

PG 7 (W01b) Pernyataan yang benar tentang MLOps...

A. Model bisa di-serve sebagai microservice REST API untuk online prediction
B. Model bisa di-embed di edge atau mobile device
C. Model bisa menjadi bagian batch prediction system
D. MLOps adalah irisan Machine Learning, Data Engineering, dan DevOps
E. Reproducibility tidak termasuk langkah model development

Kunci: A, B, C, D. E salah: reproducibility adalah langkah ke-7.

PG 8 (W02) Perbandingan VM dan container...

A. Setiap container membawa guest OS sendiri
B. VM berjalan di atas hypervisor dengan guest OS masing-masing
C. Container berbagi kernel host dengan isolasi level proses
D. Container start dalam milidetik sampai detik, VM puluhan detik sampai menit
E. VM lebih cocok untuk microservice yang kecil dan disposable

Kunci: B, C, D. A salah: container tanpa guest OS. E salah: container yang cocok untuk microservice; VM saat butuh kernel/OS berbeda.

PG 9 (W02) Ciri aplikasi yang baik untuk di-containerize...

A. Stateless
B. Dijaga oleh specialized team
C. Sering di-update
D. Dependency dan konfigurasi bisa didefinisikan dari luar aplikasi
E. Di-maintain oleh banyak tim sekaligus

Kunci: A, C, D. B dan E justru disebut "the app is **NOT**".

PG 10 (W02) Best practice security container...

A. Jalankan container sebagai root agar tidak terkendala permission
B. Minimal base image untuk mengurangi attack surface
C. Image yang sudah di-deploy boleh di-patch langsung di dalam container
D. Digital signature untuk memverifikasi authenticity dan integrity image
E. Network segmentation untuk membatasi dampak container yang compromised

Kunci: B, D, E. A salah: least privilege, hindari root. C salah: immutable infrastructure, build image baru.

PG 11 (W02) Tentang monitoring dan logging...

A. User reaction to business offerings cukup memakai data aplikasi
B. Intruder detection cukup memakai data infrastruktur
C. Capacity planning memakai data aplikasi dan infrastruktur
D. Postmortem adalah analisis setelah failure terjadi
E. Log rotation dan retention policy mencegah masalah storage

Kunci: A, C, D, E. B salah: intruder detection butuh aplikasi **dan** infrastruktur.

PG 12 (W02) Foundations: Dockerfile, Compose, 12-factor...

A. `COPY requirements.txt` dan install sebelum `COPY . .` supaya layer dependency ter-cache
B. Compose yang menjalankan service beserta db dan MinIO mencerminkan Factor IV
C. Config sebaiknya ditulis di kode agar konsisten antar environment
D. MinIO S3-compatible sehingga pindah ke cloud storage cukup lewat config
E. Branch `feature/*` sebaiknya long-lived sampai semua fitur selesai

Kunci: A, B, D. C salah: Factor III, config di environment. E salah: feature branch short-lived per milestone.

PG 13 (W02) Empat pilar XOps...

A. DevOps diukur dengan delivery metrics: deployment frequency, lead time, change failure rate, time to restore

B. Continuous Training adalah retraining otomatis yang dipicu data baru atau performance decay

C. SecOps adalah tahap terpisah yang dijalankan di akhir sebelum release

D. Lineage dan provenance memungkinkan menelusuri angka di dashboard ke raw record

E. Taksonomi XOps dibuat untuk memisahkan tegas tim per disiplin

Kunci: A, B, D. C salah: SecOps adalah **lensa di setiap tahap** (shift-left saat build time), bukan ditempel di akhir. E salah: taksonomi untuk **shared vocabulary**, bukan tembok antar-disiplin.

PG 14 (W02) 12-Factor App...

A. Factor V memisahkan build, release (build + config), dan run

B. Factor VI: data persisten disimpan di dalam proses agar cepat

C. Factor XI: log diperlakukan sebagai event stream yang ditulis ke stdout

D. Factor VIII: scaling dilakukan dengan membesarkan satu proses

E. Factor I: fork codebase per environment boleh selama tetap di Git

Kunci: A, C. B salah: proses **stateless**, data persisten di backing service. D salah: **scale out** dengan menambah process. E salah: satu codebase, **jangan pernah fork per environment**.

PG 15 (W02) Microservices dan containerization theory...

A. Service sebaiknya dipecah per technical layer (UI, logic, data)

B. Mengakses database service lain secara langsung merusak encapsulation

C. API Gateway berperan sebagai single entry point, routing, dan auth

D. Container memakai isolasi level OS (namespaces, cgroups), bukan virtualisasi hardware

E. Config separation sudah cukup untuk mencegah distributed monolith

Kunci: B, C, D. A salah: pecah per **business capability**; batas per technical layer memaksa cross-service call di hampir setiap aksi. E salah: config separation tidak mengatasi **blast radius**; menurut Conway's Law, tim yang kusut menghasilkan distributed monolith.

PG 16 (W03) Tentang shift-left...

A. Biaya bug: IDE/pre-commit 1x, CI 15x, production 100x+

B. Shift-left memindahkan pengecekan ke tahap akhir sebelum release

C. SAST pada diff dilakukan di tahap Commit

D. Policy checks di tahap Deploy adalah last line of defense

E. Container scan dan sign artifact dilakukan di tahap Code

Kunci: A, C, D. B salah: ke tahap **awal** (kiri). E salah: di tahap **Package**.

PG 17 (W03) Prinsip CI...

A. Urutkan stage dari yang paling murah dan cepat

B. Build ulang image di tiap environment agar sesuai environment

C. Pipeline yang lambat cenderung di-bypass

D. Feature flag memungkinkan pekerjaan yang belum selesai di-merge dengan aman

E. Pipeline merah boleh di-merge bila mendesak

Kunci: A, C, D. B salah: build once, promote everywhere; perbedaan env lewat config saat deploy. E salah: pipeline adalah gate, tidak ada "merge anyway".

PG 18 (W03) Testing pyramid...

A. Unit test paling banyak dan memakai mock/stub

B. Integration test menangkap SQL salah dan serialization mismatch

C. Contract test memverifikasi ekspektasi consumer terhadap API provider

D. Ice cream cone adalah banyak unit test dan sedikit E2E test

E. Integration test sebaiknya mengulang seluruh business logic

Kunci: A, B, C. D terbalik: ice cream cone berat di E2E/manual, sedikit unit test. E salah: scope ketat di titik integrasi.

PG 19 (W03) SAST, SCA, image scanning...

A. SAST bisa menangkap hardcoded secrets

B. SCA hanya memeriksa direct dependency

C. Image scan dijalankan setelah build dan sebelum push ke registry

D. Deploy dengan tag `latest` menjamin yang di-deploy sama dengan yang di-scan

E. `trivy image --exit-code 1 --severity CRITICAL,HIGH` membuat job gagal bila ada temuan Critical/High

Kunci: A, C, E. B salah: direct dan **transitive**. D salah: tag bisa berpindah; pakai **digest**.

PG 20 (W03) Registry, SLSA, pitfalls...

A. SLSA membuktikan artifact dibangun oleh proses tepercaya lewat provenance

B. Credential build boleh punya hak deploy ke production

C. Memblokir build untuk setiap temuan Low bisa menyebabkan alert fatigue

D. Flaky test sebaiknya dibiarkan karena kadang lulus

E. Job GitLab CI dalam satu stage berjalan paralel secara default

Kunci: A, C, E. B salah: credential build tidak boleh deploy ke prod. D salah: perbaiki atau karantina segera.

PG 21 (W04) IaC...

A. Declarative mendeskripsikan end state, tool menghitung langkahnya

B. Script `create_server.sh` yang dijalankan dua kali aman dan idempotent

C. Idempotency adalah sifat logika reconciliation milik tool

D. Immutable infrastructure berarti mem-patch server in place

E. Environments as data: dev/staging/prod memakai module sama dengan variabel berbeda

Kunci: A, C, E. B salah: shell script naif biasanya membuat server dua kali. D terbalik: server **diganti**, bukan di-patch.

PG 22 (W04) Terraform...

A. `terraform plan` menampilkan diff tanpa menyentuh infrastruktur

B. `terraform apply` bisa menjalankan hal berbeda dari plan yang direview

C. Plan membandingkan state file, infrastruktur aktual, dan desired config

D. State sebaiknya disimpan di laptop masing-masing

E. Provider menerjemahkan blok HCL menjadi API call ke platform

Kunci: A, C, E. B salah: apply mengeksekusi **persis** plan yang direview. D salah: remote storage yang shared dan ter-lock.

PG 23 (W04) Kubernetes...

A. etcd menyimpan state cluster

B. `kubectl` berkomunikasi langsung dengan kubelet di tiap node

C. Deployment mendeklarasikan N replica dan terus mencocokkan kenyataan

D. Service memberi alamat stabil di depan pod yang berubah-ubah

E. Scheduler adalah komponen yang menjalankan container di node

Kunci: A, C, D. B salah: `kubectl` = HTTP request ke **API server**. E salah: scheduler memutuskan penempatan; kubelet yang mengelola container di node.

PG 24 (W04) RBAC dan admission control...

A. RBAC mengatur authorization, bukan authentication

B. Role berlaku di seluruh cluster

C. Admission controller berjalan setelah authentication dan authorization, sebelum disimpan ke etcd

D. Validating admission controller bisa meng-inject sidecar

E. Kyverno mode `Audit` hanya mencatat pelanggaran tanpa memblokir

Kunci: A, C, E. B salah: Role berlaku di **satu namespace** (ClusterRole yang cluster-wide). D salah: itu **mutating**.

PG 25 (W04) Defense in depth...

A. Build-time gate = CI scans
B. Provisioning gate = RBAC
C. Access gate = RBAC
D. Runtime gate = Kyverno admission policy
E. Memberi `cluster-admin` sementara agar tidak terblokir adalah praktik yang baik

Kunci: A, C, D. B salah: provisioning gate = **Terraform plan/review**. E salah: anti-pattern *overly broad ClusterRoles*.

PG 26 (W05) Strategi deployment...

A. Rolling update menjalankan dua versi bersamaan selama proses
B. Blue-green butuh 2x kapasitas, tetapi rollback instan
C. Canary butuh observability yang kuat dan traffic splitting
D. Rolling update ideal untuk perubahan schema DB yang tidak backward-compatible
E. Big bang deployment memiliki blast radius rendah

Kunci: A, B, C. D salah: justru limitation rolling update. E salah: big bang blast radius tinggi.

PG 27 (W05) Deployment, release, progressive delivery...

A. Deployment bersifat teknis, release bersifat bisnis
B. Feature flag membuat kode yang sudah di-deploy tetap dorman
C. Pada progressive delivery, manusia memutuskan Go/No-Go lewat dashboard
D. Rollback yang butuh 30 menit menandakan strategi deployment sedang gagal
E. Blast radius adalah jumlah user/sistem terdampak bila deployment gagal

Kunci: A, B, D, E. C salah: **sistem** yang memutuskan Go/No-Go berdasarkan telemetry.

PG 28 (W05) GitOps...

A. Empat prinsipnya: declarative, versioned & immutable, pulled automatically, continuously reconciled
B. Pada pull model, CI server butuh credential Kubernetes
C. Pull model bekerja di balik firewall karena cluster memulai koneksi outbound
D. Hotfix darurat boleh `kubectl apply` langsung tanpa lewat Git
E. ArgoCD dengan sync Automated (prune & self-heal) mengembalikan perubahan manual agar sesuai Git

Kunci: A, C, E. B salah: itu masalah **push** model. D salah: hotfix lewat **emergency PR**.

PG 29 (W05) Helm, DevSecOps, anti-pattern...

A. Helm memakai `values.yaml` untuk mengelola environment berbeda

B. "Release" di Helm adalah instance chart yang berjalan di cluster

C. Raw secret boleh di-commit ke Git selama repo private

D. Canary tanpa observability tetap merupakan canary yang aman

E. Cosign/Notary dipakai untuk image signing

Kunci: A, B, E. C salah: jangan pernah commit raw secret (pakai External Secrets, Sealed Secrets, Vault). D salah: itu hanya partial deployment berisiko.

PG 30 (W06) DataOps vs DevOps...

A. Primary artifact DataOps adalah data + code (stateful)

B. Bug data biasanya gagal dengan keras (crash)

C. Rollback data berarti menulis ulang histori, sehingga sulit

D. Lima pilar DataOps: automation, quality & testing, observability, incident response, governance & security

E. Observability DataOps cukup memastikan pipeline berjalan

Kunci: A, C, D. B salah: bug data sering gagal **diam-diam**. E salah: juga kesehatan **data** (akurat, lengkap, fresh).

PG 31 (W06) Kafka...

A. Ordering dijamin di seluruh partition dalam satu topic

B. Event dengan key yang sama masuk ke partition yang sama

C. Dalam satu consumer group, satu partition dibaca tepat satu consumer

D. Menambah consumer dalam satu group melebihi jumlah partition tetap menambah paralelisme

E. Log append-only dengan offset memungkinkan replay

Kunci: B, C, E. A salah: hanya **dalam satu partition**. D salah: scaling maksimal sebanyak jumlah partition; consumer lebihnya menganggur (tambahan).

PG 32 (W06) Delivery semantics...

A. At-most-once: commit dulu baru proses, risiko data loss

B. At-least-once: proses dulu baru commit, risiko duplikat

C. Consumer crash setelah menulis ke MinIO tetapi sebelum commit menghasilkan duplikat saat restart

D. Effectively-once = at-most-once + retry

E. Nama object deterministik `topic-partition-startOffset` membuat penulisan idempotent

Kunci: A, B, C, E. D salah: effectively-once = **at-least-once + idempotency**.

PG 33 (W06) Schema dan data contract...

A. Menambah field optional dengan default termasuk safe change

B. Rename field required termasuk safe karena datanya tetap ada

C. Backward compatibility: schema baru bisa membaca data lama

D. Schema Registry dengan compatibility mode aktif menolak schema yang tidak compatible (409 Conflict)

E. Data contract hanyalah schema

Kunci: A, C, D. B salah: rename field required = **breaking**. E salah: contract = schema + semantics + ownership. (Tanpa compatibility mode di subject, registry tidak melindungi contract; itu salah satu pitfall.)

PG 34 (W06) Batch, streaming, medallion, DLQ...

A. Batch memproses data bounded, streaming memproses data unbounded

B. Fraud check saat checkout cocok memakai nightly batch

C. Bronze layer boleh diubah untuk membersihkan data

D. DLQ mencegah poison message membuat consumer macet

E. Retry tanpa batas untuk pesan buruk bisa membuat partition stalled

Kunci: A, D, E. B salah: fraud check butuh keputusan dalam detik, jadi **streaming**. C salah: bronze tidak pernah diubah; silver/gold diturunkan darinya.

PG 35 (W07) Pemetaan dimensi DQ pada PayLend...

A. `currency_code = "USDD"` → Validity

B. `transaction_id` terkirim dua kali → Uniqueness

C. `user_id` tidak ada di tabel users → Accuracy

D. `transaction_amount = 5.000.000` → Reasonability

E. Timestamp 3 hari lalu → Completeness

Kunci: A, B, D. C salah: **Integrity** (satu versi slide menyebut Consistency), bukan Accuracy. E salah: **Timeliness**.

PG 36 (W07) Security dan privacy...

A. Security tanpa privacy mungkin terjadi

B. Privacy tanpa security mungkin terjadi

C. PCI-DSS adalah undang-undang privacy

D. Data pseudonymized tetap dianggap data pribadi oleh GDPR

E. Data minimization mengurangi beban confidentiality

Kunci: A, D, E. B salah: privacy tanpa security hanya janji. C salah: PCI-DSS standar **security** industri.

PG 37 (W07) Penanganan PII...

A. Tokenization deterministik menjaga referential integrity

B. Masking menjaga utilitas data untuk ML

C. Hash SSN tanpa salt bisa di-brute-force

D. Validasi format SSN dilakukan setelah tokenize

E. `result_format="BOOLEAN_ONLY"` di Great Expectations mencegah leaky evidence

Kunci: A, C, E. B salah: masking merusak utilitas. D salah: validasi **sebelum** tokenize, karena token selalu terlihat valid.

PG 38 (W07) Saat check gagal dan pemilihan tool...

A. Block dipakai untuk defect yang tidak aman, misalnya PII bocor

B. Quarantine mengirim record buruk ke DLQ dan meneruskan yang baik

C. 40% record masuk DLQ masih dianggap masalah data quality biasa

D. Great Expectations untuk data sebelum mendarat, dbt tests untuk tabel di warehouse

E. DLQ tidak perlu dienkripsi karena isinya record yang gagal

Kunci: A, B, D. C salah: itu **incident**. E salah: DLQ menyimpan raw record, sama sensitifnya dengan sumber.

3 Soal Essay

3.1 Essay 1 (W01a). Automated dan Frequent

Soal

Sebuah tim merilis aplikasi tiga bulan sekali. Deploy dilakukan dengan menyalin file ke server dan mengubah konfigurasi production secara manual. Setiap rilis sering gagal dan rollback sulit. Analisis masalahnya dengan pandangan Fowler dan usulkan perbaikan berdasarkan prinsip software delivery.

Langkah menjawab: kenali anti-pattern → hubungkan ke dua kata kunci → usulkan prinsip → kaitkan ke feedback dan DevOps.

Contoh jawaban

1. **Masalah:** tiga anti-pattern Fowler muncul sekaligus: deploy manual, konfigurasi production manual, dan (kemungkinan) environment mirip production baru dipakai di akhir. Rilis tiga bulan sekali berarti **delta** perubahan besar.
2. **Kenapa gagal:** tanpa **otomasi**, proses tidak repeatable (hasil berbeda tiap kali karena software, konfigurasi, environment, dan proses berubah). Rilis **jarang** membuat delta besar, risiko besar, rollback sulit, feedback lambat.
3. **Perbaikan:**
   - **Automate almost everything**: deployment pipeline (commit stage, automated acceptance, capacity test, manual test, release).
   - **Keep everything in version control**: kode, konfigurasi, skrip environment.
   - **Repeatable, reliable process** dan **if it hurts, do it more frequently**: rilis lebih sering dengan delta kecil.

- **Build quality in**, **done means released**, **everybody responsible**, **continuous improvement**.

4. **Hasil:** feedback dari setiap perubahan datang cepat dan ditindaklanjuti; benefit: empowering teams, reducing errors, lowering stress, deployment flexibility. Ini juga inti lifecycle DevOps (loop discover sampai continuous feedback, menghapus silo).

3.2 Essay 2 (W01b). Menaikkan Level MLOps

Soal

Tim data science melatih model churn di notebook, lalu mengirim file model ke tim operations untuk di-deploy. Model jarang diperbarui dan performanya di production tidak dipantau. Tentukan level MLOps tim ini dan jelaskan langkah menuju level 1 dan level 2.

Contoh jawaban

- **Level 0 (manual)**: script-driven tanpa CI/CD, ML terputus dari operations (model "diserahkan"), rilis jarang, tidak ada monitoring performa aktif. Risikonya **model drift** tidak terdeteksi.
- **Ke level 1 (ML pipeline automation)**:
  - Ubah eksperimen menjadi **orchestrated experiment** dan simpan sebagai source code di repository.
  - Bangun **automated pipeline** di production (extraction, validation, preparation, training, evaluation, validation) yang dipicu **trigger** (data baru atau penurunan performa; jadwal juga lazim, tambahan).
  - Tambah **feature store**, **ML metadata store**, dan **performance monitoring** (model registry sudah ada sejak level 0) yang bisa memicu retraining (continuous training).
- **Ke level 2 (CI/CD pipeline automation)**: komponen pipeline itu sendiri lewat **CI** (build, test, package) dan **CD** (pipeline deployment), lalu model serving lewat CD, dengan monitoring memicu trigger.
- **Peran DataOps**: data as a foundation; pipeline data yang andal (quality checks, validation) menjadi dasar retraining.

3.3 Essay 3 (W02). Migrasi Aplikasi ke Container

Soal

Sebuah aplikasi monolit berjalan di VM. Aplikasi menyimpan sesi user di memori, konfigurasi ditulis langsung di kode, dan dijaga satu tim khusus. Perusahaan ingin memindahkannya ke container. Nilai kesiapannya, jelaskan langkah migrasi, dan sebutkan best practice yang wajib diterapkan.

Contoh jawaban

1. **Penilaian terhadap kriteria containerize:**
   - Menyimpan sesi di memori → **tidak stateless**; perlu dipindah ke backing service (mis. Redis).
   - Konfigurasi di kode → melanggar **config eksternal** (Factor III); pindahkan ke environment variable.
   - Monolit → **cohesion rendah**; pertimbangkan pemecahan menjadi microservice berdasarkan business capability.
   - Dijaga tim khusus → kriteria menyebut app sebaiknya **tidak** dijaga specialized team.

2. **Langkah migrasi:** discovery (bagian mana yang bisa di-containerize, scope dan goal) → requirement assessment (pecah atau tidak, risiko, prinsip migrasi) → container infrastructure design → containerize (Dockerfile per service) → testing → deployment & rollout, diulang per bagian.
3. **Best practice:**
   - Security: minimal base image, scan rutin, **non-root**, image immutable, TLS antar-container, network segmentation, signature, registry aman.
   - Performance: **multi-stage build**, resource requests/limits, caching, health check (liveness/readiness), orchestration (Kubernetes).
   - Logging/monitoring: centralized structured logging (JSON), level yang tepat, rotation & retention; alert pada CPU, memory, latency.
4. **Hasil:** "works on my machine" hilang karena image identik dan reproducible di lokal, CI, production.

3.4 Essay 4 (W02). Service yang Melanggar 12-Factor

Soal

Service `orders` menyimpan password database langsung di kode, membaca tabel milik service `inventory` secara langsung, dan menyimpan isi keranjang belanja di memori proses. Evaluasi desain ini memakai prinsip 12-Factor App dan pola microservices, lalu usulkan perbaikannya.

Contoh jawaban

1. **Password di kode** melanggar **Factor III (Config)**: secret bocor ke version control dan kode yang sama tidak bisa jalan di dua environment. Perbaikan: baca dari environment variable (`os.environ["DB_PASSWORD"]`), `.env` lokal yang tidak masuk Git, ConfigMaps dan Secrets di Kubernetes.
2. **Membaca tabel service lain** melanggar **database-per-service**: merusak encapsulation dan membuat kedua service terikat. Perbaikan: `inventory` memiliki `inventory_db` sendiri; `orders` mengakses lewat API (di belakang **API Gateway** untuk client) atau event (tambahan). Konsistensi antar-service ditangani dengan saga/eventual consistency.
3. **Keranjang di memori** melanggar **Factor VI (Processes)**: proses harus stateless, data persisten di **backing service** (Factor IV, mis. Redis/DB). Tanpa itu, **Factor IX (Disposability)** gagal: orchestrator tidak bisa kill dan reschedule container tanpa kehilangan data, dan **Factor VIII** (scale out) tidak bisa dilakukan.
4. **Batas service**: pastikan dipecah per **business capability** (orders, inventory, billing), bukan per technical layer; satu tim memiliki service end to end. Ingat **Conway's Law**: batas yang buruk menghasilkan **distributed monolith**, dan config separation saja tidak mengatasi blast radius.
5. **Hasil**: image menjadi 12-factor artifact (tanpa baked-in config), dibangun sekali lalu dipromosikan; lingkungan lokal direproduksi dengan Docker Compose (service + db + MinIO) sesuai **Factor IV** (backing services as attached resources), sekaligus mendukung dev/prod parity (Factor X).

3.5 Essay 5 (W03). Merancang CI Pipeline Build-Time

Soal

Rancang CI pipeline untuk service Python berbasis container. Sebutkan urutan stage, gate di tiap stage, dan alasan urutannya. Jelaskan bagaimana pipeline itu memenuhi prinsip CI.

Contoh jawaban

Urutan	Stage	Gate	Tool
1	Lint	Style/syntax lulus	flake8
2	Unit test	Semua lulus, coverage	pytest
3	SAST	Tidak ada temuan Critical/High (quality gate)	SonarQube
4	Build	Image ter-build sekali per commit	docker build
5	SCA	Tidak ada CVE Critical/High pada dependency	Trivy/SCA
6	Image scan	Tidak ada Critical/High di image	`trivy image --exit-code 1`
7	Publish	Hanya di main dan bila semua lulus; push by digest	registry

- **Fail fast fail loud**: stage termurah dan tercepat duluan; pipeline merah memblokir merge.
- **Immutable artifacts**: build sekali, promosi dev → staging → prod lewat **digest**; config dan secret di-inject saat deploy.
- **Fast feedback**: lint, unit test, SAST bisa paralel; cache dependency dan layer.
- **Trunk-based & small batches**: feature branch pendek, feature flag.
- **Testing pyramid**: banyak unit test, integration test seperlunya (DB container ephemeral), contract test untuk API antar service.
- **Supply chain**: registry content-addressed, credential build tanpa hak deploy prod, provenance menuju SLSA.
- **Hindari**: alert fatigue (blok hanya Critical/High), flaky test, tag `latest`, secret di log.

3.6 Essay 6 (W04). Provisioning dan Pengamanan Cluster

Soal

Tim ingin men-deploy image hasil CI ke Kubernetes. Jelaskan bagaimana Terraform, Kubernetes, RBAC, dan Kyverno dipakai, serta bagaimana desain ini membentuk defense in depth. Jelaskan juga kenapa pendekatan deklaratif aman dijalankan berulang.

Contoh jawaban

1. **Provisioning (Terraform)**: tulis `.tf` (namespace + resource quota) → `terraform plan` (diff, direview di PR) → `terraform apply` (persis plan) → state di remote storage ter-lock. Deklaratif dan **idempotent**: dijalankan ulang tidak membuat duplikat karena tool merekonsiliasi desired dengan aktual.
2. **Orkestrasi (Kubernetes)**: Deployment 3 replica dengan image by digest dan `securityContext.runAsNonRoot: true`; Service sebagai alamat stabil; Kubernetes self-healing me-reschedule pod yang crash.
3. **Akses (RBAC)**: ServiceAccount `ci-deployer` dengan Role terbatas (get, list, create pods di namespace tertentu) lewat RoleBinding; hindari ClusterRole lebar.
4. **Runtime (Kyverno)**: ClusterPolicy `require-non-root` dengan `validationFailureAction: Enforce`; berjalan sebagai validating admission controller setelah authn + authz, sebelum disimpan ke etcd. Rollout policy baru boleh mulai dengan `Audit`, lalu dinaikkan ke `Enforce`.
5. **Defense in depth**: build-time gate (CI scan) → provisioning gate (plan/review) → access gate (RBAC) → runtime gate (Kyverno). Setiap lapis independen; satu lapis gagal, lapis lain masih menahan.

6. **Hindari**: `kubectl edit` di resource live (drift), state tanpa locking, `cluster-admin`, audit-only selamanya. **Immutable infrastructure**: perubahan dengan mengganti, bukan patch.

3.7 Essay 7 (W05). Strategi Rilis Fitur Checkout

Soal

Sebuah e-commerce akan merilis fitur checkout baru yang berisiko. Bandingkan rolling update, blue-green, dan canary untuk kasus ini, pilih satu, dan jelaskan alur GitOps pull model serta rollback otomatisnya.

Contoh jawaban

1. **Perbandingan** (lihat tabel bagian 1.7): rolling menjalankan dua versi bersamaan dan berbahaya bila schema DB berubah tidak backward-compatible; blue-green rollback instan tetapi 2x kapasitas; canary membatasi blast radius dan memakai user nyata, tetapi butuh observability dan traffic splitting.
2. **Pilihan: canary dengan progressive delivery**, karena fitur berisiko dan blast radius harus kecil. (Bila ada perubahan schema stateful yang berat, blue-green bisa dipertimbangkan.)
3. **Pisahkan deployment dan release**: kode di-deploy, fitur disembunyikan di balik **feature flag** sampai siap.
4. **Alur GitOps pull**: CI build, test, scan, push image ke registry, lalu update manifest (`values.yaml`, image tag/digest) di Git lewat PR → ArgoCD (di dalam cluster) mendeteksi perubahan, render, compare, apply → **Argo Rollouts** membuat canary 10% → **Analysis Template** meng-query Prometheus (mis. error rate $> 5\%$, p99 latency) → lolos: naikkan bertahap; gagal: **abort otomatis** ke versi stabil.
5. **Kenapa pull**: CI tidak memegang credential cluster, aman di balik firewall, drift detection terus-menerus.
6. **Keamanan**: image signing, manifest scanning, RBAC ArgoCD per namespace, secret lewat Sealed Secrets/Vault, PR approval.
7. **Hindari**: big bang Friday, canary tanpa observability, push-based "GitOps", snowflake cluster.

3.8 Essay 8 (W06). Pipeline Event ShopStream

Soal

`orders-api` sebelumnya mengirim data lewat export CSV manual. Rancang pipeline streaming dari `orders-api` ke data lake. Jelaskan desain topic, partisi dan key, delivery semantics, cara menghindari duplikat, penanganan pesan rusak, dan pengelolaan perubahan schema.

Contoh jawaban

1. **Masalah awal**: CSV manual stale, tergantung orang, tanpa schema; semua pilar DataOps tidak ada.
2. **Desain**: producer `orders-api` publish event Avro (`OrderPlaced`, `PaymentAuthorized`, …) ke topic `shopstream.orders.v1`, key `order_id`, beberapa partition; dead-letter topic `shopstream.orders.dlq`.
3. **Ordering**: key `order_id` membuat semua event satu order masuk partition yang sama, sehingga urutannya terjaga (ordering hanya dalam satu partition).

4. **Consumer group** landing: tiap partition dibaca satu consumer; scaling sampai jumlah partition. Tulis ke **bronze** MinIO dengan layout `dt=/hour=`, buffer agar tidak small-file.
5. **Delivery**: pakai **at-least-once** (proses lalu commit setelah file tertulis). Duplikat dicegah dengan **idempotency**: nama object deterministik `part-<partition>-<startOffset>.avro` atau dedupe `event_id`; producer `enable.idempotence=true`, `acks=all` → **effectively-once**.
6. **Pesan rusak**: validasi; yang gagal dikirim ke **DLQ** dengan metadata error, consumer lanjut; DLQ di-alert, di-triage, diperbaiki, lalu di-replay.
7. **Schema evolution**: Schema Registry dengan compatibility mode (mis. backward); field baru dibuat optional `["null","string"]` default null; **compatibility check di CI** menolak perubahan breaking (409).
8. **Observability**: consumer lag dimonitor dan dikaitkan dengan freshness SLI; reconciliation jumlah event diproduksi = jumlah event di lake.
9. **Security**: kredensial lewat env/secret, image consumer di-scan Trivy, TLS dan topic ACL di production, akses bucket bronze dibatasi karena berisi PII.

3.9 Essay 9 (W07). Mengamankan dan Memvalidasi Event PayLend

Soal

Diberikan event PayLend berisi `ssn`, `credit_card_num`, `full_name`, `home_address`, `transaction_amount = -50.00` tanpa `transaction_type`, `currency_code = "USDD"`. Identifikasi masalah quality dan privacy-nya, lalu rancang pipeline yang menanganinya secara otomatis.

Contoh jawaban

1. **Quality**: `USDD` melanggar **Validity** (bukan ISO 4217); amount negatif tanpa tipe melanggar **Completeness**; bila event terkirim dua kali, **Uniqueness**.
2. **Privacy/security**: `ssn`, `full_name`, `home_address` = PII; `credit_card_num` = PCI. Klasifikasikan lewat metadata (catalog, glossary, tags) lalu policy engine menerapkan kontrol.
3. **Kontrol per field**: `ssn` → **tokenize** (deterministik, HMAC berkunci dari secret store, menjaga join); kartu → **mask** jadi 4 digit terakhir; `home_address` → **delete** (minimisasi); `full_name`: diagram slide menulis "keep, low risk", tetapi payload hasil dan tabel before/after men-drop-nya; sebut keduanya dan pilih drop bila tidak dibutuhkan.
4. **Shift-left & contract**: Avro enum untuk `currency_code` (USDD tidak bisa di-serialize); contract berisi domain, semantics (`amount` ≥ 0, `transaction_type` $\in$ {charge, refund}), SLA freshness, owner.
5. **Pipeline**: Kafka → **Presidio** (validasi format SSN dulu, lalu tokenize/mask per field) → staging → **Great Expectations** (`in_set`, `not_null`, `unique`, `between`, `BOOLEAN_ONLY`) → lolos ke feature store, gagal ke **DLQ** terenkripsi → alert.
6. **Saat gagal**: quarantine untuk record buruk parsial, block bila PII bocor, circuit breaker bila failure rate tinggi (mis. 40% ke DLQ = incident).
7. **Akses**: dynamic masking per role (ML engineer dan analyst melihat `tkn_********`, fraud investigator melihat token penuh) dan audit akses.
8. **Prinsip**: perbaiki di sumber; privacy diperbaiki dengan menghapus, quality dengan menolak.

3.10 Essay 10 (Integratif). Satu Taksonomi X-Ops

Soal

Slide menyebut DevOps, DataOps, MLOps, dan SecOps berbagi satu lifecycle untuk artifact berbeda. Jelaskan pernyataan ini dan tunjukkan bagaimana prinsip shift-left dan immutability muncul di ketiga bidang.

Contoh jawaban

Aspek	DevOps	DataOps	MLOps
Artifact	Code, container image	Data + code (pipeline)	Model + data + code
Testing	Unit, integration, E2E	Data quality, schema, anomaly	Evaluation, validation, monitoring drift
Failure	Outage, bug	Silent bad data	Model drift, prediksi salah
Versioning	Git, image digest	Git + schema registry / data versioning	Model registry, metadata store

- **Lifecycle bersama** (slide W02 Foundations): Plan → Build → Verify → Release → Operate + feedback, dilihat lewat empat lensa DevOps, DataOps, MLOps, SecOps; XOps menerapkan automation, versioning, testing, monitoring ke code, data, dan models.
- **Shift-left**: DevOps (lint, SAST, SCA, image scan di build-time); DataOps (validasi dan contract di edge/ingestion, compat check di CI); MLOps (data validation sebelum training, model validation sebelum serving); SecOps (scan di build, policy di admission).
- **Immutability**: image build once promote by digest; immutable infrastructure; bronze layer raw immutable; Git sebagai source of truth (GitOps).
- **Otomasi + feedback**: CI/CD, continuous training, pipeline data otomatis, rollback berbasis metrik.

4 Ringkasan Inti

Bawa ke ruang ujian (di kepala)

- **Fowler**: automated + frequent, feedback, 8 prinsip; DevOps = practices + tools + culture, hapus silo.
- **MLOps level 0/1/2**: manual → pipeline training otomatis → CI/CD untuk pipeline. **DataOps**: practices/procedures/technologies, quality-speed-collaboration.
- **Foundations**: 4 pilar XOps (DevOps delivery metrics, DataOps, MLOps CT, SecOps sebagai lensa); 12 faktor (pisahkan code dari config/environment); microservices per business capability, API Gateway, database-per-service; Conway's Law.
- **Container**: tanpa guest OS, ringan; kriteria stateless, high cohesion, config eksternal; best practice non-root, immutable, multi-stage.
- **CI**: shift-left (1x/15x/100x+), fail fast, build once by digest, pyramid bukan cone, gate SAST/SCA/image scan.
- **IaC**: declarative, idempotent, plan sebelum apply, state ter-lock; K8s + RBAC + admission (Kyverno) = defense in depth.
- **Release**: blast radius; rolling/blue-green/canary; deployment ≠ release; GitOps pull model dengan reconciliation dan rollback otomatis.

- **DataOps**: Kafka partition/key/consumer group, at-least-once + idempotency, DLQ, schema compatibility, data contract.
- **DQ & privacy**: dimensi DMBOK, shift-left, security $\neq$ privacy, tokenize/mask/delete, validasi sebelum tokenize, block/quarantine/warn/circuit breaker.