# 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**
   * Define prod/stage clusters (K8s), Windows pools/sites, ARM cohorts.
   * Catalog services and OS support matrix.
2. **Define Strategy Policies**
   * Thresholds for **risk→strategy** mapping (Rolling < Canary < Blue/Green).
   * Allowed time windows; change freeze rules.
3. **Collect Decision Signals**
   * Standardize metrics labels (RED/USE) and user KPIs.
   * Normalize Jenkins build/test outcomes and dependency changes.
4. **Risk Scoring (Heuristic v1)**
   * Weight LOC, dep bumps, critical modules, past failure rate, coverage delta.
   * Produce risk\_score 0–100; persist with the build record.
5. **Select Strategy & Pace**
   * Map score→strategy; define pace schedule (e.g., 10→25→50→100 or Blue/Green cutover gate).
6. **Wire Orchestrators**
   * K8s: prepare Helm/Argo Rollouts/Flagger specs.
   * Windows: prepare Blue/Green slots, ARR/LB weights.
   * ARM: define OTA cohorts and promotion rules.
7. **Gate with Observability**
   * Pre‑deploy smoke checks; ensure metrics/alerts present for each service.
8. **Execute & Record**
   * Run the selected strategy; record decisions, windows, SLO status.
9. **Finalize & Publish**
   * On success: promote to 100%, tag **latest** in Artifactory; on failure: mark rollback and freeze.
10. **Review & Tune**

* 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**
   * Instrument apps with OpenTelemetry; ensure consistent labels.
   * Enable exporters for K8s (kube‑state, cAdvisor), Windows (IIS/Win exporters), ARM beacons.
2. **Define SLOs & Windows**
   * Set service SLOs (e.g., error<1%, p95<300ms); choose window length (1–2m) and required consecutive passes.
3. **Create Baselines**
   * Snapshot metrics for the last stable version; align to comparable load periods.
4. **Recording Rules & Alerts**
   * Create summarized series (error ratio, latency quantiles, CPU/mem utilization).
5. **Anomaly Methods (Phase‑in)**
   * Phase 1: thresholds + EWMA smoothing.
   * Phase 2: baseline z‑scores; seasonal decomposition (STL).
   * Phase 3: multi‑metric outlier detection (Isolation Forest) for hard drifts.
6. **Decision Policy**
   * Encode promote/slow/rollback based on SLO + anomaly results and risk level.
7. **Integrate with Rollouts**
   * Connect analysis steps to rollout gates (K8s AnalysisTemplates, Jenkins stages, Windows checks).
8. **Evidence & Notifications**
   * Store verdicts, plots, and links; notify Slack/Jira with context and runbook.
9. **Shadow Period**
   * Run detection in observe‑only mode; measure false positives/negatives.
10. **Enforce**

* 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**
   * Criteria by stage (% traffic) and severity; freeze rules post‑rollback.
2. **Catalog Playbooks**
   * K8s (abort/promo to stable, helm rollback, restart, HPA/VPA), Windows (slot swap, ARR weight, AppPool restart), ARM (cohort revert, exclude devices).
3. **Establish Safety**
   * Idempotency, locks per service, exponential backoff, and caps on retries.
4. **Wire Controllers**
   * Create discrete rollback and self‑healing controllers invoked by CD gates.
5. **Config & Schema Safety**
   * Maintain LKG configs; apply expand/contract for DB migrations; use feature flags for risky paths.
6. **Verification**
   * Post‑action checks (health endpoints, KPIs) before unfreezing or re‑promoting.
7. **Evidence & Comms**
   * Persist decisions, timestamps, affected cohorts/nodes; auto‑open incident tickets with links.
8. **Chaos & Drills**
   * Regular game days to validate automation; update runbooks from findings.
9. **Governance**
   * 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**
   * Consolidate logs/metrics/build data into a unified schema (parquet in object store).
2. **Feature Store Setup**
   * Create offline/online features (LOC, dep bumps, fanout, flakiness, cacheability, early anomaly scores).
3. **Define Targets & KPIs**
   * Predictors for failure/rollback; objectives for time saved, precision/recall, false‑skip caps.
4. **Train & Evaluate Policies**
   * Train models (risk, test selection, build skip, pacing); evaluate vs rolling baselines.
5. **Shadow Policies**
   * Run counterfactual simulations; compare to status quo without affecting prod.
6. **Controlled Rollout**
   * Canary the policies on subset of repos/services; enable rollback to prior policy version.
7. **Governance & Versioning**
   * Track experiments and register policies; store lineage linking builds, data, and decisions.
8. **Feedback Integration**
   * Feed improved policies back into Methodologies 1–3; schedule periodic retraining.
9. **Reporting**
   * 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.