If your team is used to managing all of your infrastructure by hand, switching to infrastructure as code requires more than just introducing a new tool or technology. It also requires changing the culture and processes of the team. Changing culture and process is a significant undertaking, especially at larger companies. Because every team's culture and process is a little different, there's no one-size-fits-all way to do it, but here are a few tips that will be useful in most situations:

- Convince your boss
- Work incrementally
- Give your team the time to learn

Convince Your Boss

I've seen this story play out many times at many companies: a developer discovers Terraform, becomes inspired by what it can do, shows up to work full of enthusiasm and excitement, shows Terraform to everyone… and the boss says "no." The developer, of course, becomes frustrated and discouraged. Why doesn't everyone else see the benefits of this? We could automate everything! We could avoid so many bugs! How else can we pay down all this tech debt? How can you all be so blind??

The problem is that although this developer sees all the benefits of adopting an IaC tool such as Terraform, they aren't seeing all the costs. Here are just a few of the costs of adopting IaC:

Skills gap

> The move to IaC means that your Ops team will need to spend most of its time writing large amounts of code: Terraform modules, Go tests, Chef recipes, and so on. Whereas some Ops engineers are comfortable with coding all day and will love the change, others will find this a tough transition. Many Ops engineers and sysadmins are used to making changes manually, with perhaps an occasional short script here or there, and the move to doing software engineering nearly full time might require learning a number of new skills or hiring new people.

New tools

Software developers can become attached to the tools they use; some are nearly religious about it. Every time you introduce a new tool, some developers will be thrilled at the opportunity to learn something new, but others will prefer to stick to what they know and may resist having to invest lots of time and energy learning new languages and techniques.

Change in mindset

If your team members are used to managing infrastructure manually, they are used to making all of their changes *directly*: for example, by SSHing to a server and executing a few commands. The move to IaC requires a shift in mindset where you make all of your changes *indirectly*, first by editing code, then checking it in, and then letting some automated process apply the changes. This layer of indirection can be frustrating; for simple tasks, it'll feel slower than the direct option, especially when you're still learning a new IaC tool and are not efficient with it.

Opportunity cost

If you choose to invest your time and resources in one project, you are implicitly choosing not to invest that time and resources in other projects. What projects will have to be put on hold so that you can migrate to IaC? How important are those projects?

Some developers on your team will look at this list and become excited. But many others will groan—including your boss. Learning new skills, mastering new tools, and adopting new mindsets may or may not be beneficial, but one thing is certain: it is not free. Adopting IaC is a significant investment, and as with any investment, you need to consider not only the potential upside but also the potential downsides.

Your boss in particular will be sensitive to the opportunity cost. One of the key responsibilities of any manager is to make sure the team is working on the highest-priority projects. When you show up and excitedly start talking about Terraform, what your boss might really be hearing is, “Oh no, this sounds like a massive undertaking. How much time is it going to take?” It’s not that your boss is blind to what Terraform can do, but if you are spending time on that, you might not have time to deploy the new app the search team has been asking about for months, or to prepare for the Payment Card Industry (PCI) audit, or to dig into the outage from last week. So, if you want to convince your boss that your team should adopt IaC, your goal is not to prove that it has value but that it will bring more value to your team than anything else you could work on during that time.

One of the least effective ways to do this is to just list the features of your favorite IaC tool: for example, Terraform is declarative, it’s popular, it’s open source. This is one of many areas where developers would do well to learn from salespeople. Most salespeople know that focusing on features is typically an ineffective way to sell products. A slightly better technique is to focus on benefits: that is, instead of talking about what a product can do (“product X can do Y!”), you should talk about what the customer can do by using that product (“you can do Y by using product X!”). In other words, show the customer what new superpowers your product can give them.

For example, instead of telling your boss that Terraform is declarative, talk about how your infrastructure will be far easier to maintain. Instead of talking about the fact that Terraform is popular, talk about how you’ll be able to leverage lots of existing modules and plugins to get things done faster. And instead of explaining to your boss that Terraform is open source, help your boss see how much easier it will be to hire new developers for the team from a large, active open source community.

Focusing on benefits is a great start, but the best salespeople know an even more effective strategy: focus on the problems. If you watch a great salesperson talking to a customer, you’ll notice that it’s actually the customer that does most of the talking. The salesperson spends most of their time listening and looking for one specific thing: What is the key problem that customer is trying to solve? What’s the biggest pain point? Instead of trying to sell some sort of features or benefits, the best salespeople try to solve their customer’s problems. If that solution happens to include the product they are selling, all the better, but the real focus is on problem solving, not selling.

Talk to your boss and try to understand the most important problems they are working on that quarter or that year. You might find that those problems would not be solved by IaC. And that’s OK! It might be slightly heretical for the author of a book on Terraform to say this, but not every team needs IaC. Adopting IaC has a relatively high cost, and although it will pay off in the long term for some scenarios, it won’t for others; for example, if you’re at a tiny startup with just one Ops person, or you’re working on a prototype that might be thrown away in a few months, or you’re just working on a side project for fun, managing infrastructure by hand is often the right choice. Sometimes, even if IaC would be a great fit for your team, it won’t be the highest priority, and given limited resources, working on other projects might still be the right choice.

If you do find that one of the key problems your boss is focused on can be solved with IaC, then your goal is to show your boss what that world looks like. For example, perhaps the biggest issue your boss is focused on this quarter is improving uptime. You’ve had numerous outages the last few months, many hours of downtime, customers are complaining, and the CEO is breathing down your manager’s neck, checking in daily to see how things are going. You dig in and find out that more than half of these outages were caused by a manual error during deployment: e.g., someone accidentally skipped an important step during the rollout process, or a server was misconfigured, or the infrastructure in staging didn’t match what you had in production.

Now, when you talk to your boss, instead of talking about Terraform features or benefits, lead with the following: “I have an idea for how to reduce our outages by half.” I guarantee this will get your boss’s attention. Use this opportunity to paint a picture for your boss of a world in which your deployment process is fully automated, reliable, and repeatable so that the manual errors that caused half of your previous outages are no longer possible. Not only that, but if deployment is automated, you can also add