HashiCorp Certified: Terraform Associate (002)

<https://www.hashicorp.com/certification/terraform-associate>

The Terraform Associate certification is for Cloud Engineers specializing in operations, IT, or development who know the basic concepts and skills associated with open source HashiCorp Terraform. Candidates will be best prepared for this exam if they have professional experience using Terraform in production, but performing the exam objectives in a personal demo environment may also be sufficient. This person understands which enterprise features exist and what can and cannot be done using the open source offering. Visit our exam partner to [schedule and take the exam](https://hashicorp-certifications.zendesk.com/hc/en-us/articles/360049382552).

**Prerequisites**

* Basic terminal skills
* Basic understanding of on premises and cloud architecture

**Product Version Tested**

Terraform 1.0 and higher.

**Preparing for the Exam**

Certification preparation learning guides for the new version of the Terraform Associate will be posted soon. For now, use the current [Terraform Associate Tutorial List](https://learn.hashicorp.com/collections/terraform/certification-associate-tutorials) to start your studying.

**Exam Details**

|  |  |
| --- | --- |
| **Assessment Type** | Multiple choice |
| **Format** | Online proctored |
| **Duration** | 1 hour |
| **Price** | $70.50 USD plus locally applicable taxes and fees Free retake **not included** |
| **Language** | English |
| **Expiration** | 2 years |

# Exam Objectives

## **Understand infrastructure as code (IaC) concepts**

## Explain what IaC is

* Human readable files that defince infrastructure for both cloud and on-prem
* Immutable
* declarative
* Providers Utilizes APIs
* Allows scalability to be automated
* Workflow stages (same on any platform)
  + Write
    - Define resources
    - Can be across multiple cloud providers and services
  + Plan
    - Terraform creates an execution plan describing infrastructure it will
      * Create
      * Update
      * Destroy
  + Apply
    - On approval, terraform performs proposed operations in correct order
* Types of IaC
  + AdHoc Scripts
    - Better than nothing still not framework
  + Configuration Management Tools
    - Chef
    - Puppet
    - Ansible
    - SaltStack
    - Install and manage software o existing servers
  + Server Templating tools
    - Docker
    - Packer
    - Vagrant
    - Full server template including any software
  + Orchestration tools
  + Provisioning Tools
    - Terraform
    - CloudFormation
    - OpenStack
    - Create the servers (infrastructure) themselves

### Describe advantages of IaC patterns

* Allows you to use only what you need instead of keeping all the time
* Immutable
* Versioning
* Reuse components
* Self service
* Configuration consistency
  + Idempotent
  + Consistent
  + Repeatable
  + Predictable
* Minimize risk
  + Automates process
  + Form of documentation
* Increased efficiency in dev
  + Sandbox environment same for all and built quickly
* Reduce cost
  + Tear down infrastructure when not needed
  + elastic

## **Understand the purpose of Terraform (vs other IaC)**

https://developer.hashicorp.com/terraform/intro/vs

## Explain multi-cloud and provider-agnostic benefits

* What is Terraform
  + Open source
  + Used to do this to infrastructure
    - Build
    - Change
    - Manage
  + Used files ending in
    - .tf
    - .tf.json
  + Same workflow
* Multi-Cloud
  + Single workflow
  + Works on public and private clouds
  + Cloud agnostic
* Infrastructure defined as code
  + Shared and reused
  + Modulare
* Execution plan
  + Terraform plan
  + Information about what terraform wil do
  + Shows what will change
* Resource Graph
  + Efficient build
  + Parallel execution of anything not dependent
* Comparison
  + CloudFormation
    - Not Cross Platform only for AWS
    - Uses YAML or JSON
    - Good Conditional functions
    - Wait condition and creation policy
    - Not quite as modular (nested stacks)
    - Not as good validation
    - Harder to read than terraform
    - State file stored native to AWS
  + Azure ARM
    - Only Azure
    - JSON
    - Not as modular
    - Not as good validation
    - Harder to read JSON
    - State stored in Azure
  + Google Cloud Deployment Manager
    - Only GCP
    - YAML, Jinja or Python
    - Not as modular
* Terraform Architecture
  + Terraform Core
    - Binary written in GO
    - Communicates with terraform plugins via RPC (remote procedure calls)
    - Terraform CLI
    - Responsibilities
      * IAC: reading and interpolating configuration files and modules
      * Resource state management
      * Resource graph construction
      * Plan execution
      * Communication with plugins
  + Terraform Plugins
    - Exposes an implementation for a specific service ( AWS, Azure, VMWare etc) or provisioner (BASH)
    - Defined in the terraform configuration file
    - Responsibilities Provider Plugin
      * Init of any included libraries
      * Auth with infra provider
      * Def resources map to specific services
    - Responsibilities Provisioner Plugin
      * Execute commands on resource following creation or destruction
    - Terraform init looks for plugins in default locations
      * Can be overridden with -plugin-dir=<PATH>
    - Upgrading Plugins
      * -upgrade checks found version with hashicorp for newer version

## Explain the benefits of state

* State files are stored on the local file system or remote backend ( think S3 )

## **Understand Terraform basics**

### Install and version Terraform providers

### Describe plugin-based architecture

### Write Terraform configuration using multiple providers

### Describe how Terraform finds and fetches providers

## **Use Terraform outside of core workflow**

### Describe when to use terraform import to import existing infrastructure into your Terraform state

### Use terraform state to view Terraform state

### Describe when to enable verbose logging and what the outcome/value is

## **Interact with Terraform modules**

### Contrast and use different module source options including the public Terraform Module Registry

### Interact with module inputs and outputs

### Describe variable scope within modules/child modules

### Set module version

## **Use the core Terraform workflow**

### Describe Terraform workflow ( Write -> Plan -> Create )

### Initialize a Terraform working directory (terraform init)

### Validate a Terraform configuration (terraform validate)

### Generate and review an execution plan for Terraform (terraform plan)

### Execute changes to infrastructure with Terraform (terraform apply)

### Destroy Terraform managed infrastructure (terraform destroy)

### Apply formatting and style adjustments to a configuration (terraform fmt)

## **Implement and maintain state**

### Describe default local backend

### Describe state locking

### Handle backend and cloud integration authentication methods

### Differentiate remote state back end options

### Manage resource drift and Terraform state

### Describe backend block and cloud integration in configuration

### Understand secret management in state files

## **Read, generate, and modify configuration**

### Demonstrate use of variables and outputs

### Describe secure secret injection best practice

### Understand the use of collection and structural types

### Create and differentiate resource and data configuration

### Use resource addressing and resource parameters to connect resources together

### Use HCL and Terraform functions to write configuration

### Describe built-in dependency management (order of execution based)

## **Understand Terraform Cloud capabilities**

## Explain how Terraform Cloud helps to manage infrastructure

## Describe how Terraform Cloud enables collaboration and governance