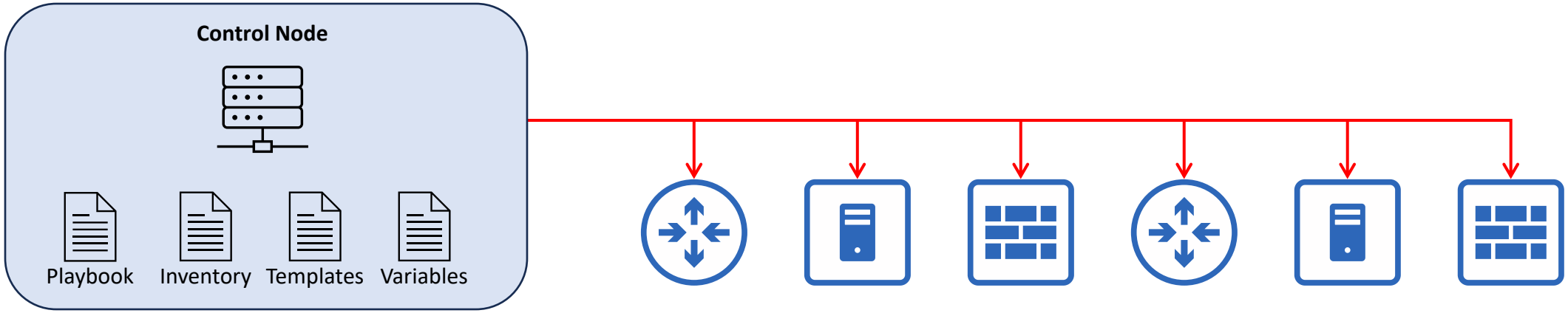
Infrastructure as Code

- **Infrastructure as Code (IaC)** is the practice of provisioning and managing infrastructure (servers, networks, cloud resources) using machine-readable configuration files (code) instead of manual configuration (e.g., CLI/GUI).

- Ansible, Puppet, and Chef are examples of IaC configuration management tools.
  - Ansible manages device configurations using several files:



- **Terraform** is an IaC-based provisioning tools that automates the creation of infrastructure resources.

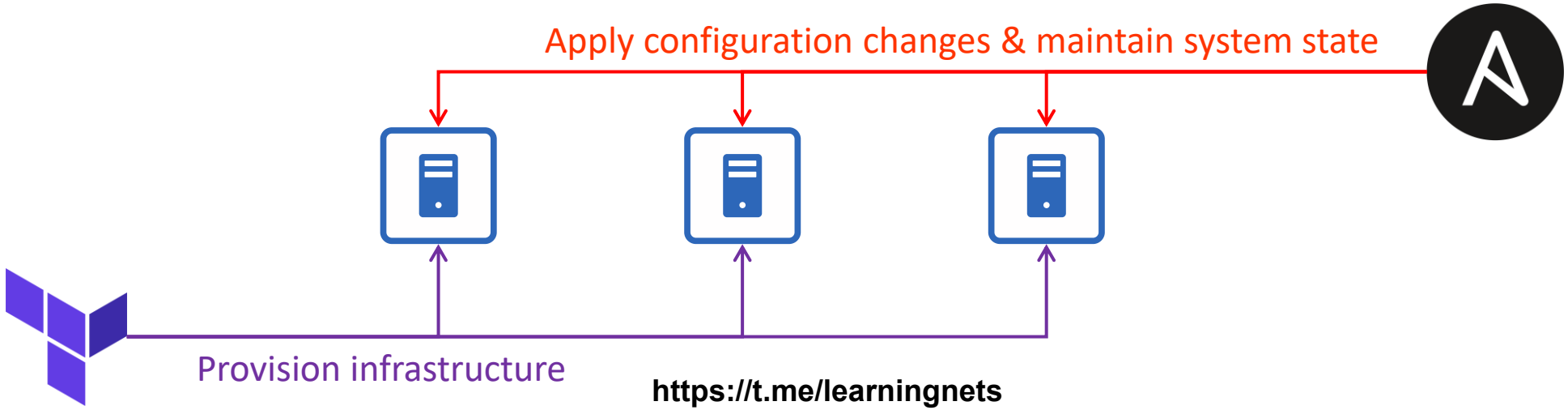
- IaC automates infrastructure deployment and management, ensuring consistency, scalability, and repeatability.

Provisioning vs management

- **Configuration management** (e.g., Ansible, Puppet, Chef)
  - Manages existing infrastructure by installing software, configuring settings, and maintaining system state.
  - Ensures consistency by applying and enforcing configurations across multiple devices.

- **Infrastructure provisioning** (e.g., Terraform)
  - Creates, modifies, and deletes infrastructure resources such as servers and network infrastructure.
  - Focuses on initial setup rather than ongoing configuration management.

- Configuration management tools work on already existing systems, whereas provisioning tools build infrastructure from scratch.
- **Terraform** (provisioning) and **Ansible** (configuration management) can work together:
  - Terraform provisions infrastructure (VMs, networks, storage, etc.), and Ansible provides ongoing configuration and management.



Mutable vs immutable infrastructure

- Configuration management tools typically use a **mutable infrastructure** approach.
  - Infrastructure can be modified after deployment (e.g., applying updates, patches, or configuration changes)
  - Changes are made *in place*, meaning existing resources are updated rather than replaced.

- Provisioning tools employ an **immutable infrastructure** approach.
  - Infrastructure cannot be changed after deployment.
    - "Changes" involve replacing the previous resource with a new version.
  - No **configuration drift**, since each deployment starts from a fresh, predefined state.

Procedural vs declarative

- **Procedural approach** (a.k.a. Imperative)
  - Follows **explicit steps** in a **specific order** to achieve the desired outcome.
  - The user must define each action to configure the infrastructure.
  - Provides greater control compared to a declarative approach.

- **Declarative approach**
  - Defines the **desired end state**.
  - The tool (e.g., Terraform) figures out the steps needed to achieve the goal.
  - Easier to maintain and ensures consistency across deployments.

- A **procedural** approach focuses on **how** to make changes (explicit steps).
  - "Configure router's hostname with **`hostname R1`**"
  - "Configure the IP address of G0/1 with **`ip address 192.168.1.1 255.255.255.0`**"
  - "Enable the interface with **`no shutdown`**"
  - etc.
- A **declarative** approach focuses on **what** the final state should be.
  - "Create a router named R1 with IP address 192.168.1.1/24 on its G0/1 interface, ensuring the interface is enabled".

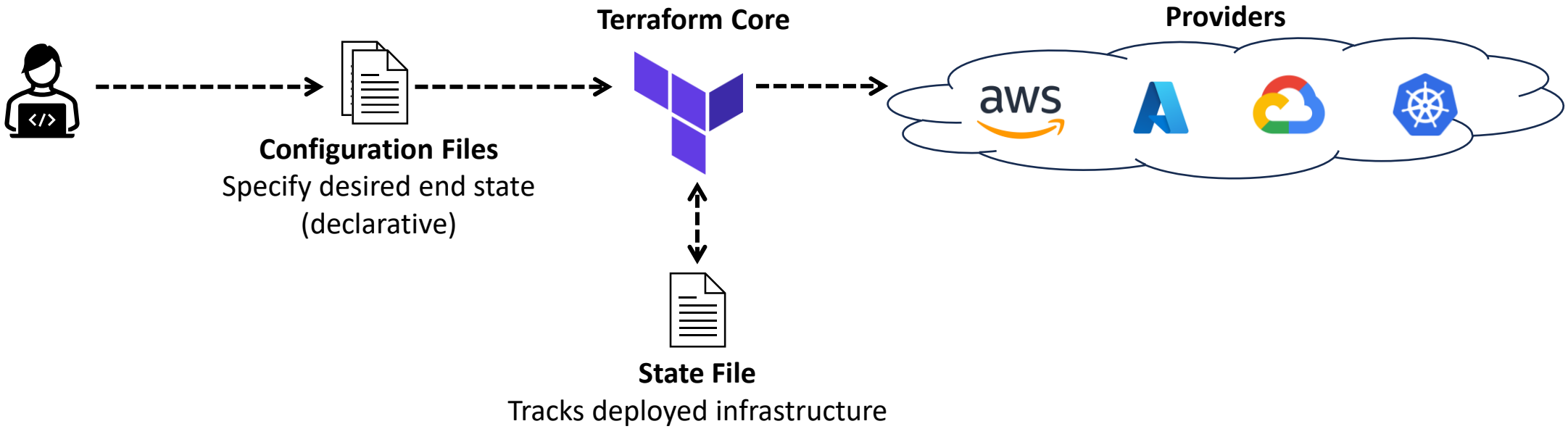
PROCEDURAL	DECLARATIVE
Ansible	Terraform
Chef	Puppet

Terraform

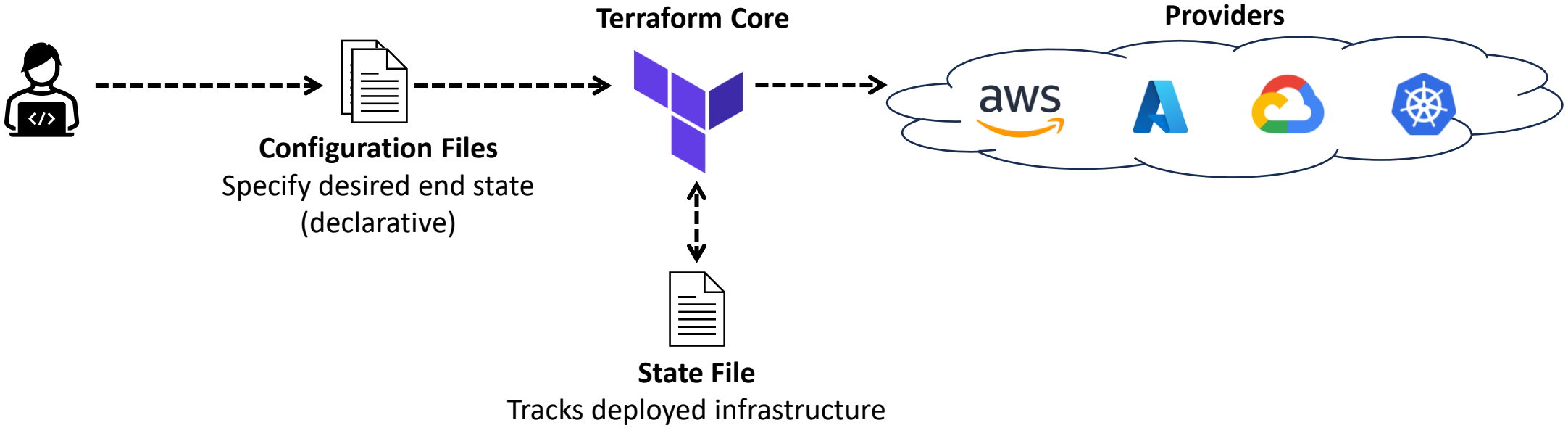
- **Terraform** is an open-source IaC tool developed by HashiCorp (acquired by IBM in 2025).

- It is primarily a **provisioning** tool, focused on deploying infrastructure resources on various cloud & on-prem platforms.
  - These platforms are called **providers** and include AWS, Azure, GCP, Kubernetes, and many more (1000+).
    - This includes integrations with Cisco platforms like **Catalyst Center**, **ACI**, and **IOS XE**.

- Like Ansible, Terraform uses a **push** model and is **agentless**; it doesn't require a software agent on infrastructure it provisions or manages.



Terraform



- The basic Terraform workflow consist of three main steps:
  - **Write:** Define the desired state of your infrastructure resources in configuration files.
  - **Plan:** Verify the changes that will be executed before applying them.
  - **Apply:** Execute the plan to provision and manage the infrastructure resources.

- While Terraform Core is written in **Go**, configuration files are written in **HashiCorp Configuration Language (HCL)**.
  - HCL is an example of a **domain-specific language (DSL)**, a type of language specialized for a particular purpose.
    - Because DSLs are specialized, they allow users to perform complex tasks with much less effort than a general language (like Python or Go) would require.

Summary

- **Infrastructure as Code (IaC)** is the practice of provisioning and managing infrastructure (servers, networks, cloud resources) using machine-readable configuration files (code) instead of manual configuration (e.g., CLI/GUI).

- **Configuration management** tools (e.g., Ansible, Puppet, Chef) focus on managing existing infrastructure by installing software, configuring settings (e.g., router configurations), and maintaining system state.
- **Infrastructure provisioning** tools (e.g., Terraform) focus on creating, modifying, and deleting infrastructure resources.

- **Mutable infrastructure** can be modified after deployment (e.g., applying updates, patches, or configuration changes).
- **Immutable infrastructure** cannot be changed after deployment; changes involve replacing the previous resource with a new version.

- A **procedural** (imperative) approach follows explicit steps in a specific order to achieve the desired outcome.
- A **declarative** approach defines the desired end state, and the IaC tool figures out the steps needed to achieve the goal.

- **Terraform** is an open-source IaC tool developed by HashiCorp.
  - It is primarily a provisioning tool used for deploying infrastructure on **providers** like AWS, Azure, GCP, Kubernetes, etc.
    - It interacts with these providers via their APIs.
  - Like Ansible, it uses a **push** model and is **agentless**.
  - Main components: **Terraform Core**, **configuration files**, **state file**, **providers**.
  - The basic Terraform workflow consists of three main steps:
    - **Write:** Define the desired state of your infrastructure resources in configuration files.
    - **Plan:** Verify the changes that will be executed before applying them.
    - **Apply:** Execute the plan to provision and manage the infrastructure resources.
  - Terraform Core is written in **Go**, and configuration files are written in **HashiCorp Configuration Language (HCL)**, a DSL.

Quiz 1

Which of the following are characteristics of Terraform? (select two)

- A) Procedural
- B) Declarative
- C) Immutable infrastructure
- D) Mutable infrastructure

Quiz 2

What are three the main steps of the Terraform workflow?

- A) Write, Test, Apply
- B) Plan, Test, Apply
- C) Plan, Write, Apply
- D) Write, Plan, Apply

Quiz 3

Which of the following IaC tools are focused on configuration management? (select three)

- A) Ansible
- B) Terraform
- C) Chef
- D) Puppet