

**Ansible (Config. Mgmt):**

Ansible is an open source automation platform. It is very, very simple to setup and yet powerful. Ansible can help you with configuration management, application deployment, task automation. It can also do IT orchestration, where you have to run tasks in sequence and create a chain of events which must happen on several different servers or devices.

**Example**:

An example is if you have a group of web servers behind a load balancer. Ansible can upgrade the web servers one at a time and while upgrading it can remove the current web server from the load balancer and disable it in your Nagios monitoring system. So in short you can handle complex tasks with a tool which is easy to use.

Another simple example, let's say if you want install nginx package in a group of servers, you can just do it from ansible and even you can upgrade/remove the existing packages.

There are lot of use cases in the below link

<https://www.ansible.com/use-cases>

**Features/Benefits:**

Unlike Puppet or Chef it doesn’t use an agent on the remote host. Instead Ansible uses SSH which is assumed to be installed on all the systems you want to manage, it’s written in Python. You don’t have to setup a client server environment before using Ansible, you can just run it from any of your machines and from the clients point of view there is no knowledge of any Ansible server.

It can do,

1. configuration management

2. provisioning

3. Application deployment

4. Orchestration

5. continuous delivery

6. Security

**Terraform (Infra as Code):**

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions.

Configuration files describe to Terraform the components needed to run a single application or your entire datacenter. Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, Terraform is able to determine what changed and create incremental execution plans which can be applied.

The infrastructure Terraform can manage includes low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc.

Examples work best to showcase Terraform. Please see the use cases give in below link.

<https://www.terraform.io/intro/use-cases.html>

**Sample use case:**

Disposable Environments

It is common practice to have both a production and staging or QA environment. These environments are smaller clones of their production counterpart, but are used to test new applications before releasing in production. As the production environment grows larger and more complex, it becomes increasingly onerous to maintain an up-to-date staging environment.

Using Terraform, the production environment can be codified and then shared with staging, QA or dev. These configurations can be used to rapidly spin up new environments to test in, and then be easily disposed of. Terraform can help tame the difficulty of maintaining parallel environments, and makes it practical to elastically create and destroy them.

**Features/Benefits:**

» Infrastructure as Code

Infrastructure is described using a high-level configuration syntax. This allows a blueprint of your datacenter to be versioned and treated as you would any other code. Additionally, infrastructure can be shared and re-used.

» Execution Plans

Terraform has a "planning" step where it generates an execution plan. The execution plan shows what Terraform will do when you call apply. This lets you avoid any surprises when Terraform manipulates infrastructure.

» Resource Graph

Terraform builds a graph of all your resources, and parallelizes the creation and modification of any non-dependent resources. Because of this, Terraform builds infrastructure as efficiently as possible, and operators get insight into dependencies in their infrastructure.

» Change Automation

Complex changesets can be applied to your infrastructure with minimal human interaction. With the previously mentioned execution plan and resource graph, you know exactly what Terraform will change and in what order, avoiding many possible human errors.

**Jenkins (CI):**

What is CI?

Basically Continuous Integration is the practice of running your tests on a non-developer machine automatically everytime someone pushes new code into the source repository.

This has the tremendous advantage of always knowing if all tests work and getting fast feedback. The fast feedback is important so you always know right after you broke the build (introduced changes that made either the compile/build cycle or the tests fail) what you did that failed and how to revert it.

If you only run your tests occasionally the problem is that a lot of code changes may have happened since the last time and it is rather hard to figure out which change introduced the problem. When it is run automatically on every push then it is always pretty obvious what and who introduced the problem.

Built on top of Continuous Integration are Continuous Deployment/Delivery where after a successful test run your instantly and automatically release the latest version of your codebase. Makes deployment a non-issue and helps you speed up your development.

What is jenkins?

Jenkins is an open-source continuous integration software tool written in the Java programming language for testing and reporting on isolated changes in a larger code base in real time. The software enables developers to find and solve defects in a code base rapidly and to automate testing of their builds.

**Benefits/Features:**

Jenkins is an open source tool with much support from its community.

Installation is easier.

Platform Independent.

It has more than 1000 plug-in to make the work easier.

It is a tool which is written in Java. Hence it can be portable to almost all major platforms.