Infrastructure As Code:

Infrastructure as Code (IaC) is managing and provisioning of infrastructure through code instead of manual processes, with IaC configuration files are created that contains infrastructure specifications, which makes easier to maintain and distribute. Version control is an important part of IaC and the configuration files should be under source control just like any other software source code files. It is one of the key DevOps practices that enable teams to deliver infrastructure, and the software running on it, rapidly and reliably, at scale

Key Principles:

Idempotency:

Idempotency means no matter how many times we run IaC configuration and, what’s our starting state is, will end up with the same end state. This simplifies the provisioning of Infrastructure and reduces the chances of inconsistent results. Idempotency can be achieved by using a stateful tool with a declarative language, like Terraform, we defined desired end state of the infrastructure we want, and it is Terraforms job to get us to that end state.

Immutability:

Configuration drift is a huge problem with infrastructure, it occurs when over a period there are changes made to infrastructure that are not recorded, and various environments drift from each other in ways that are not easily reproducible. These issues can be resolved by using immutable infrastructure.

Immutable infrastructure means instead of changing an existing infrastructure we replace it with new. By provisioning new infrastructure every time, we are making sure it is reproducible and doesn’t allow for configuration drift over time.

#### Benefits of IaC:

* Cost reduction
* Increase in speed of deployments
* Reduce errors
* Improve infrastructure consistency
* Eliminate configuration drift

Approaches for writing Infrastructure As Code?

There are 2 approaches for writing IaC

1. Declarative or Functional: The declarative approach defines the desired state of the target, i.e., what should be the target actual configuration. The steps to set up are not defined, Instead the list of requirements of third-party software’s needed to setup the infrastructure or server is needed.
2. Imperative or Procedural: The imperative approach defines the commands that must be executed to achieve the desired result. The imperative focuses on how the infrastructure is to be changed to meet the desired result.

Server automation and configuration management tools can often be used to achieve IaC.

These are a few popular choices:

* Ansible
* Terraform
* Azure Source Manager
* Google Cloud Deployment manager
* AWS CloudFormation
* Chef
* Puppet
* Red Hat Ansible Automation Platform
* Saltstack
* Vagrant

How to choose best IaC tools?

As mentioned above there are different types of tools in IaC, because not the same tool can perform all the tasks. So tools are categorized to 3 main categories

1. Infrastructure Provisioning
2. Configuration of provisioned infrastructure
3. Application deployment

The phases in which they automate the tasks

1. Initial Setup Phase
2. Maintaining Phase

There is one more category in which we can divide the tools is “how they work”

1. Declarative or Imperative
2. Mutable or Immutable
3. Agent or Agentless

Now let us understand when we should choose which IAC tool

### Configuration Management vs. Provisioning:

Ansible, Chef, Puppet, and salt stack are configuration management tools that can install and configure the applications on existing infrastructures. Whereas terraform and cloud formation are infrastructure provisioning tools, they can be used to provision the servers and other infrastructure like load balancers, databases, networking configuration. Ansible and Terraform are the tools that can perform both provisioning and Configuration management. Ansible and Terraform can work together. We can use Terraform for provisioning of infrastructure, [databases](https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjv9bbe3cnzAhWtuksFHYuFBsM4ChAWegQIEhAB&url=https%3A%2F%2Fcloud.google.com%2Fblog%2Fproducts%2Fdatabases&usg=AOvVaw2qXoZziWrBaiRYURZ4WFIa), load balancer, network topology, etc. Ansible can be used to deploy applications on top of these infrastructures.

Some tools can perform both tasks. For example, Terraform can do both infrastructure provisioning as well application deployment

Mutable Infrastructure Vs Immutable Infrastructure:

Chef, Puppet, Ansible, and SaltStack are all mutable infrastructure tools by nature. If you configure these tools to install a new version of an existing package or software, it will install the software update on our existing servers and make the changes in real-time.  
Whereas for a tool like Terraform (an immutable IAC tool), every “change” is a new server deployment.

Declarative Vs Imperative:

Chef and Ansible follow the imperative approach as we must write code that specifies the step-by-step to achieve the target state. Terraform, CloudFormation, SaltStack, and Puppet all use a declarative method, in which we write code that defines the target state we want to achieve, and the IAC tool figures out how to get there

Agent Vs Agentless:

Chef, Puppet, and SaltStack are agent-based tools that involve the installation of an agent on the target computer. This agent usually runs in the background on the target machine and carries out the operations necessary to achieve the goal.  
On the other hand, No agents are needed for Ansible, CloudFormation, or Terraform. Some of them need agents, but these are usually included in the infrastructure we use.

## Conclusion:

Infrastructure as Code is thus practice and process of representing the underlying resources or infrastructure in the form of Code that can be versioned and stored in the source control system

The only tool that satisfies all our requirements is Terraform.