liatrio

SOLUTIONS BRIEF

Understanding Usage Patterns for Terraform & Azure Resource Manager (ARM) Templates

TERRAFORM & AZURE RESOURCE MANAGER (ARM) TEMPLATES

This brief outlines Liatrio's point of view and comparison between Hashicorp's Terraform and Microsoft's Azure Resource Manager (ARM) templates.

Problem Statement

With the ever-expanding need for dynamic, reliable, and repeatable infrastructure, traditionally manual approaches to creation and management cannot scale. A way to define Infrastructure-as-Code (IaC), which enables automation, is necessary to meet demands.

Terraform and ARM templates are distinct tools for defining and orchestrating IaC.

Our Background

This recommendation is grounded in our expertise and real-world experience with multiple clients with various delivery patterns. Liatrio engineers bring a diverse background of hands-on demonstrations of value through reference implementations as well as cross-team collaboration. In this brief, we will discuss both the challenges and benefits of using Terraform and ARM templates in the context of an enterprise.

Top Value Points

While there is much that can be said for both tools, Liatrio believes these points to be the main differentiators between the two solutions:

Terraform

- Universally extendable through providers, to include the capability for any infrastructure, service, and application configuration.
- Can handle complex order-of-operations and composability of individual resources and encapsulated modules.
- Broad open source community for hundreds of providers and thousands of modules that are typically leveraged in an enterprise environment, with easy-to-find documentation and examples.
- Microsoft supports and collaborates directly with Hashicorp on building and maintaining related Terraform providers.
- There are over 50 million installations of the main Azure provider.
- Tracks state of real-world resources, which makes Day 2 and onward operations easier, and more powerful.

```
resource "azurerm_key_vault" "this" {
  location                        = var.location
  name                            = local.key_vault_name
  resource_group_name             = var.resource_group
  sku_name                        = var.sku
  tenant_id                       = var.tenant_id
  enable_rbac_authorization       = var.enable_rbac
  purge_protection_enabled        = var.purge_protection_enabled
  enabled_for_template_deployment = var.enabled_for_template_deployment

  lifecycle {
    ignore_changes = [
      tags
    ]
  }

  dynamic "network_acls" {
    for_each = length(var.ip_rules) > 0 || length(var.virtual_network_subnet_ids) > 0 ? [1] : []

    content {
      bypass                     = var.network_acls_bypass
      default_action             = "Deny"
      ip_rules                   = var.ip_rules
      virtual_network_subnet_ids = var.virtual_network_subnet_ids
    }
  }
}
```

ARM Templates

- Lockstep support for all Azure resources, since ARM templates are a native representation of the cloud services.
- Provides the ability to both authors from scratch, or start from existing services or infrastructure via an export.
- Provides more effective validation and a preview of changes for Azure resources specifically, when compared to general IaC tooling.

A Closer Look

Terraform and ARM templates can seem very similar on the surface. Both self-described as tools to facilitate infrastructure as code; allowing users to level up their DevOps practices with improved speed and confidence. Both have ways to handle reuse with parameterization and forms of modularization, the ability to preview changes before applying them, and are generally friendly to automation and source code management (SCM) tools. With all that these tools appear to have in common, it may give the impression that there is not much to strongly debate. But as stated prior in the "Top Value Points", there are critical differentiators that we believe make the choice definitive.

First is the universal extensibility of Terraform, achieved through the design of providers. The ability to define virtually any category of infrastructure, service, application, or other configuration cannot be understated when compared to ARM templates that are solely for the definition of Azure services. That capability translates into:

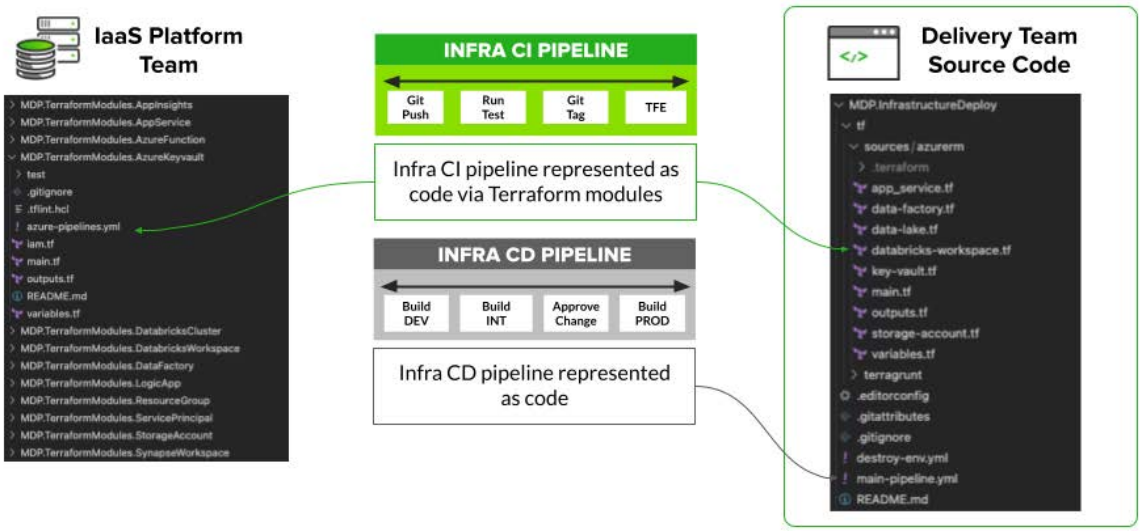
- A single investment in learning and building around a new tool. This does not mean that Terraform is a Golden Hammer. But it is best-in-class for orchestrating many common infrastructure platforms and services, not just Azure. Enterprise-size organizations are extremely likely to have other areas that could benefit from Terraform as well. This single investment will pay long-term dividends, as new teams and problem areas look to improve with DevOps best practices. The new in-house knowledge and practices with Terraform will prove quickly adaptable.
- The community of users and support around Terraform is naturally larger than that for ARM templates. This is because it can be used in infinitely more cases, and the basics of Terraform translate to all. This larger community will mean more sources of open knowledge, such as for example usage patterns or common troubleshooting issues that others have already solved. As well, finding candidates with Terraform experience during hiring will be much more common.
- The extensibility means that an organization has more opportunities for customization or innovation, as needed. Most Terraform providers welcome open source contributions for fixes or features, and an organization can wholly author a new provider to solve their specific use-case.

Another important distinction is the concept of tracking the state of real-world resources. State is a core component of Terraform. Every resource, whether a virtual machine, storage bucket, or IAM policy, has all of its defining properties recorded by Terraform. Each time Terraform is "applied", it checks what the current IaC defines, as well as refreshes its view of the real world resources based on the previously recorded state information. ARM templates do not have functionality that resembles state tracking. Each "application" of an ARM template takes a simpler approach of making idempotent requests for each defined resource to its respective REST API endpoint. The ability to keep state is what powers Terraform to be a better tool beyond Day 1 operations. The easiest example is the destruction of resources when they are no longer needed. When a resource is no longer present in the IaC, Terraform still knows that resource exists in the real world, and will handle destroying its real-world counterpart. ARM templates cannot handle this type of operation by default. If a resource is no longer defined in the ARM template, ARM has no way of knowing that a real-world resource needs to be destroyed. This leaves real infrastructure in a dangling condition, over utilizing and in a public cloud scenario incurring unneeded costs. This same scenario can even occur with simple renames of infrastructure, if that rename would constitute a new instance of a resource.

Recommendation

Liatrio recommends that Terraform should be the first choice for Infrastructure-as-Code (IaC) tooling. As such, Liatrio recommends Terraform over ARM templates.

Terraform provides a more consistent code-first (engineering) approach to modern source control, versioning, testing, and deployment pipeline practices. This helps align organizations on DevOps principles.



Terraform provides cross-platform and cross-service functionality. Whether your infrastructure is running on-prem or in the various public cloud providers, Terraform is portable and reusable, ultimately making Terraform the best return on investment. Learning and adopting new tools and approaches can be costly, but once a team has learned Terraform the modules and approaches can be applied to new and future platforms, tools, and configurations.

We believe that Terraform lays the foundation and skill set for attracting new talent and for building a strong engineering culture. This will make hiring additional engineers easier and more appealing.

In addition to producing Infrastructure as a Service (IaaS), Terraform can easily be used for Platform as a Service (PaaS) offerings as well as “Day 2” operations, e.g. maintenance, upgrades, and new features.

Other Considerations

What if my organization is all in on Microsoft's Azure, does it make sense to use ARM templates over Terraform?

Liatrio believes Terraform should still be the first choice for IaC. Terraform is a single, well-constructed orchestration tool with a vast library of plug-and-play integrations. These integrations include things outside of infrastructure, ranging from configuration of SaaS Data offerings like Databricks, to self-hosted Identity Providers like Keycloak. While ARM templates may be an okay choice to meet the business pressure of today, Terraform will give your team the ability to grow and adapt your automation to support future needs with a tool you are already familiar with.

What if I adopt Terraform and encounter a gap in a Terraform provider functionality?

From our experience, it is rare to encounter a situation where a particular provider from a vendor or open-source community would not exist. We have encountered providers that exist, but maybe missing a particular resource or resource configuration. In those instances, Liatrio partners with our customer to contribute back to the provider, implementing the missing feature. This usually means creating a pull request to an open source repository, but may involve other steps depending on the provider's contribution model.

I already have existing Azure resources that were created manually, why not export them as ARM templates?

It is not uncommon to have Azure resources that were created manually due to timeline pressure or the initial scope of a project, and as that project matures the need to have those resources be handled as IaC becomes apparent. ARM templates have the unique ability to be exported directly from Azure resources through the Portal or CLI. This can seem like a quick answer or stop-gap since you can have your existing infrastructure as code in a moment. We believe that this route should be avoided for both organizational and technical reasons.

Organizationally, IaC can only be as successful as the discipline of delivery teams to use it. If a project has been delivered on manually created infrastructure thus far, it is unlikely to have all of the surrounding automation built around the newly exported ARM templates, or skill sets to maintain the ARM templates. This will lead to constantly

out-of-date IaC as manual changes will inevitably still occur, and frustrations around misaligned or misguided expectations of IaC.

On a technical side, we still believe that most organizations will ultimately arrive at the conclusion that Terraform is the better tool for their needs, as they discover the limitations of ARM templates over time. In this case, you will still have to learn and build processes around Terraform, but with the additional sunk cost of time and effort put into existing IaC with ARM templates.

ABOUT LIATRIO

Author

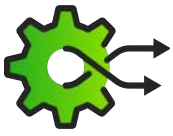
Eric Chapman is a Technical Principal at Liatrio, helping our clients deliver software faster and safer. With a background in software development, he has spent the past 20 years architecting, building, and enabling software systems for large complex organizations ranging

We Help Your Business Deliver Faster And Safer

Liatrio guides complex organizations through their digital transformation journey, enabling them to deliver value faster and safer with Enterprise DevOps and Cloud Native delivery.

We partner with large, successful enterprises to drive systemic change and transformation that helps you scale your entire approach to software delivery.

Our Core Capabilities



Enterprise DevOps Transformation	Cloud Native Delivery	Modern Platform Engineering	DevSecOps
Accelerate business results and scale your organization with a lean, value-driven approach to software delivery and IT operations.	Empower your teams to build scalable apps in dynamic environments and make high-impact changes frequently and predictably with little toil.	Reliable applications are built on modern self-service platforms that reduce engineering friction.	Speed of delivery while always staying safe and secure in an automated way. Remove untimely, manual, last gate siloed approvals and validations.

liatrio

DEVOPS AND CLOUD TRANSFORMATIONS

LIATRIO.COM