**Infrastructure as Code (IaC)**

* Executive Summary
* IaC Categories
  + Infrastructure as a Service (IaaS)
  + Platform as a Service (PaaS)
  + Software as a Service (SaaS)
* Viewpoint and approach
* Benefits of IaC
* Risks Involved with IaC
* Current State of IaC
* Infrastructure Provisioning using Terraform
  + what is terraform
  + Why we need terraform
  + Where we can use
  + Advantages
* Configuration Management using Ansible
  + what is ansible
  + Why we need ansible
  + Where we can use
  + Advantages
* CICD Workflow and Demo with IaC

**Executive Summary:**

Businesses must deliver software with higher quality and velocity to their target clientele. One

way to enable this is to automate the infrastructure setup and application configuration and

deployment. Each stage of the software development lifecycle (SDLC) from local dev to

acceptance and eventually to production is performed in an automated, repeatable, and

deterministic manner. This process supports the enabling capability of Infrastructure as Code

(IaC).

Infrastructure as Code (IaC) is a method to provision and manage IT infrastructure through the use of source code, rather than through standard operating procedures and manual processes.

You’re basically treating your servers, databases, networks, and other infrastructure like software. And this code can help you configure and deploy these infrastructure components quickly and consistently.

IaC helps you automate the infrastructure deployment process in a repeatable, consistent manner, which has many benefits.

The concept of IaC is that the description of the infrastructure resources and their topology is

modeled through code and then managed like software. These descriptor files are managed

through version control. Executions are orchestrated via configuration management and

Continuous Integration/Continuous Deployment (CI/CD) platforms.

**IaC Categories:**

IaC processes fall into three categories:

Infrastructure as a Service (IaaS)

Platform as a Service (PaaS)

Software as a Service (SaaS).

**Infrastructure as a Service (IaaS):**

Cloud infrastructure services, known as Infrastructure as a Service (IaaS), are self-service models for accessing, monitoring, and managing remote datacenter infrastructures, such as compute (virtualized or bare metal), storage, networking, and networking services (e.g. firewalls). Instead of having to purchase hardware outright, users can purchase IaaS based on consumption, similar to electricity or other utility billing.

Compared to [SaaS](https://apprenda.com/white-papers/saas-hub/?utm_source=library&utm_medium=post&utm_term=saaspaasiaas&utm_campaign=saas-hub) and [PaaS](https://apprenda.com/white-papers/streamlining-software-development-lifecycle-sdlc-process-private-paas/?utm_source=library&utm_medium=post&utm_term=saaspaasiaas&utm_campaign=sdlcwp), IaaS users are responsible for managing applications, data, runtime, middleware, and OSes. Providers still manage virtualization, servers, hard drives, storage, and networking. Many IaaS providers now offer databases, messaging queues, and other services above the virtualization layer as well. What users gain with IaaS is infrastructure on top of which they can install any required platform. Users are responsible for updating these if new versions are released.

The IaaS technology helps the users to avoid the cost and complexity of purchasing and managing their own physical servers.

IaaS Examples: Amazon Web Services (AWS), Cisco Metapod, Microsoft Azure, Google Compute Engine (GCE)

Common IaaS Use-Case: Extends current data center infrastructure for temporary workloads (e.g. increased Christmas holiday site traffic)

[Infrastructure-as-a-Service (IaaS)](https://www.ibm.com/cloud/infrastructure) is a cloud-computing offering in which a vendor provides users access to computing resources such as servers, storage and networking. Organizations use their own platforms and applications within a service provider’s infrastructure.

Key features

* Instead of purchasing hardware outright, [users pay for IaaS on demand](https://cloud.ibm.com/docs/vsi/vsi_about_transient.html#transient-virtual-servers).
* Infrastructure is scalable depending on processing and storage needs.
* Saves enterprises the costs of buying and maintaining their own hardware.
* Because data is on the cloud, there can be no single point of failure.
* Enables the virtualization of administrative tasks, freeing up time for other work.

**Platform as a Service (PaaS):**

[Platform as a service (PaaS)](https://www.ibm.com/blogs/cloud-computing/2014/02/what-is-platform-as-a-service-paas/) is a cloud computing offering that provides users with a cloud environment in which they can develop, manage and deliver applications. In addition to storage and other computing resources, users are able to use a suite of prebuilt tools to develop, customize and test their own applications.

Key features

* PaaS provides a platform with tools to test, develop and host applications in the same environment.
* Enables organizations to focus on development without having to worry about underlying infrastructure.
* Providers manage security, operating systems, server software and backups.
* Facilitates collaborative work even if teams work remotely.

**Enterprise PaaS Examples:** [Apprenda](https://apprenda.com/?utm_source=library&utm_medium=post&utm_term=saaspaasiaas&utm_campaign=apprenda-com)  
**Common PaaS Use-Case:** Increases developer productivity and utilization rates while also decreasing an application’s time-to-market

[**Software as a service (SaaS)**](https://www.ibm.com/cloud/saas)**:**

[Software as a service (SaaS)](https://www.ibm.com/cloud/saas)  is a cloud computing offering that provides users with access to a vendor’s cloud-based software. Users do not install applications on their local devices. Instead, the applications reside on a remote cloud network accessed through the web or an API. Through the application, users can store and analyze data and collaborate on projects.

Key features

* SaaS vendors provide users with software and applications via a subscription model.
* Users do not have to manage, install or upgrade software; SaaS providers manage this.
* Data is secure in the cloud; equipment failure does not result in loss of data.
* Use of resources can be scaled depending on service needs.
* Applications are accessible from almost any internet-connected device, from virtually anywhere in the world.

**SaaS Examples:** Google Apps, Salesforce, Workday, Concur, Citrix GoToMeeting, Cisco WebEx, Adobe  
**Common SaaS Use-Case:** Replaces traditional on-device software

we believe successful software & solution firms will embrace these concepts to

realize the following improvements:

* Eliminate Waste - IaC will allow our clients to eliminate manual tasks and increase process velocity.
* Reduce Errors - Reduce errors associated with manual infrastructure, platform, and software provisioning, deployment, and modification.
* Increase Efficiency - Standardized configurations combined with automation enable teams to deliver higher quality at greater business velocity. This provides higher confidence regarding the changes moving through the delivery pipeline for both a bespoke or COTS solution.

These capabilities are critical as organizations take on projects such as data center migrations,

migrating to the cloud, creating a self-service portal for developers, automating provisioning

and deployments to environments (either Greenfield or Legacy), or adopting new technologies

like containers or microservices. Without IaC, organizations will not keep up with the demands

of the business to deliver faster with higher productivity and will put themselves at higher risk

to quality of service, compliance, or security violations.

**Viewpoint and Approach:**

1. Define the degree of infrastructure configuration that'll be exposed using IaC approaches. It's typically not as extensible as public clouds, but not static, either. What's controlled, and by which mechanisms?
2. Define the configuration management approach, as well as tool sets to leverage. They should be able to record thousands of instances of infrastructure configuration, with the understanding that the infrastructure will ultimately be configured through complete automation, and is thus forever dynamic.
3. Define how the developers and DevOps organization should use these mechanisms, while placing governance and policies over and above the use of IaC.
4. Create automated processes for monitoring and accounting of IaC and DevOps. You need to determine what's working and what needs improvement.

**We believes the following high-level approach will help clarify the major steps needed to introduce IaC to our teams. Within each step there are details specific to your business where we can work closely with our team to architect and build the right tech stack detailed implementation plan and enabling capabilities.**

**Various tools are available to support the provisioning, configuration, and CI/CD platform described above.**

**The provisioning tools (i.e. Terraform and Packer) coupled with the config management systems (i.e. Chef, Salt, Ansible, Puppet, CFEngine, or homegrown) automatically provision and deploy the topology to a target environment via a CI/CD framework (i.e. Jenkins or TeamCity), leveraging the topology file that describes the infrastructure for a given platform and system.**

**This topology file is checked into source control and versioned – this is now your self-documented environment blueprint. The resultant CI/CD pipelines enable you to build standard box images to fit your technical need (bare metal, vm, or containers). In addition, as your topology changes, these same pipelines will be able to construct the new topology in automation. You are using code to configure and provision environments. This approach allows you to monitor the deployed configuration by running compliance/drift reports which your business can then schedule the remediation or automatically adjust the infrastructure to remain current.**

### **Principle 1: Easy Reproducibility of the Systems**

IaC should allow you to painlessly recreate any given chunk of an infrastructure. You should be able to rebuild without putting too much effort into it. It removes the ambiguity that could come in the form of risk or doubtfulness. And IaC instills confidence when it comes to provisioning new services and environments.

### **Principle 2: High Level of Flexibility**

There’s a  need to ensure that your infrastructure has the answers to all the questions that your software asks. These questions could be about a variety of things, ranging between the desired storage, the compatibility with connected networks, and the configuration. IaC should present itself as flexible blocks that can be assembled instantly if and when the requirements arise.

### **Principle 3: Dynamic Design**

While it’s always a good bet that some component or another will need a change in the form of an upgrade, it’s not always easy to make such changes to the existing system. We can’t predict how, over time, the requirement of a system can change. That’s why one of the core principles of IaC is that the design is always changing. So you should design your infrastructure and software to be as lucid as possible.

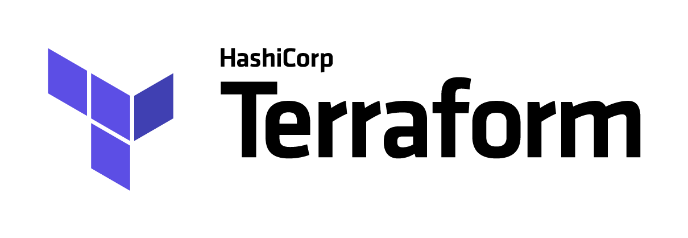
**Current state of IaC:**

Configuration Orchestration vs. Configuration Management

* The first thing that should be clarified is the difference between “configuration orchestration” and “configuration management” tools, both of which are considered IaC tools and are included on this list.
* Configuration orchestration tools, which include Terraform and AWS CloudFormation, are designed to automate the deployment of servers and other infrastructure.
* Configuration management tools like Chef, Puppet, and the others on this list help configure the software and systems on this infrastructure that has already been provisioned.
* Configuration orchestration tools do some level of configuration management, and configuration management tools do some level of orchestration. Companies can and many times use both types of tools together.

**15 Infrastructure as Code tools:**

**Terraform**



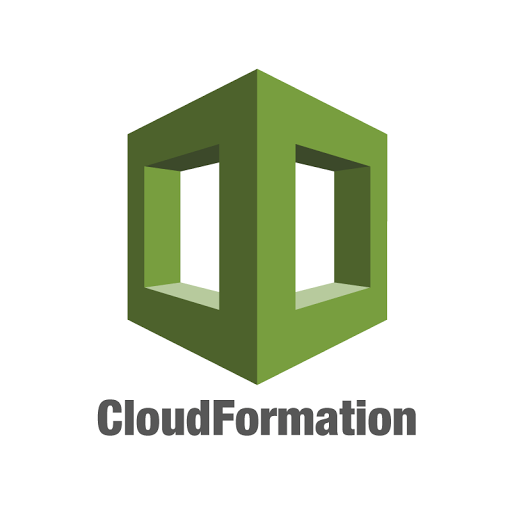
[Terraform](https://www.terraform.io/) is an infrastructure provisioning tool created by Hashicorp. It allows you to describe your infrastructure as code, creates “execution plans” that outline exactly what will happen when you run your code, builds a graph of your resources, and automates changes with minimal human interaction.

Terraform uses its own domain-specific language (DSL) called Hashicorp Configuration Language (HCL). HCL is JSON-compatible and is used to create these configuration files that describe the infrastructure resources to be deployed.

Terraform is cloud-agnostic and allows you to automate infrastructure stacks from multiple cloud service providers simultaneously and integrate other third-party services.

You even can write [Terraform plugins](https://www.terraform.io/docs/plugins/index.html) to add new advanced functionality to the platform.

**AWS CloudFormation**



Similar to Terraform, [AWS CloudFormation](https://aws.amazon.com/cloudformation/) is a configuration orchestration tool that allows you to code your infrastructure to automate your deployments.

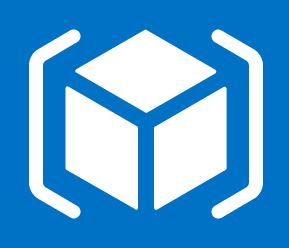
Primary differences lie in that CloudFormation is deeply integrated into and can only be used with AWS, and CloudFormation templates can be created with YAML in addition to JSON.

CloudFormation allows you to preview proposed changes to your AWS infrastructure stack and see how they might impact your resources, and manages dependencies between these resources.

To ensure that deployment and updating of infrastructure is done in a controlled manner, CloudFormation uses Rollback Triggers to revert infrastructure stacks to a previous deployed state if errors are detected.

You can even deploy infrastructure stacks across multiple AWS accounts and regions with a single CloudFormation template. And much more.

**Azure Resource Manager and Google Cloud Deployment Manager**



**Azure Resource Manager**

If you’re using Microsoft Azure or Google Cloud Platform, these cloud service providers offer their own IaC tools similar to AWS CloudFormation.

[**Azure Resource Manager**](https://azure.microsoft.com/en-us/features/resource-manager/) allows you to define the infrastructure and dependencies for your app in templates, organize dependent resources into groups that can be deployed or deleted in a single action, control access to resources through user permissions, and more.



**Google Cloud Deployment Manager**

[**Google Cloud Deployment Manager**](https://cloud.google.com/deployment-manager/) offers many similar features to automate your GCP infrastructure stack. You can create templates using YAML or Python, preview what changes will be made before deploying, view your deployments in a console user interface, and much more.

**Chef**

[](https://www.chef.io/chef/)

Chef is one of the most popular configuration management tools that organizations use in their continuous integration and delivery processes.

Chef allows you to create “recipes” and “cookbooks” using its Ruby-based DSL. These recipes and cookbooks specify the exact steps needed to achieve the desired configuration of your applications and utilities on existing servers. This is called a “procedural” approach to configuration management, as you describe the procedure necessary to get your desired state.

Chef is cloud-agnostic and works with many cloud service providers such as AWS, Microsoft Azure, Google Cloud Platform, OpenStack, and more.

**Puppet**



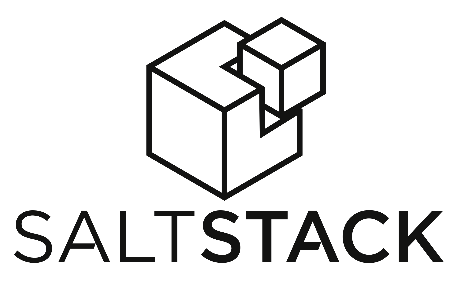
Similar to Chef, [Puppet](https://puppet.com/) is another popular configuration management tool that helps engineers continuously deliver software.

Using Puppet’s Ruby-based DSL, you can define the desired end state of your infrastructure and exactly what you want it to do. Then Puppet automatically enforces the desired state and fixes any incorrect changes.

This “declarative” approach – where you declare what you want your configuration to look like, and then Puppet figures out how to get there – is the primary difference between Puppet and Chef. Also, Puppet is mainly directed toward system administrators, while Chef primarily targets developers.

Puppet integrates with the leading cloud providers like AWS, Azure, Google Cloud, and VMware, allowing you to automate across multiple clouds.

**Saltstack**



[Saltstack](https://saltstack.com/) differentiates itself from tools like Chef and Puppet by taking an “infrastructure as data” approach, instead of “infrastructure as code.”

What this means is that Saltstack’s declarative configuration patterns, while written in Python, are language-agnostic (i.e. you don’t need to learn a specific DSL to create them) and thus are more easily read and understood.

Another differentiator is that Saltstack supports remote execution of commands, whereas Chef and Puppet’s configuration code needs to be pulled from their servers.

**Ansible**



[Ansible](https://www.ansible.com/) is an infrastructure automation tool created by Red Hat, the huge enterprise open source technology provider.

Ansible models your infrastructure by describing how your components and system relate to one another, as opposed to managing systems independently.

Ansible doesn’t use agents, and its code is written in YAML in the form of Ansible Playbooks, so configurations are very easy to understand and deploy.

You can also extend Ansible’s functionality by writing your own Ansible modules and plugins.

**Juju**



[Juju](https://jujucharms.com/) is an IaC tool brought to you by Canonical, the company behind Ubuntu.

You can create Juju charms, which are sets of scripts that deploy and operate software, and bundles, which are collections of charms linked together to deploy entire app infrastructures all at once.

You can then use Juju to manage and apply changes to your infrastructure with simple commands.

Juju works with bare metal, private clouds, multiple public cloud providers, as well as other orchestration tools like Puppet and Chef.

**Docker**

[](https://www.docker.com/)

[Docker](https://www.docker.com/) helps you easily create containers that package your code and dependencies together so your applications can run in any environment, from your local workstation to any cloud service provider’s servers.

YAML is used to create configuration files called Dockerfiles. These Dockerfiles are the blueprints to build the container images that include everything – code, runtime, system tools and libraries, and settings – needed to run a piece of software.

Because it increases the portability of applications, Docker has been especially valuable in organizations who use hybrid or multi-cloud environments.

The use of Docker containers has grown exponentially over the past few years and many consider it to be the future of virtualization.

**Vagrant**



[Vagrant](https://www.vagrantup.com/) is another IaC tool built by HashiCorp, the makers of Terraform.

The difference is that Vagrant focuses on quickly and easily creating development environments that use a small amount of virtual machines, instead of large cloud infrastructure environments that can span hundreds or thousands of servers across multiple cloud providers.

Vagrant runs on top of virtual machine solutions from VirtualBox, VMware, AWS, and any other cloud provider, and also works well with tools like Chef and Puppet.

**Pallet**

[](http://palletops.com/)

[Pallet](http://palletops.com/) is an IaC tool used to automate infrastructure in the cloud, on server racks, or virtual machines, and provides a high level of environment customization.

You can run Pallet from anywhere, and you don’t have to set up and maintain a central server.

Pallet is written in Clojure, runs in a Java Virtual Machine, and works with AWS, OpenStack, VirtualBox, and others, but not Azure nor GCP.

You can use Pallet to start, stop, and configure nodes, deploy projects, and even run administrative tasks.

**(R)?ex**



**(R)?ex**

[(R)?ex](https://www.rexify.org/) is an open-source, weirdly-spelled infrastructure automation tool. “(R)?ex” is too hard to type over and over again, so I’m going to spell it “Rex” from now on.

Rex has its own DSL for you to describe your infrastructure configuration in what are called Rexfiles, but you can use Perl to harness Rex’s full power.

Like Ansible, Rex is agent-less and uses SSH to execute commands and manage remote hosts. This makes Rex easy to use right away.

**CFEngine**



[CFEngine](https://cfengine.com/) is one of the oldest IaC tools out there, with its initial release in 1993.

CFEngine allows you to define the desired states of your infrastructure using its DSL. Then its agents monitor your environments to ensure that their states are converging toward the desired states, and reports the outcomes.

It’s written in C and claims to be the fastest infrastructure automation tool, with execution times under 1 second.

**NixOS**



[NixOS](https://nixos.org/) is a configuration management tool that aims to make upgrading infrastructure systems as easy, reliable, and safe as possible.

The platform does this by making configuration management “transactional” and “atomic.” What this means is that if an upgrade to a new configuration is interrupted for some reason, the system will either boot up in the new or old configuration, thus staying stable and consistent.

NixOS also makes it very easy to rollback to a prior configuration, since new configuration files don’t overwrite old ones.

These configuration files are written in Nix expression language, its own unique functional language.

**Benefits:**

* The team now has a master image that can quickly spin up a new box or apply changes infrastructure is disposable.
* Elimination of manual adjustment to configuration on multiple servers. This reduces

mistakes. Changes are applied via a consistent process allowing for improved delivery times (velocity), auditability, and consistency/repeatability.

* Source code and change control allows for backups, rollbacks, and auditing.
* Reduces risk of one person having all the knowledge (key man failure/risk) or having knowledge distributed like snowflakes in many peoples’ heads.
* Better collaboration is enabled across roles.
* A common set of tools and framework enforces standards and improves efficiency –both Development and Operations are using the same SDLC to get features and changes through the environment pipeline to production.
* Developers can now provision hosts that include base operating systems, and are already compliant with security and operations controls.
* This approach also allows for self-service provisioning, deployments and testing.
* Teams now have a process for testing infrastructure code. Start with basic syntax validation and linting, then move on to unit testing, and finally to integration and acceptance testing.
* Mirror production monitoring in the test environment to ensure that infrastructure components don’t cause test failures. Note: This does not replace automated testing.

### **Benefits of Infrastructure as Code**

#### **Speed and simplicity**

IaC allows you to spin up an entire infrastructure architecture by running a script.

Not only can you deploy virtual servers, but you can also launch pre-configured databases, network infrastructure, storage systems, load balancers, and any other cloud service that you may need.

You can do this quickly and easily for development, staging, and production environments, which can make your software development process much more efficient (more about this later).

Also, you can easily deploy standard infrastructure environments in other regions where your cloud provider operates so they can be used for [backup and disaster recovery](https://www.thorntech.com/2017/05/cloud-disaster-recovery/).

You can do all this by writing and running code.

#### **Configuration consistency**

Standard operating procedures can help maintain some consistency in the infrastructure deployment process. But human error will always rear its ugly head, which may leave you with subtle differences in configurations that may be difficult to debug.

IaC completely standardizes the setup of infrastructure so there is reduced possibility of any errors or deviations. This will decrease the chances of any incompatibility issues with your infrastructure and help your applications run more smoothly.

#### **Minimization of risk**

Imagine having a lead engineer be the only one who knows the ins and outs of your infrastructure setup. Now imagine that engineer leaving your company.

Not only does IaC automate the process, but it also serves as a form of documentation of the proper way to instantiate infrastructure and insurance in the case where employees leave your company with institutional knowledge.

Configurations are bound to change to accommodate new features, additional integrations, and other edits to the application’s source code. If an engineer edits the deployment protocol, it can be difficult to pin down exactly what adjustments were made and who was responsible.

Because code can be version controlled, IaC allows every change to your server configuration to be documented, logged, and tracked. And these configurations can be tested, just like code.

So, if there is an issue with the new setup configuration, it can be pinpointed and corrected much more easily, minimizing risk of issues or failure.

#### **Increased efficiency in software development**

Developer productivity drastically increases with the use of IaC. Cloud architectures can be easily deployed in multiple stages to make the software development life cycle much more efficient.

Developers can launch their own sandbox environments to develop in. QA can have a copy of production that they can thoroughly test. Security and user acceptance testing can occur in separate staging environments. And then the application code and infrastructure can be deployed to production in one move.

Infrastructure as Code allows your company to use Continuous Integration and Continuous Deployment techniques while minimizing the introduction of human errors after the development stage.

You can also include in your IaC script the spinning down of environments when they’re not in use. This will shut down all the resources that your script created, so you won’t end up with a bunch of orphan cloud components that everyone is too afraid to delete. This will further increase the productivity of your engineering staff by having a clean and organized cloud account.

#### **Cost savings**

Automating the infrastructure deployment process allows engineers to spend less time performing manual work, and more time executing higher-value tasks. Because of this increased productivity, your company can save money on hiring costs and engineers’ salaries.

As mentioned earlier, your IaC script can automatically spin down environments when they’re not in use, which will further save on cloud computing costs.

* **Improving Customer Satisfaction** – Being able to deliver a quality service component within a short period of time contributes to customer satisfaction and improved perception of IT within an organization (as measured by Net Promoter Score or other method).
* **Scalable and Immutable Infrastructure** – Provides the ability for additional resources to be provisioned during burst periods allowing horizontal scaling and the ability to replace resources in the event of failure

## Risks Involved with Infrastructure as Code

There is some risk that needs to be accounted for:

* **Missing proper planning** – Once a company decides to move towards having an IaC capable IT Landscape in place, there is the mandatory need to define Infrastructure that will allow the implementation, configuration, and operation of IaC tools.  
    
  A simple example is that you can only create and operate virtual machines if you have in place a physical server infrastructure that can run a tool like VMware and is powerful and scalable enough (CPU, RAM, HDD) to support several heavy demanding simultaneous virtual machines running in parallel without any performance impact, plus redundancy that allows all to continue working within normal operational standards if problems occur.
* **IaC requires new skills** – Most existing IaC tools require expertise to be handled, and reaching such levels requires significant time in learning and training. Some companies are likely to start by resorting to outsourcing services until the tools become more user-friendly, staff is trained on the new tools, or new experts are brought on to the team.
* **Error replication** – Since the initial code is developed by humans, there is always the chance that it contains minor errors that will only produce impact after some time. The problem here is that meanwhile, several machines may have been automatically created where such errors exist. So there is the need for applying a solid auditing process to the creation of IaC generating code.
* **Configuration Drift** – Once a machine is created via an IaC workflow, it should not suffer intervention outside of an automated, aligned, and compliant maintenance workflow. Manual or external updates (even if just security patching) may result in configuration drifting which in time has the potential of producing massive non-compliance or even service failure.
* **Accidental Destruction** – Some IaC tools that maintain state have the ability to automatically destroy resources should the code reflect that action. IaC in an automation pipeline can sometimes have undesired outcomes.

What is Ansible?

Ansible is an open-source automation tool, or platform, used for IT tasks such as configuration management, application deployment, intraservice orchestration and provisioning. Automation is crucial these days, with IT environments that are too complex and often need to scale too quickly for system administrators and developers to keep up if they had to do everything manually. Automation simplifies complex tasks, not just making developers’ jobs more manageable but allowing them to focus attention on other tasks that add value to an organization. In other words, it frees up time and increases efficiency. And Ansible, as noted above, is rapidly rising to the top in the world of automation tools. Let’s look at some of the reasons for Ansible’s popularity.

**Advantages of Ansible**

Free. Ansible is an open-source tool.

Very simple to set up and use. No special coding skills are necessary to use Ansible’s playbooks (more on playbooks later).

Powerful. Ansible lets you model even highly complex IT workflows.

Flexible. You can orchestrate the entire application environment no matter where it’s deployed. You can also customize it based on your needs.

Agentless. You don’t need to install any other software or firewall ports on the client systems you want to automate. You also don’t have to set up a separate management structure.

Efficient. Because you don’t need to install any extra software, there’s more room for application resources on your server.

Ansible’s Features and Capabilities

Configuration Management

Ansible is designed to be very simple, reliable and consistent for configuration management. If you’re already in IT, you can get up and running with it very quickly. Ansible configurations are simple data descriptions of an infrastructure, and are both readable by humans and parsable by machines. All you need to start managing systems is a password or an SSH (Secure Socket Shell, a network protocol) key. An example of how easy Ansible makes configuration management: If you want to install an updated version of a certain type of software on all the machines in your enterprise, all you have to do is write out all the IP addresses of the nodes (also called remote hosts) and write an Ansible playbook to install it on all the nodes, then run the playbook from your control machine.

Application Deployment

Ansible lets you quickly and easily deploy multitier apps. You won’t need to write custom code to automate your systems; you just list the tasks needed to be done by writing a playbook, and Ansible will figure out how to get your systems to the state you want them to be in. In other words, you won’t have to manually configure the applications on every machine. When you run a playbook from your control machine, Ansible uses SSH to communicate with the remote hosts and run all the commands (tasks).

Orchestration

As the name suggests, orchestration involves bringing different elements into a beautifully run whole operation—similar to the way a musical conductor brings the notes produced by all the different instruments into a cohesive musical work. For example, with application deployment, you need to manage not just the front-end and backend services but the databases, networks, storage and so on. You also need to make sure that all the tasks are handled in the proper order. Ansible uses automated workflows, provisioning and more to make orchestrating tasks easy. And once you’ve defined your infrastructure using the Ansible playbooks, you can use that same orchestration wherever you need to, thanks to the portability of Ansible playbooks.

Security and Compliance

As with application deployment, sitewide security policies (such as firewall rules or locking down users) can be implemented along with other automated processes. If you configure the security details on the control machine and run the associated playbook, all the remote hosts will automatically be updated with those details. That means you won’t need to continually manually monitor each machine for security compliance. And for extra security, an admin’s user ID and password aren’t retrievable in plain text on Ansible.

Cloud Provisioning

The first step in automating your applications’ life cycle is automating the provisioning of your infrastructure. With Ansible you can provision cloud platforms, virtualized hosts, network devices and bare-metal servers.

Ansible Architecture

Now let’s talk a bit about the pieces that make up the Ansible environment.

Modules

Modules are like small programs that Ansible pushes out from a control machine to all the nodes, or remote hosts. The modules are executed using playbooks (see below), and they control things such as services, packages and files. Ansible executes all the modules for installing updates or whatever the required task is, and then removes them when finished. Ansible provides more than 450 modules for common tasks.

Plugins

As you probably already know from many other tools and platforms, plugins are extra pieces of code that augment functionality. Ansible comes with a number of its own plugins, but you can write your own as well. Action, cache and callback plugins are three examples.

Inventories

All the machines you’re using with Ansible (the control machine plus nodes) are listed in a single simple file, along with their IP addresses, databases, servers and so on. Once you list the inventory, you can assign variables to any of the hosts using a simple text file. You can also pull inventory from sources like EC2 (Amazon Elastic Compute Cloud).

Playbooks

Ansible playbooks are like instruction manuals for tasks. They are simple files written in YAML, which stands for YAML Ain’t Markup Language, a human-readable data serialization language. Playbooks are really at the heart of what makes Ansible so popular because they describe the tasks to be done easily and without the need for the user to know or remember any special syntax. Not only can they declare configurations, but they can orchestrate the steps of any manually ordered task, and can execute tasks at the same time or at different times.

Each playbook is composed of one or multiple plays, and the goal of a play is to map a group of hosts to well-defined roles, represented by tasks.

APIs

Various APIs (application programming interfaces) are available so you can extend Ansible’s connection types (meaning more than just SSH for transport), callbacks and more.

What Is Ansible Tower?

Ansible Tower is Red Hat’s commercial web-based solution for managing Ansible. Its best-known feature is an easy-to-use UI (user interface) for managing configurations and deployments, which is a big improvement over the original UI. Ansible Tower contains the most important features of Ansible, especially those that are easier to see in a graphical format rather than a text-based format. It is free for up to 10 nodes.

Advantages of Using Ansible With Docker

Ansible does a great job of automating Docker and operationalizing the process of building and deploying containers. If you’re managing a traditional IT system, for example, it can be hard to add container-tooling functionality. But Ansible removes the need to do processes manually. There are four main advantages of using Ansible with Docker:

Portability/Flexibility. The fact that Ansible playbooks are portable, meaning they can be used anywhere, as well as repeatable, can save you a lot of time and effort. For example, if you use a pure Dockerfile to build a container, then you can reproduce the application only in a Docker container. If you use an Ansible playbook to build a container, on the other hand, then you can reproduce the application in Docker, on the cloud and so on.

Auditability. Even if you create containers, you’ll still need to monitor code and track vulnerabilities. Using Ansible with Docker, you can easily track who has deployed which containers as well as what’s in all of the containers, and know that you can rebuild any containers as necessary.

Management of Entire Environments. With Ansible, you already know you can manage your Docker containers. But you can also manage the environment that all the containers are in, even in highly complex environments. Ansible can monitor containers and non-containers at the same time, which is important because containerized applications often need to “talk” with noncontainerized applications.

Similar Syntax. As mentioned, Ansible used YAML files for its playbooks. Docker uses its own non-YAML scripts, but they are very similar and can do almost the same things.