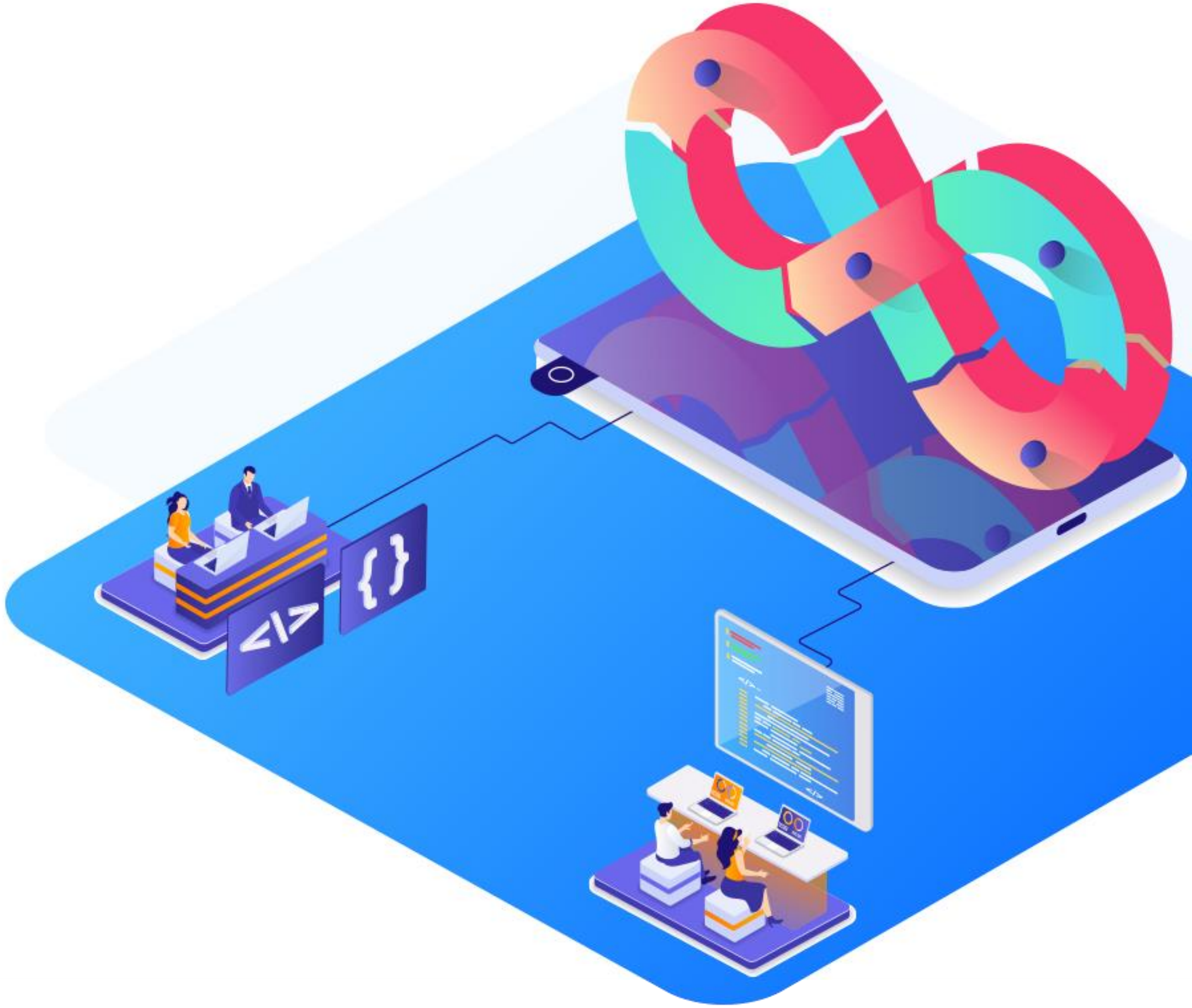
Configuration Management with Ansible and Terraform



Getting Started with Infrastructure as Code (IaC)

Learning Objectives

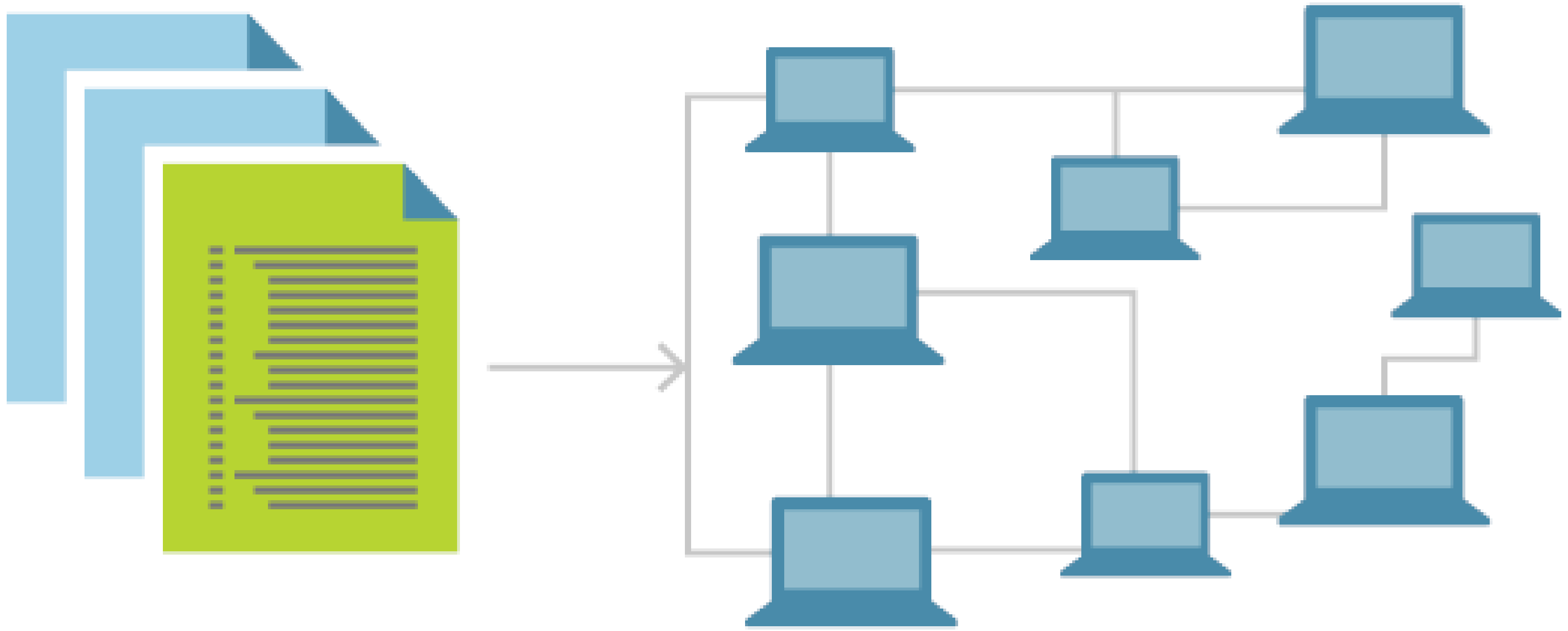
By the end of this lesson, you will be able to:

- Define Infrastructure as Code (IaC) along with its use cases
- Outline the features of HashiCorp Language (HCL) for writing effective configuration management scripts
- Apply the best practices for using Terraform modules and directories for optimized scripting

Terraform: Driving Multi-Cloud Deployments with IaC

What Is Infrastructure as Code (IaC)?

It is a practice followed by IT companies to improve infrastructure deployments, increase users' ability to scale quickly, and improve the application development process.

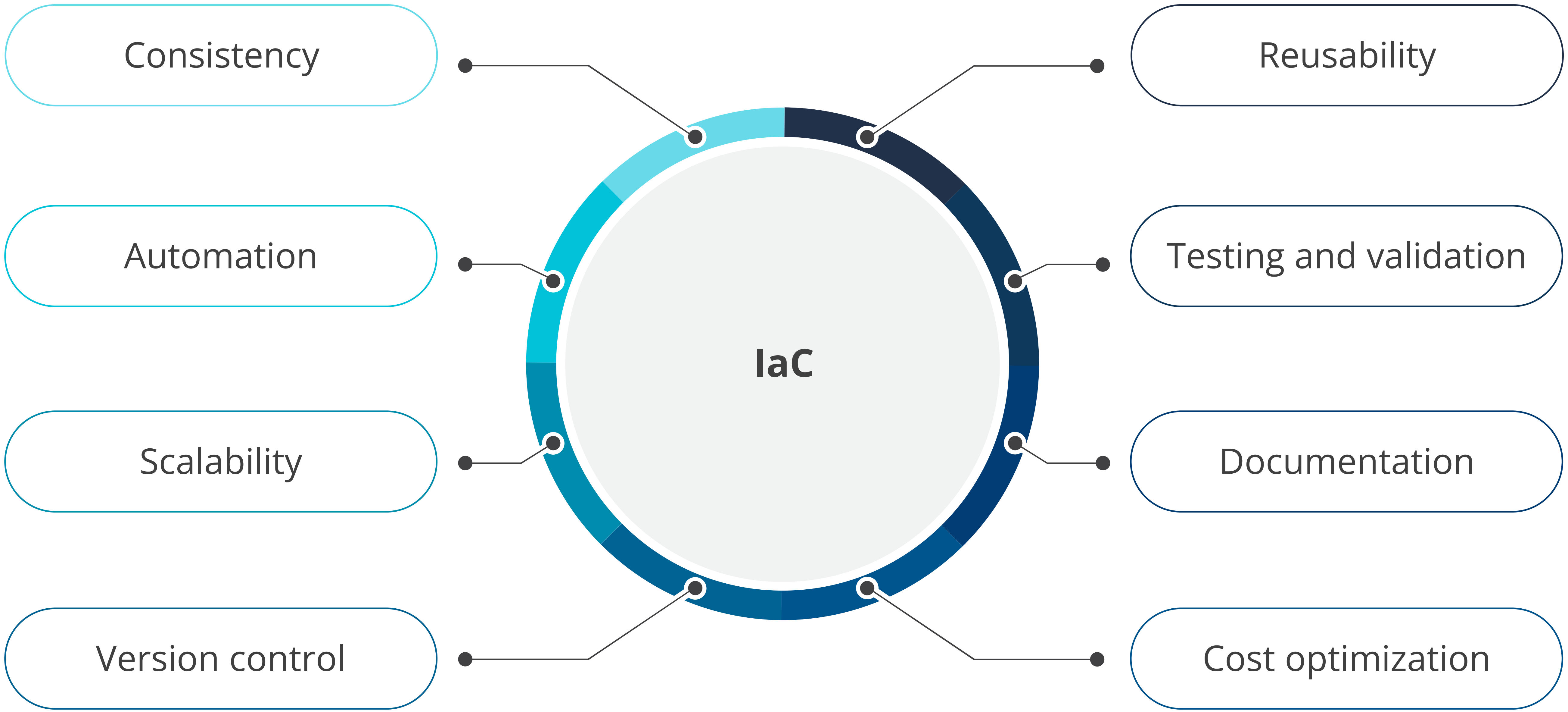


Uses of Infrastructure as Code (IaC)

- It automates the provisioning of development and production environments.
- It ensures disaster recovery and high availability through infrastructure replication.
- It manages scalable infrastructure by automatically adjusting resources.
- It maintains consistent configurations across servers to prevent drift.
- It enforces security policies and compliance across the infrastructure.

Benefits of IaC

It provides many benefits for managing and provisioning software development and deployment infrastructure, including:



IaC

Consistency

Automation

Scalability

Version control

Reusability

Testing and validation

Documentation

Cost optimization

IaC Tools

Some of the popular IaC tools available in the market are as follows:

AWS CloudFormation

Puppet

Ansible

Terraform

Chef

Azure Resource Manager

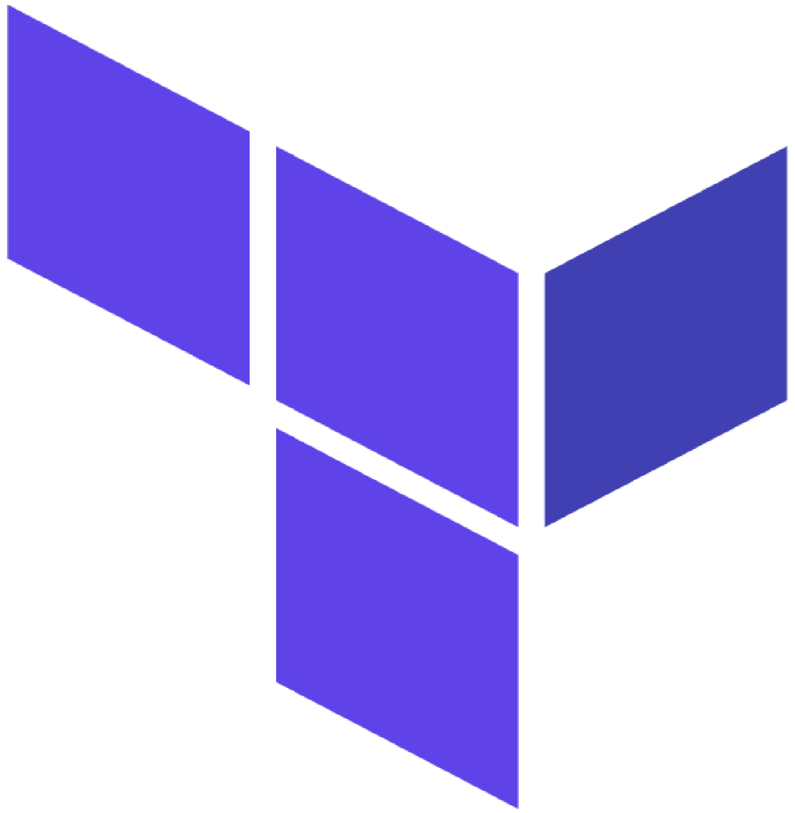
Saltstack

Vagrant

Pulumi

What Is Terraform?

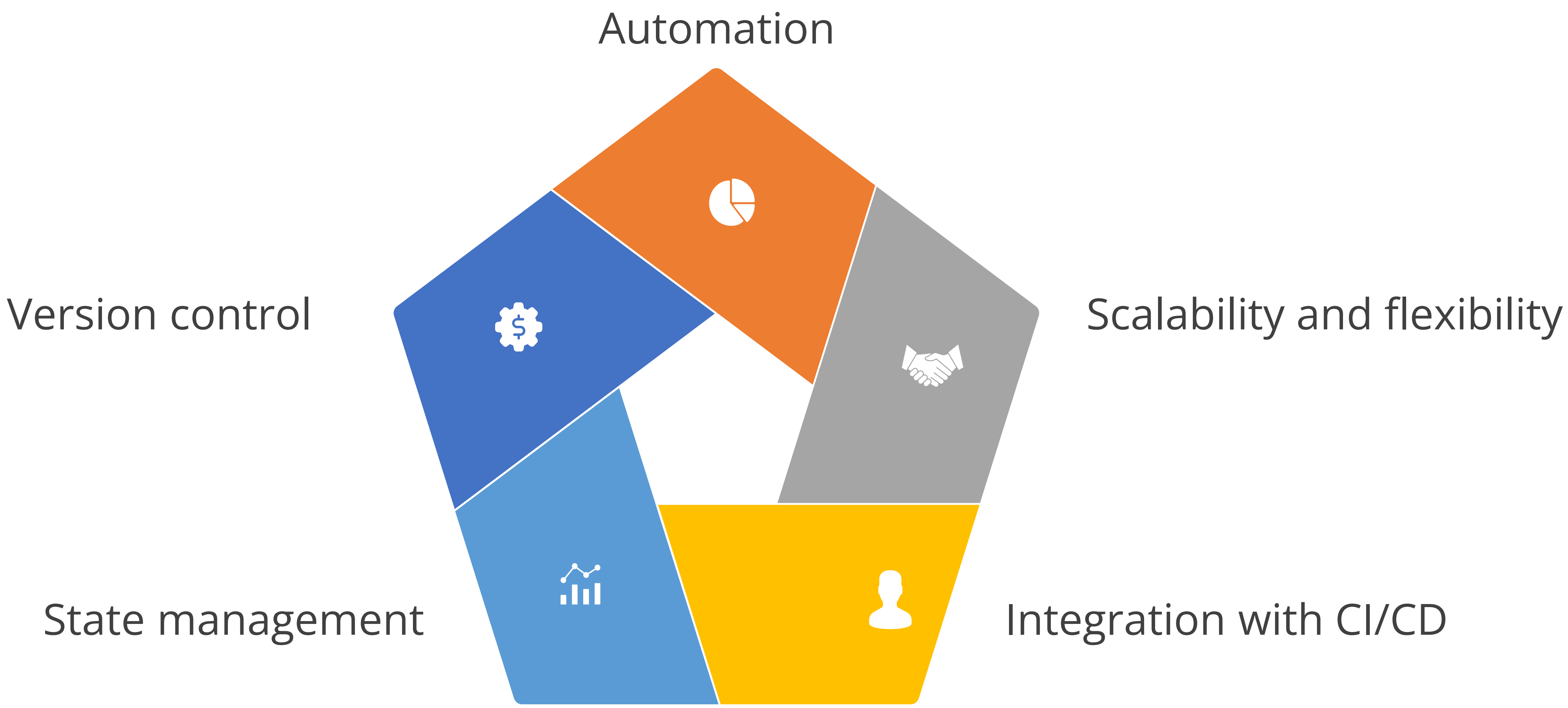
It is an Infrastructure as Code (IaC) tool that enables users to construct, modify, and version infrastructure securely and efficiently.



It facilitates the provisioning of infrastructure and services across various cloud providers, such as AWS, Azure, and GCP, as well as on-premises data centers and beyond.

Terraform as an IaC Tool

It plays a fundamental role in Infrastructure as Code (IaC) by enabling organizations to manage and provision infrastructure resources through code rather than manual processes. Its role includes:



Terraform: Role

Automation

Automates the provisioning and management of infrastructure across cloud providers and on-premises environments

Scalability and flexibility

Scales deployments easily across diverse environments and configurations

Integration with CI/CD

Integrates seamlessly with CI/CD pipelines for automated testing, validation, and deployment of infrastructure changes

State management

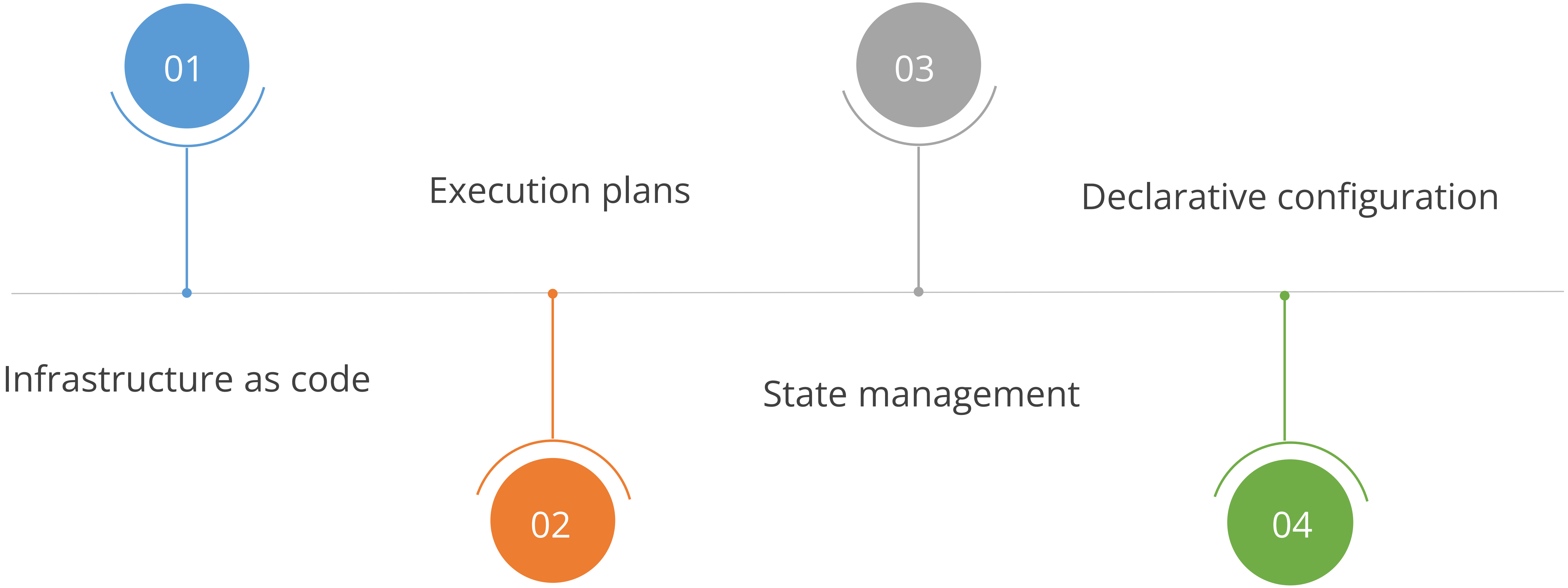
Tracks infrastructure state to ensure desired configurations match actual deployments

Version control

Treats infrastructure configurations as code, allowing versioning and collaboration

Terraform: Features

It offers several features, including:



01 Infrastructure as code

02 Execution plans

03 State management

04 Declarative configuration

Terraform: Features

Infrastructure as code

Terraform treats infrastructure as code, allowing users to manage infrastructure configurations like software code.

Execution plans

The Terraform plan offers a preview of modifications, allowing users to grasp the impact of changes before applying them.

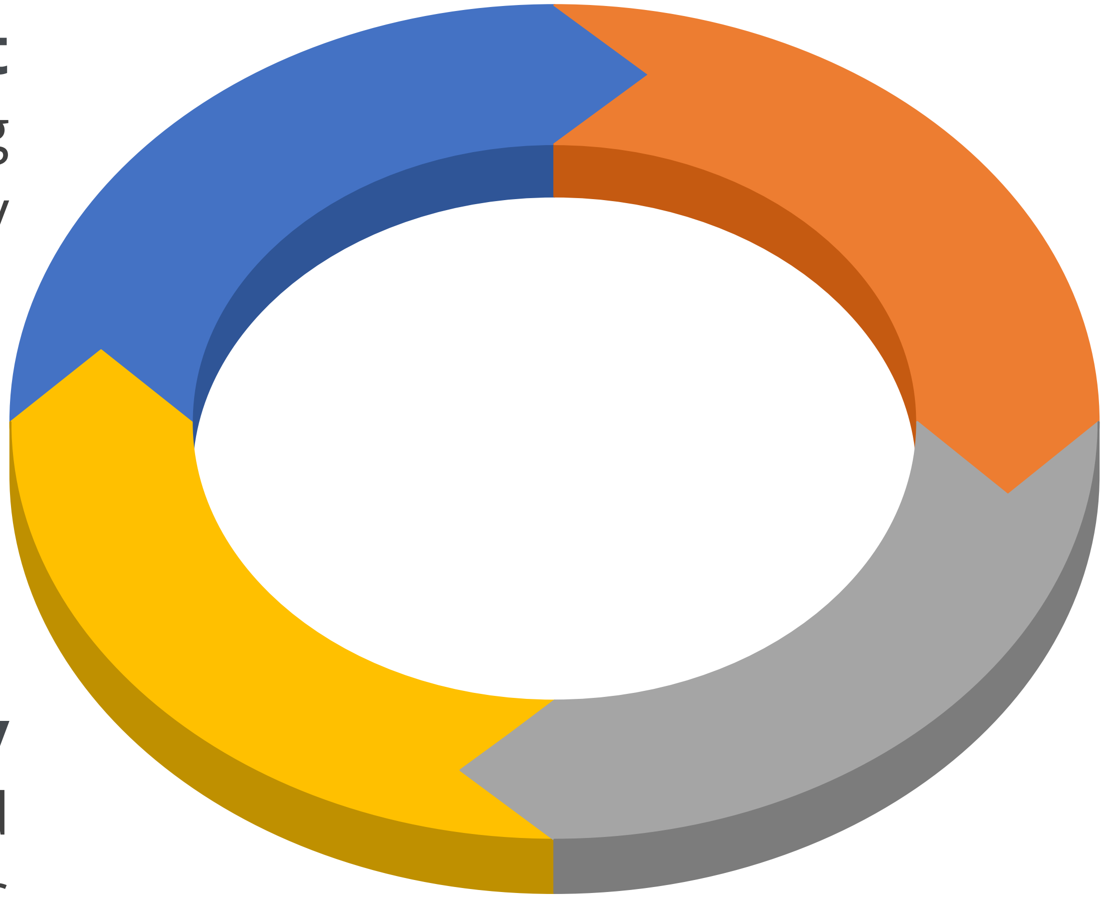
State management

This state plans and applies changes incrementally, ensuring that Terraform only makes the necessary modifications.

Declarative configuration

Terraform uses declarative language to describe the desired state of infrastructure.

Terraform Lifecycle



Init
Initializes the working directory

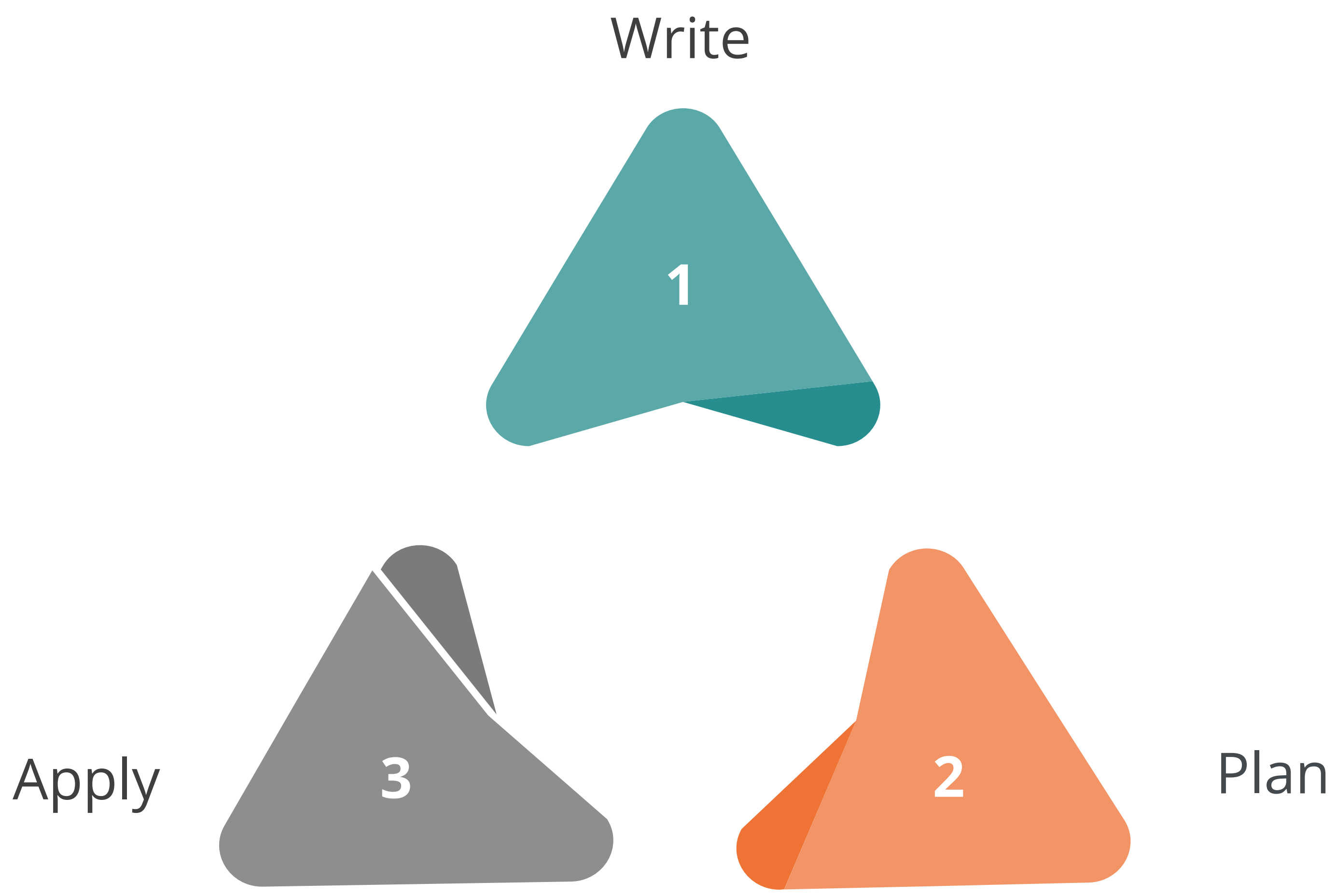
Plan
Creates an execution plan to reach the desired state of the infrastructure

Apply
Makes the changes to the infrastructure as defined in the plan

Destroy
Deletes all the old infrastructure resources

Terraform Workflow

It involves a few steps to efficiently manage infrastructure, such as:



Terraform Workflow

Write

Compose your Terraform configuration as you write code, utilizing your preferred editor.

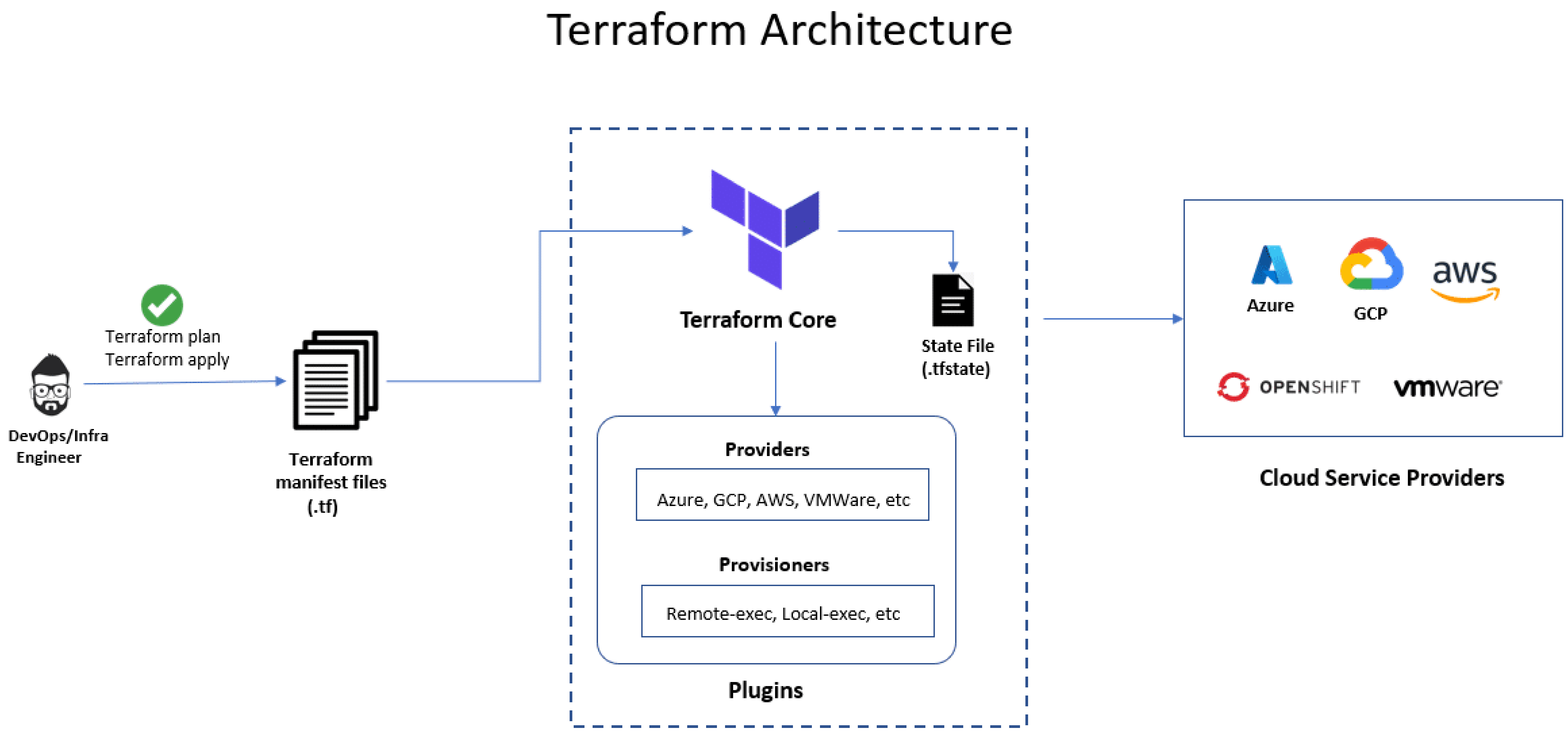
Plan

When the feedback loop of the Write step yields a good change, it's time to commit to your work and review the final plan.

Apply

After one last check, you can tell Terraform to provide actual infrastructure.

Terraform Architecture



Source: https://spacelift.io/blog/terraform-architecture

Terraform Architecture

Users can effectively manage infrastructure and maintain consistency across different environments, using the Terraform workflow.

Terraform core

It is the foundation of Terraform, constructed from a statically compiled binary created using the Go programming language.

Providers

These are modular components that empower Terraform to interface with an extensive array of services and resources, encompassing cloud providers and databases.

State file

It is a JSON file containing details about the resources handled by Terraform, including their present state and dependencies.

Terraform: Benefits

- Improved collaboration
- Consistency
- Reusability
- Portable across cloud providers
- Reduced development costs

Terraform Blocks

Provider block

Specifies a cloud platform and authentication for resource management

Resource block

Defines infrastructure components with specific configurations

Variable block

Declares input variables for flexible configuration adjustments

Output block

Specifies values displayed after deployment, aiding integration

Data block

Retrieves external data or queries existing resources efficiently

What Is CloudFormation?

It is a service provided by Amazon Web Services (AWS) that allows you to model, provision, and manage AWS and third-party resources by treating infrastructure as code.

It allows you to automate the setup and management of AWS resources using code, ensuring consistency in your cloud environment.

Terraform vs. AWS CloudFormation

Terraform

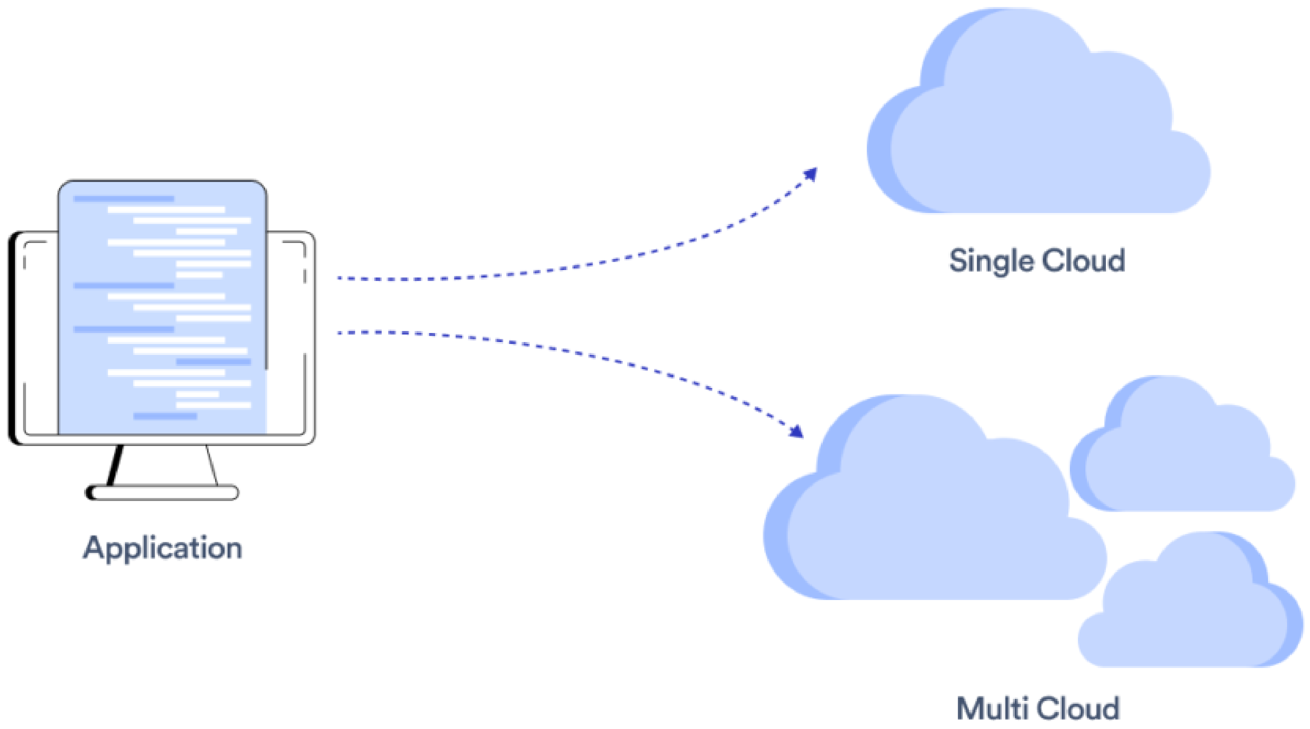
- Multi-cloud support
- Large community and extensive documentation
- User-friendly syntax
- Suitable for small to large-scale infrastructures

AWS CloudFormation

- AWS-specific resource types
- Strong community support
- Organize resources into stacks
- Automatically revert changes in case of deployment errors

Multi-Cloud Deployment

It refers to utilizing services and resources from multiple cloud providers simultaneously to optimize performance, redundancy, and flexibility in IT infrastructure management.



This involves distributing workloads, applications, and data across multiple cloud computing environments from different providers.

Uses of Multi-Cloud Deployment

High availability

Ensures continuous operation by spreading applications across multiple clouds

Performance optimization

Places workloads closer to users or leverages specialized services for improved performance

Cost efficiency

Optimizes costs by choosing the most economical cloud services and taking advantage of pricing variations

Innovation

Expedites innovation by accessing diverse technologies and services from multiple providers and fostering agility

Quick Check

In a multi-cloud deployment scenario, a company experiences an outage in AWS during planned maintenance. How does using multiple cloud providers benefit the company?

A. By reducing operational costs through consolidating services under one provider

B. By ensuring all applications remain unaffected during maintenance

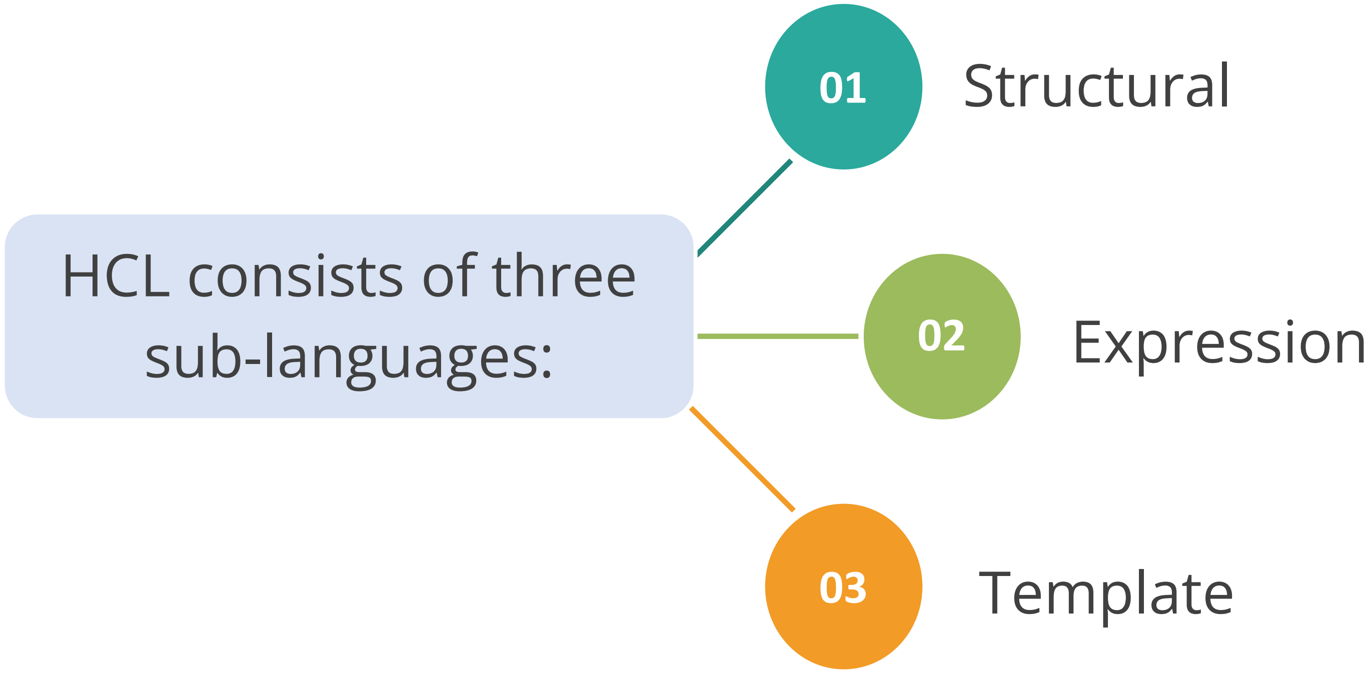
C. By allowing applications to seamlessly failover to Azure, maintaining continuity

D. By providing faster data processing speeds through optimized networks

HashiCorp Configuration Language (HCL)

HashiCorp Configuration Language (HCL)

It is a declarative configuration language developed by HashiCorp.



HCL consists of three sub-languages:

01 Structural

02 Expression

03 Template

HCL is primarily used for defining configurations for provisioning and managing infrastructure resources.

Why Use HCL in Terraform?

HCL can be used for several reasons, including:

1. Defining infrastructure with declarative syntax
2. Simplifying and reading configurations easily
3. Extending functionality with new features and tools
4. Integrating seamlessly with multiple HashiCorp products.
5. Utilizing variables and expressions for flexible configuration management

HCL: Features

Declarative syntax

HCL uses a declarative style, enabling users to define their infrastructure's desired state without specifying procedural steps.

Simple and readable

HCL is crafted for simplicity and readability, ensuring accessibility for both developers and operations teams.

Extensible

HCL's simplicity and extensibility enable its use with other systems, promoting interoperability.

Support for multiple HashiCorp tools

It serves as the configuration language for various HashiCorp tools facilitating a unified approach to infrastructure management.

Support for variables and expressions

It supports variables, enabling dynamic configurations with computations, conditionals, and other operations.

HCL Syntax: Overview

It is crafted for clarity and readability, making it ideal for expressing infrastructure configurations in HashiCorp tools. Here are some key elements of HCL syntax:

1. Blocks and Nesting
2. Attributes and Values
3. Variables
4. Expressions

HCL Syntax: Overview

Here are some key elements of HCL syntax:

5. Quoted Strings
6. Comments
7. Dynamic Blocks

Basic Syntax of HCL

HCL syntax revolves around the following core components:

Blocks

Structured containers that encapsulate configurations for resources or modules and simplify infrastructure management

Arguments

Key-value pairs within blocks that define attributes and parameters and allow precise configuration and customization

Expressions

Dynamic elements for data manipulation, including variables, functions, and resource references

HCL Syntax: Identifiers

It encompasses the names of various elements, including:

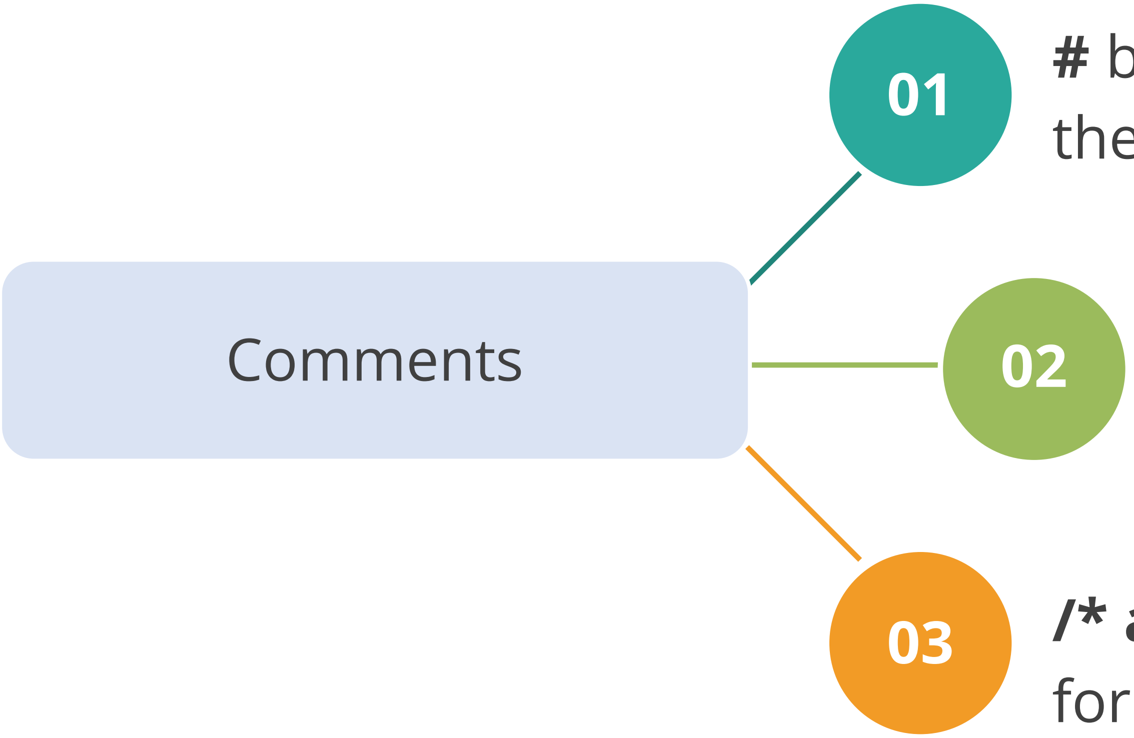
- Argument names
- Block type names
- Terraform-specific entities like resources and input variables

Creating Identifiers

Here are the rules for creating identifiers in HCL:

- Identifiers must start with a letter (a-z or A-Z).
- After the initial letter, identifiers can include letters, numbers (0-9), and underscores (_).
- They are case-sensitive, meaning Variable and variable would be considered different identifiers.
- Identifiers cannot contain spaces or special characters (for example, @, #, and $) except for underscores.

HCL Syntax: Comments



Comments

01 **#** begins a single-line comment, ending at the end of the line.

02 ***//*** can be used as an alternative to **#** to begin a single-line comment.

03 ***/* and */*** are the start and end delimiters for a comment that might span over multiple lines.

The **# single-line** comment style is the default comment style and should be used in most cases.

Assisted Practice

Deploying and Managing a VPC with Public and Private Subnets in AWS

Duration: 20 Min.

Problem Statement:

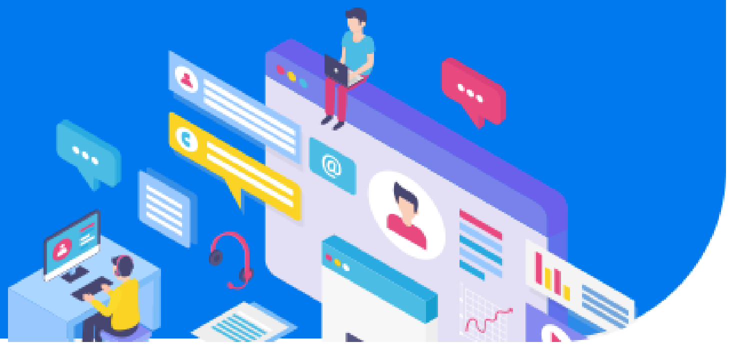
You have been assigned a task to deploy and manage a resilient VPC with public and private subnets across multiple availability zones in AWS, ensuring proper routing and connectivity for failover and disaster recovery testing.

Outcome:

A robust VPC infrastructure with effective routing and connectivity that supports failover and disaster recovery, ensuring high availability and resilience of the deployed resources.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Create a new VPC in your account in the US-East-1 region
2. Create public and private subnets in three different Availability Zones
3. Deploy an Internet Gateway and attach it to the VPC
4. Provision a NAT Gateway (a single instance will do) for outbound connectivity
5. Ensure that route tables are configured to properly route traffic
6. Delete the VPC

ASSISTED PRACTICE

Assisted Practice

Deploying AWS Infrastructure with Terraform

Duration: 20 Min.

Problem Statement:

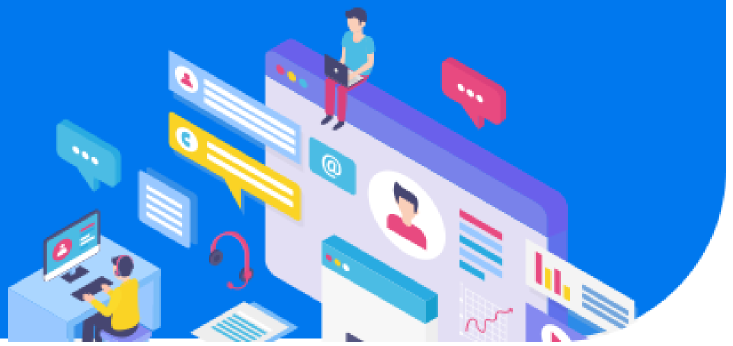
You have been assigned a task to deploy AWS infrastructure using Terraform, addressing challenges such as ensuring consistent and automated resource provisioning, managing configurations and dependencies, and maintaining infrastructure as code.

Outcome:

A streamlined and automated deployment process that reduces manual errors and enables efficient, scalable, and repeatable infrastructure management within the AWS environment.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Prepare files and credentials for using Terraform to deploy cloud resources
2. Set credentials for Terraform deployment
3. Deploy the AWS infrastructure using Terraform
4. Delete the AWS resources using Terraform to clean up our AWS environment

Assisted Practice

Validating Terraform Configuration File

Duration: 20 Min.

Problem Statement:

You have been assigned a task to validate Terraform configuration files, addressing challenges such as ensuring the correctness of syntax and semantics, detecting security vulnerabilities, adhering to compliance requirements, and managing infrastructure dependencies.

Outcome:

An automated validation process that prevents deployment failures, enhances security, and ensures compliance with organizational policies, leading to more reliable and secure Terraform deployments.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines

Steps to be followed:

1. Validate the Terraform script

HCL Functions and Configuration

HCL Functions

It enhances infrastructure code with tools for data manipulation, including string, numeric, collection, and type conversion functions. Some commonly used **string functions** are:

lower()	Converts a string to lowercase
upper()	Converts a string to uppercase
trimspace()	Removes leading and trailing spaces from a string
substr()	Extracts a substring from a given string
tostring()	Converts a value to a string

HCL Functions

Some commonly used **numeric functions** are:

max()	Returns the maximum value from a list of numbers
min()	Returns the minimum value from a list of numbers
abs()	Returns the absolute value of a number
ceil()	Rounds a number up to the nearest integer
floor()	Rounds a number down to the nearest integer
tonumber()	Converts a value to a number

HCL Functions

Some commonly used **collection functions** are:

contains()	Checks if a list contains a specific element
merge()	Merges multiple maps into one
keys()	Returns the keys of a map as a list
length()	Returns the length of a list or string

HCL Functions

Some commonly used **type conversion functions** are:

tomap()	Converts a value to a map
toset()	Converts a value to a set
tolist()	Converts a value to a list

These functions extend the functionality of HCL to meet specific needs that are not covered by built-in functions.

HCL Configuration: Example

Here is an example of a simple HCL configuration that defines an AWS EC2 instance using Terraform:

Example:

```
# Provider Block - Specifies AWS as the cloud provider
provider "aws" {
  region = "us-west-2"
}

# Resource Block - Defines an AWS EC2 instance
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"  # Amazon Linux 2 AMI ID
  instance_type = "t2.micro"
  key_name      = "my-keypair"

  tags = {
    Name = "ExampleInstance"
  }
}
```

Assisted Practice

Creating a Simple HCL Configuration File **Duration: 20 Min.**

Problem Statement:

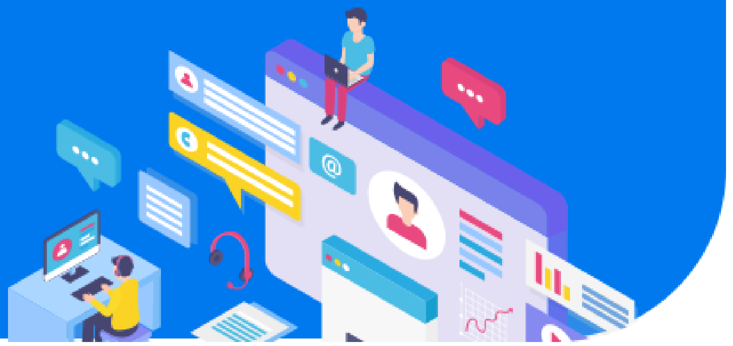
You have been assigned a task to create a simple HCL (HashiCorp Configuration Language) configuration file, which involves understanding the basic syntax and structure of HCL, defining resources and their properties, and managing configurations and dependencies.

Outcome:

A straightforward HCL configuration file that clearly and effectively defines infrastructure resources, facilitating easy management and understanding of the infrastructure setup.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Prepare files and credentials for using Terraform
2. Write the HCL configuration in the main.tf
3. Validate and apply the Terraform configuration
4. Clean up resources

Assisted Practice

Implementing HCL Functions

Duration: 20 Min.

Problem Statement:

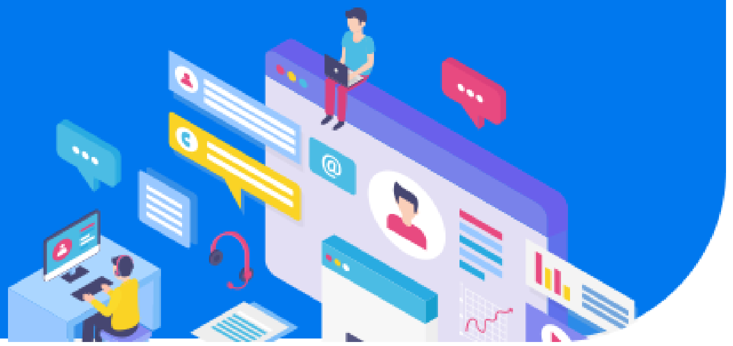
You have been assigned a task to implement HCL (HashiCorp Configuration Language) functions, which involves understanding and utilizing the built-in functions provided by HCL to manipulate and manage data within Terraform configuration files.

Outcome:

A straightforward HCL configuration file that clearly and effectively defines infrastructure resources, facilitating easy management and understanding of the infrastructure setup.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Prepare files and set up Terraform
2. Write the HCL configuration using functions
3. Validate and apply the Terraform configuration

Quick Check

In a cloud infrastructure project using HashiCorp tools, a team is tasked with automating the deployment of AWS resources for a new application. How does HashiCorp Configuration Language (HCL) facilitate this process?

A. By securing network communications between AWS instances.

B. By defining infrastructure configurations in a human-readable format.

C. By automating the creation of Kubernetes clusters on AWS.

D. By optimizing database performance across AWS regions.

Key Takeaways

- It is a practice followed by IT companies to improve infrastructure deployments, increase users' ability to scale quickly, and improve the application development process.
- Terraform is an Infrastructure as Code (IaC) tool that enables users to construct, modify, and version infrastructure securely and efficiently.
- HCL is crafted for clarity and readability, making it ideal for expressing infrastructure configurations in HashiCorp's tools.
- Multi-cloud refers to the practice of utilizing services and resources from multiple cloud providers simultaneously.
- HCL function enhances infrastructure code with tools for data manipulation, including string, numeric, collection, and date functions.

Optimizing Resilience in AWS Infrastructure Deployment Using Terraform

Duration: 25 Min.

Project agenda: To deploy and manage a resilient AWS infrastructure using Terraform for optimal performance and scalability

Description: Imagine you are a cloud engineer tasked with deploying and managing a resilient AWS infrastructure using Terraform. The project involves creating a VPC with public and private subnets, deploying necessary resources such as an Internet Gateway and a NAT Gateway, and demonstrating the use of HCL functions for string manipulation, collections, and encoding using Terraform. This project aims to provide a comprehensive understanding of infrastructure as code (IaC) practices.

LESSON-END PROJECT

Optimizing Resilience in AWS Infrastructure Deployment Using Terraform

Duration: 25 Min.

Perform the following:

1. Prepare files and set up Terraform
2. Deploy the AWS infrastructure
3. Implement HCL functions for string manipulation, collections, and encoding
4. Validate the Terraform configuration
5. Clean up AWS resources

Expected Deliverables: Terraform configuration files for deploying a resilient AWS infrastructure with VPC, public and private subnets, Internet Gateway, and NAT Gateway. Includes HCL functions for string manipulation and collections, with clear documentation of installation and execution steps.

Thank You