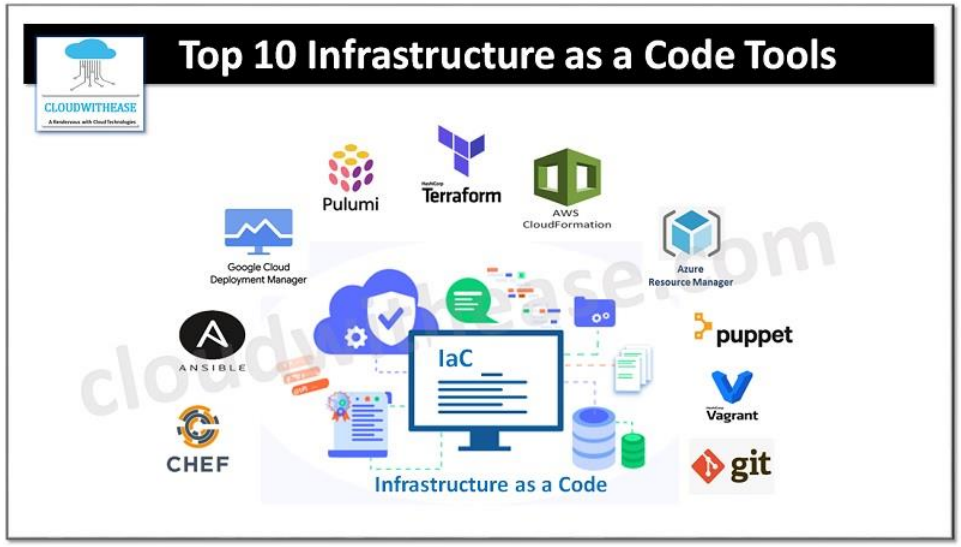
Infrastructure As Code (IAC)

What is infrastructure as code?



Infrastructure as Code or IAC is the process of provisioning and managing infrastructure defined through code, instead of doing so with a manual process. As infrastructure is defined as code, it allows users to easily edit and distribute configurations while ensuring the desired state of the infrastructure. This means you can create reproducible infrastructure configurations.

Infrastructure as code benefits

1. Decreased risk

Provisioning your entire infrastructure by hand is risky. It requires manual work that is error-prone. It may require a single person to do. That person could leave the company, taking all that knowledge with them. Infrastructure as code minimizes both of these risks. By representing infrastructure as reproducible blocks of code we are far less error-prone. Infrastructure as code lives in a source code repository. Its history and changes are visible to everyone on the team.

2. Stable & consistent environments for faster iterations

When environments have to be manually configured or modified it slows down product development. This is especially true if the product wants to change its architecture to better serve its users. With infrastructure as code environments are stable, consistent, and easily modifiable. They live in code alongside the product, so when we want to change one we can change the other at the same time. This harmony means that new features can be developed for the product faster. There is less overhead to managing a given environment.

3. Cost optimization

When all resources are represented in code you can see what is running and what shouldn't be. Optimizing cost maintains product profit margins. Those optimizations become much easier with infrastructure as code.

4. Self-documenting

There is a philosophy in software development that says good code is easy to read. It often doesn't need extensive comments because it's clear what it's doing. The idea is that a new developer should be able to come in, read the code, and understand the logic that is happening. With infrastructure as code, it is self-documenting like any other code. This can benefit the product by making it easier to add more people to the team. With self-documenting code, you can reduce the time it takes for a new developer to onboard into the team.

5. Enhanced security

If all compute, storage, and networking services are provisioned with code, then they are deployed the same way every time. This means that security standards can be easily and consistently deployed across company without having to have a security gatekeeper review and approve every change.

Infrastructure as Code tools & platforms

Under the big IaC umbrella, there are all sorts of tools, from dedicated infrastructure management tools to configuration management, from open-source tools to platform-specific IaC options.

Let's look at some of the most popular IaC tools and platforms.

Terraform

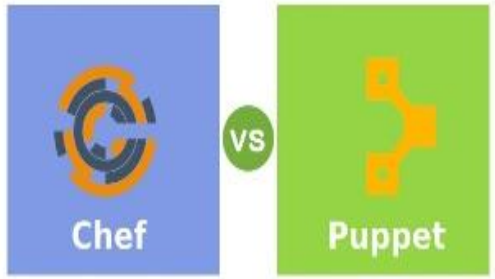
Terraform by HashiCorp is the leading IaC tool specialized in managing infrastructure across various platforms from AWS, Azure, GCP to Oracle Cloud, Alibaba Cloud, and even platforms like Kubernetes.

Ansible

Ansible is not a dedicated Infrastructure management tool but more of an open-source configuration management tool with IaC capabilities. Ansible supports both cloud and on-prem environments and can act through SSH or WinRM as an agentless tool.

Chef/Puppet

Chef and Puppet are two powerful configuration management tools. Both aim to provide configuration management and automation capabilities with some infrastructure management capabilities across the development pipeline.



AWS CloudFormation

CloudFormation is the AWS proprietary platform specific IaC tool to manage AWS infrastructure. CloudFormation has deep integration with all AWS services and can facilitate any AWS configuration as a first-party solution.

Categories of IAC

Let us now see the four methodologies of Infrastructure as Code,

Ad Hoc Scripts :

Ad Hoc Scripting is the most straightforward approach for the automation of processes. These scripts convert manual processes to automated processes just by simply breaking them down into break steps. You can achieve this with the help of scripting languages like Ruby, Python, Bash, PowerShell, etc.

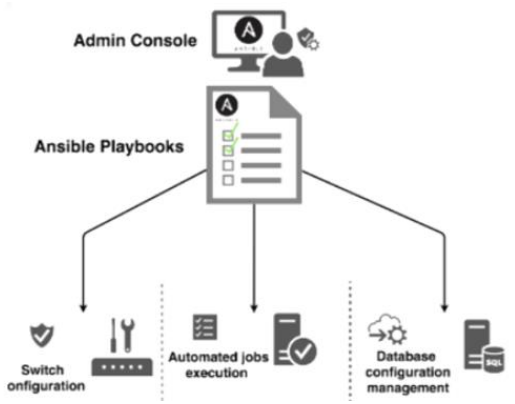
```
# Update the apt-get cache
sudo apt-get update

# Install PHP
sudo apt-get install -y php

# Install Apache
sudo apt-get install -y apache2

# Start Apache
sudo service apache2 start
```

Configuration Management Tools



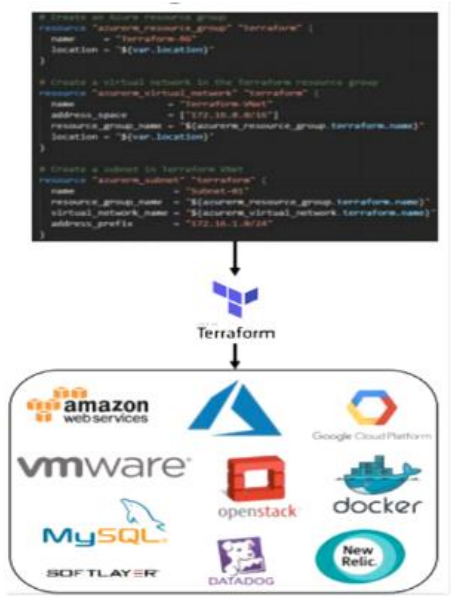
Configuration Management Tools are designed to install and manage software on existing servers. These tools have a code with a consistent and predictable structure, clearly named parameters, proper secrets management, and even a simple file layout.

Some of the Configuration Management tools include Ansible, Chef, Puppet, and SaltStack. These tools are also known as Idempotent codes.

Server Templating

Server Templating tools are used to create an image of a server. This image captures a self-contained 'snapshot' of the operating system, software, files, and all other relevant details. They are considered to be an alternative to Configuration Management Tools. Some of the popular Server Templating tools include Docker,

- **Docker:** Docker is used to create isolated environments for applications called containers.

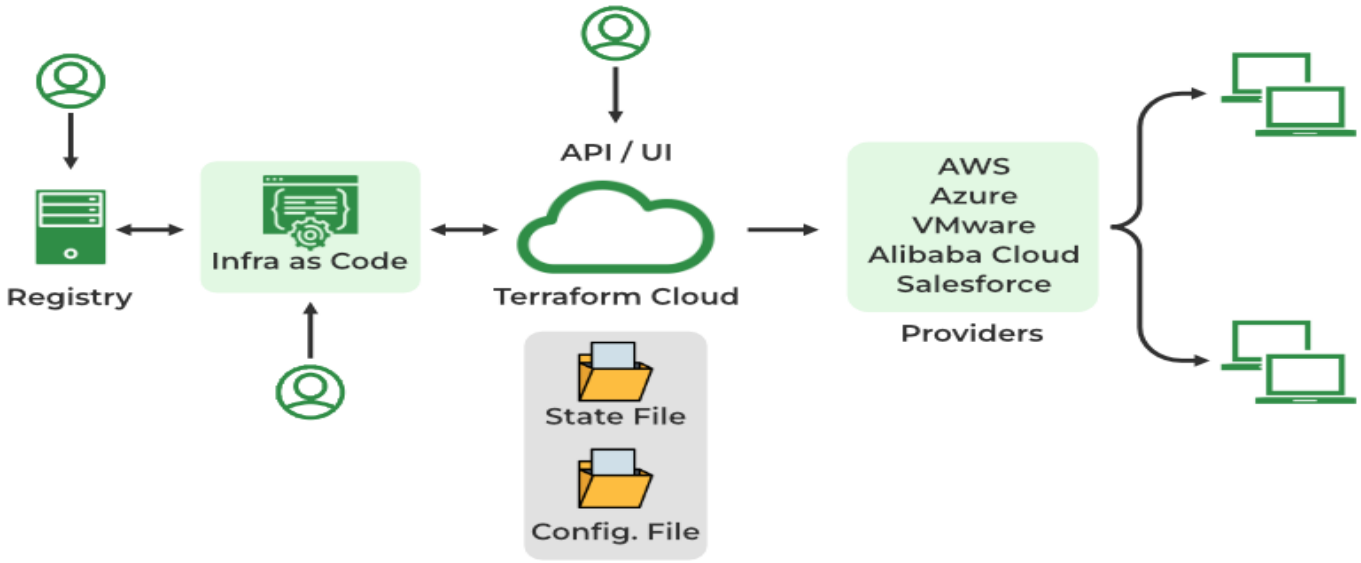


Server Provisioning

Server provisioning is the process of setting up a server so that it could be used in a network based on required resources. It consists of all the operations needed to create a new machine and bring it to a working state and includes defining the desired state of the system.

How Terraform Works?

Terraform has two main components that make up its architecture:



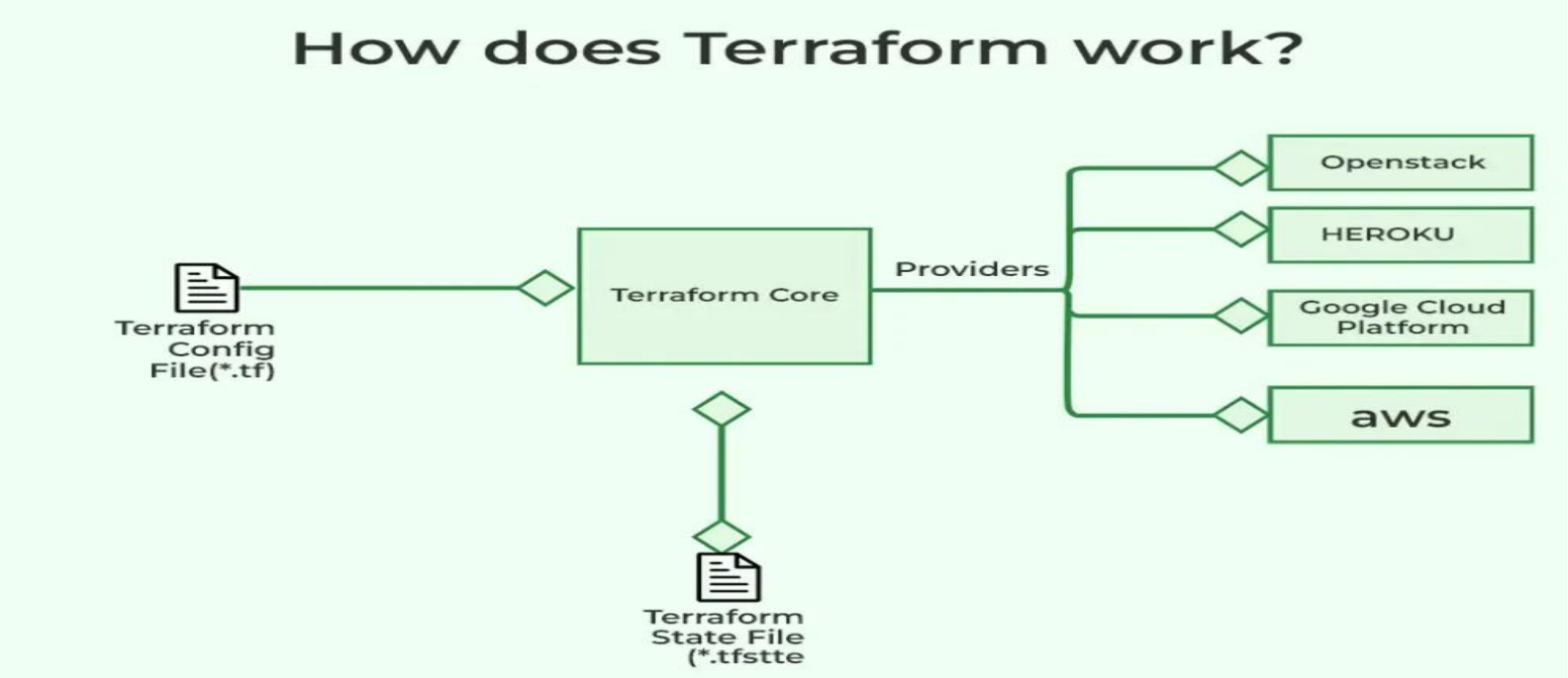
- **CORE**

Terraform's Core takes two input sources, which are your configuration files (your desired state) and second the current state (which is managed by Terraform).
With this information the Core then creates a plan of what resources needs to be created/changed/removed.

- **Provider**

The second part of the Architecture is providers. Providers can be **IaaS** (like AWS, GCP, Azure), PaaS (like Heroku, Kubernetes) or SaaS services (like Cloudflare).
Providers expose resources, which makes it possible to create infrastructure across all this platforms.

State File: Terraform stores information about your infrastructure in a state file. This state file keeps track of resources created by your configuration files and maps them to real-world resources



Choosing A Right IaC Tool?

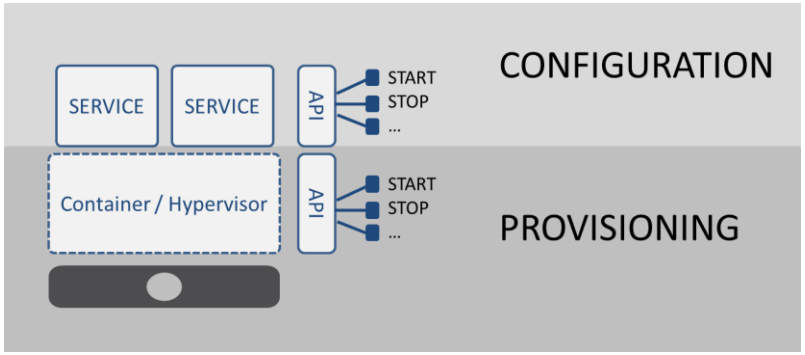
There are a lot of **IaC (Infrastructure as Code)** tools available in the market and we will be discussing a few of them in this slide,like **Terraform**, **Chef**, **Puppet**, **Ansible,** and **CloudFormation** and will try to resolve the dilemma of which one to pick for the automation of your cloud resources.

Terraform is an open-source, cloud-agnostic provisioning tool that supported immutable infrastructure, a declarative language, a masterless and agentless architecture, and had a large community and a mature codebase.

Terraform

CHEF ANSIBLE CloudFormation puppet

Configuration Management vs Provisioning



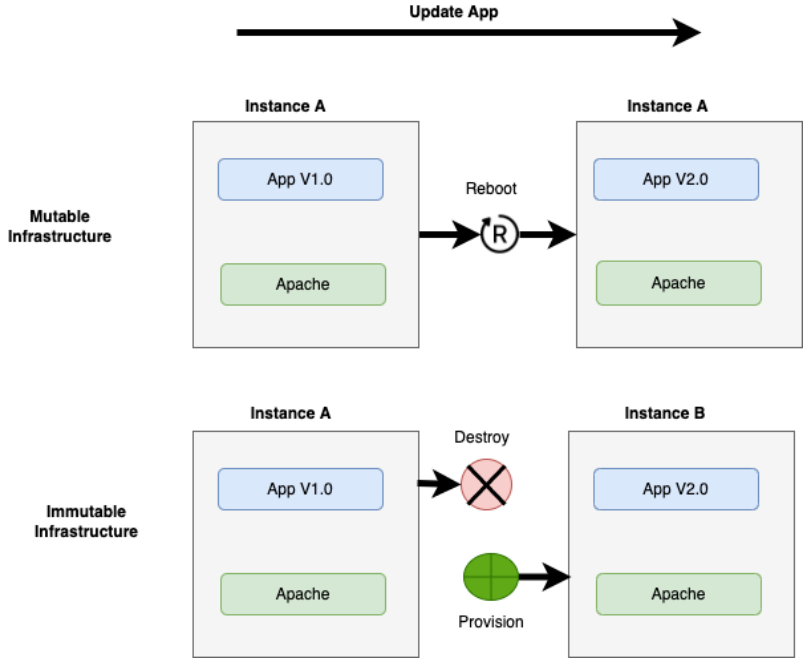
Provisioning: In IT, provisioning is the process of creating infrastructure and making it available to end users.

Configuration: It is the process of configuring the provisioned IT infrastructure resources

Chef, Puppet, and Ansible are all **configuration management tools**, designed to install and manage software on existing servers whereas CloudFormation and Terraform are **provisioning tools**.

most of the time a good alternative is to use a configuration management and provisioning tool together. For example, using Terraform to provision your servers while running Ansible to configure them.

Mutable vs Immutable Infrastructure



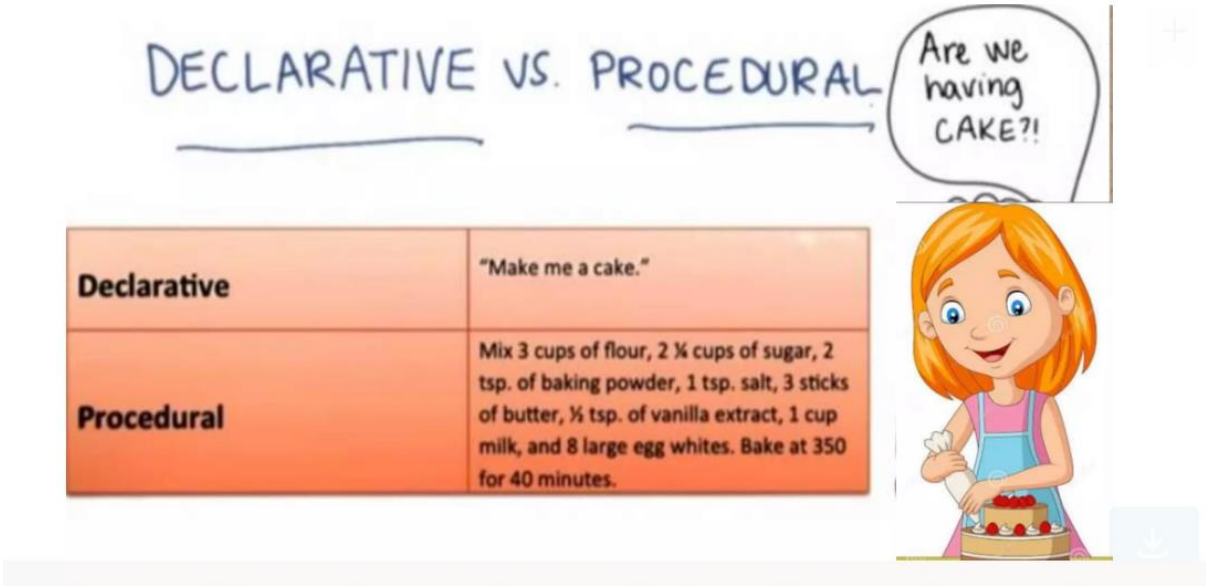
Mutable: Something that can be changed. Meaning you can continue to make changes to it after it is created.

Immutable: Something that cannot be changed. Once it is created, you cannot change anything in that.

Configuration management tools such as Chef, Puppet, and Ansible typically creates a **mutable** infrastructure. For example, using Chef to install a new version of a software, it'll run the software update on the existing servers and the changes will happen in-place.

While in Terraform, every “change” is the deployment of a new server. **Immutable** components are recreated and replaced instead of updating in-place the existing components. Here, the servers are never modified after they’re deployed.

Procedural vs Declarative



The **imperative/procedural** approach takes action to configure systems in a series of actions.

The **declarative approach** requires that users specify the end state of the infrastructure they want, and then Terraform makes it happen.

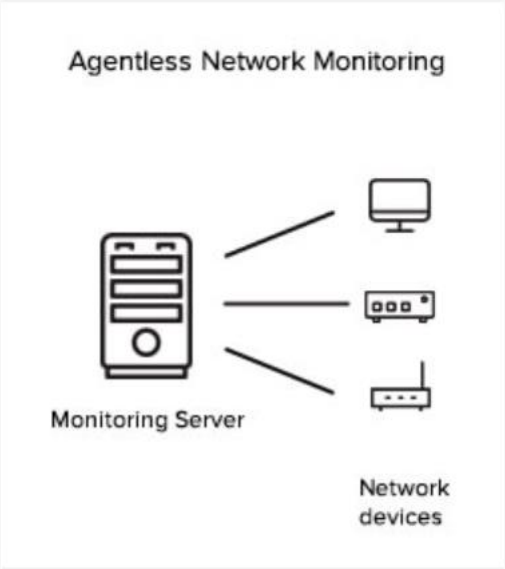
Chef and Ansible encourage a *procedural* style where you write code that specifies, step-by-step, how to to achieve some desired end state.
Terraform, CloudFormation, Pulumi, Heat, and Puppet all encourage a more *declarative* style where you write code that specifies your desired end state, and the IAC tool itself is responsible for figuring out how to achieve that state.

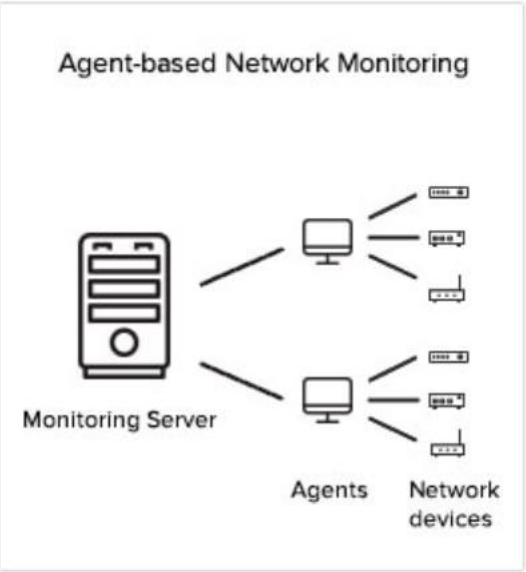
Agent vs Agentless

Chef and Puppet require you to install **agent** software (e.g., Chef Client, Puppet Agent) on each server you want to configure. The agent typically runs in the background on each server and is responsible for installing the latest updates.

Ansible, CloudFormation, and Terraform do not require you to install any extra agents. As in Terraform, you just issue commands and the cloud provider's agents execute them for you on all of your servers. With Ansible, your servers need to run the SSH Daemon, which is common to run on most servers anyway.

Agentless, meaning that it does not install software on the nodes that it manages





Lab: installation of Terraform.