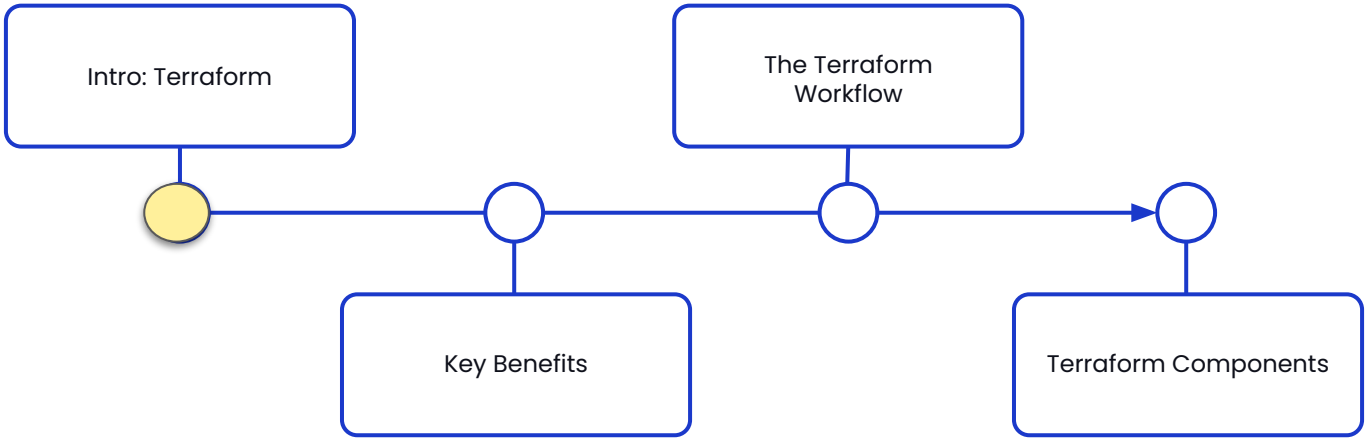
Digital Career Institute

AWS SSA

Terraform | Week 1

DAY : 1
Introduction to Terraform

Architecture and VPC



Introduction

Introduction to Terraform:

- Terraform is an **Infrastructure as Code (IaC)** tool created by HashiCorp.
- It enables users to **define** and **provision infrastructure** using a **declarative configuration language.**
- Terraform treats infrastructure as code, allowing for versioning, automation, and collaboration.

Why use Terraform for Infrastructure as Code (IaC)?

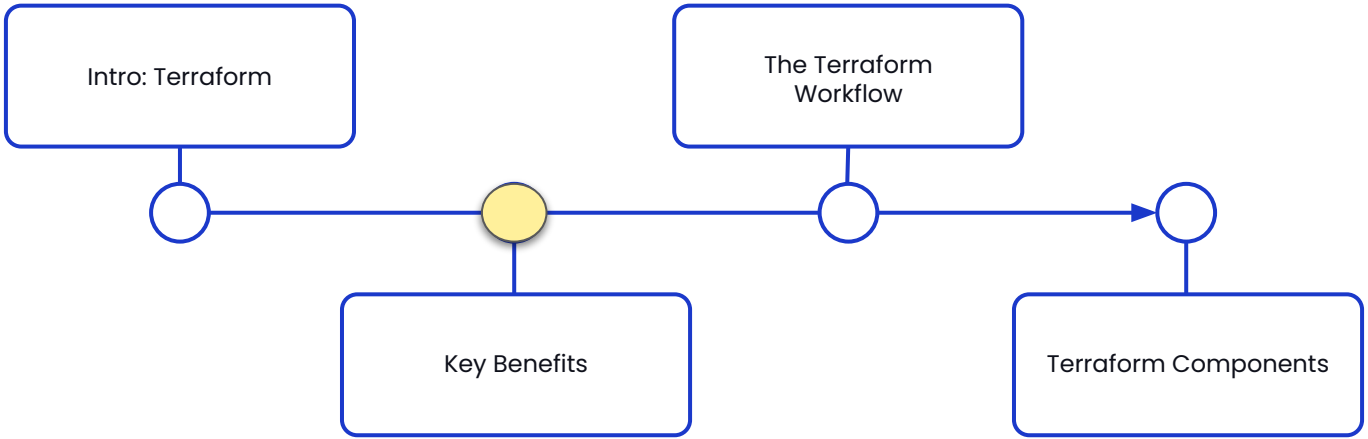
- **Declarative Infrastructure**: Terraform offers a declarative approach, defining the desired infrastructure state, not the step-by-step process.

- **Multi-Cloud Support**: Terraform provides multi-cloud compatibility, enabling management of resources across various cloud providers.

- **Scalability**: Terraform scales with your infrastructure, making it suitable for small projects to enterprise-level applications.

- **Version Control**: Infrastructure configurations are versioned, enhancing collaboration and change tracking.

Architecture and VPC



Key Benefits – IaC

- Defines infrastructure in ***versioned, human-readable configuration files***.
- Enables ***automation*** and ***consistency*** across environments.
- Reduces ***manual errors*** by automating infrastructure management.
- Facilitates infrastructure changes and ***scaling*** through code updates.

The HashiCorp Configuration Language (HCL)

The HashiCorp Configuration Language (HCL) is the language used for defining Terraform configurations.

- **HCL Overview**: HCL is a domain-specific language tailored for configuring infrastructure resources. It's designed for readability and ease of use.
- **Syntax and Structure**: HCL's syntax and structure encompassing blocks, attributes, and expressions, through practical examples.
- **Variables and Data Types**: Variables and the supported data types in HCL enables dynamic configuration and reusable code.

HCL Syntax

Syntax of HashiCorp Configuration Language (HCL), the language used for defining Terraform configurations.

- **Blocks**: Blocks are defined as the structure and type of resources. Blocks are identified by their resource type and encapsulate configuration settings within curly braces.

- **Attributes**: Attributes are residing within Blocks and determine the properties and characteristics of resources. Attributes are assigned values that configure the behavior of resources.

- **Expressions**: Expressions enable dynamic and reusable configurations. Expressions are used for calculations, transformations, and referencing values within your configurations.

Key Benefits – Cloud-Agnostic

- Works with ***multiple cloud providers*** like AWS, Azure, and Google Cloud.
- Supports ***on-premises systems***, offering flexibility across infrastructures.
- Provides a ***unified approach*** to managing different platforms.
- Reduces vendor lock-in by allowing cross-cloud strategies.

Key Benefits – Version Control & Collaboration

- Tracks infrastructure changes using Git or other ***version control systems***.
- Facilitates ***collaboration*** through pull requests and code reviews.
- Enables ***easy rollback*** to previous infrastructure versions if needed.
- Increases ***transparency and accountability*** in infrastructure management.

Key Benefits – Resource Dependency Management

- Automatically ***detects and manages dependencies*** between resources.
- Ensures resources are created, updated, or deleted in the ***correct order***.
- ***Minimizes*** the risk of ***failed deployments*** due to dependency issues.
- ***Optimizes*** infrastructure ***updates*** by handling resource relationships efficiently.

```
resource "aws_lambda_function" "example" {
  depends_on = [aws_s3_bucket.example]
  ...
}
```

Key Benefits – State Management

- ***Tracks the current state*** of your infrastructure in a state file.
- Allows for incremental updates, only ***applying necessary*** changes.
- Improves efficiency and ensures infrastructure ***consistency***.
- Facilitates accurate management of complex, large-scale environments.

Key Benefits – Modular and Reusable Code

- Allows creation of ***reusable modules*** for common infrastructure components.
- Promotes ***code reuse*** across different projects and environments.
- Ensures ***consistency*** and ***standardization*** in infrastructure deployments.
- Simplifies infrastructure management by ***reducing code duplication***.

Key Benefits – Drift Detection

- ***Detects manual changes*** made outside Terraform.
- Ensures infrastructure ***remains consistent*** with the desired state.
- Helps identify and correct unintended or unauthorized modifications.
- Reduces risk and maintains stability across environments.

Key Benefits – Multi-Environment Management

- Manages ***multiple environments*** (e.g., dev, staging, production) with ease.
- Uses ***workspaces*** or separate configuration files for isolation.
- Ensures consistency across environments, reducing configuration drift.
- Facilitates efficient promotion of changes between environments.

Key Benefits – Scalability

- Scales efficiently to manage thousands of resources in complex infrastructures.
- Handles infrastructure growth while maintaining performance.
- Optimizes resource provisioning and updates, even in large-scale environments.
- Supports both small and large infrastructures with the same workflow.

Key Benefits – Extensible with Providers

- Integrates with various cloud platforms, SaaS tools, and custom providers.
- Adapts to a wide range of use cases, including on-premises and hybrid environments.
- Expands functionality through community and custom providers.
- Enables comprehensive infrastructure management across diverse platforms.

Key Benefits – Automation and CI/CD Integration

- Integrates with CI/CD pipelines for automating infrastructure provisioning.
- Reduces manual intervention and potential for errors during deployments.
- Streamlines infrastructure updates as part of automated workflows.
- Ensures continuous, reliable deployments across environments.

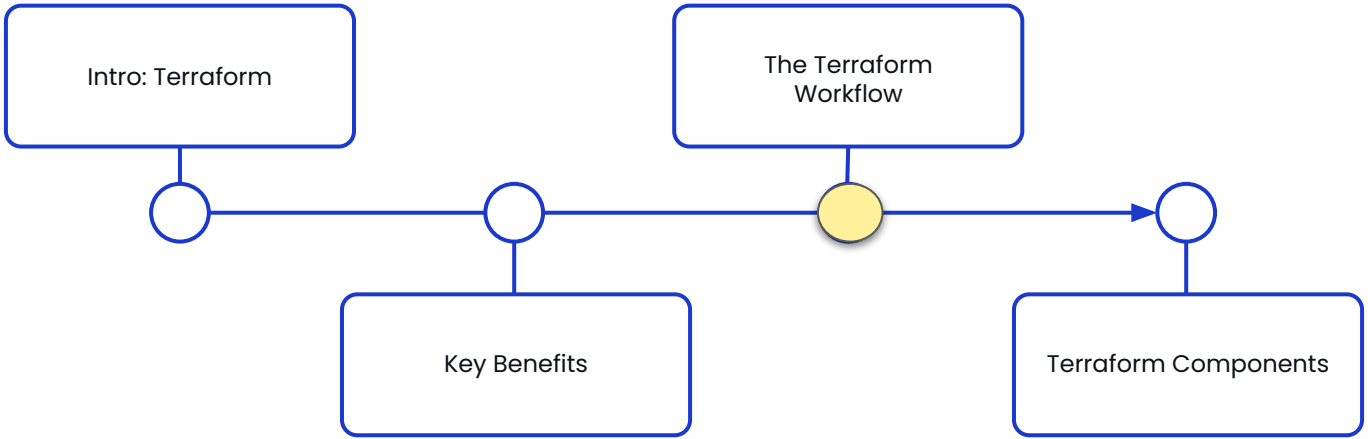
Key Benefits – Consistent Workflow

- Provides a predictable, uniform workflow across different platforms.
- Simplifies infrastructure management by reducing tool complexity.
- Ensures teams can manage resources consistently, regardless of cloud provider.
- Reduces the learning curve for managing multi-cloud and hybrid infrastructures.

Key Benefits – Cost Efficiency

- Optimizes resource usage by automating provisioning and scaling.
- Reduces manual errors, leading to cost savings and more efficient deployments.
- Helps prevent over-provisioning and underutilization of resources.
- Ensures cost-effective management of infrastructure, minimizing waste.

Architecture and VPC

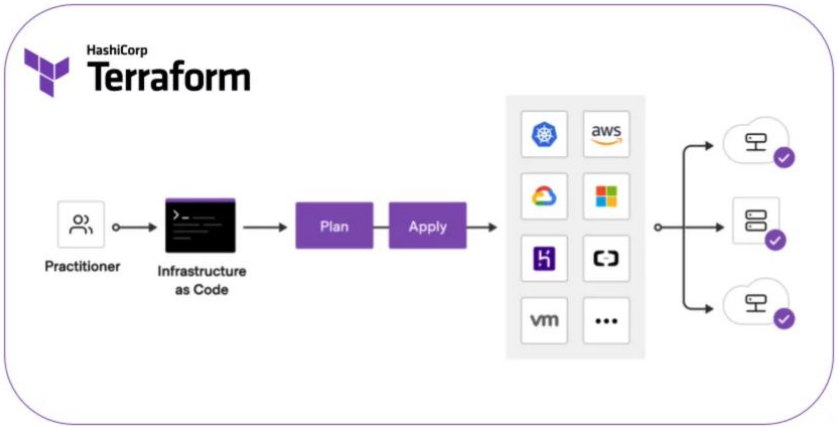


The Terraform Workflow

- **Planning:** Define resources and predict changes.
- **Applying:** Execute the plan to create or update resources.
- **Managing:** Maintain infrastructure over time.

Overview of Infrastructure as Code (IaC) with Terraform

- Infrastructure as Code (IaC) is a methodology for managing and provisioning infrastructure resources using code.
- Terraform, developed by HashiCorp, is a leading IaC tool that automates infrastructure provisioning and configuration.
- It empowers us to automate the provisioning and configuration of infrastructure, offering a more efficient and scalable way to manage our resources.



Step 1 – Planning

- Reviews the current infrastructure state.
- Compares desired state (in code) with actual state.
- Generates an execution plan that shows what changes will be made.

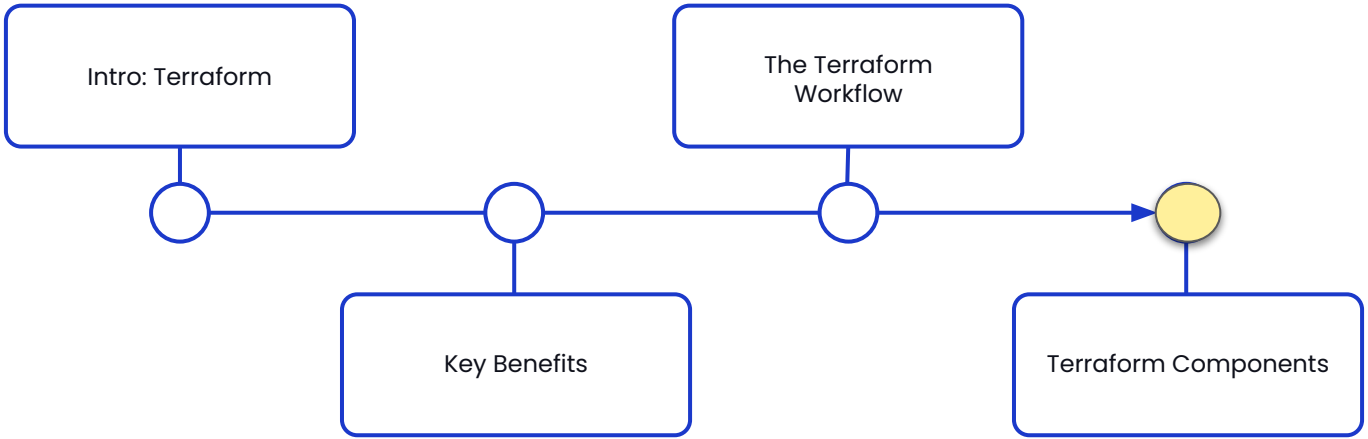
Step 2 – Applying

- Executes the changes defined in the plan.
- Creates, updates, or deletes resources as necessary.
- Provides an output of what changes were made.

Best Practices for Terraform Workflow

- Always review the plan before applying.
- Store your state file securely (e.g., use remote backends like S3).
- Use version control for your Terraform files.
- Automate the workflow with CI/CD pipelines.

Architecture and VPC



Providers

Description: Plugins that allow Terraform to interact with external APIs, such as cloud platforms (AWS, Azure, GCP), SaaS tools, or on-premises infrastructure.

Key Points:

- Manage different cloud and service resources via API interactions.
- Enables integration with various cloud platforms, SaaS tools, and more.
- Each provider translates Terraform configuration into platform-specific API calls.

Resources

Description: The fundamental building blocks representing infrastructure components, such as virtual machines, databases, or networking elements.

Key Points:

- Resources define infrastructure in your Terraform configuration files.
- Examples include AWS EC2 instances, Google Cloud buckets, Azure VMs.
- Each resource defines parameters like size, location, and dependencies.

Modules

Description: A collection of multiple resources grouped together for reusability and easier management.

Key Points:

- Encapsulates reusable infrastructure code.
- Enables consistent deployments across different environments.
- Simplifies infrastructure management by organizing resources.

Variables

Description: Input values that make configurations flexible and reusable by allowing parameterization.

Key Points:

- Allows customization of infrastructure configurations.
- Used to pass environment-specific values like region, instance sizes.
- Increases flexibility and reusability in your Terraform code.

Outputs

Description: Values returned after resource creation, often used to expose resource information.

Key Points:

- Outputs provide key information, such as instance IDs or IP addresses.
- Can be used by other modules or external systems.
- Helps manage and expose information from Terraform-managed resources.

State

Description: A file that keeps track of the current state of your managed infrastructure.

Key Points:

- Tracks deployed infrastructure to know what exists and needs to be updated.
- Enables Terraform to perform incremental changes instead of full redeployments.
- Can be stored locally or remotely for team collaboration.

Execution Plans

Description: Previews the actions Terraform will take to create, update, or destroy resources.

Key Points:

- Provides visibility into what will change before applying it.
- Helps teams review changes to avoid unintended modifications.
- Generated using the terraform plan command.

Workspaces

Description: Isolated environments that enable management of different instances of the same infrastructure.

Key Points:

- Manage multiple environments (e.g., dev, staging, production) using the same code.
- Each workspace has its own isolated state.
- Allows easy environment separation without duplicating code.

Backends

Description: Mechanisms to store and manage the state file remotely for collaboration.

Key Points:

- Store state files in remote locations like AWS S3, Azure Blob Storage.
- Enables state locking and collaborative infrastructure management.
- Supports versioning and security for state files.

Providers Registry

Description: A public repository of Terraform providers and modules.

Key Points:

- Provides pre-built modules and providers for quick setup.
- Expands Terraform functionality through community-driven contributions.
- Simplifies finding and integrating third-party services with Terraform.

Data Sources

Description: Used to query information from external services or existing resources.

Key Points:

- Fetch data without managing the resource directly.
- Useful for querying existing resources (e.g., existing AWS VPCs).
- Helps integrate external data into the infrastructure plan.

Digital Career Institute

Next up

Terraform | Week 1

DAY 2:
Resources and Providers

LEARN FOR **A NEW LIFE.**

DCI