Assessing Terraform for Provisioning Cloud Infrastructure

Published 13 November 2017 - ID G00328206 - 32 min read

By Analysts Gregg Siegfried

Supporting Key Initiative is Accelerating Infrastructure Innovation and Agility

Technical professionals seek consistency when setting up automated provisioning in AWS, GCP and Microsoft Azure. Terraform provides a common workflow to automate provisioning, which facilitates multicloud and hybrid IT. This note assesses Terraform's features and automated provisioning capabilities.

Overview

Key Findings

- Terraform fundamentally enables multicloud IT by establishing a uniform language, state management model and life cycle for software-defined infrastructure that spans multiple cloud service providers.
- Terraform's service provider coverage is extensive and offers a consistent interface to a variety of resource types. The open architecture allows any resource-based control plane to be encapsulated.
- Terraform goes beyond the management of container infrastructure by also supporting interaction with container schedulers.
- Terraform's support for infrastructure as code (IaC), immutable infrastructure and modular provisioning make it ideal for optimizing workflow within agile or DevOps-aware environments.

Recommendations

When provisioning infrastructure with Terraform, technical professionals charged with infrastructure agility should:

- Isolate dependencies in configuration assets by creating modules to layer infrastructure. Study contributions in the module registry for guidelines, best practices and prebuilt examples.

- Protect the Terraform state by persisting in remote shared state storage (called "backends"), such as HashiCorp Consul or Amazon S3. While most important for production deployment, its use adds little or no overhead and is inexpensive insurance.
- Prepare for multicloud IT by provisioning their infrastructure with Terraform, even when a single provider is currently in use. Cloud infrastructure is rarely static, and this approach facilitates common scenarios, such as provisioning platform resources, along with monitoring resources.
- Include collaboration and governance in their provisioning pipeline by deploying Terraform Enterprise in organizations that require these capabilities. Although not universally applicable, the enterprise features substantially broaden the audience for the tool.

Analysis

Managing hybrid IT workloads based on services delivered by the enterprise, as well as external cloud providers, has become the next infrastructure and operations (I&O) challenge, but creating cloud resources consistently is not always straightforward. As multicloud workload deployment becomes routine, along with increasing adoption of platform as a service (PaaS) and SaaS, the number of service provider platforms into which I&O teams must provision resources is growing seemingly without bound.

HashiCorp's Terraform is designed to simplify the process of managing software-defined infrastructure, particularly when multiple infrastructure providers are in use. This research:

- Examines the Terraform product set
- Discusses its architecture, capabilities and toolchain context
- Provides guidance and examples for its deployment

> **Although most cloud providers include native tools to automate provisioning, Terraform can automate simple to complex provisioning tasks in a uniform, platform-neutral syntax. Service coverage, which can be inconsistent, is evolving rapidly. Suitability for agile infrastructure use cases is emphasized by CI/CD pipeline readiness and deep container support.**

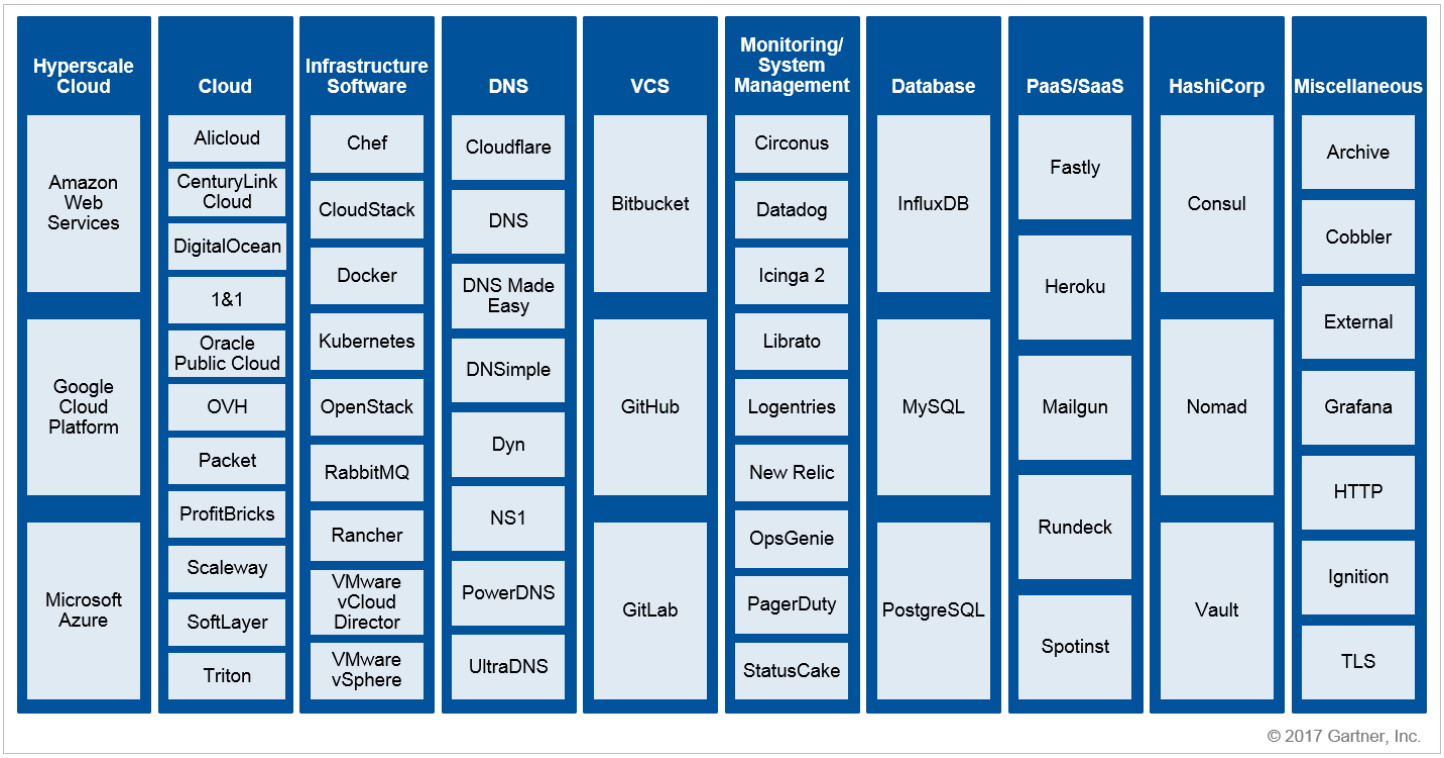
Most cloud service providers have addressed provisioning challenges by creating and distributing their own mechanisms to create and manage resources within their platforms. These commonly

include a web-based management console, as well as an application programming interface (API) and corresponding client-side toolkit. Larger providers have also created their own orchestration tools and domain-specific languages (DSLs), such as the Amazon Web Services (AWS) CloudFormation. Some organizations favor these native tools due to their pedigree. Others, out of concern for lock-in, capabilities or learning curve, would prefer a multiplatform provisioning tool.

As the number of disparate environments that I&O — no matter how agile — must manage, the challenge of meeting business needs increases. This can place infrastructure management organizations back in the familiar position of acting as "gatekeeper," attempting to constrain the business in order to maintain control. A mechanism that unifies the provisioning workflow across the universe of software-defined architecture providers would accelerate an organization's transformation into a service broker. For more information, see "Adapt IT to Become the Broker of Cloud Services." (https://www.gartner.com/document/code/328145?ref=grbody&refval=3823221)

Terraform is emerging as one of the first platform- and environment-agnostic automated provisioning tools. In addition, when the definition of "infrastructure" has become as malleable as it is, Terraform has proven to be equally malleable. It has long supported AWS, Google Cloud Platform (GCP) and Microsoft Azure, as well as many other providers, as illustrated in Figure 1. New and updated resources and providers are being added continually.

Figure 1. Terraform Provider Plugins



TLS = Transport Security Layer; VCS = version control system

Source: Gartner (November 2017)

The first generation of cloud management platforms (CMPs) attempted a similar approach — that of acting as an intermediary between I&O and multiple cloud infrastructure service providers. These products have largely failed to achieve success in the marketplace. It will remain to be seen if Terraform's simple, modular and composable architecture, along with its emphasis on utility, will prevent a similar fate.

Assembling Infrastructure as Code

IaC initiatives arise as part of a DevOps transformation or other agility-focused infrastructure automation process. In this context, Terraform may be used as:

- A rapid prototyping tool, as the resources necessary for a given solution are discovered and refined.
- For assembling modules to comprise a catalog of infrastructure resources available to application teams. This can help to reduce variations across similar deployments.
- As part of a continuous deployment pipeline, where infrastructure for an application release is created as part of the development/testing/deployment processes.

The tool itself will support both traditional and agile infrastructure management models. However, to obtain the greatest benefits, Gartner recommends the adoption of the IaC paradigm that Terraform facilitates as part of a larger, agile infrastructure or DevOps transformation.

The Normalization of Multicloud Workloads

Gartner characterizes a number of different deployment models as "multicloud." The first, and most straightforward, is defined in "Decision Point for Selecting Single or Multicloud Workload Deployment Models" (https://www.gartner.com/document/code/322970?ref=grbody&refval=3823221) as follows:

> **A multicloud workload deployment is achieved when components of the same solutions are deployed across multiple public clouds, or when different workloads are deployed across different clouds but still need to communicate, interconnect or otherwise be managed uniformly.**

This type of multicloud workload would include resources primarily from providers in the hyperscale cloud, cloud and infrastructure software categories in Figure 1. Gartner often recommends this as the default approach for organizations developing cloud migration strategies.

Another type of multicloud workload — which Gartner calls composite multicloud — includes resources provisioned from providers in any or all of the categories available. Until recently, this workload style may have been the exception, but it is increasingly becoming the rule.

> **The benefits of being able to use a common syntax, workflow and state management are substantial in both reducing the learning curve and increasing uniformity, productivity and velocity.**

It is not obvious which is the cause and which is the effect here:

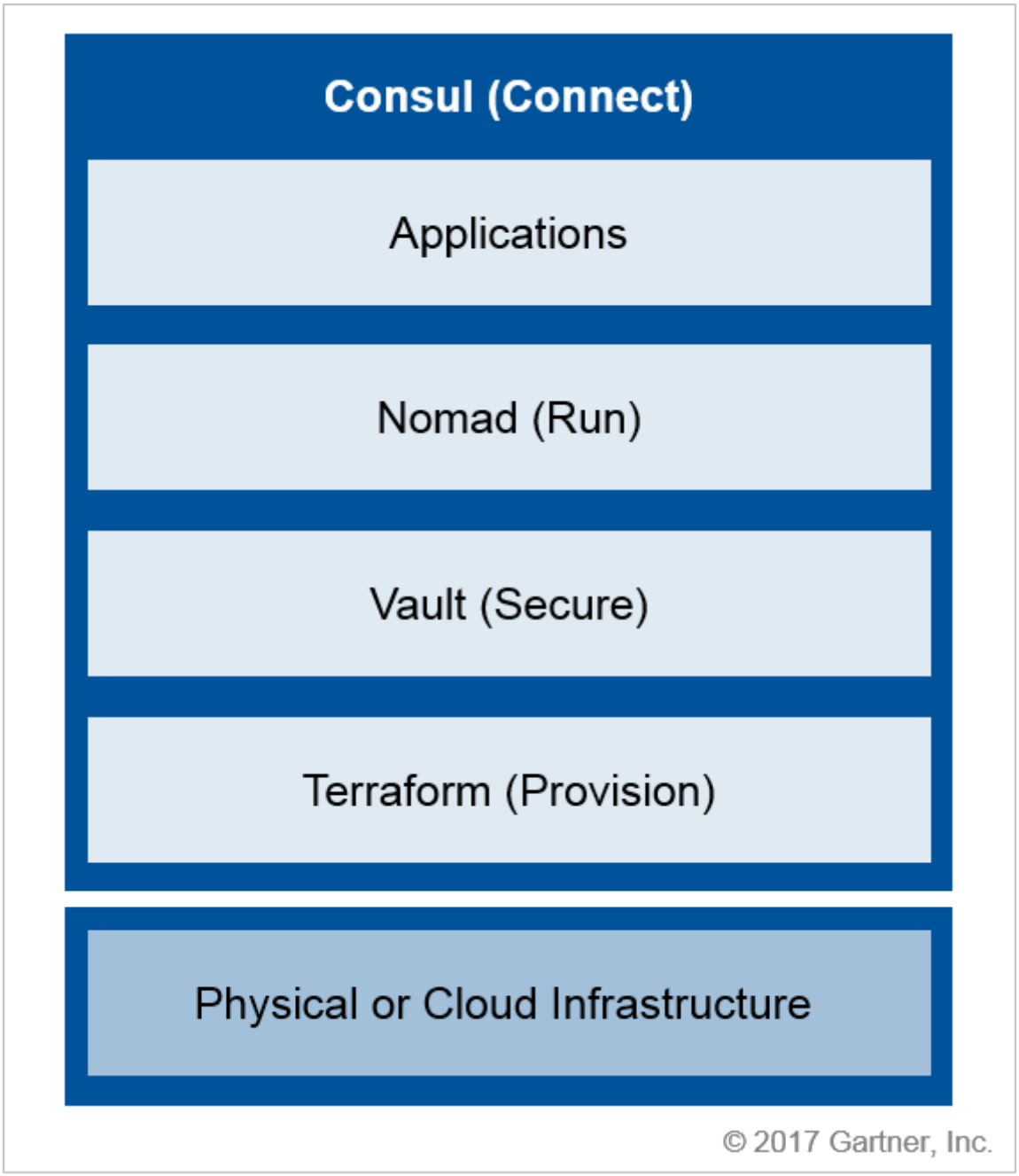
- Are composite multicloud workloads driving the adoption of tools like Terraform
- Or, has Terraform made such workloads so practical that organizations are designing their cloud infrastructure to support them?

The former is more likely, of course, although in either case, Gartner client inquiry data indicates that interest in multicloud deployment has doubled over the last 12 months.

What Is Terraform?

Although this research is focused on Terraform, understanding HashiCorp's product portfolio helps to place Terraform in context. The HashiCorp ethos is: "Consistent workflows to provision, secure, connect and run any infrastructure for any application." Each verb in that sentence refers to an individual tool, as illustrated in Figure 2.

Figure 2. HashiCorp Portfolio



Source: Gartner (November 2017)

Use of one HashiCorp tool does not require the use of others, although organizations may be able to fill additional gaps in their DevOps toolchain by doing so. The architectural cohesion and similarity in operation facilitates the adoption of additional tools as the organization's capabilities evolve.

Terraform is available in three packaging tiers:

- An open-source version that is freely available (referred to hereafter as "Terraform Open Source").
- A commercial edition called "Terraform Enterprise Pro" that extends Terraform Open Source with features that support team collaboration and includes product support.

- A commercial edition called "Terraform Enterprise Premium" that extends Terraform Enterprise Pro with features that support governance and includes product support.

For comparison, Table 1 enumerates the features available in all three tiers. This document will subsequently refer to the commercial editions (both Pro and Premium) collectively as Terraform Enterprise.

Table 1: Terraform Editions

↓	Open Source ↓	Enterprise Pro ↓	Enterprise Premium ↓
Features	■ IaC ■ Multiprovider ■ Pluggable Provider Architecture ■ Public Module Registry	All Open Source and the Following: ■ Workplace Management ■ VCS Connection ■ Remote Workflows ■ Rollback ■ Audit Logs ■ Pipeline Support	All Enterprise Pro and the Following: ■ Multifactor Authentication ■ Approval ■ Policy as Code (Sentinel) ■ SAML Support
Support	None	Silver (9x5 With SLA)	Gold (24/7 with SLA)
Install Option	On-Premises	SaaS	Private
SAML = Security Assertion Markup Language			

Source: Gartner (November 2017)

Most software tools today come with their own nomenclature, and Terraform is no exception. Relevant terminology is included in Table 2.

Table 2: Terraform Terminology

Terminology ↓	Description ↓

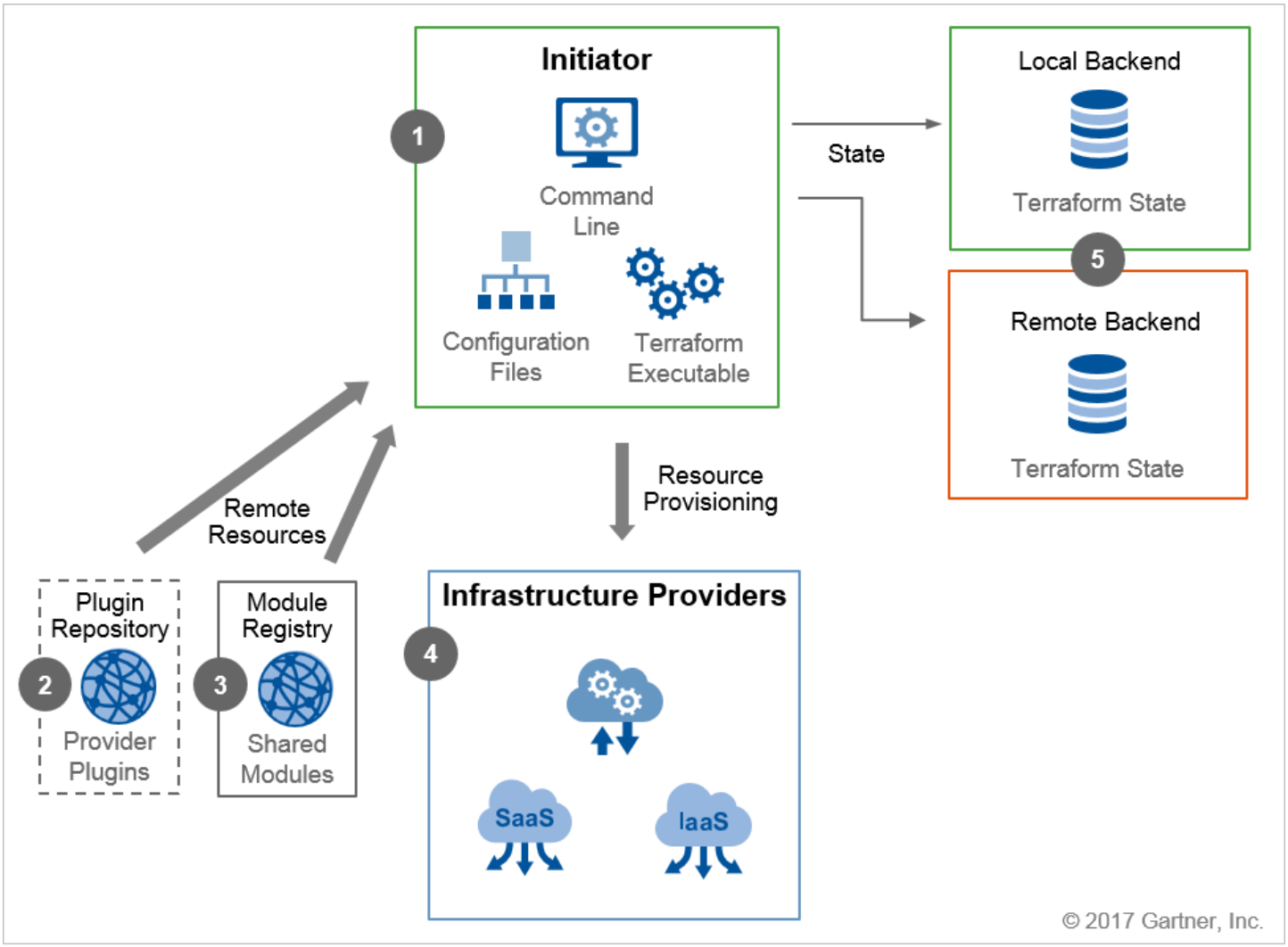
Terminology ↓	Description ↓
Configuration	The "source code" that defines, or models, the desired infrastructure. It can be specified in either the declarative HashiCorp Configuration Language (HCL) or JavaScript Object Notation (JSON)
Module	A way to describe complex infrastructure technologies in codified form.
Provider	Can refer either to an infrastructure provider or to a Terraform plugin that implements resource life cycle and data source methods.
Provisioner	A mechanism that enables scripted configuration as part of resource creation or destruction.
State	Terraform's representation of managed infrastructure. Used to map actual deployed resources to configuration.
Workspace	A unit of organization for the Terraform state that allows one configuration to be applied to different resource sets. It is not supported by all backends.
Backend	A mechanism that defines how and where the state is loaded and stored between executions. By default, the "local" backend stores the state in a directory along with the configuration. Collaboration and security can be facilitated by managing the state using a nonlocal backend.
Resource	A component of infrastructure.
Data Source	Allows data to be fetched or computed for use elsewhere in the Terraform configuration. Use of data sources allows a Terraform configuration to build on information either defined outside of Terraform or defined by another separate Terraform configuration.

Source: Gartner (November 2017)

Terraform Open Source

Figure 3 represents the Terraform Open Source tool's visible components.

Figure 3. Terraform Open Source Theory of Operation



Source: Gartner (November 2017)

The numbers below refer to the callouts in Figure 3.

Initiator (1)

The initiator is the workstation or server that includes the Terraform executable and the HCL-based configuration files, and it is the endpoint from which the provisioning activities are launched.

Provider Plugin Repository (2)

HashiCorp provides a public repository of provider plugins, which are machine-executable modules that are retrieved on an as-referenced basis. Provider plugins encapsulate an infrastructure provider's API into an interface consumable by Terraform. These executable modules are released and maintained by HashiCorp, and their retrieval by reference is largely transparent to users of the product. Due to the number and size of these plugins, distributing all of them with every Terraform program update represents a potentially unnecessary storage burden, given the relatively small subset of plugins used by individual Terraform users. Unless users are creating their own provider plugins or compiling them from source code, the storage, distribution and retrieval of provider plugins

is managed entirely by HashiCorp on the user's behalf. The first step in the Terraform execution workflow ensures that all referenced provider plugins and modules are available.

As of this writing, there are on the order of 70 Terraform provider plugins available. Figure 1 lists the majority of them. Each provider supports multiple resources and/or data sources. Because the core product remains open source and the provider interface is documented, organizations can develop additional provider plugins as necessary.

Terraform Module Registry (3)

The Terraform Module Registry (https://registry.terraform.io/) , introduced in September 2017, is a new, public repository for community-contributed Terraform modules.

The Terraform Module Registry is designed to be a community of, and mechanism for, the sharing and distribution of reusable, HCL-based Terraform resources. While many of these modules have been developed and are maintained by HashiCorp or infrastructure service providers, it is possible for anyone with a GitHub account and a Terraform module to share to participate in the registry as well.

Infrastructure Providers (4)

These are the infrastructure as a service (IaaS), PaaS, SaaS and other service providers from which infrastructure can be provisioned. A single Terraform execution plan can reference multiple providers. For example, users can create and configure a Google Compute Engine (GCE) n1-standard-8 compute instance, register a DNS record for it in NS1, and establish monitoring for the instance in Datadog in a single plan/apply.

Backends (5)

A Terraform configuration can define where to store its internal representation of the provisioned infrastructure, for safe-keeping, collaboration or both. If no explicit backend is defined, the "local" backend will maintain this state in the same directory as the configuration itself, in a terraform.tfstate file. Gartner recommends the use of remote backends for long-lived state, or for resources whose provisioning may be shared among multiple operators.

Terraform Platform Support

The Terraform Open Source executable is available across a wide variety of operating systems and processors (see Table 3).

Table 3: Supported Terraform Platforms

Platform ↓	x86 ↓	AMD64 ↓	ARM ↓
FreeBSD	✔	✔	✔

Platform ↓	x86 ↓	AMD64 ↓	ARM ↓
Linux	✔	✔	✔
macOS		✔	
OpenBSD	✔	✔	
Solaris		✔	
Windows	✔	✔	

Source: Gartner (November 2017)

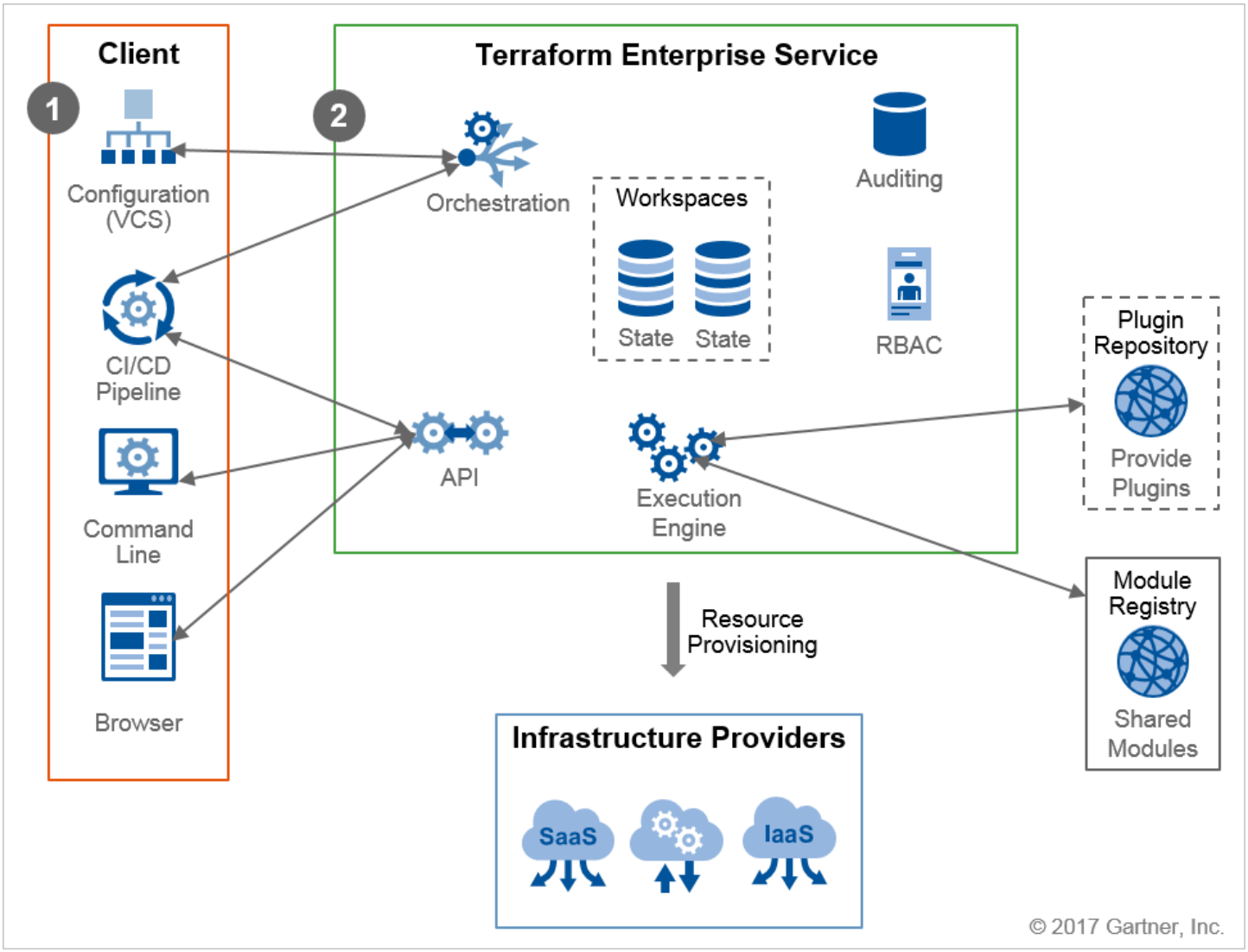
Terraform Enterprise

Unlike the Terraform Open Source version, which is largely stand-alone and usually deployed locally, Terraform Enterprise can be consumed as a:

- SaaS product
- Private installation within a public cloud platform (Enterprise Premium)
- Virtual appliance for installation on private infrastructure (Enterprise Premium)

Terraform Enterprise includes browser and API-based interfaces, in addition to the command line. Figure 4 illustrates the product in context.

Figure 4. Terraform Enterprise Theory of Operation



CI/CD = continuous integration and continuous delivery

Source: Gartner (November 2017)

The components that differ materially from Terraform Open Source are identified with numbered callouts. The unnumbered elements in Figure 4 were identified in Figure 3.

Client (1)

Terraform Enterprise can be accessed from a number of client interfaces. The command line-based client interface via the Terraform command mirrors the open-source product, although execution takes place on the centralized platform. More commonly, Terraform Enterprise will be accessed via the API or an event, such as a commit hook from a VCS. This allows Terraform execution to be plumbed into continuous integration pipelines. Finally, there is a browser-based presentation layer interface as well.

Terraform Enterprise Service (2)

This is the SaaS-based execution environment that manages provisioning activities, state, access control and orchestration. With some subscription levels, organizations may be able to deploy their own private instances of this service. Outside of the governance and collaboration features, the provider interfaces and state management operate as they do in the open-source product.

Terraform Enterprise uses the same terminology as the open-source version, with a few additions (see Table 4).

Table 4: Terraform Enterprise Terminology

Terminology ↓	Description ↓
Organization	A set of users and resources for sharing and policy purposes
Run	A sequence of "plan" and "apply" operations — analogous to a provisioning job

Source: Gartner (November 2017)

Significant efficiencies for larger teams or more complex environments are available in the enterprise delivery model. Some of these efficiencies can reduce the amount of disparate Terraform configuration that has to be written. The workspace abstraction supports the creation of multiple sets of resources — for example, a "test" environment, a "staging" environment and a "preproduction" environment — using the same configuration by varying the workspace. Many Terraform Open Source users have simulated workspaces by adopting a physical separation model using different directory hierarchies. A private instance of the module registry also will be available. This enables organizations to express their own standards, best practices and policies for infrastructure provisioning, as well as to make them available to others. This capability has been announced, but it is not yet available.

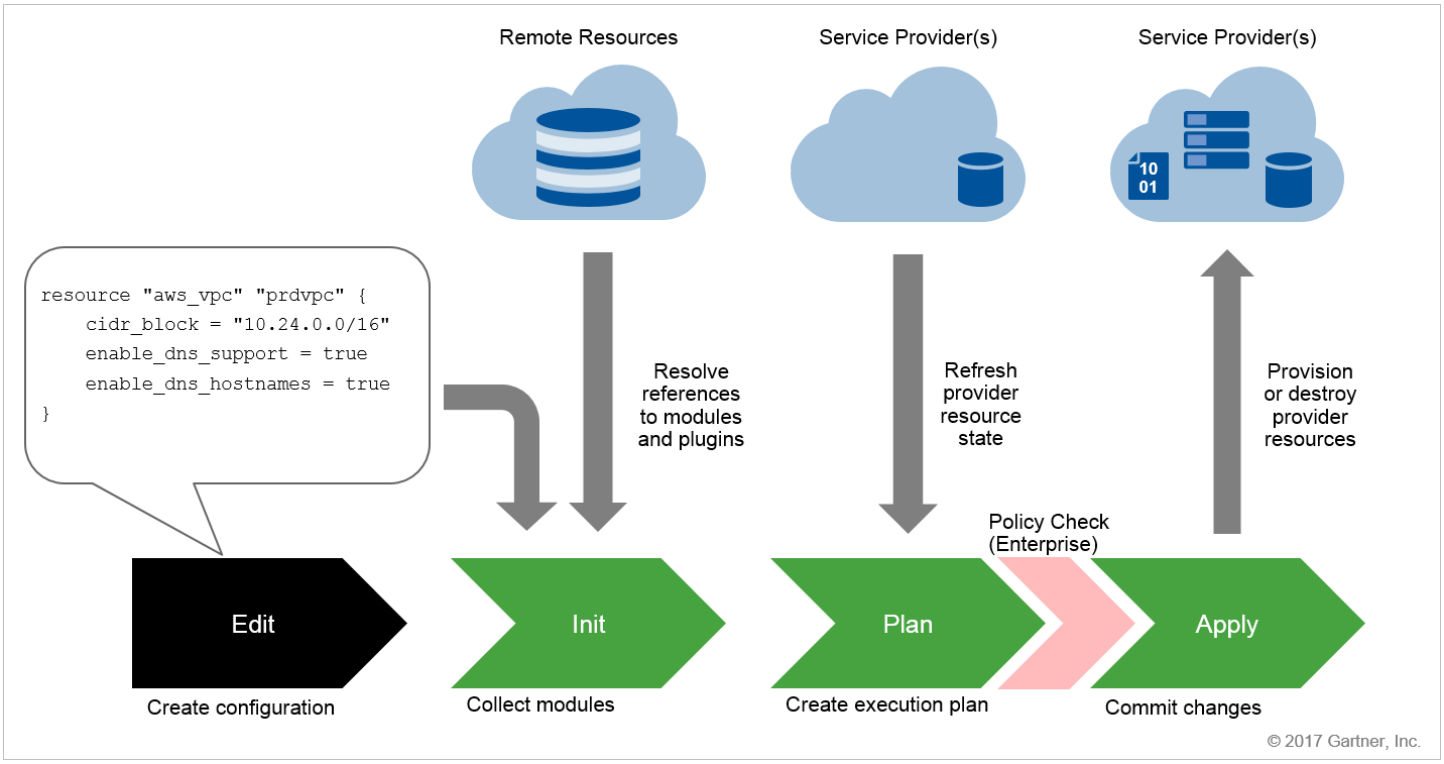
Terraform Enterprise includes a Policy as Code capability called "Sentinel." With this, organizations can develop codified business rules and compliance policies that are enforced on every Terraform run. Some examples include:

- Do not allow resources to be provisioned without tags.
- Only allow test environments in the us-west-2 region.
- Do not allow security groups to open public ingress to port 80.

Terraform Provisioning Workflow

Figure 5 shows the sequence used to provision a virtual private cloud (VPC) with Terraform in AWS.

Figure 5. Terraform Provisioning Workflow



Source: Gartner (November 2017)

- Edit — This is not an actual Terraform command, but this step represents the creation of the configuration files. These should be named with a ".tf" extension.
- Init — This command resolves any external references that the configuration makes to modules and provider plugins. This, as well as subsequent commands, operate on all configuration files found in the current directory.
- Plan — This command refreshes the external provider state and builds an execution plan to resolve any differences between the current and desired state.
- Policy check — In Terraform Enterprise, this is the point that the Sentinel Policy as Code capability can be leveraged to ensure that the provisioning activity is consistent with established governance policies.
- Apply — This command commits the changes to the external provider(s) and stores the resulting state in the current directory or the configured backend.

Additional commands are available, but these represent the most common interactive workflows. Errors can be generated in any execution phase.

As technical professionals become familiar with the available resources to be provisioned, be cognizant of the following:

In many cases, changes to a resource or set of resources can be applied directly, meaning that the attributes being changed are mutable. However:

- Changes to other attributes may not be mutable, and applying this type of change will result in the destruction and recreation of the resources in question.
- Some resources may not be mutable at all, and any attempt to change them via Terraform will cause the resource to be destroyed and recreated.

In properly managed infrastructure, whether a resource is altered in place or destroyed and recreated should have limited impact on the desired state. Changes being made outside of Terraform that are not represented in the resource creation workflow would normally be considered an error.

Use Cases

Automating infrastructure provisioning and configuration is one of the steps on the journey to continuous delivery (see "Solution Path for Achieving Continuous Delivery With Agile and DevOps" (https://www.gartner.com/document/code/342688?ref=grbody&refval=3823221)) and is fundamental for any organization that is practicing DevOps. While tools to automate cross-platform configuration activities, such as continuous configuration automation (CCA), have existed for some time, provisioning has been such a platform-dependent process that native tools are generally the default.

Infrastructure as Code

Gartner has previously (see "Using DevOps Tools for Infrastructure Automation" (https://www.gartner.com/document/code/326084?ref=grbody&refval=3823221)) described the term "infrastructure as code" as:

- The creation, provisioning and configuration of software-defined compute, network and storage infrastructure as source code
- The application of the process and disciplines traditionally associated with software development to manage IT infrastructure

Terraform is archetypal, cloud-native IaC, in that it supports the creation and provisioning of many types of resources across an array of providers, and it has the ability to compose highly complex, multiplatform and multicloud solutions.

Immutable Infrastructure

Immutable infrastructure is an architectural pattern in which the system and application infrastructure, once instantiated, is never updated in place. Instead, when changes are required, the infrastructure is simply replaced.

While it is most commonly practiced with cloud-based infrastructure, the immutable approach is a management paradigm, not a technology capability. Any environment that enables a fully automated and repeatable software-defined deployment process can support the immutable pattern. These include the use of images or Amazon Machine Images (AMIs) and CCA tool-based customization, as well as containerization.

Terraform's Plan/Apply cycle and its ability to operate in a pipeline provides near-native support for workflows that desire an immutable infrastructure approach, whether it includes cloud platforms, hypervisors, containers or a combination of them.

Additional, tailored use cases are illustrated in the Examples section.

Strengths

- **Readability:** HashiCorp Configuration Language (HCL) resembles Universal Configuration Language (https://github.com/vstakhov/libucl) (UCL), supports commenting, is more readable than JSON and is fully compatible. This reduces the learning curve and contributes a self-documenting character.
- **Modularity:** The ability to structure Terraform configuration into modules encourages a layering and encapsulation approach that not only supports reuse, but also assists in isolating dependencies. The Terraform Module Registry adds a source for expert-written and provider-written modules, as well as a sharing community.
- **Versatility:** The recently introduced shared-state backends support a voluntarily collaborative workspace model that works well for individuals and for small teams. Terraform Enterprise supports collaboration and coordination capabilities that are suitable for very large teams.
- **Community support:** The number and types of provider plugins provided by HashiCorp and the Terraform user community have increased steadily since Terraform was introduced. The plugin architecture enables proprietary providers to be created and incorporated into the local organization's workflow, in the event a standard provider is not available.
- **Team support:** Terraform Enterprise includes the same provisioning capabilities, along with the guardrails necessary to support the needs of larger organizations and more complex deployment pipelines.
- **Platform availability:** Due largely to the Go programming language's inherent cross-compilation and build capabilities, executables for Terraform Open Source are available for a large number of platforms and operating systems, as listed in Table 3.

Weaknesses

- **Backward compatibility:** As a relatively new tool that operates at the nexus of the rapidly changing cloud provisioning landscape, Terraform's rate of change is understandable. There have been periodic breaking changes, but these have been documented and staged so as to minimize impact.
- **Scope:** The Terraform Open Source project and community are well-governed by HashiCorp. However, as much of the provider support is contributed by the extended community, keeping pace with changes taking place at all of the supported service providers is a formidable task. The recent unbundling of provider plugins reduces pressure on the core executable by enabling the independent release of each component. This new process is still being proven.
- **Durability:** The state management in Terraform Open Source is elementary and does not prevent operators from accidentally destroying or recreating infrastructure resources. More advanced state management and coordination are available in Terraform Enterprise.
- **Packaging:** To date, HashiCorp has tried several different consumption models and price structures for Terraform Enterprise. While the feature set and capabilities have continued to improve, potential commercial edition clients at any tier should ensure that they know what the current offering includes. Table 1 enumerates the current differences.

Guidance

Managing tool sprawl is a real concern for technical professionals looking to optimize their maturity and throughput in the face of constant change. Terraform provides an agile-friendly provisioning tool that works with multiple cloud services, as well as on-premises infrastructure. Consider the following as you evaluate or adopt Terraform for provisioning within your organization:

- **Adopt and reinforce the IaC paradigm by using provisioning automation tools like Terraform rather than a cloud provider's management console to provision resources.** Most infrastructure providers, particularly hyperscale cloud providers like AWS, GCP and Microsoft Azure, offer sophisticated and full-featured browser-based management consoles. These are excellent for exploration, learning and high-level monitoring, but any resources provisioned or changed using the console are effectively uncontrolled. Gartner recommends using an automated mechanism like AWS CloudFormation, Azure Resource Manager or Terraform to provision managed cloud assets.
- **Store Terraform assets in a VCS, just as you would for any source code.** This strategy not only safeguards the configuration assets, but also facilitates launching configuration jobs from continuous integration pipelines. Terraform Enterprise enforces the use of a VCS by associating each workspace with a VCS repository.
- **Protect your provider credentials by not storing them directly in the Terraform configuration.** Accidentally committing a secret or access token to a public-facing VCS like GitHub can be costly and disruptive. Various strategies are available to avoid credential leakage, including:

- Use of the terraform.tfvars file
- Environment variables
- Provider-specific mechanisms such as Amazon Elastic Compute Cloud (EC2) roles and GCP application default credentials

- **Configure compute instances using CCA or configuration management tools rather than Terraform provisioners.** Aside from ad hoc, prototyping or simple Stage 1 configuration, a CCA tool supports profile reuse, better control and more sophisticated configuration activities. Terraform's destroy-time provisioners are useful for cleanup and deregistration steps. For more information on this, see the More On Terraform Provisioners section.
- **Search the Module Registry before building your own from scratch.** Although only recently introduced, the Terraform Module Registry already includes over 100 valuable, reusable modules, many contributed by the infrastructure providers themselves. As with any community resource, particularly one that creates billable resources within a cloud provider, they should be inspected before use. As a corollary to this, consider broadening the community by contributing some of your own modules to the registry as well.
- **Avoid copying terraform.tfstate files.** The open-source version's remote back-end storage capability and the enhanced state storage available in Terraform Enterprise is intended to represent the single source of truth (aside from the provisioned resources themselves, of course). Copying the state manually and processing it in two different places will have potentially undefined results.
- **Evaluate the adoption of Terraform Enterprise if collaboration or governance requirements are substantial.** The same command line workflow is available for scripting and orchestration, yet execution and state are centralized. The more robust workspace mechanism better supports multiple, similar environments as well. Organizations considering adopting Terraform Enterprise can evaluate the open-source version and be confident that the provisioned resources will be identical to those that would be created using Terraform Enterprise.
- **Import natively provisioned resources when Terraform provider coverage does not include all options required.** The Terraform import command is an evolving capability that enables Terraform to manage resources that were initially created using another mechanism. When a gap exists between native cloud provider resources and the corresponding Terraform provider's ability to provision them, import can sometimes bridge that gap.
- **Leverage Terraform in your CI/CD pipeline to provision test environments as part of validation.** On the journey to continuous deployment, provisioning, configuring and testing infrastructure as part

of application validation will build confidence in your procedures, tools and test suites, as well as in the applications themselves.

The Details

Examples

The sections below represent examples of resources that can be provisioned with Terraform. The examples are designed to illustrate certain characteristics and might not represent desirable architecture patterns for real workloads. As they are provisioning rather than policy or orchestration use cases, it would be possible to use either the open-source or enterprise editions to create them.

These examples emphasize the complexity of infrastructure solutions that Terraform can compose. The tool also functions well as a substitute for the resource creation that often takes place in a cloud provider's management portal. Gartner recommends developing a personal or organizational catalog of Terraform configuration files that supports provisioning an individual resource or sets of related resources as a controlled substitute for ad hoc provisioning via a management portal.

Finally, it should be understood that some resources depend naturally on others — for example, storage volumes, Internet Protocol (IP) addresses and "association" activities. Assume that any dependent resources not listed explicitly would be created using Terraform as well.

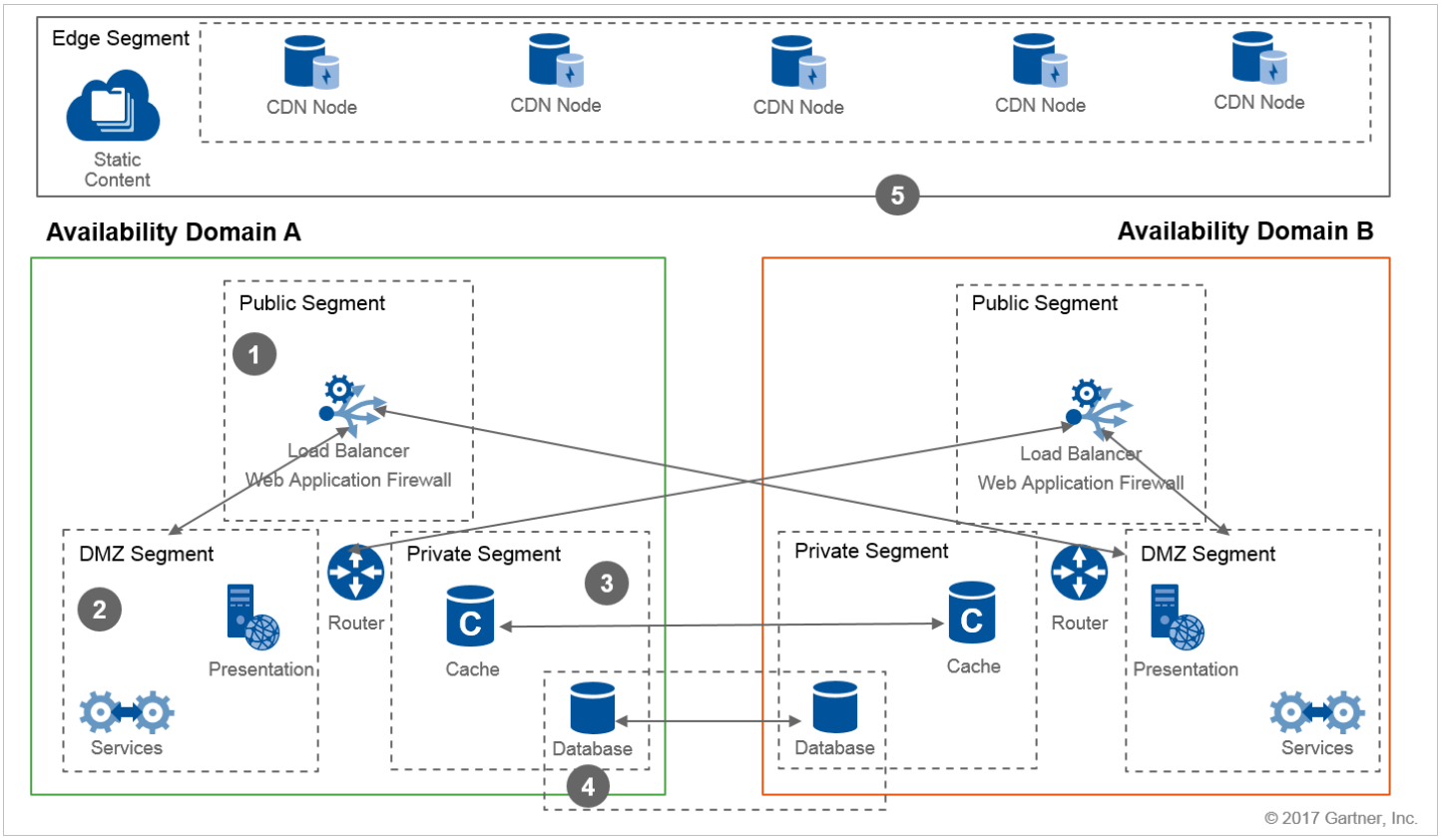
This section illustrates two examples:

1. Multitier application infrastructure — A common architecture for web application delivery or database-driven sites
2. Multicloud Kubernetes workload — A set of container orchestration systems deployed using multiple provider platforms

Example Configuration No. 1

This is a common deployment model for web-enabled applications and services. As illustrated, this infrastructure is intended to be deployed within a single cloud provider and distributed across two or more availability domains, such as AWS or Microsoft Azure availability zones or Google zones, for high availability. Within any of the hyperscale providers, the assumption is that all resources would be provisioned within a virtual network or virtual private cloud equivalent. For purposes of the discussion, the virtual networks and segments, complete with bidirectional internet access, are assumed to exist at time of resource provisioning. Any application-specific network provisioning would be limited to flow enablement or segmentation implemented via security groups (see Figure 6).

Figure 6. Multitier Application Infrastructure



CDN = content delivery network; DMZ = demilitarized zone

Source: Gartner (November 2017)

Resource Types in the Solution

The numbers in the list refer to the callouts in Figure 6. More information about the listed resource types are available in the Terraform documentation (https://www.terraform.io/docs/index.html) .

1. The publicly facing infrastructure is limited to traffic management in the form of a load balancer. Each load balancer can distribute traffic to multiple availability domains. Whether the provisioned resource comprises a single load balancer with multiple domain segments, or multiple load balancers, is dependent on cloud provider capabilities. Examples of available Terraform load balancer resources include:

   - AWS aws_alb
   - Microsoft azurerm_lb
   - Google google_compute_backend_service

2. Two types of application servers comprise the resources in the DMZ segments. The "service" application represents a set of API services and the "presentation" application represents a web-

based UI, perhaps using the API included in the service application. These are traditional compute instances, such as:

- AWS aws_instance
- Google google_compute_instance
- Oracle opc_compute_instance

3. The private segments include two kinds of state — the database tier, as well as a caching layer, to reduce latency and traffic impact on the database:
   - AWS aws_elasticache_cluster and aws_rds_cluster
   - Microsoft azurerm_redis_cache and azurerm_sql_elasticpool
4. In the example solution, the cache and the database are illustrated as being synchronized across availability zones (AZs). This may or may not be possible with all providers.
5. The edge segment represents a CDN and the backing store used to load the CDN edge nodes with static content. While most hyperscale cloud providers include CDN resources, there are also third-party CDNs that can be provisioned with Terraform:
   - AWS aws_cloudfront_distribution
   - Microsoft azurerm_cdn_profile
   - Fastly fastly_service_v1

DNS for the platform is not pictured. It is a common practice to establish multiple external-facing DNS provider relationships to reduce the risk of being taken out of service in the event the DNS provider fails or experiences a distributed denial of service (DDoS) event. In addition to cloud platform-native DNS resources, Terraform supports myriad third-party DNS providers.

This example illustrates how a single-provider deployment can evolve into a composite multicloud provisioning problem very easily. While some hyperscale cloud providers support all of the infrastructure resources included in the example, this is not universally the case. Even if the majority of the delivery platform is contained within one provider, an optimized environment will potentially include the following resources from other providers:

- CDN resources
- DNS resources

- Monitoring resources

Organizations managing platforms like these will benefit from the flexibility of being able to add or substitute resources from multiple providers using a common language and workflow.

Example Configuration No. 2

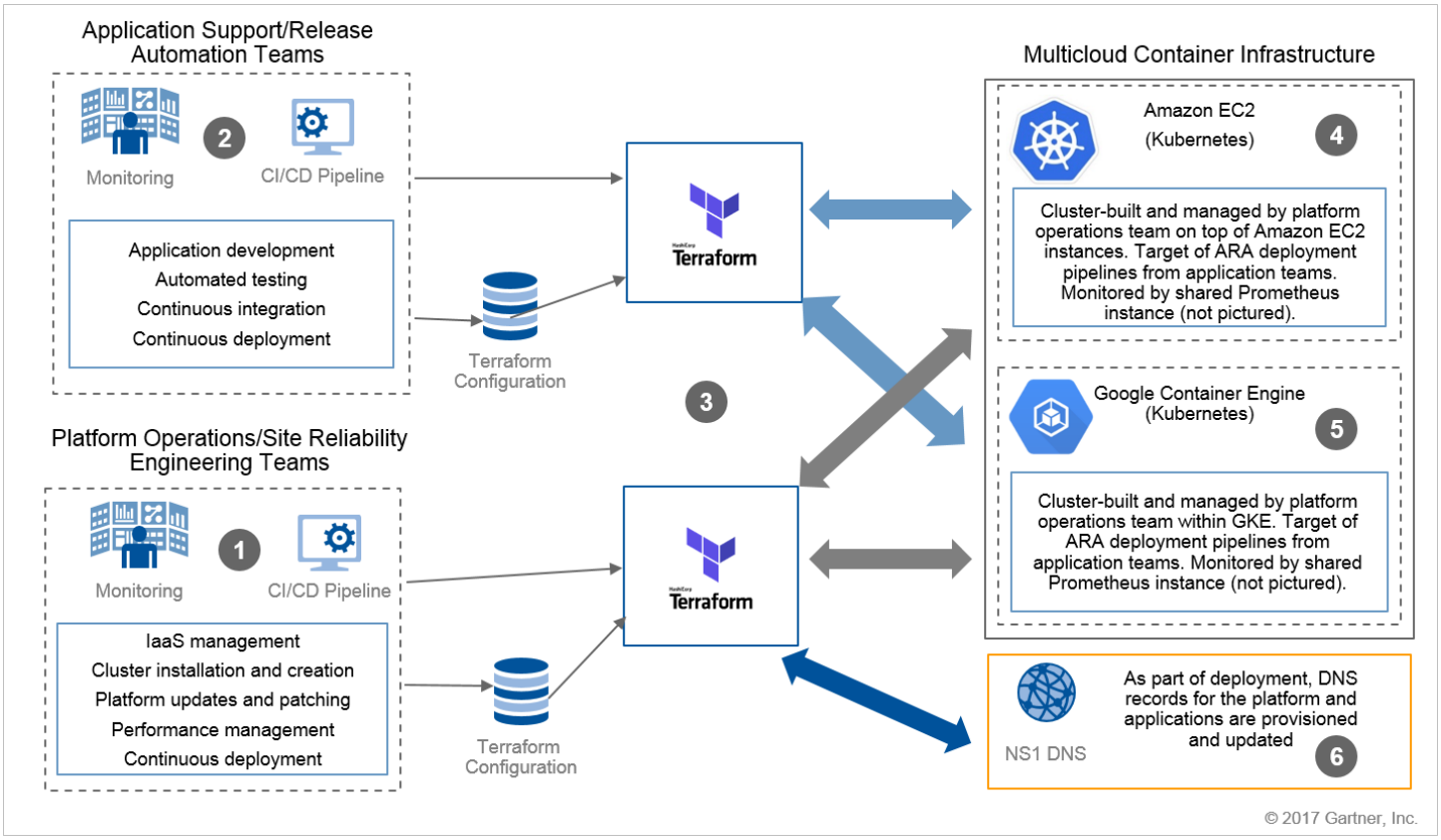
Rather than enumerating resource types, this example illustrates Terraform's multilevel support for containerized workloads.

Kubernetes is an open-source container orchestration and scheduling framework. For more information and background on Kubernetes, see:

- "Assessing Container Management Frameworks Running On-Premises and in the Cloud" (https://www.gartner.com/document/code/323943?ref=grbody&refval=3823221)
- "Container Services in the Public Cloud" (https://www.gartner.com/document/code/323006?ref=grbody&refval=3823221)

Kubernetes can be installed and configured on multiple types of base infrastructure (including bare-metal servers, on top of a hypervisor like VMware's vSphere or on a set of compute instances at a cloud provider). In addition, both Google and Microsoft offer "managed," single-tenant Kubernetes clusters in the form of Google Container Engine (GKE) and Azure Container Service (AKS), respectively. Once a Kubernetes cluster has been installed and configured — often a complex task in and of itself — containerized application workloads must be applied to the cluster for it to perform useful work (see Figure 7).

Figure 7. Multicloud Kubernetes Workloads



ARA = application release automation

Source: Gartner (November 2017)

Terraform supports the provisioning of both the Kubernetes environment itself and the application workloads that will be executed on the cluster. Applications tend to undergo revision more frequently than the container orchestration system, so the creation and updating of applications on the cluster should be the more common provisioning activity of the two.

As with previous examples, the numbered list below corresponds to the callouts in Figure 7:

1. The platform operations and Site Reliability Engineering (SRE) teams include system administration and engineering personnel responsible for installing and managing Kubernetes clusters. In this case, they have chosen two platforms to host them, covered in No. 4 and No. 5. Once constructed, this team is responsible for cluster performance and health management, which may involve adding or subtracting the number of worker nodes in the clusters, as well as patching and updating the Kubernetes software itself. In this example, the platform operations team is also responsible for managing DNS.

2. The application support and release automation teams manage the testing, packaging and deployment of application workloads onto the Kubernetes clusters. Terraform's Kubernetes provider is used to deploy and update services. As application containers are built, deployed to test environments and validated, they may be promoted to production clusters automatically, or

scheduled and approved for update during specific windows. Kubernetes has its own mechanisms for rolling application updates, and like the rest of Kubernetes, these mechanisms are evolving very rapidly.

3. With the open-source product, each of the support teams may have their own "provisioning workstation," and separate credential and source repositories that support the various levels of access they have to the cloud platforms and resources. The need for this sort of role-based access control is also a good example of an environment that could benefit from Terraform Enterprise.

4. Amazon EC2 cluster — This is a Kubernetes installation from scratch. Terraform is used to create the base infrastructure for the cluster, including compute instances, VPCs and/or network segments, security groups, and block storage. Additional configuration management software will likely be used to perform the installation and configuration of Kubernetes itself. This could be Red Hat Ansible, Pivotal's Kubo or CoreOS Matchbox. The CoreOS Tectonic Kubernetes distribution can often be installed completely with Terraform.

5. GKE — Terraform can provision and configure this type of managed Kubernetes environment directly and completely, using the google_container_cluster and google_container_node_pool resources. Once up and running, it can be managed by the platform operations team and deployed to the stand-alone Kubernetes cluster by the application support team.

6. In an externally facing application, data at one or more DNS providers is used to provide access to the deployments, replication controllers and services within the illustrated Kubernetes clusters. As pictured in Figure 7, the Terraform NS1 DNS provider is used by the platform operations and SRE teams to map external names to Kubernetes workloads.

This example was multicloud from the beginning. It illustrates first using Terraform to provision compute resources and container schedulers in two different cloud platforms by a platform operations team. Once the platform is created, the application teams also use Terraform to provision and manage container-based workloads on the two Kubernetes clusters.

Ideally, the application teams practice continuous deployment (CD), and orchestrate regular application updates by invoking Terraform from their CI/CD orchestration platform.

Organizations deploying container orchestration and scheduling software locally or to any cloud platform will benefit by leveraging common tools to provision both platform and application resources. These organizations may require the additional access control and auditing available in the enterprise product.

More on Terraform Provisioners

Terraform supports the use of what are called "provisioners" to execute scripts as part of compute instance creation or destruction. While the provisioner can be a useful ad hoc capability, the above

guidance recommends the use of CCA or CM tools instead, particularly during creation. This section includes the rationale for that recommendation.

During discovery, testing and debugging, the Terraform provisioner capability is an excellent mechanism for operators to use to obtain rapid feedback and to test resource configuration or modules. Feel free to use provisioners in this capacity and, in some cases, to act as a "bootstrap" for higher-level tooling if a user-data-like feature is not available.

Most cloud providers include support for script execution during instance creation. In Amazon EC2, the mechanism that supports script execution during instance creation is called user data. GCP calls this mechanism a startup script and Microsoft Azure calls it a startup task. Terraform supports — and Gartner recommends — using these cloud-native instance initialization (such as cloud-init for Linux) capabilities as part of compute instance creation. The advantages of this over the Terraform provisioner approach include:

- It does not require the Terraform host to connect to the instance. In many cases, the host from which resource provisioning activities are being launched does not have or need direct, Secure Shell (SSH) or Windows Remote Management (WinRM) access to the instances being provisioned. To use Terraform provisioners, such access is required.
- Terraform provisioners limit concurrency. One benefit of Terraform's execution planning step is its ability to concurrently provision independent resources. Use of Terraform provisioning introduces a dependency on the host executing the "Terraform apply" to each compute instance being created.
- The cloud-native instance initialization capabilities often have a greater ability to interoperate with other features of the cloud platform, such as tagging. For example, it is common practice to apply metadata tags to resources as they are provisioned. The cloud-native instance initialization script can examine this metadata and customize the instance based on metadata tags.
- Gartner recommends the use of CCA tools to perform operating system provisioning and instance configuration as a best practice. Creation-time provisioning, using cloud-native instance initialization or a Terraform provisioner, should be extremely modest and consist largely of installing the CCA agent, if necessary, and handing the instance over to the CCA master server.

Recommended by the Author

Solution Path for Achieving Continuous Delivery With Agile and DevOps (https://www.gartner.com/document/3823865?ref=ddrec&refval=3823221)

Using DevOps Tools for Infrastructure Automation (https://www.gartner.com/document/3745722?ref=ddrec&refval=3823221)

Decision Point for Selecting Single or Multicloud Workload Deployment Models (https://www.gartner.com/document/3681027?ref=ddrec&refval=3823221)

A Guidance Framework for Architecting Portable Cloud and Multicloud Applications (https://www.gartner.com/document/3540043?ref=ddrec&refval=3823221)

Adapting IT to Become the Broker of Cloud Services (https://www.gartner.com/document/3760566?ref=ddrec&refval=3823221)

The Automation Architect: How to Become, and Succeed as, This Emerging IT Specialist (https://www.gartner.com/document/3787268?ref=ddrec&refval=3823221)

Assessing Container Management Frameworks Running On-Premises and in the Cloud (https://www.gartner.com/document/3728217?ref=ddrec&refval=3823221)

Container Services in the Public Cloud (https://www.gartner.com/document/3666417?ref=ddrec&refval=3823221)

Decision Point for Selecting a Public Cloud Management Strategy (https://www.gartner.com/document/3655817?ref=ddrec&refval=3823221)

Recommended For You

Assessing Amazon Web Services, Google Cloud Platform and Microsoft Azure for Your GDPR Readiness (https://www.gartner.com/document/3875768?ref=ddrec&refval=3823221)

Decision Point for Choosing a Cloud Migration Strategy for Applications (https://www.gartner.com/document/3893681?ref=ddrec&refval=3823221)

In-Depth Assessment of Oracle Cloud Infrastructure IaaS, July 2018 (https://www.gartner.com/document/3884666?ref=ddrec&refval=3823221)

In-Depth Assessment of Microsoft Azure IaaS, July 2018 (https://www.gartner.com/document/3884485?ref=ddrec&refval=3823221)

Assessing the Strengths and Weaknesses of High-Value IaaS and PaaS Multicloud Use Cases (https://www.gartner.com/document/3846474?ref=ddrec&refval=3823221)

research is produced independently by its research organization without input or influence from any third party. For further information, see "Guiding Principles on Independence and Objectivity."