**An understanding of Terraform architecture**

Figure 1.12 – Terraform architecture


As you can see in the preceding Terraform architecture, there are two key components on which Terraform's workings depend: Terraform Core and Terraform plugins. Terraform Core uses **Remote Procedure Calls** (**RPCs**) to communicate with Terraform plugins and offers multiple ways to discover and load plugins to use. Terraform plugins expose an implementation for a specific service, such as AWS, or a provisioner, and so on.

**Terraform Core**

Terraform Core is a statically compiled binary written in the Go programming language. It uses RPCs to communicate with Terraform plugins and offers multiple ways to discover and load plugins for use. The compiled binary is the Terraform CLI. If you're interested in learning more about this, you should start your journey from the Terraform CLI, which is the only entry point. The code is open source and hosted at [github.com/hashicorp/Terraform](http://github.com/hashicorp/Terraform).

The responsibilities of Terraform Core are as follows:

* IaC: Reading and interpolating configuration files and modules
* Resource state management
* Resource graph construction
* Plan execution
* Communication with plugins via RPC

**Terraform plugins**

Terraform plugins are written in the Go programming language and are executable binaries that get invoked by Terraform Core via RPCs. Each plugin exposes an implementation for a specific service, such as AWS, or a provisioner, such as Bash. All providers and provisioners are plugins that are defined in the Terraform configuration file. Both are executed as separate processes and communicate with the main Terraform binary via an RPC interface. Terraform has many built-in provisioners, while providers are added dynamically as and when required. Terraform Core provides a high-level framework that abstracts away the details of plugin discovery and RPC communication, so that developers do not need to manage either.

Terraform plugins are responsible for the domain-specific implementation of their type.

The responsibilities of provider plugins are as follows:

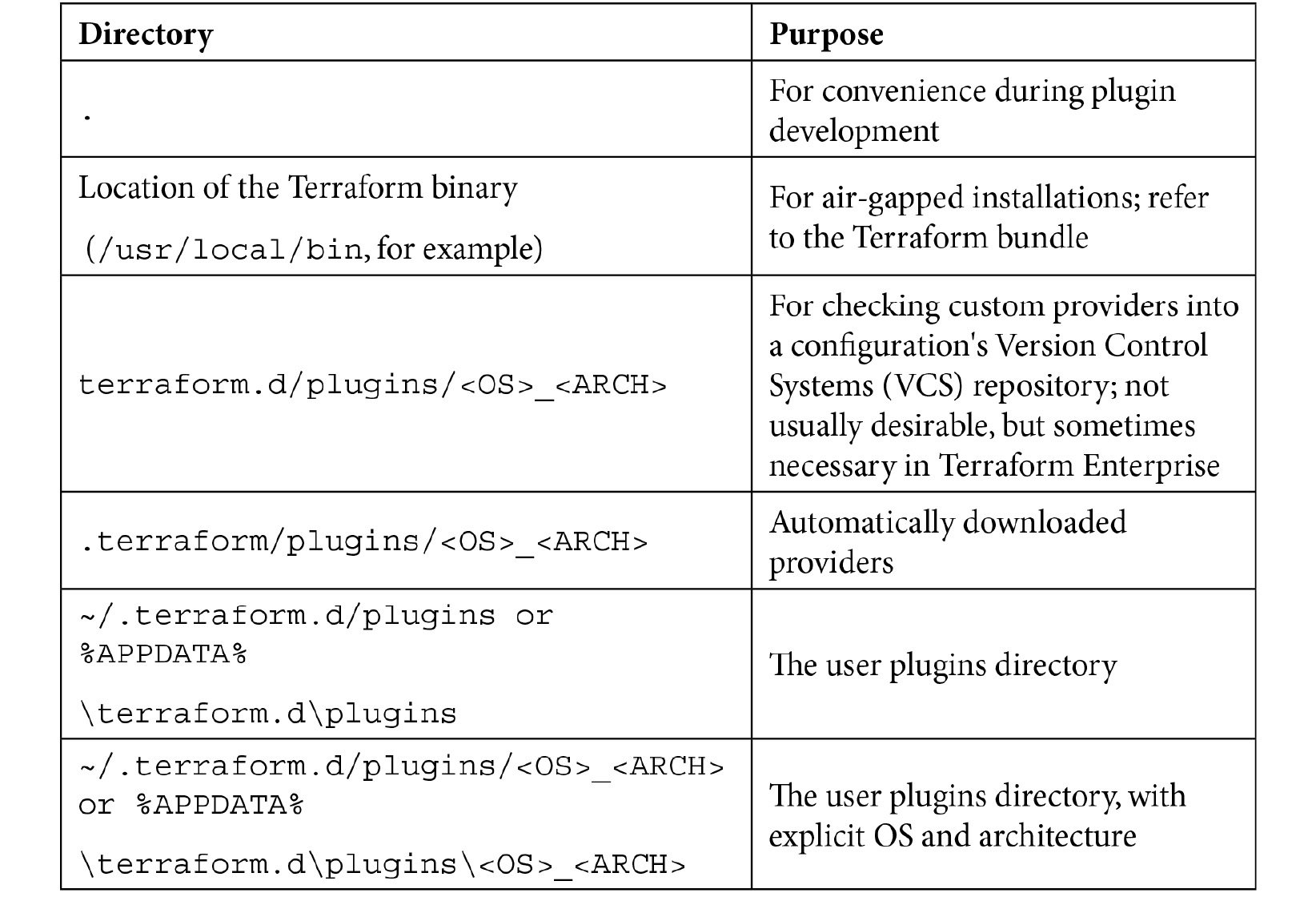
* Initialization of any included libraries used to make API calls
* Authentication with the infrastructure provider
* The definition of resources that map to specific services

The responsibilities of provisioner plugins are as follows:

* Executing commands or scripts on the designated resource following creation or destruction

**Plugin locations**

By default, whenever you run the terraform init command, it will be looking for the plugins in the directories listed in the following table. Some of these directories are static, while some are relative to the current working directory:



You can visit the following link for more information on plugin locations:

https://www.terraform.io/docs/extend/how-terraform-works.html#plugin-locations

Important note

<OS> and <ARCH> use the Go language's standard OS and architecture names, for example, darwin\_amd64.

Third-party plugins should usually be installed in the user plugins directory, which is located at ~/.terraform.d/plugins on most OSes and %APPDATA%\terraform.d\plugins on Windows.

If you are running terraform init with the -plugin-dir=<PATH> option (with a non-empty <PATH>), this will override the default plugin locations and search only the path that you had specified.

Provider and provisioner plugins can be installed in the same directories. Provider plugin binaries are named with the scheme terraform-provider-<NAME>\_vX.Y.Z, while provisioner plugins use the scheme terraform-provisioner-<NAME>\_vX.Y.Z. Terraform relies on filenames to determine plugin types, names, and versions.

**Selecting plugins**

After finding any installed plugins, terraform init compares them to the configuration's version constraints and chooses a version for each plugin as defined here:

* If there are any acceptable versions of the plugin that have already been installed, Terraform uses the newest *installed* version that meets the constraint (even if [releases.hashicorp.com](http://releases.hashicorp.com) has a newer acceptable version).
* If no acceptable versions of plugins have been installed and the plugin is one of the providers distributed by HashiCorp, Terraform downloads the newest acceptable version from [releases.hashicorp.com](http://releases.hashicorp.com) and saves it in .terraform/plugins/<OS>\_<ARCH>.

This step is skipped if terraform init is run with the -plugin-dir=<PATH> or -get-plugins=false options.

* If no acceptable versions of plugins have been installed and the plugin is not distributed by HashiCorp, then the initialization fails and the user must manually install an appropriate version.