Atlassian

DevOps Case Study:

JAMF Software's DevOps Journey

About the Author

Michael Kren

Manager, Development Operations,
JAMF Software

From growing up on a Wisconsin dairy farm to Manager of Development Operations at JAMF Software, with a little volleyball in between (okay, a lot of volleyball), Michael Kren doesn't mind working hard, but he'd rather be working smart. 'Automate yourself out of a job' has been his mantra since becoming a System Center Configuration Management administrator in the Education setting. Michael was able to save countless hours managing classrooms and labs using SCCM for configuration management. Moving on to the software development space has brought even more opportunities. He has been able to save time and speed up deployments using the Atlassian tool set. Michael is always looking for new ways to achieve greater efficiency and transparency. And automate himself out of a job.

Contents

Why we have DevOps at JAMF Software

A quick trip back to 2012

It was 1 a.m. and we weren't having fun anymore. We'd worked around the clock all week to get a release out the door, and we were almost there—but it had been extraordinarily difficult. The whole team was coding as fast as we could, but our code was stepping all over each other. Still, we couldn't slow down, or QA wouldn't have anything to test.

We quit for the night. The next morning, with just a few hours of sleep, we rallied and pushed the release out the door by the end of the day. Our entire team breathed a collective sigh of relief together, a combination of excitement and extreme fatigue. We'd done the impossible—and we decided at that moment that we never wanted to do it that way again.

And that's how we started our journey to DevOps at JAMF. We knew we weren't working optimally, in a sustainable fashion, and we were committed to making a change before our customers suffered. It's been the best decision we've made as a team—and I'm excited to share a bit about how we got started, what we've learned, and where we're headed.

One server to rule them all

But first a quick bit of background: JAMF Software is all about device management and helping IT do more with less. One of our products, Casper Suite, allows IT

personnel to manage thousands of Apple devices from a single web server—so we're big on efficiency and scalability.

When I was first hired in 2012, DevOps wasn't even a thing yet, or at least not at JAMF. I came onboard to assist the engineering team with infrastructure deployment and maintenance, and we had just one build server, a handful of developers, and a couple QA folks.

For the most part, it worked just fine: Developers coded all day, and committed to a single "trunk" branch. At night, the build server would build all our products and dependencies (over a dozen separate builds) from the single branch. The next morning, QA would grab the artifacts and start testing.

See what's wrong with this picture? In hindsight, of course you do—but before you start bashing, look at the positives (with your 2012 glasses on):

- It was simple. All code was built at the same time, together, with only one server to maintain.
- You could even call it pure continuous integration, on a daily build, since everyone was committing to the same branch and all testing was done against that one build.

But the negatives still outweighed the good, and the biggest negative was the destructive feedback loop between QA and developers. The dev team would write code all day while QA was testing yesterday's work. As QA uncovered issues, the developer had to switch gears and remember the logic behind changes made an entire day and several issues ago.

The result? Hurry up and wait. We were painfully slow, and that's bad when customers are your top priority. Even when we iterated quickly to solve a customer issue, we still had to wait for our build, which was "all or nothing"—whether you changed a feature or component or not, every single piece of every product still had to be built, every single time. When the team comes together to help a customer in trouble, but the build takes an hour or more—that's a long hour to wait, particularly if it takes a couple builds to get it right.

It was clear to us that it was time to change. Our customers deserved better, and "customer" is the magic word at JAMF. We're always asking, "How does this help the customer?" We even send our engineering team members on customer site visits, for professional development—and in the process, it shows the customer the "face behind the code" and strengthens our relationships.

Chapter 1

People shock: Building a DevOps team and culture

We started with our culture

I promise we'll get to process and tools**,** since I know that's the fun part. But no amount of tools can make up for a deficiency in how you work together.

At JAMF, we've always had a GSD (get stuff done) attitude, but at a certain point, everyone couldn't do everything anymore. We realized we needed to spend some time making sure we had the right people and the right culture to be successful, and then add the right tools and processes to that strong foundation.

As a "build guy," I'm always looking for ways to automate the device management process. But at a certain point at JAMF, I started to worry that I might be automating myself into the unemployment line. At some point, I mentioned it to my boss, and he told me what has now become my biggest mantra, and my first tip for implementing DevOps successfully.

1.) Automate yourself out of a job. We'll give you another one.

If your culture is one where the other teams are simply trying to hold on to their jobs, exactly as they are, and won't let go, it's time for change. A good culture will embrace creative and smart approaches to automation and efficiency, and reward the teams and individuals who uncover them with new roles and responsibilities.

Before you automate every process and trigger the robot takeover, though, take time to truly understand every process. Automating any process incorrectly will only decrease the trust you are trying to build. A good way to earn the trust you need to make changes people believe in is to spend time every day with the teams that follow these processes. Help them solve their problems, and find ways they can do more with less.

At JAMF, one of our early automation projects was verifying hyperlinks in our documentation. Before, we had to verify every link manually, and it was boring, arduous, and inefficient—and the team hated the work. By hearing these pains, my team wrote a script that automatically verifies that every link returns a 200 code, and flags the ones that don't.

The team member who previously spent hours verifying those links isn't unemployed now. They're doing more important, fulfilling work, and they're thrilled to be doing it. So spend some time each day talking to your engineering department. Ask them, "How are things going?" and "Are there any pain points I can help you with?"

Talk to other teams like Tech Comm, too. At JAMF, the Tech Comm team bridges the gap between the user and the software by making sure users have the information they need to use our products effectively. They've been the target for some of my team's biggest projects to date, so I enjoy hearing their feedback about how things can get better.

If you have the right culture, these teams will be thrilled for your help. If you don't, they won't—and you'll know it's time to enact change before you dive head first into DevOps tools and processes.

2.) Hire people who do your job better than you

We'll talk shortly about how we moved beyond a single build server, and what we use today. For now, let's just say that more efficiency meant more new projects getting spun up, and it became pretty clear that I needed help. Business continuity was becoming a concern as well, since I was the only person who had intimate knowledge of our evolving build infrastructure. So I requested an addition to my humble team of one.

The criteria I was looking for, and that I have found to be most effective, are as follows:

- *Can do my own job better than I can.* I know I'm not the smartest person out there. I wanted to bring new knowledge to the company, and learn.
- *Motivated laziness.* (Bill Gates once famously said, "I choose a lazy person to do a hard job because a lazy person will find an easy way to do it.")
- *Willing to spend the time* to get work done right.
- *Not afraid to automate themselves out of a job*. It's one of our JAMF DevOps mantras, after all.
- *Someone who brings change, and thrives in it*. Change is happening all around, so hiring someone with a fixed mindset is a nonstarter.

This last point, about change, is one of the most important. Our team doesn't write the code, test the code, run the code, or even write the documentation for the product. The Engineering and Online Services teams are our customers. But in DevOps, we can't take a "Customer is Always Right" approach, since sometimes even painful change is needed.

Hire people who avoid change just for the sake of change, but aren't afraid to push for improvements when there is substantial ROI involved. Take our release plan, for example. It worked well for a while, but as the frequency and types of releases changed, my team had to adapt and create a more sustainable process. Much like software development, nothing is truly finished—it just gets moved lower on the priority list.

3.) Structure your team for success

The structure of a DevOps team should be very fluid, since roles will change as you continue to automate yourselves out of your current roles and take on new responsibilities. Team structure also depends on the products you build, and the customers you serve—so a one-size-fits-all approach seldom works.

Today, our team is made up of three DevOps engineers and a Release Manager. Some of you may say, "There is no such thing as a DevOps engineer, that doesn't make any sense." I'll just say this about that—"DevOps engineer" is a lot easier to say than "Helps other teams get stuff done better by automating as many tasks for that other team and assists with the handoffs between teams, either information or artifacts or anything else to get the job done engineer."

With that said, here's a look at what our team is responsible for:

- All things Docker, since just about everything our team does now goes through Docker
- Bamboo / build administration
- JIRA administration, which is a new responsibility we've taken on in the past year
- Release process and compliance of that process, which we've also taken on in the past year

I serve as the team's manager, and provide a vision for the team. But I've learned to step back and let the smarter people make things better—aka my team.

At JAMF, we have 120 people in the engineering department, and only 5 in DevOps—but I don't worry about the ratio of DevOps engineers to developers. If the entire DevOps team is truly focused on trying to automate themselves out of a job, it doesn't matter how many customers you support, because it motivates you to think about building efficiency at massive scale. What matters are the responsibilities the team takes on, and how well you perform against them.

4.) Finally, bring other teams along for the ride

As you address your culture, it can be helpful to start small, since change is easier for a small team. We found a two-person team that had some unique challenges and were open to change, and worked closely with them to convert their current processes over to the Atlassian toolset. In the end, they loved it, and became our biggest champions. They still tell other teams how awesome the conversion went, and earn us more trust and confidence.

We also make sure we lead by example, encouraging change within our own team. It's not fair for us to ask other teams to change if we can't show strong examples of change we've made within. Repeat this over and over: Without trust, there won't be change.

Next step: Put process before tools.

I see this happen more often than I'd like to admit: A team has a problem, and they immediately look for the best tools to solve it. Granted, sometimes the right tool is the best answer—but often it's not a toolset problem, it's a process problem, and no tool can make it better. In fact, a poor process will often cause a good tool to fail.

Along the way, it will also cause good people to grow more frustrated. So before we automate processes at JAMF, we like to make sure they are good ones to begin with.

Go epic or go home

I could dive into every single process we had to rethink along our DevOps journey, but that's for another paper. The most important decision we made was to move to a feature-branch workflow, much like git-flow with long live epic branches.

This gave us the flexibility to not have a single feature hold an entire release hostage. Now, we just needed the tools to make that happen.

Chapter 2

Automate yourself out of a job: DevOps tools

Process done? Bring on the tools.

A while back, I mentioned that we were all committing to a single trunk branch, using a ginormous build script that literally built everything all at once. We used SVN and Jenkins, and at one point had at least 50 Jenkins slaves.

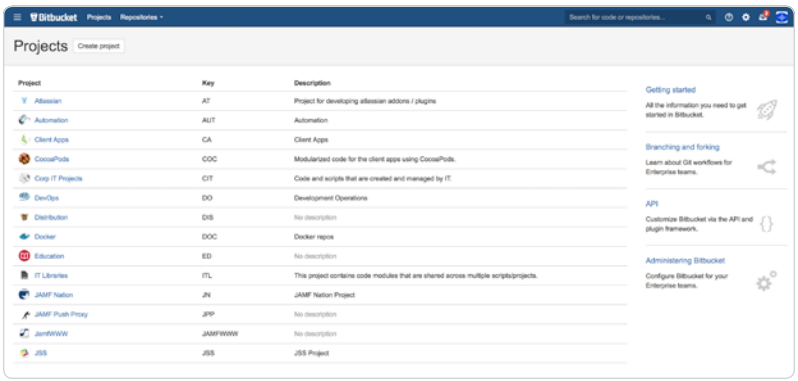
Every developer we hired back then said we needed to move to Git. I kept asking them two questions: "What's the business case for the move?" and "How does this help the customer?" I wanted to make sure that we didn't make a move "*because I like it.*" Once we decided to change our process to a branch-per-issue workflow, Git made great sense. It helped the customer by increasing code quality in the main-line (develop) branch, since only "done" issues get merged. It also allowed for features to be worked on longer than a single release cycle, which helped the engineering team scale more effectively.

Use collaboration to drive tool adoption

Ultimately, we decided to migrate to Bitbucket Server (called Stash at the time), since we still wanted to host behind our own firewall. It had everything we wanted to support our new process from a code repository standpoint.

A difficult part of the code migration was going from one huge repository to many smaller repositories. My team worked very closely with the engineering team on how we should split it up, and that collaboration really helped drive adoption.

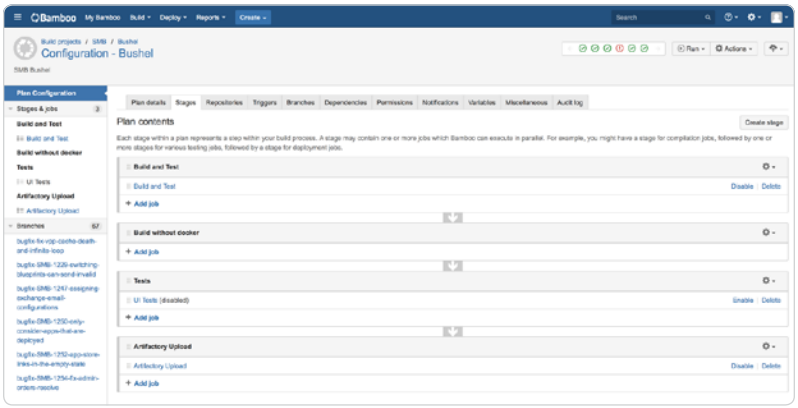
Once a plan was in place, we did a trial run of the migration and it went fairly smoothly—but before we could do a full cutover we needed to get our continuous integration environment working with Bitbucket Server:



Saying goodbye to Jenkins

Early on, Jenkins served us well. But unlike Bamboo, Jenkins didn't yet support plan branches. Plan branches were the killer feature (along with branch updater) that made the decision to switch over to Bamboo a no-brainer for us, along with the integration to Bitbucket Server. It was a perfect fit with our decision to use a branch-per-issue workflow, too. Now, a developer creates a branch, and boom—we have a build, with no additional work from our team. That's my kind of automation.

The migration from Jenkins to Bamboo was probably the hardest part of this, since we were running so many different kinds of jobs (from Java to Xcode to Windows installers). We were able to have Jenkins and Bamboo use the same build servers for a while, so it didn't have to be a hard cutover, which helped a lot—and now that it's up and running, we haven't looked back for an instant.

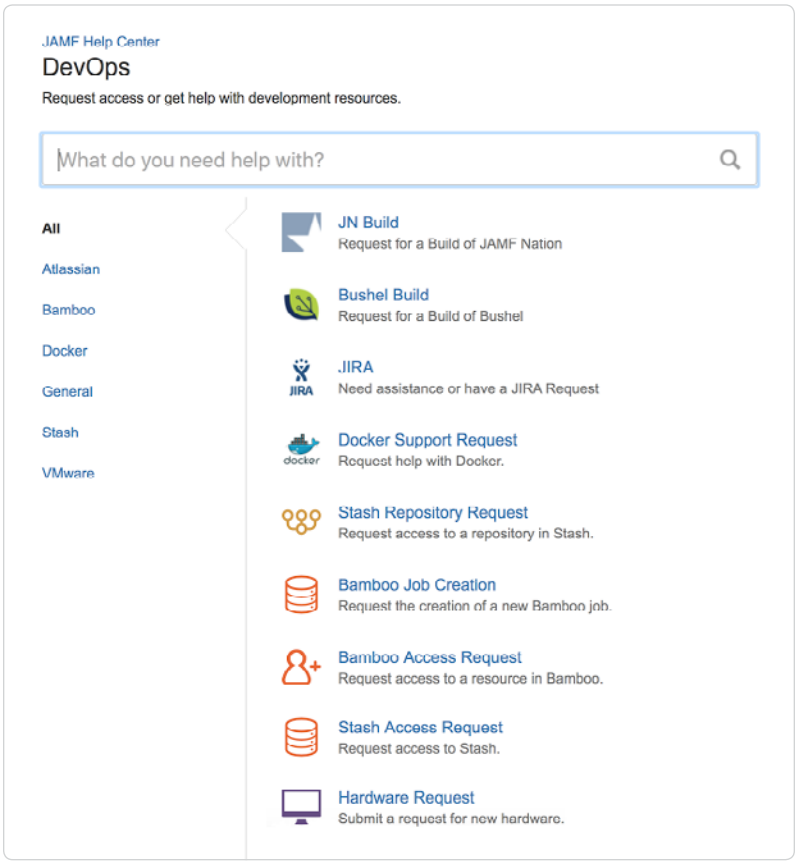


Hello, better issue tracking

Remember when I said earlier that new tools were a great way to expose bad processes? In this case, it was the opposite—our old issue tracking system wasn't allowing us to move to the more sophisticated, agile processes we needed to adopt. But the decision was still out on what system to migrate to.

We settled on JIRA, particularly due to the strong integration with Bitbucket Server and Bamboo. The Salesforce plugin was icing on the cake. The migration to JIRA was easy, and we could start with a pretty basic workflow.

JIRA Service Desk was brought in after our IT manager and I saw it at the Atlassian Summit in 2014. After they showed it at the keynote, we both looked at each other and basically said, "Uhh, yes please." DevOps was the first team to start using it, and now most internal help tickets for all departments run through JIRA Service Desk. On our DevOps team, we use it for Release Management, and we're looking into using it to automate that process for us as well.



After the Atlassian toolset migration, we needed an artifact management solution, and we decided to go with Artifactory. Having that toolset has given us much more flexibility with our Bamboo jobs and doing simultaneous releases.

With all these shiny new processes and tools, what's next?

We automate ourselves out of our jobs, of course—over and over again—by trying to solve problems or complaints in a way that the customer will never have to ask for it again.

We created a self-service testing environment for our Tech Comms team.

Now, they can choose any local branch build and have a running environment with optional test data in around two minutes. Our engineering department uses it, too. It's a lot easier than managing local VMs, since it's all running in Docker containers. QA told me it saves them 20% of their time testing issues, since it's so easy to spin up and update environments. For technical details, read my employee's blog post.

We're eliminating our build servers.

They're just a pain to maintain. There's always the fear of upgrading, or compatibility issues—or they're sitting idle while jobs are in queue because one build server doesn't have the right components to build a particular job.

In order to scale our CI environment, we also need to scale our build infrastructure—so we're switching to Docker build agents instead.

Now, we can have generic agents that can build anything by simply loading the proper Docker container. Well, almost everything (blast you, Xcode and Windows apps)!

Our automation tests also use these same agents. I can have 30 Docker agents, and the next job will just use an agent that is free. Load the appropriate container and boom, it's done—with no more servers to maintain, for the most part.

We'll also be hooking up AWS to our on-prem Bamboo server.

That way, if we need to burst past my local agents, Bamboo will automatically spin up generic agents in the cloud. Then, we'll have truly on-demand scaling of our CI environment—with no more waiting in line!

Conclusion

Looking back—and looking forward

Going back to those 1 a.m. mornings in 2012, it's hard to believe we're the same team and the same company. For us, adopting DevOps has made our smart people even smarter—but it took patience, persistence, a little bit of pushing, and a cultural commitment to collaboration and change.

By following our JAMF DevOps mantra to "Automate yourself out of a job," we've only grown in our careers—and the best part is, a few years in, we've really only just begun.

Ready to get started?

Learn more at Atlassian.com