ARTICLE · CI/CD · JENKINS · DOCKER · KUBERNETES

Jenkins, Docker and Kubernetes: where code-churn measurement fits in CI/CD

The trio moves your code from commit to cluster. None of the three can tell you what actually changed on the way through — how much of the diff a developer wrote, how much a lockfile or an AI agent produced, and whether that should block the merge. That measurement slots into the pipeline as one container image.

2 August 2026 · recipes for the CodeDelta CLI

The trio, in one breath

If you landed here from a search, the division of labour is simpler than most explainers make it: **Jenkins** is the orchestrator — it notices a commit and runs your pipeline, stage by stage. **Docker** is the packaging — it freezes an application and its dependencies into an image that runs identically everywhere. **Kubernetes** is the runtime — it keeps fleets of those containers running, restarted and scaled. Jenkins builds it, Docker boxes it, Kubernetes runs it.

What none of them measures

All three move code; none of them looks at it. A pipeline can build, containerise and deploy a 5,000-statement change without anyone learning that 4,900 of those statements came from a regenerated lockfile, or that the change quietly added a call to a foreign-hosted AI model, or that an agent's credentials got committed alongside it. CI status is green; the questions that matter to an engineering manager — *what changed, who or what wrote it, does it carry risk* — are simply never asked.

CodeDelta asks them deterministically: a compiled C++ engine diffs two versions of a tree and counts changed, deleted and added *logical statements* (not lines), separates authored churn from generated churn (TRUE_CHURN), and scans for AI-agent code, egress to foreign-hosted models and committed agent credentials. Same input, same numbers, every run — no sampling, and the churn arithmetic involves no ML.

Jenkins: a churn gate as a pipeline stage

The engine ships as a public Docker image, so a Jenkins stage needs no tool installation — just Docker on the agent. This reference stage compares the workspace against the main branch and fails the build (the engine exits 3) when gated findings appear:

```
# Jenkinsfile (declarative) - reference recipe
pipeline {
  agent any
  environment {
    // base64 of your codedelta.lic, stored as a Jenkins secret-text credential
    CODEDELTA_LICENSE_B64 = credentials('codedelta-license-b64')
  }
  stages {
    stage('Churn + agent gate') {
      steps {
        sh '''
          git worktree add --force /tmp/base origin/main
          docker run --rm \
            -v "$WORKSPACE:/work" -v /tmp/base:/base \
            -e CODEDELTA_LICENSE_B64 \
            ghcr.io/code-delta-app/codedelta \
            scan /work /base --mode churn_agent --html --csv --gate
        '''
      }
    }
  }
}
```

Exit code 3 means a policy violation — by default, egress to sanctioned-country model providers or executable-on-model artifacts — and Jenkins fails the stage. Add `--write-baseline` once and then `--baseline <file> --fail-on-new` to block only *new* findings, which is how you introduce a gate to an existing codebase without relitigating history. The reports (churn, AI audit, agent scan) land in the workspace as HTML and CSV for archiving.

Docker: the no-install measurement

This is the live path most corporate users take — hardened CI that forbids installing tools but allows approved images. The image is public on GHCR and republished with every release; the licence rides in as an environment variable:

```
export CODEDELTA_LICENSE_B64=$(base64 -w0 codedelta.lic)

docker run --rm -v "$PWD:/work" -e CODEDELTA_LICENSE_B64 \
  ghcr.io/code-delta-app/codedelta \
  scan /work/new /work/old --mode both --html --csv
```

The container holds the compiled engine and everything it needs — no Python, no dependency resolution, no network calls during the scan. Everything executes against the trees you mount; nothing leaves the machine.

Kubernetes: scheduled scans as a CronJob

CodeDelta is a build-time measurement, not a service, so it does not *deploy* to Kubernetes — it *runs on* it, the same way your K8s-hosted CI agents do. Two patterns are useful. First, if your Jenkins agents (or Tekton/Argo runners) are pods, the Docker recipe above works unchanged inside them. Second, a scheduled snapshot scan of a repository — agent detection and AI audit on a cadence, reports to a volume:

```
# reference recipe — weekly snapshot scan
apiVersion: batch/v1
kind: CronJob
metadata:
  name: codedelta-weekly-scan
spec:
  schedule: "0 6 * * 1"
  jobTemplate:
    spec:
      template:
        spec:
          restartPolicy: Never
          containers:
          - name: codedelta
            image: ghcr.io/code-delta-app/codedelta
            args: ["scan", "/repo", "--mode", "ai_audit", "--html", "--csv"]
            env:
            - name: CODEDELTA_LICENSE_B64
```

```
            valueFrom:
              secretKeyRef: { name: codedelta-licence, key: license-b64 }
          volumeMounts:
          - { name: repo, mountPath: /repo }
        volumes:
        - name: repo
          persistentVolumeClaim: { claimName: repo-checkout }
```

Run weekly, this gives you a longitudinal record of AI involvement and agent artifacts in the tree — the trend that a single PR-sized snapshot cannot show.

What the output looks like

This is real output, not a mock-up — the comment CodeDelta's own gate posted on the live public demo pull request on 1 August 2026, running the released engine against a PR whose diff was six authored statements plus a regenerated lockfile:

```
CodeDelta report

Churn: 96 logical statements (+6 / −0 / Δ90); REP_CHURN 0.06
TRUE_CHURN: 6 authored — the other 90 (94% of the total)
            came from 1 generated file

AI audit:   0 HIGH · 1 ELEVATED · AI 6% of added code
Agent scan: 0 CRITICAL · 3 HIGH · 8 ELEVATED
Governance: 4 file(s) send data to foreign-hosted models
            (4 non-allied); 1 file(s) call AI inside a loop
Agent infrastructure: 6 artifact(s) in the tree — 2 tier-3
            (rogue residue / credentials)
```

Ninety-four percent of that diff was machine-produced. A reviewer reading only the raw diff size would have concluded a substantial change; the measurement says six statements of human work. That distinction — on every PR, deterministically — is the point.

Try it

The CLI page covers every run path (GitHub Action, Docker, Codespaces, local, other CI) with copy-paste commands. Downloads for Windows, macOS and Linux are on the download page — free, fully unlocked, until 31 August 2026.