# AI-Driven Adaptive Deployment Architecture (Jenkins + Multi-OS)

This document shows a practical reference architecture to add **AI-powered Adaptive Deployment Strategies** (blue/green, canary, rolling) to your existing Jenkins pipeline spanning **ARM, Linux, and Windows** builds. It also details the **monitoring and auto-rollback** loop, artifacts/versioning, and tooling choices.

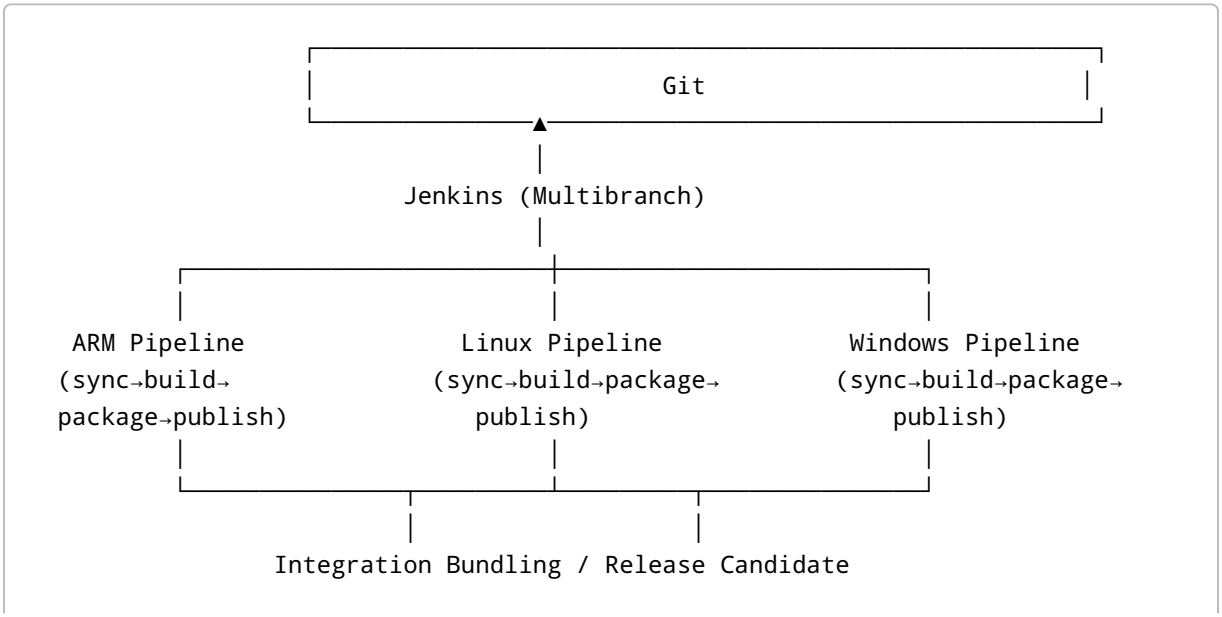
---

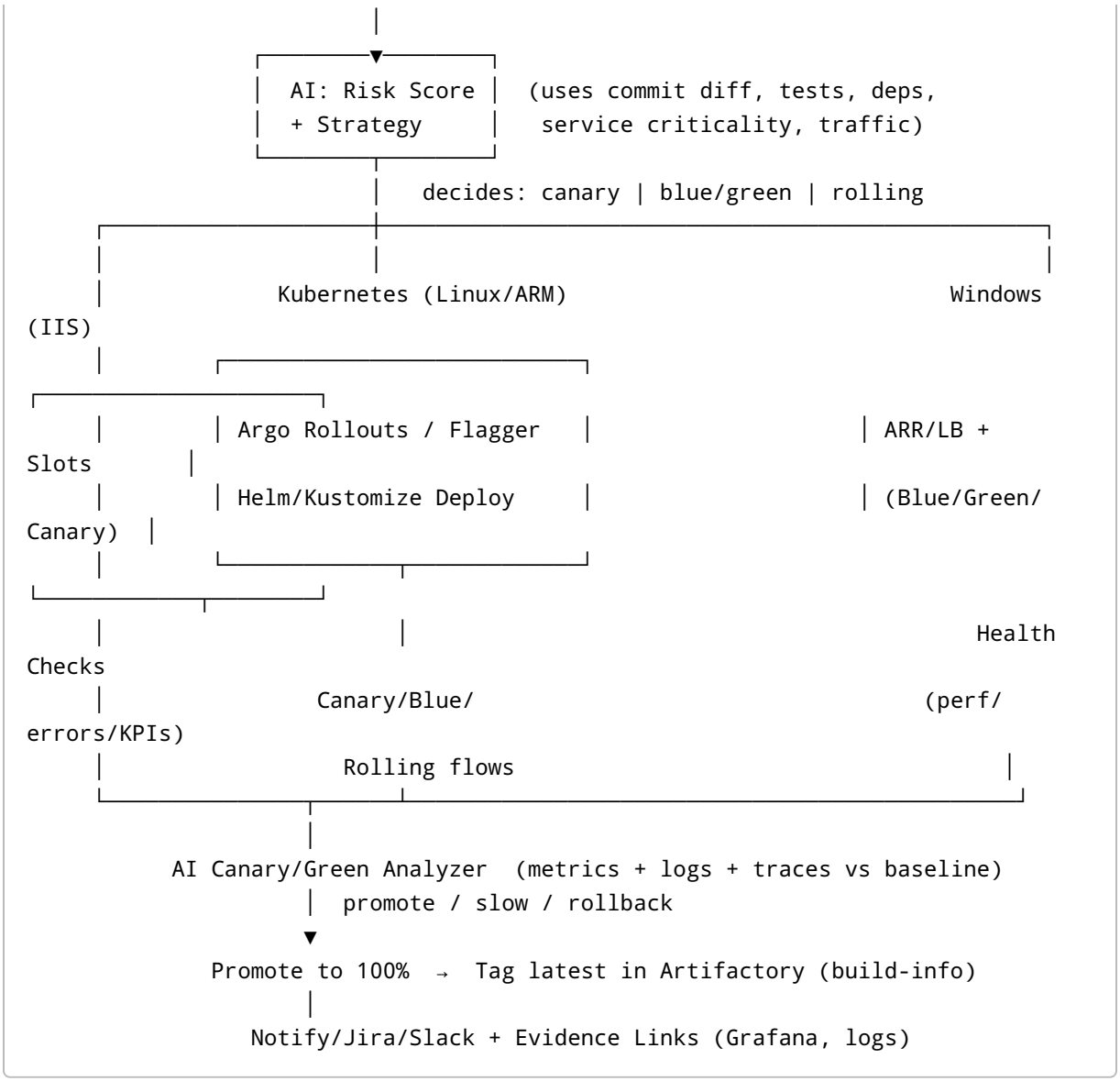
## 1) High-Level Overview

- **Code & Artifacts:** Git → Jenkins (multi-OS parallel pipelines) → Artifactory
- **Orchestration:** Kubernetes (Linux/ARM services), Windows (IIS/Windows Service + ARR/LB), ARM/edge cohorts (OTA manager)
- **Progressive Delivery:** Argo Rollouts or Flagger (K8s), ARR pool weights (Windows), cohort percentages (ARM)
- **AI Layer:**
- *Risk Scoring* (pre-deploy): choose strategy & pace
- *Canary/Blue-Green Analysis* (during deploy): promote/slow/rollback
- **Observability:** OpenTelemetry → Prometheus (metrics) + Loki/ELK (logs) + Tempo/Jaeger (traces) + Grafana (dashboards)
- **Policy/Safety:** OPA gates, SLO error budgets, change windows
- **Release Evidence:** Build-Info + deployment decisions saved to Artifactory (or DB)

---

## 2) Architecture Diagram (ASCII)






---

## 3) End-to-End Flow (Stages)

1. **Parallel builds (ARM/Linux/Windows)** → publish artifacts to Artifactory.
2. **Integration Bundle** (compose cross-OS deliverables if needed).
3. **AI: Risk Score & Strategy Selection** → emits `DEPLOY_STRATEGY` + pace.
4. **Deploy per target**
5. *K8s*: Helm + Argo Rollouts/Flagger (canary, B/G, rolling).
6. *Windows*: IIS slots (B/G), ARR pool weights (canary), rolling batches.
7. *ARM/Edge*: OTA cohorts 1% → 5% → 20% → 100%.
8. **AI Analyzer** monitors SLOs/KPIs vs last-stable baseline → promote/slow/rollback.
9. **Publish Evidence**: risk score, strategy, health windows, SLOs, result → Artifactory Build-Info (+ dashboard links).

## 4) Tooling Choices (Reference Stack)

- **CI/CD:** Jenkins (Shared Library for AI stages)
- **Packaging:** Docker/Helm for k8s; MSI/NSSM/IIS for Windows; OTA images for ARM
- **Registry/Artifacts:** JFrog Artifactory
- **Progressive Delivery:** Argo Rollouts or Flagger (k8s); IIS slots/ARR (Windows); Mender/Balena/ Custom (ARM)
- **Observability:**
- Metrics: Prometheus (+ kube-state-metrics, cAdvisor)
- Logs: Loki or ELK
- Traces: OpenTelemetry SDKs → Tempo/Jaeger
- Dashboards/Alerts: Grafana
- **AIOps/AI:** Python microservices/scripts (risk scoring, canary analysis) packaged as containers
- **Policy:** OPA/Gatekeeper or Jenkins quality gates
- **Comms:** Slack/MS Teams; Jira for incidents

---

## 5) Monitoring & Auto-Rollback Design

### 5.1 Telemetry Model (per service)

- **RED/USE metrics**: request rate, error rate (HTTP 5xx), p95/p99 latency; CPU, memory, saturation
- **Business KPIs**: success ratio (e.g., payment success), downstream error ratio
- **Windows-specific**: IIS AppPool health, Event Log error signatures
- **ARM-specific**: device error beacons, OTA success/failure

### 5.2 Baseline & Comparator

- Baseline = last stable release's metrics on the **same time-of-day**/load or synthetic load.
- Compare candidate vs baseline using **rolling windows** (e.g., 2 min windows for 10–20 minutes).
- Statistical checks: EWMA drift, z-score, non-parametric tests (Mann–Whitney) for latency distributions.

### 5.3 Decision Policy

- **Promote** if all SLOs healthy for `N` consecutive windows.
- **Slow** if marginal (near SLO), extend observation & reduce pace.
- **Rollback** on sustained SLO breach, sharp error spike, or crash loops.

### 5.4 Alerting & Evidence

- Grafana Alerting/Alertmanager routes to Slack/Jira with **runbook links**.
- Store analyzer output JSON + plots in artifact repo (tied to build number).

---

# 6) AI Components

## 6.1 Risk Scoring (pre-deploy)

**Inputs:** lines changed, files, dependency bumps, test coverage Δ, historical failure rate of module, criticality tag, current QPS/load, change window (peak/off-peak).

**Output:** `{ risk_score: 0–100, strategy: canary|bluegreen|rolling, pace }`

**Phase 1 (heuristic):** weighted sum with clamps. **Phase 2 (ML):** train logistic/XGBoost using your history (label: success/rollback).

## 6.2 Canary/Green Analyzer (during deploy)

**Inputs:** Prometheus queries for candidate & baseline; logs anomaly score; trace error spans.

**Logic:** compute SLO status + drift score → action `promote|slow|rollback`.

---

# 7) Kubernetes: Progressive Delivery (example snippets)

## 7.1 Argo Rollouts (Canary)

```
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: svc
spec:
  strategy:
    canary:
      steps:
      - setWeight: 10
      - pause: {duration: 2m}
      - analysis: { templates: [{ templateName: kpi-check }]}
      - setWeight: 25
      - pause: {duration: 2m}
      - setWeight: 50
      - pause: {duration: 3m}
      - setWeight: 100
```

## 7.2 AnalysisTemplate (Prometheus SLOs)

```
apiVersion: argoproj.io/v1alpha1
kind: AnalysisTemplate
```

```
metadata:
  name: kpi-check
spec:
  metrics:
  - name: error-rate
    provider:
      prometheus:
        address: http://prometheus.monitoring:9090
        query: sum(rate(http_requests_total{job="svc",status=~"5.."}[1m])) /
               sum(rate(http_requests_total{job="svc"}[1m]))
    failureCondition: result > 0.01
  - name: p95-latency
    provider:
      prometheus:
        address: http://prometheus.monitoring:9090
        query: histogram_quantile(0.95,
sum(rate(http_request_duration_seconds_bucket{job="svc"}[1m])) by (le))
    failureCondition: result > 0.3
```

### 7.3 Blue/Green (K8s)

- Maintain `svc-blue` and `svc-green` Deployments.
- Flip Service/Ingress to point to `green` after analysis passes.

---

## 8) Windows: Blue/Green & Canary

- **Blue/Green:** Two IIS sites (Blue = live, Green = new). Swap bindings after health checks.
- **Canary:** Use ARR/LB to route 10%/25%/50% traffic to the Green pool; advance on healthy windows.
- **Rolling:** Update node pool in batches; verify Event Logs + synthetic checks.

**Telemetry:** WinRM/Telegraf → Prometheus; ship Event Logs to Loki/ELK; expose custom app metrics endpoint.

---

## 9) ARM/Edge: Cohort Rollouts

- Tag devices into cohorts (1% → 5% → 20% → 100%).
- OTA manager (e.g., Mender/Balena or custom) pushes candidate to next cohort on **AI analyzer** approval.
- Health signals: heartbeat, error codes, crash rate, customer-visible KPIs.

---

## 10) Jenkinsfile Skeleton (key AI stages)

```
stage('AI: Decide Strategy') {
  steps {
    sh 'python3 ci/ai_decide_strategy.py --inputs ci/changes.json >
strategy.json'
    script {
      def s = readJSON file: 'strategy.json'
      env.DEPLOY_STRATEGY = s.strategy
      env.RISK_SCORE = s.risk
      env.PACE = s.pace
    }
  }
}

stage('Deploy (by strategy)') {
  steps {
    script {
      if (env.DEPLOY_STRATEGY == 'canary') {
        sh 'helm upgrade --install svc ./charts/svc --set canary.enabled=true --
wait'
      } else if (env.DEPLOY_STRATEGY == 'bluegreen') {
        sh './ci/flip_blue_green.sh'
      } else {
        sh 'helm upgrade --install svc ./charts/svc --set rolling=true --wait'
      }
    }
  }
}

stage('AI: Analyze & Decide') {
  steps {
    sh 'python3 ci/analyze_metrics.py --slo slo.yaml > verdict.json'
    script {
      def v = readJSON file: 'verdict.json'
      if (v.action == 'promote') { sh './ci/promote_rollout.sh' }
      else if (v.action == 'slow') { sh './ci/slowdown.sh' }
      else { sh './ci/rollback.sh'; error('Rollback triggered') }
    }
  }
}

stage('Publish Evidence') {
  steps {
    sh """
      jf rt build-add-props my-build ${env.BUILD_NUMBER}
```

```
        risk=${env.RISK_SCORE};strategy=${env.DEPLOY_STRATEGY};result=$
{currentBuild.currentResult}
      jf rt build-publish my-build ${env.BUILD_NUMBER}
      jf rt set-props 'repo/path/*${env.BUILD_TAG}*' latest=true
    """
  }
}
```

---

## 11) Prometheus SLO/Alert Examples

```
# Error rate (>1% for 3m)
- alert: HighErrorRate
  expr: sum(rate(http_requests_total{job="svc",status=~"5.."}[1m])) /
        sum(rate(http_requests_total{job="svc"}[1m])) > 0.01
  for: 3m
  labels: { severity: critical }
  annotations:
    summary: "svc error rate > 1%"

# p95 latency (>300ms for 5m)
- alert: HighLatencyP95
  expr: histogram_quantile(0.95,
sum(rate(http_request_duration_seconds_bucket{job="svc"}[1m])) by (le)) > 0.3
  for: 5m
  labels: { severity: warning }
  annotations:
    summary: "svc p95 latency regression"
```

---

## 12) Policy & Safety Gates

- **OPA**: block deploys during peak windows or when error budget exhausted.
- **Change windows**: calendar-based Jenkins gate.
- **Manual approval**: optional for high risk (>70) before Blue/Green flip.

---

## 13) Dashboards (Grafana)

- **Service Health**: RED/USE metrics + error budget burn-down
- **Release Compare**: baseline vs candidate (dual-axis)
- **Windows/IIS**: app pool restarts, HTTP errors, queue length
- **ARM Fleet**: cohort progress, device failures, OTA success rate

## 14) Rollout Runbook (Ops)

1. Check **Risk Score** & chosen strategy.
2. Watch **Release Compare** dashboard during canary windows.
3. If alerts fire → Jenkins will roll back; capture ticket with evidence.
4. On success → promote to 100% and mark **latest** in Artifactory.

---

## 15) Phased Adoption Plan

- **Phase 1:** Heuristic risk + Prometheus SLO checks (no ML training needed).
- **Phase 2:** Train failure-probability model; add ARR/Windows + ARM cohorts.
- **Phase 3:** Add unsupervised drift detection (Isolation Forest), OPA budgets, and feature-flag canaries at user level.

---

### Deliverables You Can Implement Next

- `ci/ai_decide_strategy.py` (heuristic now, ML-ready signature)
- `ci/analyze_metrics.py` (PromQL queries + decision JSON)
- Helm/Argo Rollouts manifests + AnalysisTemplate
- Jenkins shared library steps for **decide / deploy / analyze / promote / rollback**
- Grafana dashboards: **Release Compare**, **SLOs**, **ARM Cohorts**, **Windows/IIS**
- Alerting rules + runbook links