# How We Built a Seamless Salesforce Deployment Pipeline with Gearset, Git & Jira

## Part 1: The Human Side of DevOps – Our Story

As our Salesforce team and org began to grow, so did the complexity of our deployments. Initially, we were managing releases the old-fashioned way—manually creating change sets or directly using Gearset's compare and deploy feature between orgs, coordinating over Teams, tracking Jira ticket status separately, and double (sometimes triple) checking environments before pushing anything live.

It worked… until it didn't.

As we scaled, our release process started slowing us down. We lacked consistency, visibility, and automation. That's when we began our journey into CI/CD using Gearset, Git, and Jira.

## Why We Made the Shift

Here's what we were struggling with:

- Developers accidentally overwriting each other's changes
- No centralized source of truth — it was hard to track exactly what was deployed
- Errors in UAT and production due to missed dependencies
- A complete disconnect between Jira tickets and actual deployments
- Painful (or impossible) rollbacks
- And honestly… deploy days were stressful

### So we asked ourselves: What would a dream deployment process look like?

It looked like this:

- Every change tied to a Jira ticket
- Developers working on isolated branches
- Stakeholders knowing exactly what was going live, and when

So we rolled up our sleeves.

## Part 2: Solution Architecture Overview

Our solution is built around three key tools:

- **Gearset**: To automate comparisons, create pull requests, validate changes, and manage deployments
- **Git (GitHub)**: Our central version control and collaboration platform
- **Jira**: To manage and track development work, tie deployments to business requirements, and provide traceability

## How We Built It — Step by Step

### Step 1: Introduced Git as Our Source of Truth

We standardized our branching strategy:

- `feature/SF-####-description` for all new work
- `Dev` for sanity testing
- `UAT` for regression testing
- `main` for production-ready code

We made sure every commit referenced a Jira ticket. Pull requests were required, and we kept our review cycles tight.

## Step 2: Set Up Gearset for CI/CD

Gearset was a game-changer for us:

- We connected it to our Git repo and Salesforce orgs
- Set up an automated job that deployed to Dev when a PR was merged into the Dev branch, which then automatically created a PR to UAT.
- Set up a second job for UAT, triggered by merges into `UAT`, which then automatically created a PR to main.
- Set up a third job for production, triggered by merges into `main`

Gearset's change monitoring and validation tools helped catch common issues early — like missing field-level security or permission sets.

## Step 3: Connected Jira with Git & Gearset

Using Jira automation, we linked ticket status with Git commits and Gearset deployment jobs.

We also used release dashboards in Jira so business users could see what was going live — without needing to ask us.

## Step 4: Added Rollback and Monitoring

Gearset stores every deployment and backup. If something slips through, we can roll back in minutes — no more digging through old change sets or frantically rebuilding lost metadata.

# Our CI/CD Pipeline

## 1. Development in Personal Sandboxes

Each developer works in their own Salesforce sandbox environment. These isolated environments allow for independent development without interfering with each other's work.

## 2. Pull Requests to the Dev Branch

Once development is complete and tested in the personal sandbox, a pull request (PR) is created from the feature branch to the shared `Dev` branch. This initiates automatic validation via Gearset. The Deployment team then merges the PR into `Dev`.

## 3. Automatic PR Creation from Dev to UAT

Upon successful merge to the `Dev` branch, Gearset automatically creates a PR from `Dev` to `UAT`. However, the actual deployment to UAT is intentionally kept manual.

## 4. Controlled UAT Deployment

Deployments to UAT are handled manually by the Deployment team, and only when:

- The Developer confirms the feature is ready for testing
- The assigned Project Manager reviews and approves the work in Dev

This ensures only validated and business-approved features move forward.

## 5. UAT Testing and Regression

Once deployed, QA and business stakeholders perform functional and regression testing. Any issues are tracked in Jira and fixes follow the same Dev → UAT cycle.

## 6. Controlled Releases to Production

After UAT sign-off:

- Each developer creates their own release branch for the Jira tickets they're responsible for.
- Two code reviewers are required to approve production PRs. The Deployment team merges those release branches into `main`, triggering Gearset's deployment job for Production

## Results (That We’re Proud Of)

- Deployment time reduced by 80%
- Zero post-deployment surprises
- Improved collaboration between dev, QA, and product teams
- Rollback safety net that gives us confidence to push changes faster
- No more accidental code overwrites thanks to Git-based version control

But the biggest win? Our release process no longer feels like a battle. It feels like flow.

## Final Thoughts

Implementing CI/CD wasn’t an overnight switch. There were bumps. There were “just one more test run” late nights. But the outcome was worth it.

If you're considering bringing automation and version control into your Salesforce workflow:

- Start small
- Get team buy-in
- Show results early

For us, Gearset + Git + Jira didn’t just transform our deployment process — it made us a stronger, more collaborative team.