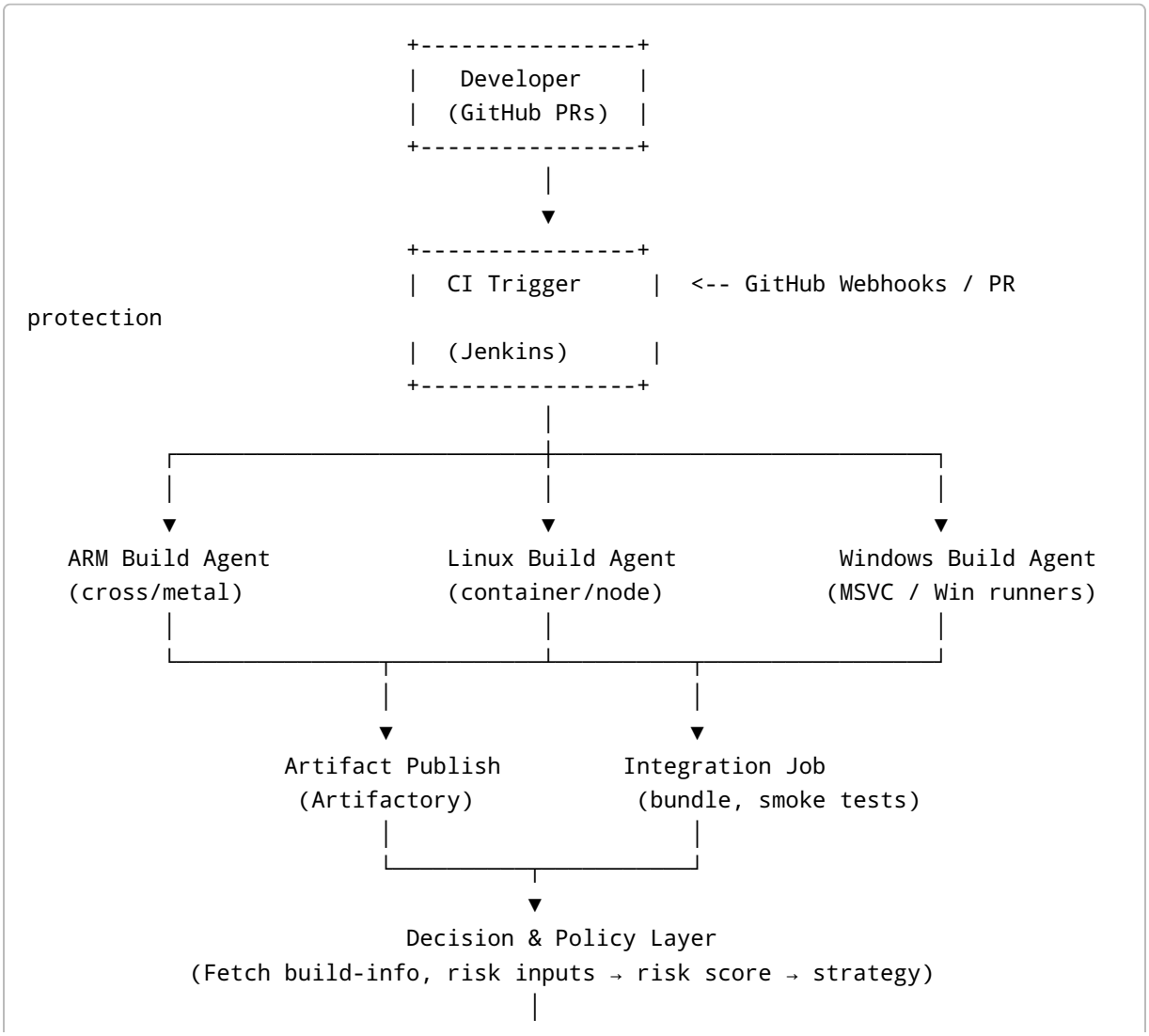
End-to-End Architecture: AI-Driven Adaptive CI/CD (Jenkins + Multi-OS)

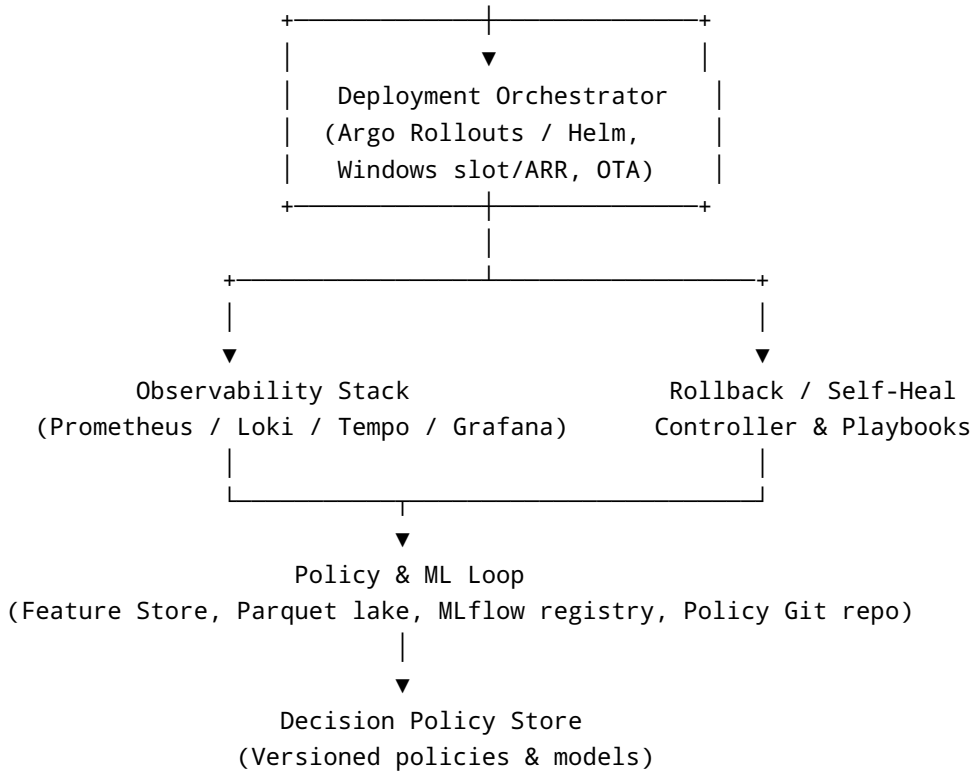
This document describes a complete, end-to-end architecture for the pipeline we built out: from developer push → multi-OS builds (ARM/Linux/Windows) → artifact publishing → AI-driven risk scoring → strategy selection (Rolling/Canary/Blue-Green) → progressive deployment + real-time monitoring → auto-rollback/self-heal → continuous learning and policy improvement.

It is written as a clear architecture diagram (ASCII + sections), component responsibilities, data flows, control flows, and operational notes so you can implement it in your organization.

---

1) High-level ASCII Architecture





---

2) Key Components & Responsibilities

- **Developer / GitHub**: Authors of C++ repos (drivers, firmware, user-space). Triggers CI via PR/push.

- **Jenkins (CI)**: Multibranch pipeline that:

- checks out code, runs linters & static analysis
- runs per-OS build stages in parallel (ARM/Linux/Windows)
- runs unit and emulator/hardware-in-the-loop smoke tests
- produces `build-info.json` and `risk-inputs.json`

- publishes artifacts to Artifactory under `candidates/` with metadata

- **Artifactory**: Stores artifacts + attached metadata (build-info, SBOM, test reports, decision audit links). Holds `latest` and `candidate` namespaces.

- **Integration Job**: Bundles multi-repo artifacts (driver+firmware) and runs cross-repo integration smoke tests in isolated Green-like environment.

- **Decision & Policy Layer**:

- Fetches `risk-inputs.json`

• Loads policy bundle (weights or model) from Policy Git or Policy Service
• Computes `risk_score` (heuristic or ML), produces `risk-decision.json`

• Persists decision and policy version to Artifactory

• **Deployment Orchestrator**:

• K8s: Argo Rollouts / Flagger + Helm manifests
• Windows: IIS slot/pool swap + ARR weight control
• ARM: OTA manager (Mender/Balena/custom) and device cohort manager

• Consumes `risk-decision.json` to select Rolling/Canary/Blue-Green flow

• **Observability Stack**:

• OpenTelemetry client libs in apps
• Prometheus (metrics) + recording rules
• Loki (logs) + Tempo/Jaeger (traces)

• Grafana dashboards for Release Compare + SLOs + Canary progress

• **Analyzer (AIOps)**:

• Runs EWMA / Z-score / STL / Isolation Forest analyses on candidate vs baseline

• Produces `verdict` (promote|slow|rollback) and `evidence` (metric deltas, plots)

• **Rollback & Self-Heal Controller**:

• Executes idempotent playbooks for K8s/Windows/ARM

• Locks execution, records action, triggers incident creation if needed

• **Policy & ML Loop**:

• Feature store (Feast or parquet lake) collects build/test/deploy outcomes
• MLflow tracks models; models predict failure probability or recommend tests
• Policies (YAML) stored in Git with review process

---

3) Data Flow (detailed)

1. Developer pushes commit → Jenkins pipeline triggered.
2. Jenkins builds per-OS, runs tests, collects SBOM & coverage → creates `build-info.json` & `risk-inputs.json`.
3. Jenkins publishes artifacts to Artifactory under `candidates/` and writes metadata.
4. Integration job assembles bundle (if needed) and publishes integration manifest.

5. Decision layer fetches metadata + current policy bundle → computes `risk_score` and `deploy_strategy`.
6. Decision stored in Artifactory and posted to Slack; pipeline branches to selected deployment path.
7. Orchestrator deploys candidate to Green / runs canary weights / executes rolling batches.
8. Observability collects metrics/logs/traces; Analyzer compares candidate vs baseline and emits `verdict`.
9. If verdict==promote → orchestrator increases rollout; if slow → extend watch; if rollback → rollback controller invoked.
10. All decisions & evidence saved; feature store updated for ML training.
11. ML/training jobs periodically run on collected data to update policies/models; policies versioned in Git and rolled out in shadow first.

---

4) Control Flow (decision & safety)

- **Policy enforcement levels**:
- Shadow: decisions logged but not enforced (collect counterfactuals)
- Advisory: decisions posted to Slack/PR but require manual approval
- Enforced: automation acts on decision (deploy path controlled)
- **Manual override**: pipeline supports manual approvals for high-risk changes (DB change, kernel driver, critical CVE).
- **Cooldown**: after an automated rollback, that candidate is quarantined for a configurable cool-down window.

---

5) Per-OS Specific Patterns & Safe Defaults

- **ARM (Firmware)**:
- Always start Green on small cohort of test devices. Prefer canary cohorts; Blue/Green for risky releases.
- Firmware images immutable and signed; previous image retained for instant rollback.
- Extra telemetry for device health (boot time, crash beacon).
- **Linux (K8s / Containers)**:
- Use Argo Rollouts / Flagger for progressive delivery. Argo AnalysisTemplate uses Prometheus queries for SLO checks.
- Use Helm values to parameterize canary weights.
- **Windows (drivers/services)**:
- Use site slots or blue/green server pools; ARR to route traffic for canary.
- Validate driver signing & digital signature checks pre-deploy.

---

6) Security & Compliance

- SBOM created for every artifact and scanned for vulnerabilities; critical CVEs block or force Blue-Green + manual approval.
- Access control: only pipeline tokens (with least privilege) can promote artifacts to production.
- Audit: all decisions (`risk-decision.json`, `verdict.json`, playbooks`) stored with build metadata for compliance.

---

7) Observability & Dashboards (must-haves)

- **Release Compare**: candidate vs baseline (error_ratio, p95, cpu, mem, KPI)
- **Canary Progress**: current weight, windows passed, verdict timeline
- **Rollback Incidents**: list of automated rollbacks with reasons and links to artifacts
- **ML/Policy Health**: shadow vs enforced policy performance (false positives, false negatives)

---

8) Operational Playbooks (short list)

- How to run a manual Blue/Green cutover (k8s helm swap / LB switch / Windows binding swap) and how to rollback.
- How to run the Analyzer in shadow vs enforced mode and tune thresholds.
- How to quarantine a firmware artifact and revert fleet if needed.

---

9) Storage & Retention

- Artifacts: retain candidates for N days (configurable); keep previous production artifacts for rollback.
- Metrics: Prometheus retention policy (e.g., hot store 15d, long-term 90d in remote storage) to support baseline comparisons.
- Logs & Traces: retain per compliance needs.
- ML Data Lake: store parquet datasets for model training (retention per data policy).

---

10) Rollout Roadmap (phased)

1. **Phase 0:** Telemetry hygiene, per-OS environment parity, artifact metadata standardization.
2. **Phase 1:** Implement CI per Step 2, publish candidate artifacts, create decision stage (heuristic, shadow).
3. **Phase 2:** Integrate Orchestrator + Analyzer gates (EWMA/z-score), run shadow for 2–4 weeks.
4. **Phase 3:** Enable enforcement for pilot services; implement auto-rollback playbooks.
5. **Phase 4:** Build feature store + ML models; transition risk scoring to model-backed decisions (shadow → canary → enforced).

---

11) Owners & RACI (summary)

- Platform (owner): CI pipelines, orchestration, rollback controllers
- Observability (owner): instrumentation, Prometheus/Loki/Tempo, dashboards
- SRE (owner): runbooks, on-call, incident triage
- Dev teams (owner): repo changes, tests, SBOMs, supporting telemetry
- Security (owner): SBOM scanning & approval
- Data/ML (owner): feature store and models

---

12) Immediate 30-Day Plan (practical next steps)

1. Standardize `build-info.json` and `risk-inputs.json` and ensure all repos emit it. (week 1)
2. Deploy Prometheus/Loki/Tempo (or confirm access) and instrument 5 pilot services. (week 1–2)
3. Add `Decide Strategy` stage in Jenkins in shadow mode, record `risk-decision.json` to Artifactory. (week 2)
4. Build Release Compare dashboards and configure EWMA alerts. (week 2–3)
5. Run shadow runs and refine thresholds, then enable enforcement for 1–2 pilot services. (week 3–4)

Closing

This architecture balances safety and automation: it gives a clear path from reproducible multi-OS builds to AI-driven deployment decisions, progressive delivery, fast detection and rollback, and a learning loop to continuously improve. If you want, I can now export a printable diagram (SVG/PDF) and a one-page operational runbook for the Release Decision step. Which would you like next?