AI-Driven CI/CD — Four Methodologies

Scope: Procedure-only, step-by-step implementation plans derived from our previous design for a Jenkins + Multi-OS (ARM/Linux/Windows) pipeline with progressive delivery, anomaly detection, auto-rollback, and continuous learning. **No code** included.

Methodology 1: AI-Driven Adaptive Deployment Architecture (Jenkins + Multi-OS)

Objectives

- Automatically select **canary / blue-green / rolling** strategy per change.
- Orchestrate deployments across **ARM, Linux (K8s), Windows (IIS/services)**.
- Capture decision evidence and publish to **Artifactory Build-Info**.

Inputs

- Git change metadata (diff, LOC, files, deps, coverage delta)
- Historical success/rollback stats
- Current traffic/load & business calendar

Outputs

- Chosen strategy + rollout pace (% weights or batches)
- Deployment status (promoted/paused/rolled back)
- Evidence bundle (metrics snapshots, decisions, links)

Procedure (Implementation Steps)

1. **Establish Environments & Targets**
2. Define prod/stage clusters (K8s), Windows pools/sites, ARM cohorts.
3. Catalog services and OS support matrix.
4. **Define Strategy Policies**
5. Thresholds for **risk→strategy** mapping (Rolling < Canary < Blue/Green).
6. Allowed time windows; change freeze rules.
7. **Collect Decision Signals**
8. Standardize metrics labels (RED/USE) and user KPIs.
9. Normalize Jenkins build/test outcomes and dependency changes.
10. **Risk Scoring (Heuristic v1)**
11. Weight LOC, dep bumps, critical modules, past failure rate, coverage delta.
12. Produce `risk_score` 0–100; persist with the build record.
13. **Select Strategy & Pace**
14. Map score→strategy; define pace schedule (e.g., 10→25→50→100 or Blue/Green cutover gate).
15. **Wire Orchestrators**

16. K8s: prepare Helm/Argo Rollouts/Flagger specs.
17. Windows: prepare Blue/Green slots, ARR/LB weights.
18. ARM: define OTA cohorts and promotion rules.
19. **Gate with Observability**
20. Pre-deploy smoke checks; ensure metrics/alerts present for each service.
21. **Execute & Record**
22. Run the selected strategy; record decisions, windows, SLO status.
23. **Finalize & Publish**
24. On success: promote to 100%, tag **latest** in Artifactory; on failure: mark rollback and freeze.
25. **Review & Tune**
26. Weekly review of risk thresholds vs. outcomes; adjust mappings and paces.

Success Criteria

- <2% unexpected rollbacks; >30% reduction in mean deploy time; full decision evidence attached to artifacts.

Methodology 2: Real-Time Monitoring with Anomaly Detection

Objectives

- Detect regressions during rollout in **minutes**.
- Produce a **verdict**: promote / slow / rollback.

Inputs

- Metrics (error rate, p95/p99 latency, CPU/memory, throughput)
- Logs (error signatures, new patterns)
- Traces (error spans, slow endpoints)
- Baseline from last stable release (same time-of-day if seasonal)

Outputs

- Window-by-window health status
- Anomaly score + SLO compliance
- Action recommendation (promote/slow/rollback) with reasons

Procedure (Implementation Steps)

1. **Standardize Telemetry**
2. Instrument apps with OpenTelemetry; ensure consistent labels.
3. Enable exporters for K8s (kube-state, cAdvisor), Windows (IIS/Win exporters), ARM beacons.
4. **Define SLOs & Windows**
5. Set service SLOs (e.g., error<1%, p95<300ms); choose window length (1–2m) and required consecutive passes.
6. **Create Baselines**
7. Snapshot metrics for the last stable version; align to comparable load periods.

8. **Recording Rules & Alerts**
9. Create summarized series (error ratio, latency quantiles, CPU/mem utilization).
10. **Anomaly Methods (Phase-in)**
11. Phase 1: thresholds + EWMA smoothing.
12. Phase 2: baseline z-scores; seasonal decomposition (STL).
13. Phase 3: multi-metric outlier detection (Isolation Forest) for hard drifts.
14. **Decision Policy**
15. Encode promote/slow/rollback based on SLO + anomaly results and risk level.
16. **Integrate with Rollouts**
17. Connect analysis steps to rollout gates (K8s AnalysisTemplates, Jenkins stages, Windows checks).
18. **Evidence & Notifications**
19. Store verdicts, plots, and links; notify Slack/Jira with context and runbook.
20. **Shadow Period**
21. Run detection in observe-only mode; measure false positives/negatives.
22. **Enforce**
23. Activate gating once shadow metrics are acceptable; review monthly.

Success Criteria

- Median detection time < 5m; false-positive rate < 10%; no undetected critical regressions.

Methodology 3: Auto-Rollbacks & Self-Healing

Objectives

- Reduce blast radius via **fast, safe reversion**.
- Automatically fix common infra/app issues without human intervention.

Inputs

- Anomaly verdicts; SLO breaches; health checks
- Catalog of **idempotent playbooks** per platform (K8s, Windows, ARM)
- Last-known-good versions and configs

Outputs

- Completed rollback actions (version/traffic/capacity)
- Executed self-healing actions with status
- Audit trail and freeze conditions when necessary

Procedure (Implementation Steps)

1. **Define Rollback Policies**
2. Criteria by stage (% traffic) and severity; freeze rules post-rollback.
3. **Catalog Playbooks**
4. K8s (abort/promo to stable, helm rollback, restart, HPA/VPA), Windows (slot swap, ARR weight, AppPool restart), ARM (cohort revert, exclude devices).

5. **Establish Safety**
6. Idempotency, locks per service, exponential backoff, and caps on retries.
7. **Wire Controllers**
8. Create discrete rollback and self-healing controllers invoked by CD gates.
9. **Config & Schema Safety**
10. Maintain LKG configs; apply expand/contract for DB migrations; use feature flags for risky paths.
11. **Verification**
12. Post-action checks (health endpoints, KPIs) before unfreezing or re-promoting.
13. **Evidence & Comms**
14. Persist decisions, timestamps, affected cohorts/nodes; auto-open incident tickets with links.
15. **Chaos & Drills**
16. Regular game days to validate automation; update runbooks from findings.
17. **Governance**
18. Review rollbacks weekly; prune ineffective playbooks; refine thresholds.

Success Criteria

- MTTR reduction > 50%; rollback correctness ~100%; no repeated incident without runbook update.

Methodology 4: Continuous Learning from Logs → Pipeline Efficiency

Objectives

- Turn operational data into **policy improvements** that speed up builds, tests, and safer deployments.

Inputs

- Jenkins build/test outcomes, durations, cache hits
- Deploy decisions & outcomes (promote/slow/rollback)
- Observability metrics; log templates; trace errors
- Artifact metadata & security scans

Outputs

- Updated policies: risk thresholds, test/build selection, rollout pacing
- Flaky test quarantine lists; cache optimization guidance
- Governance reports and dashboards

Procedure (Implementation Steps)

1. **Ingestion & Normalization**
2. Consolidate logs/metrics/build data into a unified schema (parquet in object store).
3. **Feature Store Setup**
4. Create offline/online features (LOC, dep bumps, fanout, flakiness, cacheability, early anomaly scores).
5. **Define Targets & KPIs**

6. Predictors for failure/rollback; objectives for time saved, precision/recall, false-skip caps.
7. **Train & Evaluate Policies**
8. Train models (risk, test selection, build skip, pacing); evaluate vs rolling baselines.
9. **Shadow Policies**
10. Run counterfactual simulations; compare to status quo without affecting prod.
11. **Controlled Rollout**
12. Canary the policies on subset of repos/services; enable rollback to prior policy version.
13. **Governance & Versioning**
14. Track experiments and register policies; store lineage linking builds, data, and decisions.
15. **Feedback Integration**
16. Feed improved policies back into Methodologies 1–3; schedule periodic retraining.
17. **Reporting**
18. Dashboards for CI time saved, defect detection efficiency, rollback rate, canary dwell time, cost per change.

Success Criteria

- ≥25% CI time saved; ≥50% less flaky test noise; stable or reduced rollback rate; positive cost trend.

Cross-Cutting Readiness Checklist

- 🔲Metrics coverage for every service & environment
- 🔲Baseline references for last stable release per service
- 🔲Clear SLOs and alert routes with runbooks
- 🔲Risk→Strategy policy file versioned in Git
- 🔲Idempotent rollback & healing playbooks tested in staging
- 🔲Evidence pipeline to Artifactory/DB + notification hooks
- 🔲Weekly review cadence and owners for thresholds/models/policies

---

RACI Snapshot (Who does what)

- **Owners:** Platform/DevOps team (policies, controllers, observability)
- **Contributors:** Service teams (SLOs, KPIs, coverage mapping)
- **Reviewers:** SRE & Security (gates, runbooks, failure classes)
- **Approvers:** Engineering leadership (risk thresholds, freeze policies)

Implementation Roadmap (Phased)

- **Phase 0 (2–3 weeks):** Telemetry hygiene, SLOs, environment matrix, artifact tagging.
- **Phase 1 (3–4 weeks):** Adaptive strategy (heuristic), anomaly thresholds, evidence pipeline.
- **Phase 2 (4–6 weeks):** Auto-rollback playbooks, shadow anomaly, controlled enforcement.
- **Phase 3 (6–10 weeks):** Continuous learning loop (shadow → canary → full), flaky/test selection, build skip predictor.

- **Ongoing:** Monthly policy review, chaos drills, governance reports.