# Step 3 – Auto‑Rollbacks & Self‑Healing (Architecture + Implementation)

## Objectives

* **Minimize blast radius** by reverting fast when anomalies occur.
* **Self‑heal** common failures automatically (pods, services, configs, nodes).
* Provide **auditable, idempotent** rollback with clear evidence and links.

## High‑Level Architecture

Anomaly Signals (from Step 2)  
 ├─ SLO breaches (err, p95, CPU/mem)  
 ├─ Log/trace anomalies  
 └─ Business KPI dips  
 │  
 ▼  
 [Decision Engine / Remediator]  
 - Policy: promote | slow | rollback | self‑heal  
 - Playbooks: k8s, Windows, ARM, DB, Infra  
 - Safety: impact checks, idempotency, backoff  
 │  
 ┌───────┴────────┐  
 │ │  
 Rollback Controller Self‑Healing Controller  
 (version/traffic revert) (restart, reschedule, fix drift)  
 │ │  
 ▼ ▼  
 CD System (Jenkins/Argo/Flagger) + Infra (Helm, IIS, OTA, Terraform, Ansible)  
 │  
 ▼  
 Evidence store (Artifactory build‑info + logs + Grafana links) & Notifications

## Failure Classes → Actions

* **App regression**: revert traffic/version (canary weight ↓, abort, Helm/IIS/OTA rollback).
* **Infra flake** (node crash, pod OOM): self‑heal (restart/reschedule/scale) then retry.
* **Config drift/secret error**: restore last known good (LKG) config; re‑sync via GitOps.
* **DB migration issue**: pause rollout; run backward‑compatible fallback or toggle feature flag; perform **contract/expand** pattern.

## Policies & Decision Tree (example)

if critical SLO breach ≥ 2 consecutive windows:  
 if rollout stage < 50% traffic: abort canary + rollback version  
 else: immediate traffic switch to stable + freeze further deploys  
elif anomaly score high but SLO OK:  
 slow rollout + extend observation window  
elif infra flake detected (no code signal):  
 attempt self‑healing (N retries, exponential backoff); if persists → rollback

## Kubernetes (Linux/ARM) – Rollback & Self‑Heal

### Rollback (Argo Rollouts / Helm)

# Abort active canary  
argo rollouts abort svc -n prod  
  
# Promote back to stable (sets weight 100% to stable ReplicaSet)  
argo rollouts promote --to-stable svc -n prod  
  
# Or Helm rollback to previous revision  
helm history svc -n prod  
helm rollback svc 1 -n prod --wait

### Self‑Healing Playbooks

* **Pod crashloop/OOM**: kubectl rollout restart deploy svc; increase resources via HPA/VPA if triggered by saturation.
* **Bad config/secret**: restore LKG ConfigMap/Secret (kubectl apply -f cm-lkg.yaml); re‑deploy.
* **Node issue**: cordon+drain node; reschedule workloads; autoscaler to add node.
* **Service mesh/circuit break**: enable outlier detection; trip circuit on failing endpoint to protect users.

# HPA example  
apiVersion: autoscaling/v2  
kind: HorizontalPodAutoscaler  
spec:  
 minReplicas: 3  
 maxReplicas: 15  
 metrics:  
 - type: Resource  
 resource: {name: cpu, target: {type: Utilization, averageUtilization: 70}}

## Windows (IIS/Services) – Rollback & Self‑Heal

### Blue/Green Swap Back (PowerShell)

Import-Module WebAdministration  
# Assume Blue=live, Green=candidate  
# Revert bindings to Blue  
Set-ItemProperty 'IIS:\Sites\MySite' -Name bindings -Value $blueBindings  
Restart-WebAppPool -Name 'MySiteAppPool'

### Canary via ARR – Reduce Weight / Remove Green

# Set ARR to 0% for Green server group  
# (example outline; depends on ARR configuration)

### Self‑Heal

* Restart AppPool, clear ASP.NET temp, re‑attach app‑insights; if repeated 3× in 10m → rollback to Blue.
* Synthetic checks (PowerShell Invoke‑WebRequest) gate promotion.

## ARM/Edge – Cohort Rollback & Health

# Revert cohort to previous firmware  
otactl push --fleet arm-prod --cohort canary --version ${PREV\_TAG}  
# Freeze further cohort expansion until stable for N windows

* Health: device heartbeats, error beacons, OTA success %; auto‑exclude failing devices and retry later.

## Jenkins / CD Integration

stage('AI Decision') {  
 steps {  
 sh 'python3 ci/analyze\_metrics.py --out verdict.json'  
 script {  
 def v = readJSON file: 'verdict.json'  
 if (v.action == 'rollback') {  
 build job: 'rollback-controller', parameters: [string(name:'TARGET', value: env.SERVICE)]  
 currentBuild.description = 'Auto-rollback executed'  
 error('Stopped by auto-rollback')  
 } else if (v.action == 'slow') {  
 sleep time: 180, unit: 'SECONDS'  
 }  
 }  
 }  
}

### Rollback Controller Jobs (per target)

* **K8s:** run argo/helm commands above.
* **Windows:** PowerShell slot swap back / ARR weight 0%.
* **ARM:** OTA revert API.

## Database Safety (Expand/Contract + Flags)

* **Phase 1 (expand):** add new columns/tables nullable; dual‑write via feature flag.
* **Phase 2:** deploy app using new schema (read new, write both).
* **Rollback safe:** old code continues to work (columns still present).
* **Phase 3 (contract):** remove old paths after soak; migration behind flag, reversible until contract.

## Self‑Healing Library (Examples)

* **Restart unhealthy pod/service** with capped retries and jitter.
* **Config drift fix**: reconcile with GitOps desired state.
* **Auto‑scale** if saturation root cause (HPA/VPA, Windows scale set).
* **Network Heal**: recycle load balancer endpoint, rotate node.

Pseudo (Python):

if is\_crashloop(ns, app):  
 restart\_deploy(ns, app)  
 if still\_unhealthy(app): rollback(app)

## Evidence, Audit, and Comms

* Record **who/what/why**: anomaly scores, SLO breaches, commands executed, durations.
* Store JSON + logs + dashboard PNGs in Artifactory tied to build number.
* Notify Slack/Jira with links; auto‑create incident for rollbacks.

## Safety & Idempotency

* All playbooks must be **idempotent** (safe to re‑run).
* Use **locks** to avoid concurrent rollbacks on the same service.
* **Backoff & cap** retries; circuit‑break promotion for 30–60m after rollback.

## Observability of the Remediator

* Expose its own metrics: rollbacks.count, mttr\_seconds, false\_positives, played\_playbooks, retries.
* Dashboard: Rollback Rate, MTTR, Time in Canary, Success after Retry.

## Runbooks & Chaos

* Attach runbooks to alerts (how to override, manual controls).
* Periodic **chaos drills** (pod kill/node kill/latency inject) to validate self‑healing.

## Rollout Plan (Step 3)

1. Implement rollback controller jobs for K8s, Windows, ARM.
2. Encode policies in Decision Engine (thresholds from Step 2).
3. Add DB expand/contract and feature flag integration.
4. Build evidence pipeline → Artifactory + Slack/Jira.
5. Drill with staging chaos tests; then enable in prod with guardrails.

## Deliverables

* rollback-controller scripts/jobs (k8s/helm/argo, Windows PS, OTA CLI)
* Decision Engine service (policies + idempotent playbooks)
* GitOps LKG config bundles
* Evidence collectors + notification hooks
* Dashboards for rollback/self‑healing KPIs