Heartland Coop’s Transition to DevOps: Embracing Git and Automation on IBM i

HEARTLAND CO-OP

Interview with Todd Stewart, DevOps-certified Senior Application Engineer at Heartland Coop, and active member of the COMMON Americas Advisory Council.

In this frank exchange, Todd reveals the challenges faced and lessons learned in leading a modernization initiative at Heartland. His mission: to bring 30 years of IBM i development into a DevOps approach, facilitate innovation and maximize the value of their IBM i investment.

What was your goal behind DevOps on IBM i?

Todd: We were looking to achieve consistency in application lifecycle management through automation. Essentially remove “hold-ups” in the cycle for a rapid ROI.
We needed more flexibility in the change process - to deliver higher quality software more quickly, and stable/reliable/secure services to the business.

Our code base was decades old, developed in the ‘80s and ‘90s. We needed to “start somewhere”. There was no real “big bang” option. We had to take our waterfall method and automate.

DevOps drivers: “2 speed IT” – major projects + small fixes in parallel

Todd: The real driver behind DevOps for us was “2 speed IT”. We were running big projects and small fixes in parallel. Before we implemented Git, these two lines of development caused code conflicts and delays. Small fixes slowed the big project, and we were sometimes unable to deliver small fixes due to conflicts with the big project.

We had been working on one particular project for many years, stopping and starting as business priorities came in. This project involved changing the size of an attribute in the location table, 2 digits long. The change was massive, impacting 90% of the ERP. To test the change, we pushed to test, which put the rest of development to a screeching halt. Priority changes couldn’t be tested against these changed database files!

We realized that DevOps would be a long-term incremental change for us. We needed to tool ourselves to support modern methodologies.

Moving to ARCAD with Git right away solved this problem. Git meant we could branch off the massive project and clear the highway for day-to-day work.

With Git, we could regularly merge those small priority production changes into the large project, then deploy the massive project into the main branch (that’s 2,900 code modifications and 3,700 components deployed in total!).

How easy/difficult was it to adopt Git?

Todd: Prior to the ARCAD project, I had *zero* experience with Git, though some of the team had used Git on other platforms. **Even though I’m a rookie, Git works great! We can now monitor code changes & tracking at the line level.** The ARCAD project has been a huge benefit for our IT department. Even our more traditional IBM i developers are reassured that not much interaction with Git is needed.

With ARCAD, you can track objects without source (like data areas) with Git.

I’ve learned that you need to keep the repository clean, avoid leaving obsolete branches out there. When you deploy, merge immediately into master branch, and clean.

With Git, you end up with less “rogue” objects from developers (where they copy the source to modify and compile outside the system). Working rogue was a major frustration for them, when they realized that their changes get forgotten and aren't included in the release! **Git solves all that and brings developers into the fold**.

How much "cleanup" is needed before moving to DevOps?

Todd: The **ARCAD audit is an automatic process that helped us cleanup our application**. We uncovered a huge number of programs and files without source. This was the result of source migrations over the years (such as, the creation of SQL views & indexes directly on the production LPAR which makes its way over to the dev LPAR via data refresh). We reduced our technical debt by deleting obsolete objects unused for many years.

The **cross-references in ARCAD Observer track everything, everywhere, which is awesome**. ARCAD tracks relationships throughout the test environments. When you make a file change, it recompiles all the dependent views.

Moving to DevOps means taking the most tedious tasks and automating them, to make life easier for the developers.

CI/CD and Test Automation

Todd: Our ARCAD implementation uses webhooks in GitHub and Jenkins pipelines so that when we merge a feature branch into a release branch, this **triggers a process in Jenkins, builds a version, and imports the deployment instance** into DROPS for deployment to the QA environments, all automatically.

We plan to kick-off automated testing in the QA changes when the changes arrive, using ARCAD Verifier. For this, we first need to define the use cases & record the tests. Once that's done, the developers will commit their changes at the end of the day, then come back the next morning with a whole set of test results. Time to deployment will get shorter & shorter!

Your advice about Git & DevOps?

Todd: My advice is "just do it". Even if you aren't quite ready for DevOps, the benefits of Git and automation go far beyond just DevOps. For example, with Git, **you can test a large project without interfering with day-to-day development**. You can have multiple developers working on the same component at the same time. You can have 2 developers merge their changes into a single release, and each can move their work up through the test environments concurrently and have them tested together. That's a huge productivity gain right there.

I advise IBM i shops to get their source into Git. You can use Source Orbit from IBM to easily migrate source from a library into Git. This gives you **great visibility, you can manage different versions far easier, and ARCAD for DevOps gives a highly automated CI/CT/CD cycle**.

Future of DevOps on IBM i?

Todd: It takes time to reach a true DevOps methodology on IBM i. For full modern DevOps on a legacy code base, you would ideally modernize much of the code base, shrink the size of individual functions so that smaller increments can be deployed faster. Here, the new AI tools will have a major impact in transforming to modular code.

When you refactor monolithic code into smaller chunks, you can manage parallel changes in separate Git branches which makes maintenance way more efficient. At Heartland we are modernizing on case-by-case basis, incrementally. Smaller IBM i shops can make use of Watson x and CodeAssist to help modernize their code base.

We are looking to use ARCAD's Transformer Microservices to identify repeat code and replace this by modular procedures.

For successful DevOps, you need the right tools in place before you start. I confirm that **the ARCAD tool chain is a major help in maintaining your IBM i applications, regardless of how they're architected!**