A Software Engineer's Guide to DevOps

Laurie Barth

https://devopsdays.org/events/2019-washington-dc/program/laurie-barth/

- Twitter - https://twitter.com/laurieontech
- Video Recording - https://youtu.be/8VIC28AeTw8
- Slides - https://www.slideshare.net/LaurieBarth/a-software-engineers-guide-to-devops-keynote
- Audio - http://traffic.libsyn.com/devopsdays/software-engineers-guide-to-devops.mp3

LAURIE: Hi, everyone. It's so nice to be here. I have this cool opportunity to talk to everyone at once I want you to give a quick round of applause for organizers and the volunteers and the sponsors. It takes a whole lot to put on a conference.
[Applause]
I appreciate you all being here and not drowning on the way. Kudos. I'm here to talk about a software engineer's guide to DevOps. I'm Laurie Barth. You can follow me@Laurie on tech and I'm a software engineer for a company called Ten Mile Square technologies. And the reason I say that up top is not as a sales pitch, but we're a small company that works with companies big and small to solve whatever problems they have. And that means that I wear a lot of hats. And I kind of learn and do whatever it is that our clients need. And about a year ago, whatever they needed, was DevOps. So over the course of six months, I dived in to everything I could, context and vocabulary and tools and all of that. Quick question for the room. How many of you are DevOps engineers? Okay. So 50/50. There's a version of this talk that introduces you to what DevOps is. But that didn't really seem like the right fit for this crowd. So instead we're going to follow on what's on the back of our T-shirts with a house divided cannot stand and we'll get into that now.

So I'm going to be honest. As a software engineer, I haven't had the best of experiences with this thing they call DevOps. One story in particular sticks out. A few years ago I was working for a company, and the process worked like this. I would write code and in order to test it, I needed to test it with the full data set. And the full data set required some amount of manual testing. I wrote code, I committed it to git, I pushed it, and then Jenkins would take over, do an automatic build for me. I would SSH into the QA box, I would do a simple command and it

would pull the most recent build on to that box. Not too bad. Well, one day it didn't work out quite that way. Instead of success, the new builds on the box, I got a message that said this random file that you've never seen or heard of is locked. So I tried to do what I could and I had no permissions on that machine. That meant I had to track down the operations person that I knew. And he wasn't at his desk, because he was pulled in about a million different directions. So two hours later, two hours of being blocked, I finally get ahold of him. All he has to do is log on to the box, delete the files, say try again, everything is good to go. Sound familiar? Does that sound like developers trying to track you down? Now, the next story is not mine, it's a friend of mine. And full disclosure, I had no idea they were a sponsor until just this morning. So we're going to go easy. So she was working on a project for a client and she was integrating with a very specific version of elastic search. The operations team at the company had set up an environment where that version of elastic search was running. She came in one morning and the elastic search version had jumped. It needed to be pinned to a specific version. Luckily she had permissions, she spent the day dealing with versioning we all know how much of that a nightmare that could be. She comes back to the office, the elastic search version has jumped again and she calls up the operations team and says hey, what is going on. And they say, oh, it must be Chef. She has no idea what that means, and for those of you who don't know, Chef is a configuration management tool and you define your configuration using a recipe, very clever. And at some time interval, they check against each other, and if the running environment doesn't match the recipe, the recipe wins. Which is why she had great permissions to change whatever she wanted, because they didn't care. It was going to get overwritten anyway. So she goes to try and figure out what the heck they mean when they say it must be Chef. She gets a lot of marketing mumbo-jumbo, she gets a lot of terms she's never heard of and can't make sense of and she says okay, I'm going to throw me hands up and wait for them to fix it and I'll be blocked as long as they do. Sound familiar? Does this sound like kind of the push and pull that you experience with developers? In my prior experience, before I learned about this thing called DevOps, my interaction with DevOps went a little bit like this. I wrote code, I pushed it to Git, probably merged it into some major branch or the trunk. I waited for Jenkins to pick it up and build it. And then I threw it over a wall. I had no idea what happened on the other side of that wall. I didn't really want to know. It sounded confusing and obnoxious and slightly evil. And so the wall stood between me and operations and I knew my side. And this created kind of this natural tension that I think exists with a lot of teams, between developers and operations and it's what this culture of DevOps is supposed to help remediate. And the tension comes from a few

places. One of the things that I think makes a big difference is this lack of knowledge. As a developer, I don't know the tools. I don't know the context, I don't know the considerations at play. In fact, the only thing I see is constraints. To me, it's a bunch of rules that I need to follow to make sure that I don't break things. A box that I need to fit into. And that lack of knowledge, and the idea that this is all kind of making my job harder, leads to frustration. On both sides. Both on the operations side of the coin and the developer side of the coin. That wall. And it silos us so we don't want to understand the other side of the coin. Developers in particular throw their hands up and say I don't have permissions, I don't have access, I don't understand any of this, I'm not going to get involved. And I kind of picture this like two siblings on either corner of a wall with their arms crossed, being like you make my job harder, no, you make my job harder. So for developers, it's this mentality of, I just need it to work. I know my code is broken. I don't know how. I don't know what I don't know. So I need to get it on this box, I need to put it on this environment to see what these other variables are going to do to code that I pray is working but who knows? And DevOps is standing there saying stop breaking things. You avoid too much freedom, you get the run of this whole place, and all you do is make more problems for me. So how do we solve this? They gave me that nice intro at the beginning, tear down that wall!

How do we work towards the same goals? Because we have the same goals. A house divided cannot stand. This theming is working great for me today. I thought about this a lot. And one of the things that occurred to me is that we're all engineers. Our meta goals are different in terms of what our focus area is, but we all have similar justifications and motivations. So what about this novel concept of treating each other like engineers? What would that look like? What would that do? What would that change?

Engineers have something in common, regardless of what their focus area is. They want to know the why. They're not satisfied with just being told to do something. They want to understand why they're doing it, and how it works. So for DevOps, that has a lot of different ways to view it.

That can be, I need to know the edge cases; that can be I need to know what your environment needs. But what it really is, it comes down to what are the risks. What do I need to be aware of to make sure that this thing doesn't blow up in my hands. Because you don't seem to know what I need to know. And for developers, they're looking at these constraints, they're looking at these things that they think makes their job harder. And what they really want to know is where do I fit in? How am I a piece of this puzzle? What does this process look like,

and what am I doing to make it easier or harder? Because this is the crazy part. The reality is that developers' lives are made easier by DevOps. It's a bit of hyperbole but no one is going to care about their code quite as much as developers do. Their blood sweat and tears for weeks, months or years went into making these applications. They care all of this effort wasn't for naught. That it serves its purpose, that it does what they built it to do. They didn't spend all this time combing through bugs in a back room for people to get a blank screen. DevOps matters to them. They just might not understand why. One of the first things I learned in this 6-month journey was about CIDC pipelines and two things jumped out to me right away as something that mattered to me as a developer. The first was between local environment and shared dev test. In this example there's a Docker container but there's many other ways to do this, as you all know. I can put an environment that Mick I canned QA and production directly on my machine, that I could see everything that my application code would be touching, eliminate all my risk, know that I'm promoting something more than dropping it into a new pond and hoping that my fish swims.

The other thing that CI CD of pipelines that can do and do do that developers benefit from is cycle time. Too often we talk about what it takes to make development go faster. And rarely is it how smart your engineers are and how fast they can type. Most of the time it's the idea that developers write code and pigs don't fly so it's not going to work the first time. They write code, they test it and they iterate. How long does that process take? How much of that iteration is them waiting around for things to build or things to deploy? What does that look like? How much time are they waiting for to hope they can get in touch with a DevOps engineer because they broke something?

The other thing that I learned in the six months, was that DevOps itself has real reasons and benefits. That's why you do the work you do. You wouldn't do it if it didn't matter. And those reasons and benefits are things developers should know. Because it affects them too. Who cares about code more than the ones who built it?

The first thing that stuck out to me was repeatability. The idea that every time you're deploying the application, you're taking it that last mile, you're making sure that it follows that process the same way every time. And that's so important to me as a developer because I spent all that time on the bugs. And handling the edge cases and making sure that we've handled all our risks and the worst case scenario is every time we deploy is we're doing it a different way and all my work is a wash. It doesn't matter. That matters to me. Scaleability. It's always interesting to me that developers spend so much time thinking about scaleability within

their own code.

They make multithreaded applications, they make calls asynchronous. DevOps goes a step further with that. It makes sure we can have multiple instances of that application I built. It makes sure that traffic is getting routed in the right ways and things aren't going down. The worst-case scenario is that the code that I wrote doesn't get that last mile, it doesn't get in front of a customer. It doesn't matter how good of a job I did if it can't get there, it makes no difference. This matters to me. This is one of my favorite m oh meme cartoon things. You have had the time-honored tradition of seeing two threads running, about ten jobs waiting in the queue and yours is the very last one. Scaleability is something developers understand really well and this goes to that cycle time. These things affect not just the application and production, but it affects their development experience. All of these things should matter to us as developers.

And automation. I know at the beginning of the conference they made the joke about who's on call. Developers get put on call too, right? We're a part of that rotation. And I'm sure you nor I want to be the one to miss the 2:00 a.m. email saying that everything is melting down because we didn't scale. Automation has such a big impact on all of us. It affects on our on-call rotations, but it also affects our ability to keep that application in front of customers, to make sure that the work that we do matters. And failure states. DevOps is all about preventing failure, right? It's why you do what you do. And for developers, that is the ultimate. The last mile. I've side that about six times, but it's so important. Because too often developers think that their job ends with their code. But their code isn't running on their local environment, so even if they say it works on my machine, well it doesn't really apply. It doesn't help anyone. If you care that much about your code, if you put that time and that energy and that effort in, you want to make sure that it ends up on the screen of the person you built it for. So what does that look like and how do we do it, and how do developers have more of a hand in being successful? Part of this is considering everyone. There's a process and a solution and you want your developers to be involved in it, instead of throwing things over a wall and hoping that it doesn't have a bomb hidden in it. Part of this is tool solution. Choosing tools that work for everyone. Choosing tools that everyone can understand, that everyone has access to, that they can be effective in shepherding their code a little more seamlessly along a conveyor belt, instead of jumping over a wall. What you're really trying to do is blur the lines between developers and operations. That's why they made the DevOps name, right? And the reality is, that your developers aren't going to become DevOps experts and you don't want them to be. Because if

they try to do two things well, they're going to do neither of them well. You still want your experts, you still want your DevOps engineers. What you're really after is having them meet in the middle. Moving your developers just a little bit closer to your operations side, giving them some overlap. Because that overlap allows you to do really amazing things. It allows developers to give more relevant information as things move to the next gate. What systems need to talk to each other? What versions of dependencies might have an impact? What level of robustness should this application have? What kind of traffic might we expect to see? You're allowing for these smoother hand-offs between teams, because the first team understands what the second team needs from them. What information is relevant to their jobs, because they, at a certain level, understand what that job is trying to do. Now I'm sure you're sitting here thinking just because I give my developers overview of our pipeline and our process doesn't mean I'm comfortable giving them access. And I understand that. But I caution you against it because I think access is the only way that they're actively involved in those solutions. It's the only way they're not calling you up to say hey, I'm locked out of this box, I can't do anything and blocked for two hours and you're annoyed because you had six meetings today. And there's ways to do this. And I think my experience gave me a lot of good entry points. So what's interesting to me about DevOps versus writing code is that there aren't major differences. Your goal is very different. But the tools that you use and the process that they use, there are a lot of parallels between them. The most obvious is probably infrastructure as code. I found a lot of examples of this that felt very accessible to me as a developer but one really stood out. But just to say this up top, I'm not saying use this particular tool. It's just an example. So there's actually an NPM package called serverless and if you go to their GitHub repository they have an entire examples directly. In that directory, you will find pretty much every modern coding language coupled with all of the existing AWS databases, at least as of this writing, and you will find a cloud formation template that stands up an API gateway in that language, for that database. How amazing is that? What I loved about it was that I know what an API looks like. I write back end code. So for me I could look directly at that project. I could see my end points and see the one-to-one mapping with the cloud formation template. I could understand how things were getting deployed, what considerations there were, what I had to worry about, what the operations side of my company had to worry about. Another thing that was incredibly helpful was templates and GUIs and I kind of hate this mentality that we have as developers, especially that using visual aids is training wheels and you're not hardcore if you do so. It's particularly bad in DevOps because so much of DevOps is data visualization of how things move through a

process. Or what's happening when things are live. Green versus red status, that kind of thing. One of the things I found incredibly helpful and accessible was using the baked-in Helm template, just the example, go into Docker hub and pull in an image of something I recognized. I could take the name of that Docker image and the tag, and replace it in the Helm template and run a couple simple commands. The first was minikube start. As long as I put Kubernetes on my local machine, this was all I had to run. With the Helm template, I could run the Helm template against it and I ended up with this. I've got a deployment, I've got a pod, I've got a replica set. I'm not touching anything in production. It's a total playground. I got to iterate with this and see what impacts it had. I could look at fail over, I could look at making dependencies for databases. Everything I did gave me the opportunity to further see what I was accomplishing and what I could accomplish. Now, you're not all using Helm or Kubernetes or AWS or Serverless and that's okay. But these tools have a lot of similarities. They're trying to accomplish similar ends. They have different gotchas. And allowing developers time to have a bit of a playground, a safe space, will make them more comfortable using whatever your pipeline consists of. And this isn't just about making a cross-functional team. It's about having these overlaps of knowledge that we've been talking about. If developers better understand everything you do, they can better support your requests for information, and make you doing your job easier. They just need some context. So what does that look like? It's giving them those entry points. It's giving them a safe place to test things out, to iterate, to look at what it is that you're trying to accomplish, that you're trying to do day to day, to see those constraints and how hard it is to do the jobs. The deployment zone to deploy the thing that's doing the deployment. Mind-blowing. Helping them understand that will help them help you. It's helping them understand where they fit in. They're engineers, just like you are. As developers we want to know the why. We want to understand the unfamiliar terrain around us. If we get the puzzle piece that we are, we can better see our connection points, our integration points, what points of failure we may be creating, what risk we are introducing. And this is probably the most important takeaway. DevOps should matter to developers. I know that once I dove into it a step further, it became so apparent to me how integral it was to my success, and the team's success, and the company's success. And I'm still a software engineer. I enjoy being a software engineer. But learning DevOps has made me far more successful than I was before. Because I'm making sure my code goes that last mile. I'm making sure I'm giving the relevant information to the people it applies to. And if I need to jump on a box or into a Jenkins job or whatever it is, I can probably figure it out. So thank you so much for coming. I hope you enjoy the kitten in the

bow tie and enjoy the rest of the conference.

[Applause]