# **Bicep vs Terraform**

**What is Infrastructure as Code?**

Before comparing Terraform and Bicep, we need to understand the concept of Infrastructure as Code (IaC). What is it exactly, and why is it so important for modern infrastructure deployments? Infrastructure as Code is relatively intuitive in regards to the meaning of the term. You are describing your infrastructure in code blocks.

The true birth of Infrastructure as Code was driven by the move to public cloud infrastructure. In public cloud environments such as Amazon AWS, Google GCP, and Microsoft Azure, you can’t physically get your hands on the infrastructure. However, public cloud providers do provide “console-based” interfaces that allow manually spinning up VMs, networks, and other necessary resources.

The true power of provisioning resources in the public cloud comes from interacting with the infrastructure programmatically. Public cloud environments such as Microsoft Azure expose REST APIs that allow performing actions programmatically instead of using the GUI console.

With this capability, organizations can describe their infrastructure in code, perform the actions, create resources, and configure them in an automated fashion. One of the powerful concepts of Infrastructure as Code is the *declarative*nature of the code. It means you can declare what you want the infrastructure to “look” like, and the code makes that happen.

The sheer number of resources and configurations maintained by organizations deploying modern applications can be staggering with modern cloud computing. As a result, infrastructure may need to be provisioned, scaled up or down, or even destroyed. Without an IaC tool in place, it becomes increasingly difficult to maintain the scale, agility, and consistency needed across all infrastructure landscapes.

**What are the benefits?**

The benefits of IaC include, but are not limited to, the following:

* Increased speed of application deployment
* Fewer human-error mistakes
* Improved consistency of infrastructure
* Helps to reduce or eliminate configuration drift
* Reduces the cost of development and cloud infrastructure

**Declarative vs. Imperative**

## What is Hashicorp Terraform?

One of the most popular Infrastructure as Code languages today is Hashicorp Terraform. Thousands of organizations and DevOps professionals use Terraform to capture their infrastructure in code form effectively. So, what is Terraform? It is an open-source Infrastructure as Code tool that allows managing on-premises and hundreds of cloud services using declarative code.

It allows infrastructure to be expressed as code. Terraform uses Hashicorp Configuration Language (HCL) to write declarative configuration files. HCL is a very simple, human-readable

### Terraform concepts and terms

There are several concepts and terms to be aware of when working with Terraform. Note the following:

**Resources** – Resources are one of the most important components of the Terraform language. A resource is an entity that describes an infrastructure object. However, objects can include virtual networks, DNS records, compute instances, and anything that can be described by Terraform.

**Providers** – These are the “plugins” Terraform uses to interact with different types of infrastructure, from on-premises technologies to cloud environments. In your Terraform configuration, you declare the providers required. When you initialize Terraform, it then pulls down the needed providers for the specific code you are using.

**Data sources** – The data sources construct is information defined outside of Terraform and are defined by another Terraform configuration or modified by functions

**Modules** – Modules contain multiple resources that are related and commonly used together. A module is a collection of .tf or tf.json files kept in the same directory.

**State** – The Terraform state information stores the map of your real-world resources. It stores bindings between objects in a remote system and resource instances declared in your configuration. Terraform checks the state data to make decisions on what resources need added or changed.

### Terraform workflow

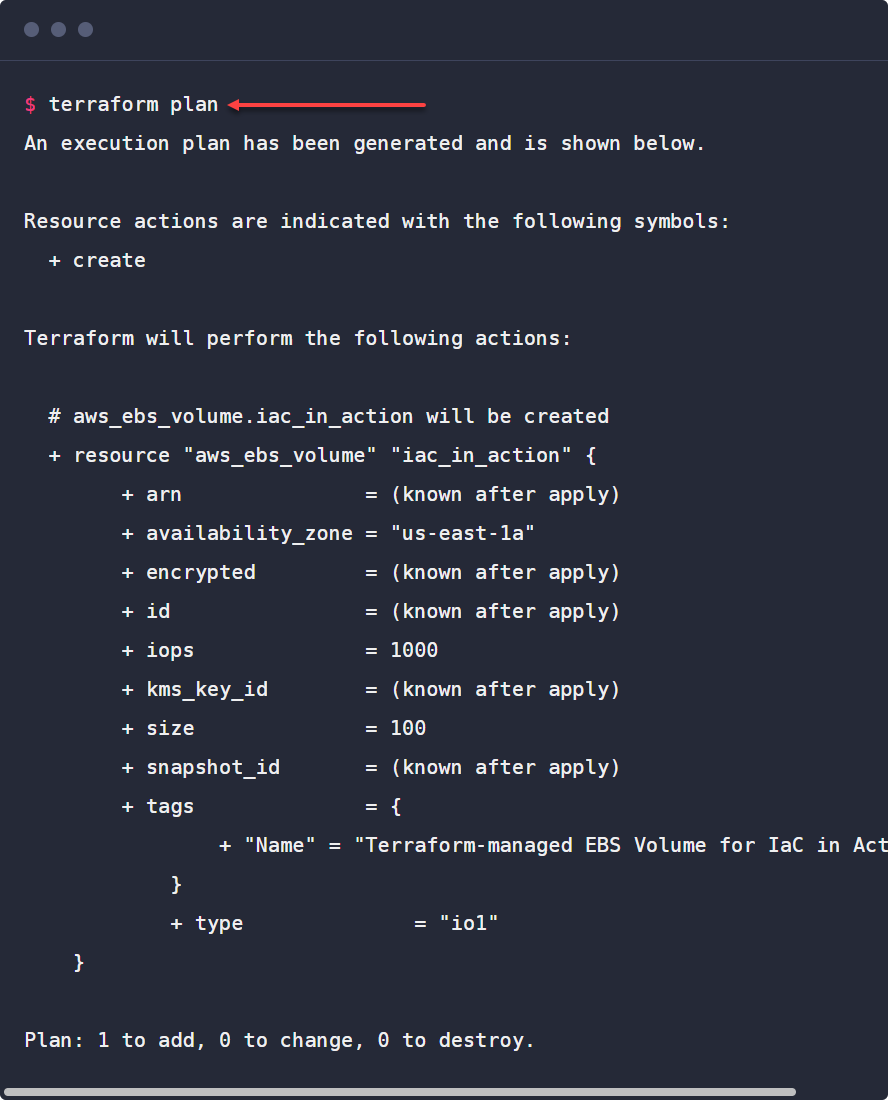
Terraform uses a specific workflow that involves the steps of write, plan, and apply. Note the following high-level look at the Terraform workflow.

Let’s look at each of these steps in the Terraform workflow and see how they relate to writing IaC.

* **write** – In the HCL, you describe the resources you want to create, change, or configure using blocks, arguments, and expressions. Below is an example of Terraform code blocks interacting with Amazon AWS.

[](https://www.starwindsoftware.com/blog/wp-content/uploads/2021/12/text-description-automatically-generated.png)

* **plan** – You can think of the Terraform plan parameter as a “what if” statement of sorts. With Terraform plan, you can check the configuration plan for configuration matches and verify the expectation of what you think the code should do before provisioning or changing your infrastructure. Entering the “terraform plan” command displays the changes that will be made in the environment.

[](https://www.starwindsoftware.com/blog/wp-content/uploads/2021/12/text-description-automatically-generated-1.png)

* **apply** – Running the Terraform apply command applies the changes to your environment. It puts in place the resources or required changes described in the configuration language.

**Other Terraform commands**

While the “terraform plan” and “terraform apply” commands are two of the most common Terraform commands used when working with infrastructure deployment and configuration, other useful and necessary commands are used with Terraform. What are some of these?

* **terraform init** – Using the “terraform init” command automatically pulls down the terraform provider needed to work with the specific type of infrastructure or public cloud environment you are working with. Using Terraform with Amazon AWS will pull down the relevant AWS modules to work with your infrastructure.
* **terraform refresh** – The refresh command is used to reconcile real-world infrastructure with the Terraform code
* **terraform destroy** – The destroy parameter allows DevOps engineers to destroy resources defined in the terraform plan

## What is Bicep?

Microsoft’s Bicep is a newcomer on the block as a declarative Infrastructure as Code language. However, it is gaining momentum, especially in the Microsoft Azure environment, as currently, it is written for Azure environments only. Bicep is fundamentally a **transpiler**. What is a transpiler?

A transpiler is a source-to-source compiler. In other words, a transpiler takes the source code written in one language and converts it to another language at the same level of abstraction. What does Bicep transpile? Bicep takes Azure Resource Manager templates and covers them into the Bicep domain-specific language, using declarative syntax to deploy Azure resources.

The official [Bicep Github page](https://github.com/Azure/bicep) describes Bicep more as a revision of ARM templates:

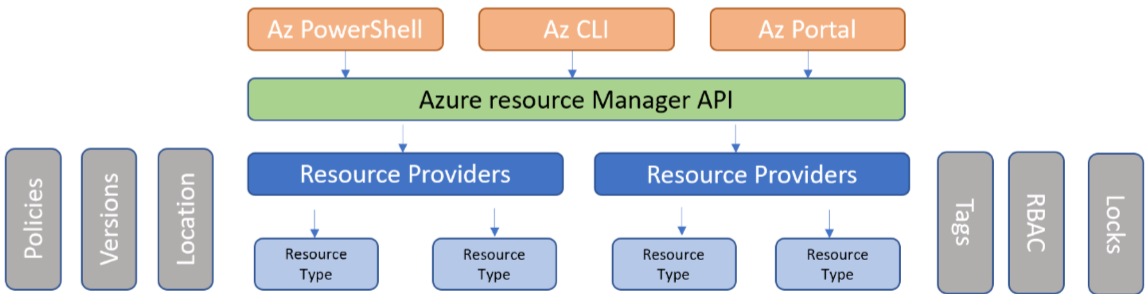
Bicep is more of a revision to the existing ARM template language rather than an entirely new language. While most of the syntax has been changed, the core functionality of ARM templates and the runtime remains the same. You have the same template functions, same resource declarations, etc.

### With Bicep there is not state data to maintain

One of the characteristics of Azure Bicep, compared to Terraform and other IaC tools, is Bicep does not require you to keep up with state data. So, there is no state file as there is with Terraform. Instead, the state used for Bicep code is automatically stored inside Azure.

### Bicep interacts with Azure Resource Manager

**Azure Resource Manager (ARM)** is the deployment and management service for Azure. ARM allows the creating, updating, and deletion of resources within your Microsoft Azure account. Any request sent to Azure using the Azure tools, SDKs, or APIs traverses the Azure Resource Manager. It then authenticates the request and authorizes the action, and sends it to the requested service.

[](https://www.starwindsoftware.com/blog/wp-content/uploads/2021/12/diagram-description-automatically-generated-3.png)

**Bicep vs. Terraform**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Bicep** | **Terraform** | **Description** |
| Can it be used across multiple platforms? | No | Yes | Terraform has support for over 1000 providers, meaning you can work across a multitude of cloud providers and infrastructure environments |
| Is there state data that must be maintained? | No | Yes | Bicep automatically stores state data inside Azure. Terraform stores state locally by default or in a shared cloud location when working as a team |
| Support in modern IDE such as VS Code? | Yes | Yes | Modern IDEs like Visual Studio Code have strong support for both Terraform and Bicep. There are currently more plugins in VS Code available for Terraform since it has been around longer |
| Native Azure services integration | Yes | No | Bicep, being a Microsoft creation specifically for Azure, has the advantage here. However, Terraform has strong Azure support as well. Most likely, you will be able to use either tool successfully for DevOps tasks in Azure |
| Microsoft support | Yes | No | Bicep is a Microsoft product and is fully supported. Terraform is a third-party tool that Hashicorp supports |
| Modularized construct | Yes | Yes | Bicep and Terraform both have the concept of modules |
| CI/CD pipeline support? | Yes | Yes | Both Bicep and Terraform can be used for CI/CD pipeline processes |
| Strong community support? | Yes | Yes | Bicep lags behind here since it is the newcomer. However, the Bicep howto’s and posts are starting to pop up across the Internet. Terraform has very strong support across the community. Most likely, you can find a code snippet that does what you need it to using Terraform. |
| Declarative IaC language? | Yes | Yes | Both provide declarative configuration languages |
| Easy to learn? | Yes | Yes | Both are comparable in the ease of learning the languages and understanding them |

**Wrapping Up**

The modern Infrastructure as Code (IaC) approach to provisioning, configuring, and changing infrastructure allows organizations to control their infrastructure in a much more robust way. In addition, as businesses move to a DevOps style approach and transition to continuous integration and development (CI/CD) processes, Infrastructure as Code allows making the infrastructure a part of the development pipeline.

Terraform is a powerful IaC tool that many organizations are using today. It provides a robust feature set and support for 1000+ providers. In addition, it works across all the major cloud providers and on-premises technologies such as VMware vSphere. If you want a tool that is flexible across many public clouds and infrastructure environments, Terraform is a great choice.

IN the realm of Microsoft Azure, Bicep is gaining momentum as a great choice for those who have been working with ARM templates already. Bicep is the evolution of the ARM template language and transpiles ARM JSON into Bicep domain-specific language. Therefore, transitioning to Bicep may make sense for organizations heavily invested in Microsoft Azure and ARM templates.

**Terraform is the better choice for the all-around tool since it is not just Azure-specific but provides IaC across a multitude of platforms.**