Friday evening, 6:47pm. A developer at one of our client companies in Westlands ran a deployment script. The wrong one. Instead of deploying the staging branch to the test server, they pushed an untested feature branch straight to production. The M-Pesa payment integration broke. Customers could not check out. By the time someone noticed, the company had lost about three hours of peak evening traffic.

The fix took eleven minutes. The revenue loss took a lot longer to calculate.

CloudSpinx is a DevOps and infrastructure consultancy based in Nairobi, Kenya. We build CI/CD pipelines and manage deployment infrastructure for over 40 businesses across East Africa. That Friday incident is not unusual. We hear some version of this story almost every month.

This post is for every Kenyan tech team that still deploys code by SSHing into a server and running commands manually. There is a better way, and it is not as complicated or expensive as you think.

The SSH Deploy: How Most Kenyan Teams Actually Ship Code

Let us be honest about what we see when we audit Kenyan tech companies. The typical deployment process looks something like this:

1. Developer finishes a feature on their laptop
2. They SSH into the production server
3. Run `git pull origin main`
4. Maybe run `npm install` or `pip install -r requirements.txt`
5. Restart the application with `pm2 restart all` or `sudo systemctl restart myapp`
6. Hope nothing broke

Sometimes there is a deploy script. Sometimes there is a shared Google Doc with "deployment steps." Sometimes the knowledge lives entirely in one developer's head, and when that person is on leave, nobody deploys anything.

We have seen all of it. The fintech in Kilimani where three developers had root SSH access to production. The e-commerce company on Mombasa Road where deployments happene via FTP. The SaaS startup where the CTO was the only person who knew how to deploy, and they were doing it from their phone over Safaricom 4G while stuck in traffic on Thika Road.

This works. Until it does not.

What CI/CD Actually Means (Without the Jargon)

CI/CD stands for Continuous Integration and Continuous Deployment. In plain language:

Continuous Integration means every time a developer pushes code, an automated system runs your tests. If the tests fail, the team knows immediately. No waiting until Friday to discover that Monday's code change broke something.

Continuous Deployment (or Continuous Delivery, depending on who you ask) means that once code passes all tests, it gets deployed automatically. No human SSHing into anything. No running scripts manually. The pipeline does it.

Think of it like a factory assembly line. Raw code goes in one end. Tests, security checks, and quality gates happen along the way. A working deployment comes out the other end. If any step fails, the line stops and everyone gets notified.

A Real GitHub Actions Pipeline (Copy This)

GitHub Actions ↗ is what we recommend for most Kenyan teams. It is free for public repositories and gives you 2,000 minutes per month on the free tier for private repos. That is enough for most small to mid-size teams.

Here is a real pipeline we set up for a Node.js application:

```yaml
name: Deploy to Production

on:
  push:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '20'
          cache: 'npm'

      - name: Install dependencies
        run: npm ci

      - name: Run linter
        run: npm run lint

      - name: Run tests
```

```
        run: npm test

      - name: Run security audit
        run: npm audit --production

  deploy:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    steps:
      - uses: actions/checkout@v4

      - name: Deploy to server
        uses: appleboy/ssh-action@v1
        with:
          host: ${{ secrets.SERVER_HOST }}
          username: ${{ secrets.SERVER_USER }}
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          script: |
            cd /var/www/myapp
            git pull origin main
            npm ci --production
            pm2 reload ecosystem.config.js
```

What is happening here:

- Every push to `main` triggers the pipeline
- Tests run first. If they fail, deployment never happens
- Only after all tests pass does the deploy step execute
- SSH credentials are stored as encrypted secrets, not in someone's `.bashrc`
- The entire process takes 2-4 minutes with zero human intervention

That Friday disaster we mentioned? It cannot happen with this setup. The wrong branch cannot reach production because only `main` triggers deployments. And code only reache `main` through pull requests that pass automated tests.

GitLab CI Alternative

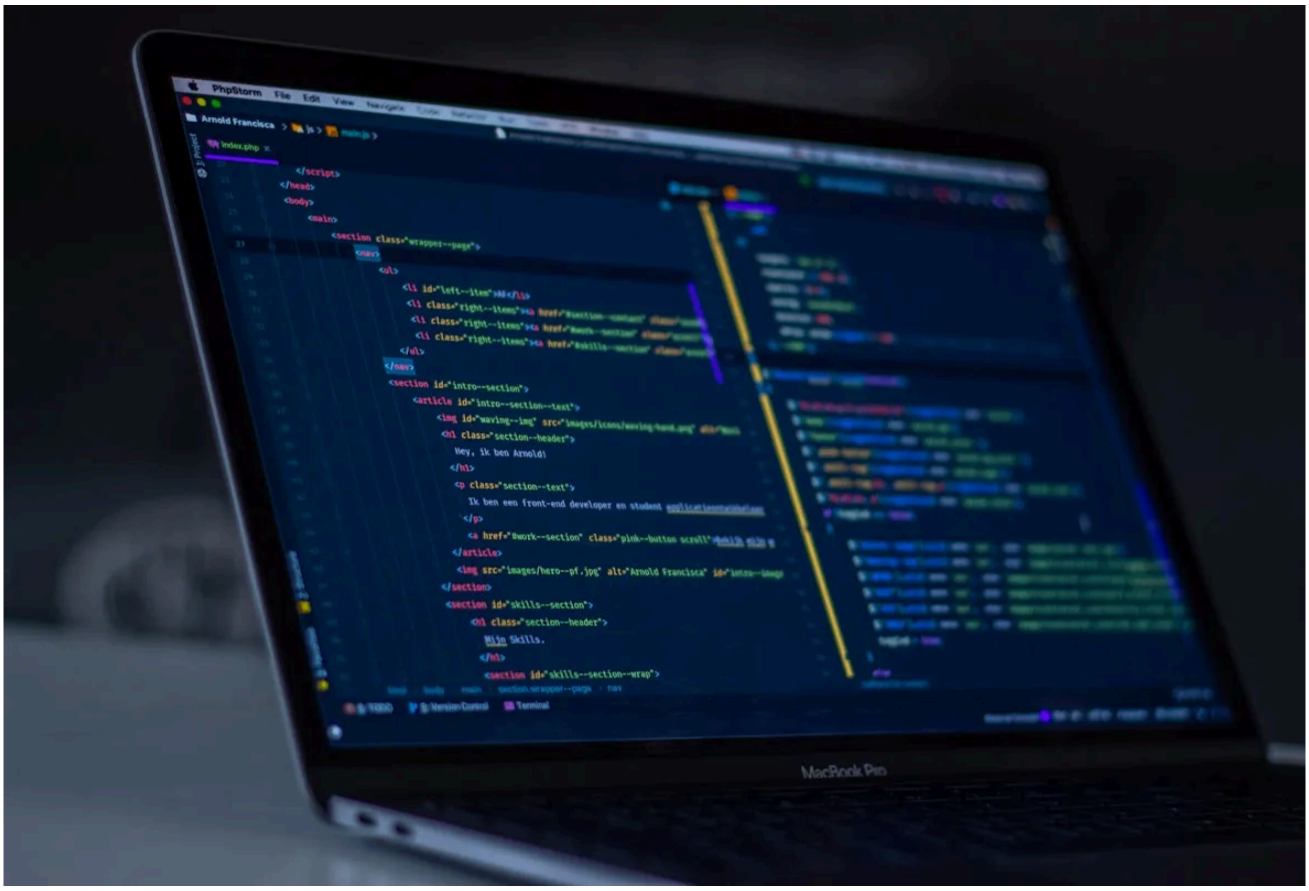
If your team uses GitLab ↗, the equivalent looks like this:

```yaml
stages:
  - test
  - deploy

test:
  stage: test
  image: node:20
  cache:
    paths:
      - node_modules/
  script:
    - npm ci
    - npm run lint
    - npm test

deploy_production:
  stage: deploy
  only:
    - main
  script:
    - apt-get update && apt-get install -y rsync
    - rsync -avz --delete ./dist/ $SERVER_USER@$SERVER_HOST:/var/www/myapp/
    - ssh $SERVER_USER@$SERVER_HOST "cd /var/www/myapp && pm2 reload all"
  environment:
    name: production
```

GitLab gives you 400 CI/CD minutes per month on the free tier. For most Kenyan teams pushing 5-10 times a day, that is plenty.



What This Actually Costs in KES

Here is the part everyone asks about. Let us break it down honestly.

GitHub Actions (recommended for most teams):

- Free tier: 2,000 minutes/month - KES 0
- Team plan: KES 520/user/month (USD 4), 3,000 minutes included
- A typical Node.js pipeline uses 3-5 minutes per run
- At 20 deployments per week, that is about 400 minutes/month. Well within the free tier.

GitLab CI:

- Free tier: 400 minutes/month - KES 0
- Premium: KES 3,900/user/month (USD 29), 10,000 minutes
- Self-hosted runners: Free minutes, but you pay for the server (a t3.small on AWS runs about KES 3,800/month)

Self-hosted runners on your own server:

- If you already have a cloud server, you can run your own CI/CD runner for KES 0 extra

- We often set this up for clients who are cost-conscious or have specific compliance requirements

Our setup cost:

- The cost depends on your application stack and deployment complexity. Reach out to our DevOps team for a quote tailored to your setup.
- After setup, the pipeline runs itself. You pay nothing to us for ongoing pipeline executior

Compare that to the cost of one production outage. The Friday incident we described cost our client roughly KES 280,000 in lost transactions. The CI/CD pipeline we built them the following week paid for itself before the end of the month.

"We Are Too Small for CI/CD"

We hear this constantly. Here is our take.

Even a 2-person team benefits from basic CI/CD. The setup is simpler for smaller teams, but the peace of mind is the same. A lightweight pipeline that runs your tests on every pusl and deploys automatically takes about an hour to set up and saves you from the "wrong branch to production" disaster regardless of team size.

That said, the urgency scales with your situation. If any of these are true, CI/CD should be a priority this week:

- More than two developers pushing code
- Deploying more than twice a week
- Running any kind of payment integration (M-Pesa, card payments, bank APIs)
- Customers depend on your uptime during business hours
- You have staging and production environments
- Your deployment process involves more than three steps

We work with teams of all sizes. One of our clients, a 4-person team building a logistics platform, was spending 6 hours per week on deployment-related tasks. That is almost a ful working day. Their senior developer was spending an entire working day each week babysitting deploys instead of writing features. We set them up with a lightweight GitHub Actions pipeline, and they got that time back immediately. Talk to us about what makes sense for your stage.

"CI/CD Is Too Complex"

It can be. Jenkins ↗ is famously painful to set up and maintain. Kubernetes-based deployment with ArgoCD ↗ is powerful but requires serious expertise.

But GitHub Actions for a standard web application? That YAML file above is the whole thing If your developers can write JavaScript or Python, they can understand a 30-line YAML file The learning curve is real but shallow.

We usually get teams up and running in two days. Day one: set up the pipeline, run through it together, explain what each step does. Day two: the team runs a few deployments on the own while we watch and answer questions. By day three, they are independent.

When CI/CD Is Genuinely Overkill

We would be dishonest if we did not mention this.

CI/CD does not make sense for:

- Static websites that change once a month (just FTP it, honestly)
- Solo developer projects with no users yet
- Internal tools used by three people where 5 minutes of downtime does not matter
- Prototypes and MVPs in the first two weeks of development

Start simple. Add automation when the pain of manual deployment exceeds the effort of setting up a pipeline. For most growing teams, that crossover point comes sooner than they expect.

The Deployment Checklist We Use at CloudSpinx

Every pipeline we build includes these checks. Non-negotiable:

1. **Code linting** - catches syntax errors and style violations before they reach production
2. **Unit tests** - your core business logic should be tested
3. **Security audit** - `npm audit`, `pip-audit`, or equivalent. Catches known vulnerabilities in dependencies
4. **Build step** - compile, bundle, whatever your framework needs. If the build fails, nothing deploys

5. **Smoke test after deploy** - hit the health endpoint. If it returns anything other than 200, roll back automatically
6. **Notification** - Slack message, email, or SMS confirming successful deployment (or alerting on failure)
7. **Rollback plan** - every pipeline should be able to deploy the previous version in under 2 minutes

Items 1-4 happen before deployment. Items 5-7 happen after. If any step fails, the pipeline stops and the team gets notified.

We also recommend adding these once your team is comfortable:

- Database migration checks (does the migration run cleanly against a copy of prod data?
- Integration tests against staging
- Performance benchmarks (did this deploy make the API slower?)
- Container image scanning if you are using Docker through our hosting infrastructure

Getting Started This Week

You do not need to automate everything at once. Here is what we recommend for teams that are currently deploying manually:

Week 1: Move your deployment steps into a single shell script. Document it. Put it in your G repo. This alone eliminates the "only one person knows how to deploy" problem.

Week 2: Add a basic GitHub Actions workflow that runs your tests on every push. Do not automate deployment yet. Just get automated testing working.

Week 3: Add the deployment step to your pipeline. Deploy to staging first. Let it run for a week.

Week 4: Once you trust the staging pipeline, point it at production.

If you want help setting this up, our DevOps team does this every week. We can also handl the full infrastructure side, server management, monitoring, and security hardening, so you developers focus purely on code.

The era of SSH-and-pray deployments needs to end. Your customers deserve better. Your developers deserve better. And your Friday evenings definitely deserve better.

Frequently Asked Questions

How long does it take to set up CI/CD for an existing project?

For a standard web application with existing tests, we typically get a working pipeline running in 1-2 days. If you have no tests at all, add another week to write the critical ones first. Complex microservice architectures can take 2-3 weeks.

Can I use CI/CD with shared hosting?

Technically yes, but it is painful. Most shared hosting providers do not give you SSH acces or support modern deployment methods. If you are serious about CI/CD, you need at least VPS. A basic cloud instance on AWS or DigitalOcean costs KES 1,300-2,600 per month.

Do I need Docker for CI/CD?

No. Docker is useful for consistency between environments, but plenty of teams run CI/CD without it. If your application runs directly on Ubuntu with Node.js or Python, your pipeline can deploy directly without containers. Add Docker when you have a specific reason, not because someone told you to.

What happens if the pipeline fails mid-deployment?

A well-built pipeline uses atomic deployments. The new version is built and prepared in a separate directory. Only after everything is ready does the system switch traffic to the new version. If anything fails, the old version is still running and serving traffic. Zero downtime.

Is GitHub Actions secure enough for production deployments?

Yes. SSH keys and credentials are stored as encrypted secrets that never appear in logs. The connection between GitHub and your server uses the same SSH protocol you would use manually. We still recommend restricting which branches can trigger production deployments and requiring pull request reviews.

Should we use Jenkins or GitHub Actions?

GitHub Actions for almost everyone. Jenkins is more powerful and flexible, but it requires it own server, regular maintenance, plugin updates, and a dedicated person who understand it. Unless you have specific enterprise requirements that only Jenkins can meet, start with GitHub Actions.

Can CI/CD work with mobile app deployments?

Yes. We build pipelines that compile Android APKs and iOS builds, run tests, and push to th Play Store and App Store. The setup is more involved than web deployments because of code signing and store review processes, but the principle is the same.

What if our tests are slow?

Parallelise them. GitHub Actions lets you run test suites across multiple workers simultaneously. A test suite that takes 20 minutes on a single machine can often finish in 5-7 minutes when split across 4 parallel workers. Also, look at which tests are actually slow. We regularly find that 80% of test time comes from 3-4 tests that are doing unnecessary database setup.

Read next

All articles →

EMAIL 9 min

Why Your Business Email Still Runs on Free Gmail (and Why That Is Costing You)

Most Kenyan SMEs send invoices and tender responses from personal Gmail...

Amina Hassan 23 March 2026

LINUX 14 min

Setting Up a Production Linux Server for Your Kenyan Business: What We Check Every Time

Our 15-point production Linux server checklist, from distro selection to...

Josphat Mutai 23 March 2026

NETWORKING 11 min

Office Network Disasters We Have Fixed in Nairobi (and How to Avoid Them)

Seven real network disasters from Nairobi offices - switching loops, rogue DHCP...

Peter Ochieng 23 March 2026