Infrastructure as Code (IaC) Using Terraform

Vasanth Babu R
rvasanthbabu@gmail.com
https://github.com/rvasanthbabu

Agenda

- Infrastructure evolution & overview of IaC
- IaC on public clouds(AWS, GCP, Azure)
- IaC Tools comparison
- Terraform basics
- Terraform Architecture
- Terraform workflow
- Terraform key terminologies
- Terraform Editions
- CDK for Terraform

Infrastructure evolution & overview of IaC

What is Infrastructure as Code(IaC)?

It is infrastructure (CPUs, memory, disk, firewalls, etc.) defined as code within definition files.

Advantages of Infrastructure as Code(IaC)

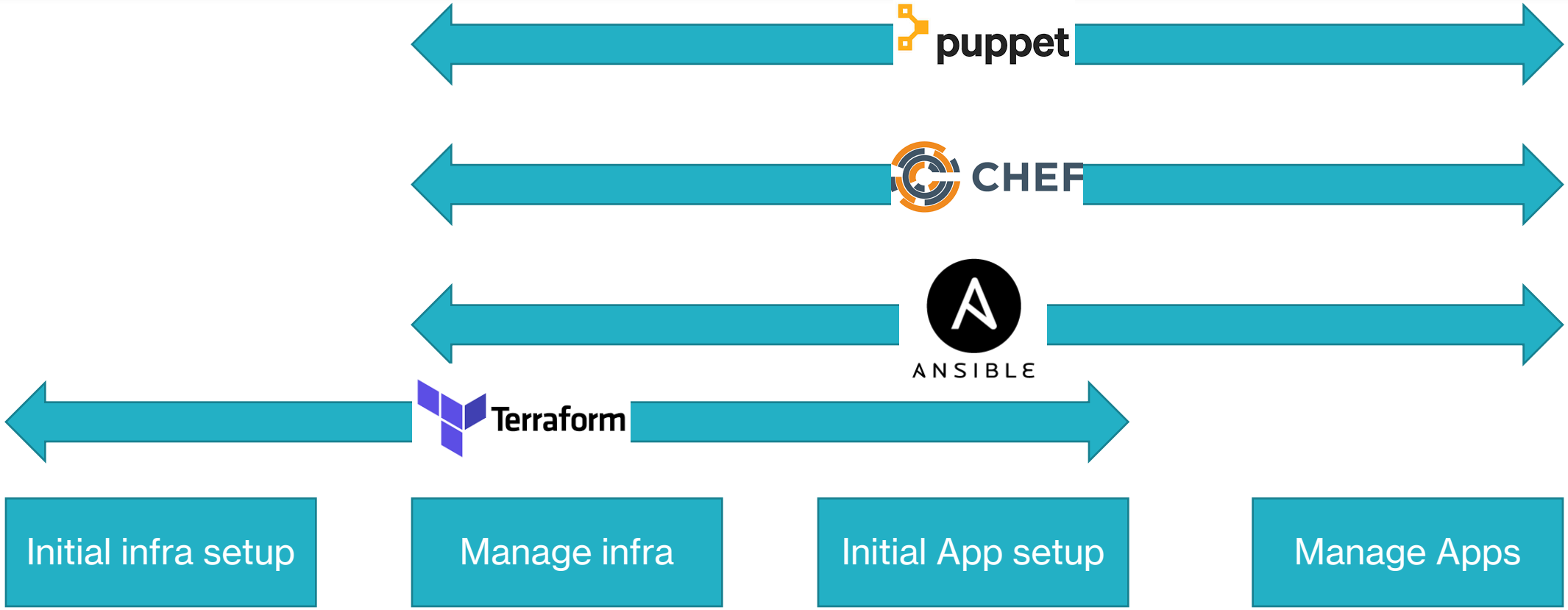
- Makes Infrastructure More Reliable
  - Makes changes idempotent, consistent, repeatable, and predictable
  - We can test the code and review the results before deployment
  - Ensuring consistency and repeatability
  - Can be version controlled
- Makes Infrastructure More Manageable
  - Only the necessary changes will be applied, leaving existing, valid infrastructure untouched.

IaC on public clouds(AWS, GCP, Azure)

	AWS	GCP	Azure
Tools	CloudFormation	Cloud Deployment Manager	Azure Resource Manager
Format	YAML	Jinja / Python Templates	JSON

NOTE: Learning cloud specific IaC tool is difficult as each are different

IaC Tools comparison



Terraform basics

What is terraform?

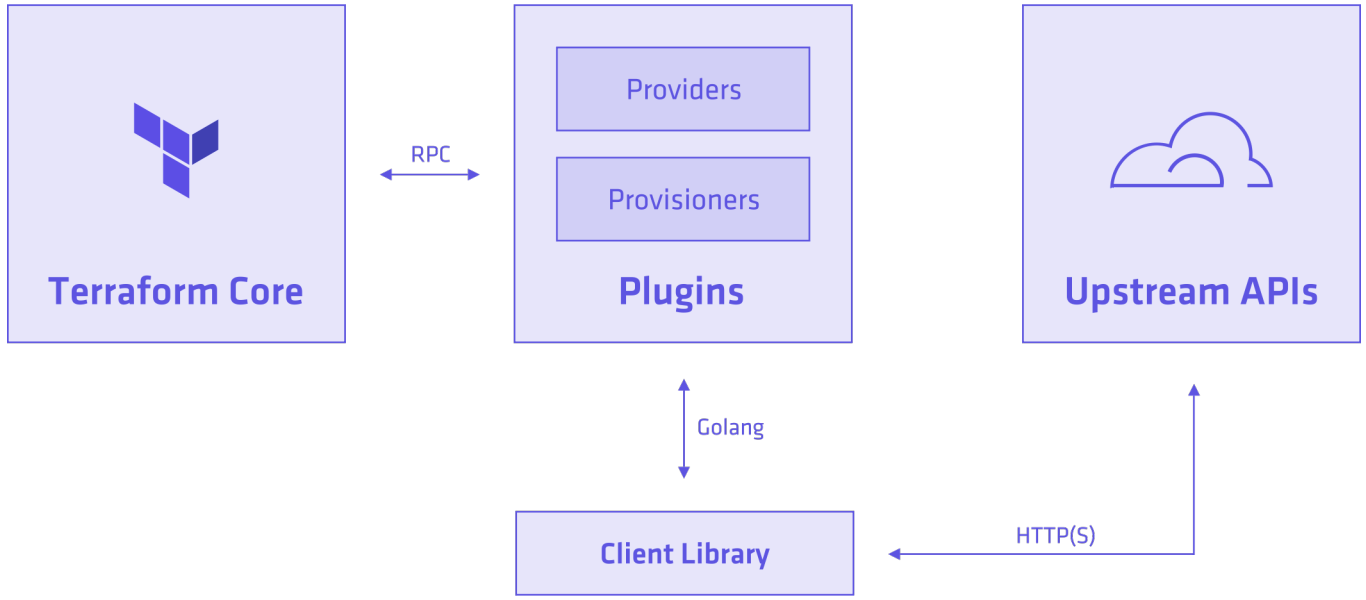
- Terraform is an infrastructure as code tool that lets you **define both cloud and on-prem resources** in **human-readable configuration files** that you can **version, reuse, and share.**
- Configurations are **written in declarative** Hashicorp configuration Language(**HCL**)
- You can then use a **consistent workflow** to provision and manage all of your infrastructure throughout its lifecycle.
- Terraform can manage low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features.

Why terraform?

- Manage any infrastructure
  - Via providers in terraform registry
- Track your infrastructure
  - track of your real infrastructure in a **state file**, which acts as a source of truth for your environment.
- Automate changes
  - Declarative approach-No need to write step-by-instructions
- Standardize configurations
  - supports reusable configuration components called **modules**
- Collaborate
  - Config files can be version controlled

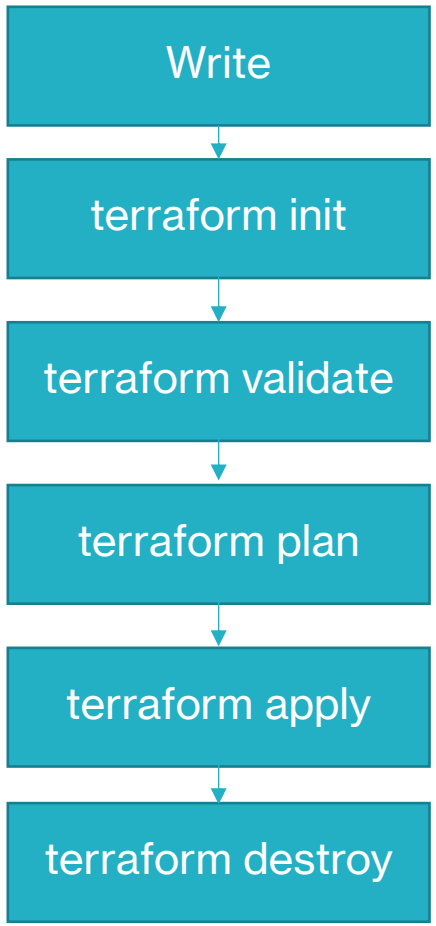
Terraform Architecture

- **Terraform creates and manages resources** on cloud platforms and other services through their **APIs**.
- **Providers** enable Terraform to **work with virtually any platform or service** with an accessible API.
- Terraform community have already written more than **1700 providers**. You can find all publicly available providers on the **Terraform registry.**



- **Terraform Core** reads the configuration and builds the resource dependency graph.
- **Terraform Plugins** (providers and provisioners) bridge Terraform Core and their respective target APIs. Terraform provider plugins implement resources via basic CRUD (create, read, update, and delete) APIs to communicate with third party services

Terraform workflow



Write terraform configuration code

Downloads plugins & initialize a working directory containing Terraform configuration files.

Validates the configuration files in a directory

Creates an execution plan, which lets you preview the changes that Terraform plans to make to your infrastructure.

Executes the actions proposed in a Terraform plan.

Destroys all remote objects managed by a particular Terraform configuration.

Terraform key terminologies

Provider

- Acts as a **translation layer** that allows Terraform to communicate with many different cloud providers, databases, and services.

Resource

- Each resource **block describes one or more infrastructure objects**, such as virtual networks, compute instances, or higher-level components such as DNS records.

Modules

- A Terraform module is a **set of Terraform configuration files in a single directory**.
- They enable reusability.

Outputs

- Output Values are like **return values for a Terraform module**.

Input Variables

- Input Variables serve as **parameters for a Terraform module**, so users can customize behaviour without editing the source.

Workspaces

- Workspaces in Terraform are simply **independently managed state files**.
- A workspace contains everything that Terraform needs to manage a given collection of infrastructure

Provisioners

- Terraform Provisioners are **used to performing certain custom actions and tasks either on the local machine or on the remote machine**.

Terraform Editions

- Terraform Open Source
  - **Command line** tool that lets you **provision infrastructure** on any cloud provider and manages configuration, plugins, infrastructure, and state.
- Terraform Cloud
  - **SaaS application** that runs Terraform in a stable, remote environment and securely stores state and secrets.
  - It includes a **rich user interface** that helps you better understand your Terraform operations and resources, allows you to define role-based access controls, and offers a private registry for sharing modules and providers.
  - **Support for version control systems** (VCS) like GitHub, GitLab, and Bitbucket.
  - Cost estimator
  - Define policy using sentinel framework
- Terraform Enterprise
  - Terraform Enterprise allows you to set up **a self-hosted distribution of Terraform Cloud.**
  - It offers **customizable resource limits** and is ideal for organizations with strict security and compliance requirements.

For detailed report refer : https://amazic.com/terraform-editions-explained-cloud-enterprise-and-oss

CDK for Terraform