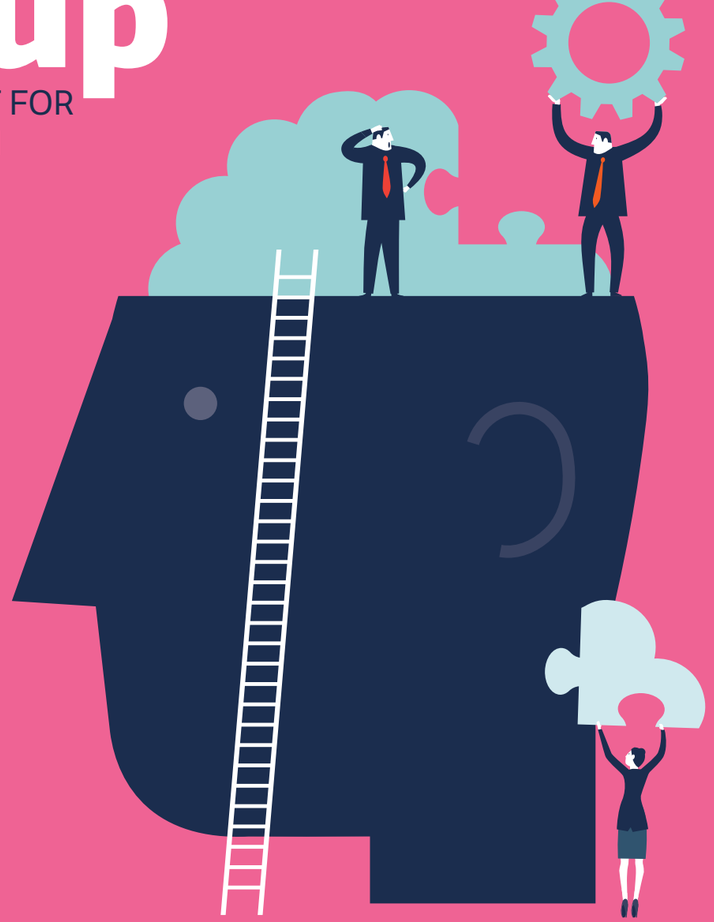
# Scaling the Startup

BY BOB DUSSAULT

## ADOPT A DEVOPS MINDSET FOR A SUCCESSFUL TRANSITION



Your customers don't care if you're small. When they report a problem, they want it fixed. When they ask for a new feature, they want it delivered as soon as possible. For the resource-constrained startup trying to mature, this can pose a significant challenge. You need to introduce more process and standards to ensure quality, resiliency, and security as you scale. But at the same time, you can't afford to squelch the innovation and agility that led to your earlier success.

### THROUGHPUT WITH QUALITY

I lead IT operations for a fast-moving educational software company. We're also a startup, with all the associated goals and challenges: commercialize as fast as possible, but with all the control and assurance necessary to meet customer expectations for the product. Then scale like hell.

Like most startups, we run lean, so adopting a platform as a service (PaaS) strategy gave us the economies of scale and ability to quickly spin up and provision the resources we needed to commercialize what began in a university lab incubator. We use Amazon Web Services (AWS), but thanks to our DevOps mindset and the process I'm about to share with you, we're not locked into any one vendor or platform that could limit our vision or ability to innovate.

That would be a constraint, and that's the number one enemy of the DevOps mindset.

### WHAT IS THE DEVOPS MINDSET?

The DevOps mindset focuses everyone's attention on one thing: removing all constraints from the value stream that produces meaningful things for your customers and your business. Nothing is sacred. Not culture. Not process. And definitely not tools.

To identify the constraints and make the right choices for accelerating value now and in the future, we developed the following framework:

1. Understand what we have
2. Know where we want to go
3. Start from the top down: culture, architecture, processes, then tools
4. Experiment, fail fast, and codify our successes

Here's a brief run-through of what that looked like for us.

#### 1: Understand What You Have and What You Can Let Burn

Before making changes, we first needed to understand what we really had. Most of what we had was in our CTO's head. So for us, the first order of business was to get everything documented and to determine our level of resilience and scalability. When you're running fast, you often leave a trail of technical debt. To find the time we needed to mature our DevOps value stream, we made a conscious effort to fix only those issues that had a meaningful impact on customer experience or were critical to our ability to deliver services. These were quick conversations. Is it important, or can we "let it burn"?

#### 2: Know Where You Want to Go

Our firefighting efforts bought us two things: a reasonably resilient and stable minimal viable product, and the time we needed to plan and implement the fully realized vision our founders intended. We hired a product strategist, reviewed all the capabilities and features that our founders wanted to get into the real world, and laid out our product and software roadmaps. Then we compartmentalized features and upgrades into bundles that were easier to digest and pushed them into our systems development life cycle so we could deliver them in a predictable manner.

#### 3: Start from the Top Down

Once you have your roadmaps in place—and they don't have to be very detailed—you can begin to focus on how to get there. This is where you will start to see the constraint points in your value stream. Constraints can be removed through culture, with processes, with tools, or with a combination of these. But you won't know until you start experimenting. And you can't experiment until you ask yourself some questions. You're still not talking about tools. These should be big architectural questions:

- Based on our product roadmap, how might we need or want to change our technology stack?
- Is our database architecture scalable?
- Where does it make sense to look at our own data center versus a hosted service?
- Should we use a different application framework?

The answers can be found through the lens of your own constraints. For instance, we knew we had an issue integrating with school information systems. One way to solve this issue would be to use application segmentation, by writing a microservice focused on that single task. Moving to a microservices architecture would mean fewer restrictions on where these loosely coupled, containerized apps needed to reside. And that decision would have a direct impact on our platform strategy and the notion of vendor lock-in I mentioned earlier.

Our entire journey is framed around adding automation into the value stream. We had to get to the point where the developers were not waiting on operations to deliver new services. But there was still the issue of security and compliance. Do we allow people to push directly into production, or should there be gates? So we were very interested in infrastructure as code, and in change control with automated assurance.

Automation removes ambiguity from a process. When you spin up an Amazon Elastic Compute Cloud (EC2) server

based on a configuration articulated in a job ticket, it may or may not meet the needs of the task. But if you create a cloud formation stack and write a script with a "deploy this when that" trigger, you know what you're going to get every single time.

You can apply this level of automation to development and integration testing as well. By having trust early in the development cycle, you can increase agility and quality, and the back-end constraints start to fall away.

When you view your existing processes through the lens of your constraints and the ideals of your product roadmap and cultural pillar, ask yourself: Do our processes still make sense? If they don't, move on. If they do, it's time to experiment.

### 4: Experiment, Fail, and Codify Successes

Our CTO has a PhD in data analytics and follows a rigorous scientific method when experimenting with potential big data solutions. The methodology I follow in operations (and am helping to push into development) is a bit more straightforward:

- What is the business need?
- Does the solution fit within our roadmap vision?
- Is it practical from a logistics standpoint?
- Is it cost-effective?

Cost is the least important factor. If it works, we know it'll save us money in the long run, because we'll get more done. If the experiment fails, you move on quickly. If it succeeds, you have something you can codify into a new published process.

## A WORD ABOUT TOOLS

Tools are important, but they're just that: tools. They're meant to serve you, not the other way around. We use Jira for job ticketing and tracking, but we don't let limitations with Jira dictate changes to our workflows. If the tool doesn't remove a constraint, or if it creates new ones, you need a new tool. There are no sacred cows.

Ultimately, the DevOps mindset is about going back to basics—to the principles of customer focus, agility, and that single-minded determination to knock down obstacles to success. You do that by understanding your value stream and the constraint you're trying to solve for. Then ask yourself questions, starting with architecture and frameworks and then moving on to individual processes and tools.

We love the kanban board tool Trello. Yet here I am with our director of engineering, sitting cross-legged on the floor, staring at a bunch of Post-it notes on the wall. It works. And we have plenty of Post-its and a lot more wall.

**About the Author** Bob Dussault is Senior Director of Data Center and Technical Operations at FastBridge Learning. Or as Bob puts it, "an infrastructure guy who has spent the last ten years figuring out how best to provide secure and agile resources for a software-as-a-service (SaaS) provider."

### Toolbox

The tools you use will be unique to your needs. In case you're wondering, here's what we're experimenting with:

- Continuous integration (CI): Jenkins
- Version control: Switched from Subversion to Git to work in a distributed fashion
- Build agent: Maven
- Test automation: Selenium
- SQL schema consistency: Redgate SQL Toolbelt
- Structural administration: Kubernetes and Puppet
- Monitoring and metrics: New Relic and Google Analytics