CASE STUDY

IXL Learning: One Stack to Rule Them All

1,001-5,000
Company size

AWS, Terraform
Stack

Senior DevOps Engineer Rob Ramirez devised the One Stack construct, a single entrypoint for an entire business unit's set of Spacelift stacks to be managed as code.

IXL uses approval policies to enable cross-business-unit collaboration and advanced visibility while also requiring expert approval for any domain changes.

IXL continues to focus on standardization so that the standard approach will be the easy, default option for present and future teams.

Summary

IXL Learning's award-winning personalized learning platform is the gold standard in educational technology. With brands such as Rosetta Stone, Teachers Pay Teachers, Dictionary.com, and Education.com, IXL is helping 96 of the top 100 U.S. school districts improve teaching and learning with its technology. However, multiple acquisitions mean multiple methods for managing Terraform deployments, and binding that diverse range of brands together with one infrastructure-as-code (IaC) management strategy is an epic undertaking that Spacelift is helping them with.

We spoke to Clayton Keleher, Senior Software Engineer on IXL's Unification team, and Rob Ramirez, IXL Senior DevOps Engineer, about how they are forging a unified approach to IaC with Spacelift.

IXL's IaC challenge

When their existing Terraform solution became cost-prohibitive due to a change in its pricing model, the company started investigating alternatives. Concerns about unsustainable costs could not be something that would restrict their design ambitions as they expanded their IaC footprint.

Other criteria included:

- Ease of use and onboarding for new users without inhibiting experienced users
- Strong governance, auditing tools, and permissions settings
- Managed state files
- Management of the platform using IaC

Spacelift's work as one of the founding companies behind OpenTofu brought it to the IXL team's attention, and its solid support and scalable pricing model made it IXL's ultimate choice.

How IXL Learning manages its infrastructure — a single, unified approach

The various business units within IXL have different methods of organizing IaC, ranging from large monorepos to small polyrepo patterns. A history of acquisitions has left IXL with multiple mature, robust Terraform practices across different business units, with a huge variance in adoption methods. However, the team is building company-wide standards for new projects, with the intent of having monorepos for each business unit, along with a business-unit-agnostic monorepo for management of shared resources and modules. Adopting this new execution platform for the entire company has given the team the opportunity to realign their practices, with the ultimate aim of converging on a single, unified approach that other teams can adopt. As Clayton Keleher explains, "our currently evolving strategy is to maximize the amount of Spacelift configuration and management done as IaC, including stack definitions."

The One Stack concept

Senior DevOps Engineer Rob Ramirez is the mastermind behind the One Stack construct, a single entrypoint for an entire business unit's set of stacks to be managed as code. He explains the rationale behind the concept: "We wanted to have all the Spacelift objects in IaC Terraform, but I realized that you can consolidate the creation of certain stacks and keep going up and up and up, but at a certain point, there is one thing that you have to create by hand. So I created One Stack — I called it 'the one stack to rule them all', but we shortened it to One Stack."

The One Stack creates a "stack" stack that generates all other "resource" and "context" stacks. This approach ensures consistency, delivering a uniform structure across all environments and applications, especially for foundational elements like IAM, security groups, and VPCs. As Rob points out, "each 'stack' stack is identical to any other 'stack' stack, the only difference being a local variable."

Standardizing IaC across IXL

Another concept that is helping standardize IaC across the company is the skeleton script, which, as Rob explains, "scaffolds up all the Spacelift objects required for a typical workspace to make it even easier for our folks to follow our standard pattern." He wrote the script after an IXL core team asked him to help them set up their Cloudflare. "I learned that most of the files in there are pretty common, but when we were setting up this structure, they were basically taking another directory and copying all these files over, which I thought was tedious." To avoid this repetitive manual effort, Rob wrote a script that the team could run to create the necessary structure. "Now the context creation stuff is all pre-set up for you. It speeds up the process, not having to copy everything over and then manually change a bunch of stuff, and it also helps us ensure we're using naming standards for particular things."

This pattern materializes as shown on the right:

Each environment has its own context, and all contexts share the same common files. These files are injected from the directory `context/$ENVIRONMENT/mounted_files` into the project root at execution time. All of this is orchestrated by IaC definitions, with the team's One Stack creating the stack stack, which in turn creates the context stacks and resource stacks. As their names imply, the context stack generates environment-specific contexts for stacks to bring in, and the resource stack creates resources by combining contexts and the "common" directory shown in the image above.

Here is an example of the local variables IXL uses to compose the many stacks they create within Spacelift:

IXL's approach to policies

Policies control the kind of resources engineers can create, their parameters, the number of approvals required for a run, the kind of tasks executed, what happens when a pull request is open, and where notifications are sent.

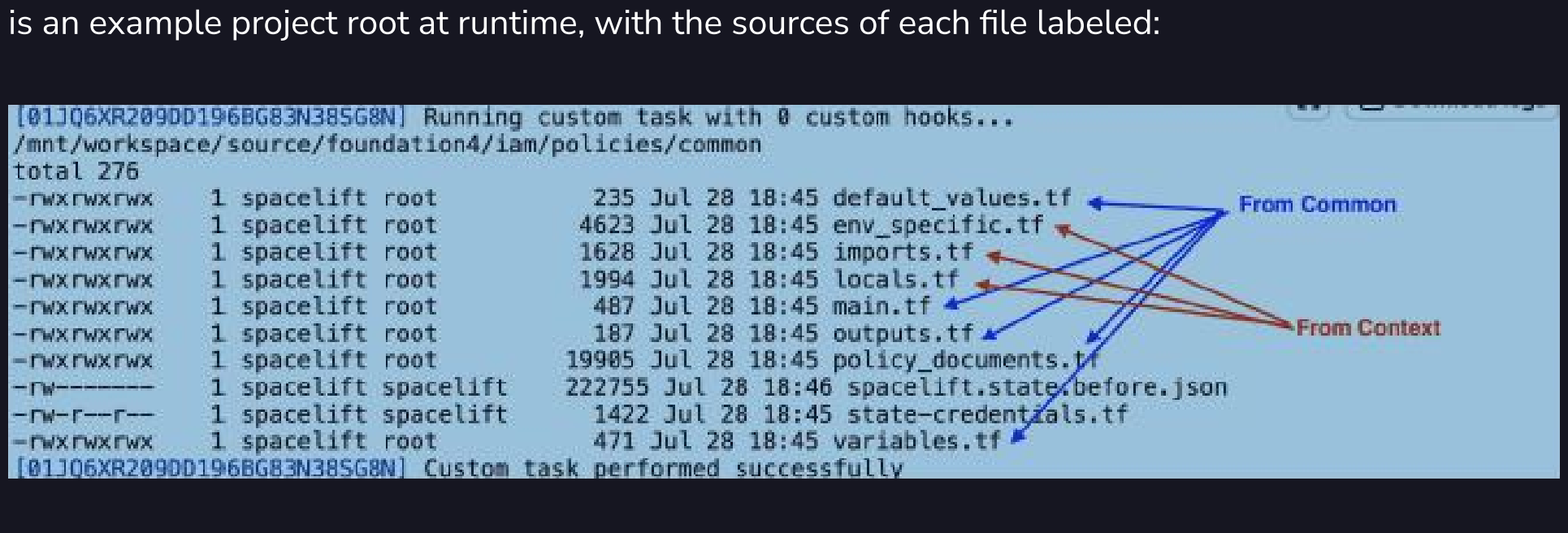
IXL uses approval policies to enable cross-business-unit collaboration and advanced visibility while also requiring expert approval for any domain changes. Specifically, all IXL Spacelift users have default visibility into every other space across the company and can even initiate runs as desired. However, so-called admin approval policies ensure that no run is applied unless an administrator of that product space has approved it.

Clayton explains how it works: "For example, all stacks for our TPT product are within our TPT Space, and while anyone can initiate a run, approval from a member of the TPT Admins group is required before it is allowed to apply. We are also experimenting with requiring that an approval come from someone other than the initiator of the run — this would give us a similar workflow to how we currently manage Pull Requests in GitHub, where peer approval is required."

This approach mirrors how almost all IXL stacks interact with policies. IXL is also in the early stages of implementing other policies, including notifications for approved applies. With the team more comfortable working in Rego (the language used for writing policies in Open Policy Agent), they are exploring other policy types. "Spacelift's policy system is decently extensible, so there's a lot we could do with it in the longer term."

Where contexts come in

IXL uses contexts extensively to define environments. Environment-agnostic IaC is combined with environment-specific contexts to keep code consistent between different environments, and contexts are also used for easy secret management when appropriate. Each environment has an environment-specific stack and a corresponding context used to inject environment-specific data into it. The stack tracks a "common" directory from the source repository, which is used across every environmental permutation. In other words, if a project has a staging stack and a production stack, both stacks are referring to the same "common" directory, and the environment-specific differences are contained in the injected contexts. The repository contains environment-specific directories, one for each environment, which then contains a "mounted_files" directory. Files from that directory are injected into the project root at runtime, to combine with the files in the "common" directory. This creates a project root with environment-specific local configuration, import blocks, and so on. Below is an example project root at runtime, with the sources of each file labeled:



The future of IXL's IaC strategy

For properties that already have IaC, IXL is moving them toward the One Stack model, but the priority is to take properties that don't have proper IaC yet and set up their foundations and eventually their application infrastructure with the model. "That's the vision," says Rob. "Basically, what I'm trying to do is present a concrete view of what it could be like, and then, as I build this, I can build or propose tools or processes that would make life easier for everybody. That will entice people to embrace this structure because I know that it is a little complicated."

The plan is that, as more properties shift to the standardized model, IXL will reach a critical mass where people are building tooling and processes for it, and the support teams are becoming more familiar with it. "Some of our properties don't have their own dedicated DevOps teams, so one of the biggest enticements is that if they follow this model, they'll gain a set of DevOps engineers who understand it and can help you take care of your infrastructure code because it will look like every other property that has onboarded it."

Clayton underlines why this kind of standardization is so important. "We have 15+ properties and they're growing, year by year — month by month in some cases — and so one of our big charges is to support the implementation of things like what Rob is doing, which is to take a common pattern that is generalized enough that anyone can use it and have just two or three experienced DevOps engineers supporting an entire fleet of brands that can extend as needed."

Rob agrees: "My job is to support Clayton and his team by designing a model that is basically universal. Every property that uses AWS is always going to need networking, so that's a VPC, security groups, IAM policies and rules, and all this other stuff that's really easy to fit in. The hard part is accommodating the application infrastructure for different services, so when I come across a new property that uses something else — like, say (Amazon) Kinesis, I need to design a universal way to accommodate that particular service, but it will have the exact same structure so that it becomes universal for every property that uses Kinesis, for example."

Much of the standardization work is now complete, and the key remaining task is a one-time effort to pull everything into the standardized model so that it's easier to follow the set approach than to try to build something new.

Clayton sums it up: "We continue to focus on standardization and easily repeatable patterns, with a goal of making our standard approach the easy-default option for those who come later."