**Balancing Centralized & Federated IT in a DevOps Transformation**

**AUG 13, 2019**

There's a trade-off between a centralized IT function, with its operational speed and simplicity, and a federated IT function that gives developers flexibility around tools and technologies. How do you balance those as you transition to DevOps?

**Speaker**

* 

**Armon Dadgar**Co-founder & CTO, HashiCorp

**Transcript**

Hi my name is Armon, and today we’re going to talk about the trade-off between a centralized IT function that provides a lot of common services versus a more federated IT in terms of letting application developers have more flexibility around tools and technologies, and how that impacts a large organization transitioning to adopting DevOps.

When we talk about central IT, an analogy I like to use is thinking about it like a tree. At the base layer, we have a shared technology trunk, responsible for providing different IT services that are common across the entire organization. Examples are things like single sign-on. Everyone might log into the company with the same Active Directory system providing the credentials. That’s an example of something that might be provided through the common trunk.

On top of that we’re going to have a series of branches. And these branches will align to maybe a business group, maybe a line of business, maybe just a very large application that has its own way of doing things.

You can think about the branches as providing a key difference from the trunk but still providing a set of common services that we share across many applications. These applications then ultimately end up being the leaf nodes. It’s available to the leaf nodes as a shared service, either provided through a branch, which is maybe differentiated from other branches and differentiated from the trunk, or it’s provided from the trunk, which is providing a common set of services across an entire organization.

**Federated IT**

As you think about this analogy, you can see 2 very different types of organizations.

On 1 extreme, you’ll get a very thin trunk with minimal branching. Effectively the leaves are located directly on top of this very small trunk. An example of this would be if I have a central IT group that provides not much beyond maybe a shared login and an ability to provide VM infrastructure—a very limited set of shared capability. What this means is a lot of the higher-level functionality, higher-level problems, need to get solved at the leaf layer.

The other extreme of this is if I have a very thick trunk that provides a high level of standardization with minimal branching and puts the leaves on top. The difference would be, in this kind of an organization [thick trunk], the central trunk is providing a lot of common services. It might be, How do we provision our infrastructure? Or security concerns like [secrets management](https://www.vaultproject.io/guides/secret-mgmt/index.html) or single sign-on. It might be things like traffic, routing, and management. It might be things like a common way of deploying our applications, logging, monitoring, metrics, etc.

So you have these trade-offs between a relatively thin central IT that allows flexibility at the leaf layer. In this world, our applications have a lot of flexibility and many degrees of freedom in terms of, “Hey, maybe I want a different login solution, or I want to use Kubernetes instead of Pivotal Cloud Foundry or instead of some other service.”

What we’re trading is a lot of flexibility, so we’re going to get more flexibility in terms of what our developers want to use. The leaf nodes represent the end applications, the developers. So, they’re going to gain that Day 1 flexibility. “Hey, I can use Kubernetes, or PCF, or Lambda, or whatever I want.” But the challenge is there’s a lot of reinvention. You have to reinvent the wheel over and over because the shared trunk isn’t providing that as a service.

**Centralized IT**

Versus: Over here you get a different set of concerns. In this model, what I’m going to lose very explicitly is flexibility. What we’re doing is saying, “I want a standardized platform that provides these capabilities.” So, by definition I can’t say, “I’m going to use a different way of deploying or a different log management tooling.” But what that gains is a lot of reuse. I don’t have to re-implement logging. I don’t have to re-implement monitoring. I don’t have to re-implement the way my app deploys times every single application.

From the developer’s perspective, there is this trade-off. Here [Federated IT] I have choice, but there’s a lot of work I have to do times every single application. Here [Centralized IT] I have less choice, but I don’t have a lot of work to do. It’s kind of prescribed by the platform in terms of how we want to solve this. If we think about it from a higher-level business perspective of “What’s the trade-off?” one of the other challenges we add into it is, as we think about governance and control over here [Federated IT]—let’s say I want to implement some new [GDPR](https://www.hashicorp.com/gdpr) rule—I need to think about my data protection a different way, or we need to encrypt all of the users’ data, things like that.

Well, the problem is, I have flexibility, which means inevitably many different choices were made in terms of database technology, in terms of deployment architecture, in terms of the way an application is run. And so, the challenge becomes: I have to think about that control times many different environments.

This becomes an operational challenge. We get a lot of operational complexity in this world [Federated IT], where over in the other side of it [Centralized IT], it’s sort of a different trade-off—because everything is standardized on 1 common core, it’s much simpler operationally if I add 1 new control, because there’s really a common standard. Over here [Centralized IT], it’s just a lot more operationally simple. These extremes end up being illustrative and are useful for talking about the trade-offs.

**Most orgs take a middle road**

In practice, most organizations look like something in the middle. On one extreme you have a lot of flexibility and limited standardization, which make the controls hard. On the other side you have very limited flexibility, but you have a high level of reuse and it’s operationally simpler.

The practical reality is most organizations are somewhere in the middle. We like to be conscious and aware about that choice we make in terms of what should live inside of this common trunk, because that’s really the leverage that we’re providing all of our developers. If I solve logging as part of the common trunk, now for every single application, they don’t have to re-implement logging. So, I want to be conscious about, “What are the things that are going to be low value for all of my developers to have to reinvent?” versus the things that are going to be high value if we’re reusing.

At the same time we want to acknowledge that these different business groups might have different requirements. If I have 1 group that only delivers user-facing microservices, this might be a really good branch to say, “You know what? You should use Kubernetes to deliver your applications because it’s better suited for that type of workload.”

Versus maybe a back-office team that is more heavily focused on data processing, big data analytics. They’re doing batch reporting. Kubernetes might not be a sensible platform for them. Here they might standardize and say, “You know what? We’re going to use Spark or a Hadoop data platform as our core, and all of our leaves are going to run on top of that common platform.”

Whereas over here we said, “It’s Kubernetes.” And maybe you’ll have a third group that PCF makes the most sense.

**Think about the trade-offs**

I think the goal here is to really think about and make those trade-offs around, “Where do we want flexibility? Where do we want a degree of freedom?” That degree of freedom might make sense if we have a ton of applications that have a shared pattern, a shared architecture.

Versus when we talk about things like provisioning workflow, or the way we do network connectivity, or the way we do logging. There’s not a lot of value in solving that 5 different ways. If you have a logging solution that can work across all of that, or a central way of doing provisioning, or security, or actual application deployment patterns, then those make sense to try to move into a centralized trunk.

I think the goal regarding centralization versus federation is being conscious of what trade-offs we’re making. And as long as we’re making those conscious choices of, “Yes, in our end state we want to give business units flexibility around runtime platform, but not [provisioning, security, connectivity](https://www.hashicorp.com/),” I think it’s useful to think about it through the lens of the branch-and-trunk analogy.

**Moving from ITIL to DevOps**

When we talk about the branch-and-leaf analogy as we think about [DevOps](https://www.hashicorp.com/devops-defined), a common question becomes, “OK, but how do I switch from my existing organization, which already delivers in an ITIL model, where we’re filing tickets and waiting between organizations, to a DevOps model, which is [self-service](https://www.hashicorp.com/cloud-operating-model) and more developer-oriented?” How does that map back to this analogy?

Instead of 1 tree, a large organization looks like a forest. There’s not necessarily a single tree in terms of how delivery is done.

You might look at it and say, “You know what? I have an existing setup in terms of my ITIL process that already looks like a large tree. We have a common trunk that maybe delivers Active Directory for security and delivers VMware on premises in terms of the infrastructure that we’re running. And then I have a branch that is, let’s say, Pivotal Cloud Foundry in terms of my application deployment, and I have applications running on top of that already. So I have an existing infrastructure in place.

Now, the challenge is, over here we have an ITIL model running. With ITIL ,the classic challenge is, we’ve organized our people differently. We’ve organized around a series of technology silos. So, let’s say I have my VMware team that’s responsible for provisioning VMs. I have my F5 team that updates my [load balancers](https://www.hashicorp.com/resources/load-balancing-modern-architecture), and my Palo Alto team that manages the firewalls.

My experience as an end-user developer is I file a ticket against the VMware team, and some number of weeks later I get a VM. And then, following from that I file a ticket against F5, wait some number of weeks, and then the load balancer gets updated. And then I file a ticket against my firewall team, wait some number of weeks, and my load balancer comes out.

This end time is the time-to-value for a customer. Because at any point in the middle of this it wasn’t useful to me. If I had a VM that didn’t get traffic and there were no firewall ports open, it’s not that helpful. It’s only once I have the infrastructure running, traffic can get to it, all the firewall walls are open, great—there’s now a real value. I can deploy my application, and it can do things.

**Organizing around provisioning, security, and connectivity**

In this model what we did is have a series of teams organized around technology and the front door, the interface to working with the team, is: You file a ticket. You file a ticket against the VMware team and then they manually create a VM, and then you go from there.

The challenge becomes, as we adopt a DevOps model, this existing tree is there already. It’s already working, there are already applications, and you can’t really change it in flight. We have too many business-critical needs to just say we’re going to take a pause for a few years and just redo everything.

So instead you almost think about it as, you’re planting a little sapling alongside. The common pattern becomes, How do I bootstrap a new tree alongside, which is DevOps-native?The goal becomes, Define that common trunk. And usually at this stage what we’re talking about is onboarding the first app, maybe the first 5 apps, so we don’t need to distinguish and create branches yet. Because we’re really only onboarding the first few applications in the new model, but most likely also operating as cloud infrastructure.

So how do we plant a new little tree that is operating in a different model? Part of what it comes down to is: It’s a different process; it’s a different structure in terms of how we deliver. We leave our existing ITIL tree, let it function. You want business continuity. Things have to continue to work. But then, as we define this new one, what do we put in this common trunk? And what we often see is there are [4 key layers](https://www.hashicorp.com/resources/divide-and-conquer-cloud-operations), which include, How does an application provision? And oftentimes for us this is using [Terraform](https://www.hashicorp.com/products/terraform) as the tool.

There’s a question around, “How do we provide security of not only application infrastructure, but data as well, as well as the infrastructure?” And so [Vault](https://www.hashicorp.com/products/vault) tends to be used here.

There’s a common network connectivity challenge. How do we connect all of the pieces together? And this includes things like our API gateways, our load balancers, our firewalls. We need some registry that knows what’s running where and then use that in an API-driven way so these other pieces can interface. This is [Consul](https://www.hashicorp.com/products/consul).

And then, at the topmost layer is the actual runtime that we end up using. And this might depend on what our organization’s comfortable with. We have a product, [Nomad](https://www.hashicorp.com/products/nomad). It’s common for customers to use Kubernetes here as well. If you’re big data–oriented, you might use a system like Spark instead.

But these are the pieces that we end up seeing as being common. Particularly these 3 tend to sit in the trunk as a shared service versus the runtime ending up being one of the key branches. One group will use Apache Spark, one will use PCF, onewill use Kubernetes, one will use Nomad.

**Leaving ITIL behind, gradually**

What we want to do is plant this new trunk, allow a team to bootstrap using a different set of technologies, but more importantly a different process. You’re not bringing ITIL with you. The goal becomes, for these applications, onboarding will be a totally self-service experience. And so we need to define this referenced stack or trunk that we’re using to expose this capability, but doing it in an API-driven way.

So when the developer comes in, they’re not filing a ticket against the Terraform team; they’re taking their own module and saying, “Great, this is a web application that runs on Kubernetes. There’s a [pre-approved module](https://www.terraform.io/docs/enterprise/registry/publish.html) and I can self-service the provisioning and deployment of. And the way I will interact with my network is in an API-driven way.

When the application gets deployed it gets automatically registered as part of the registry, and that drives the downstream network automation in terms of firewalls and load balancers and API gateways. We’re not filing a ticket against the API gateway to add a new backend instance.” So this is just as much about this process shift as it is about the technology shift.

This is how we like to talk about it: When we think about the trunk and branch, keep your existing tree. This is business as usual. Identify the 5 new applications or maybe the 1 application that’s going to migrate to DevOps, that’s going to [migrate to cloud](https://www.hashicorp.com/cloud-operating-model). Define the trunk in terms of how we want to operate in this new environment, and then onboard that 1 application.

But the existing infrastructure, the existing IT, is still business as usual. It’s going to continue to function. And what you find over time is that it becomes a strategy around stopping onboarding new applications here. All of our net new applications land here. So this sapling grows and becomes its own tree.

Over time we’ll scale this up and we’ll have a larger trunk as we onboard more applications and services. We might hit a tipping point where we say, “Great, when we got to the 50th app, there was a reason to branch, and we have a Kubernetes branch and we have a Spark branch.” And then, our applications are running on top of this.

What we’re basically doing is, as we write new greenfield apps or as we modernize our applications or over time some of these services might just be getting deprecated, they’re no longer relevant. This tree will start to grow as we move things over into the new model, and this tree will start to shrink naturally, sort of a **contain-and-drain strategy**.

This is what Gartner likes to refer to as a [bi-modal infrastructure](https://www.gartner.com/it-glossary/bimodal), meaning we’re operating in these 2 modes simultaneously. Some apps are operating and delivering in an ITIL fashion, some apps are operating and delivering in a DevOps fashion. I think this becomes a realistic transition, versus saying, “An existing business that’s in flight, we’re simply going to turn it off and tomorrow turn on as DevOps.” It doesn’t really work versus a much more gradual transition where we say, “Net new. We bleed over, existing stuff continues to run, business as usual.”

I think that’s a useful way to think about this transition from ITIL over to DevOps.

I hope you found this video useful. Please check out [hashicorp.com](https://www.hashicorp.com/). We have a lot more resources on topics related to this as well as to DevOps and the tools in general. And if you’re thinking about your managed services strategy and how this might apply to you, please [reach out](https://www.hashicorp.com/go/contact-sales) and partner with us as you go through that conversation.