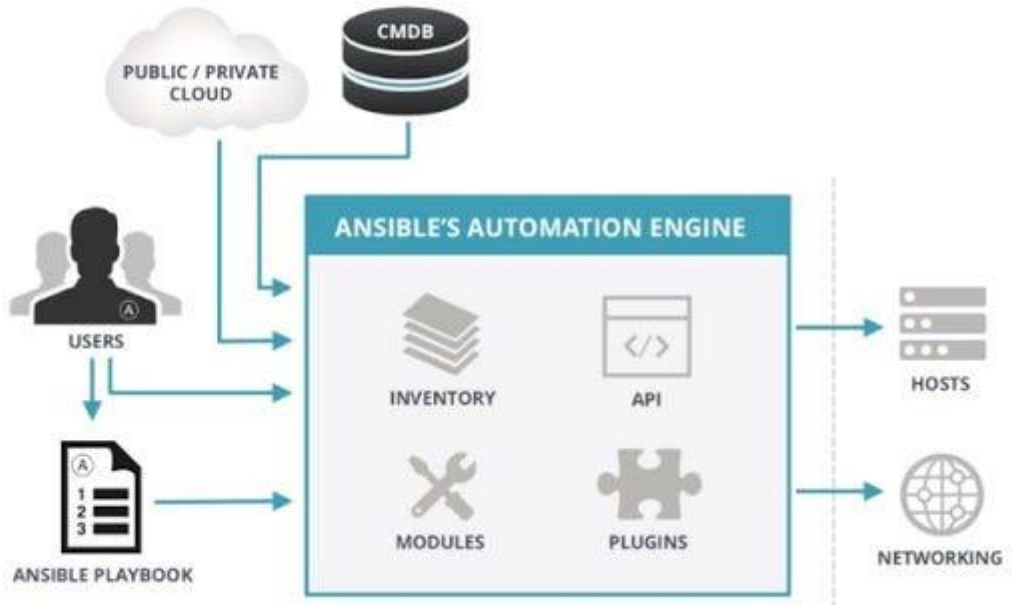
Ansible and Ansible Architecture

The Ansible orchestration engine interacts with a user who is writing the Ansible playbook to execute the Ansible orchestration and interact along with the services of private or public cloud and configuration management database. The below diagram explains it. .



Ansible Architecture

Playbooks here actually define your workflow because whatever tasks that you write in a playbook, it gets executed in the same order that you have written them. They are written in YAML format, which describes the tasks and executes through the Ansible. Also, you can launch the tasks synchronously and asynchronously with playbooks.

The above architecture has a bunch of **Host** machines to which ansible server connects and pushes the playbooks through SSH. It is not always needed to use an SSH for connecting with your host machines; you can also use a connection plug-in. For example, Ansible provides you with a docker container connection plugin and

It has an Ansible automation engine using which users can directly run a **playbook** that gets deployed on the hosts. There are multiple components in the Ansible automation engine. The first is a host inventory. It's a list of all the IP addresses of all the hosts. Let's go one by one, in the Ansible automation engine.

Ansible works against **multiple systems** in your infrastructure at the same time. It does this by selecting portions of systems listed in Ansible's inventory file, which defaults to being saved in the location `/etc/ansible/hosts`. You can specify a different inventory file using the `-i <path>` option on the command line.

- **Playbooks and Inventory**

As a best practice, you don't want your Ansible architecture that consists of playbooks and Inventory to be too specific. But on the other hand, you need to have a certain level of abstraction and keep out precise information. Therefore, to develop flexible code, you must separate site-specific information from the code, done with variables in Ansible.

Remember that when you develop dynamic code along with your static information, you can use this on any site with minor modifications to the variables themselves. However, you can have variables in different places, and where you place variables, such as a play header or Inventory, will take different precedence. So, to provide site-specific code, variables can be used in your Ansible deployment architecture.

Before you proceed, you may find the following posts helpful for pre-information:

1. Network Configuration Automation
2. Ansible Variables
3. Network Traffic Engineering

- **A key point: Video on Ansible automation and Ansible architecture.**

In this video, we will discuss **Ansible automation and Ansible architecture**. In particular, Ansible Engine is run from the CLI. This is compared to a different Ansible deployment architecture with Ansible Tower, a platform approach to security. We will discuss the challenging landscape forcing us to move to automation. At the same time, it introduces Ansible playbooks, Ansible variables, and other main components.

Ansible Architecture: The Drive for Automation

To move to an Ansible architecture has been driven by several megatrends. Firstly, the rise of distributed computing made the manual approach to almost anything in the IT environment obsolete. Not only because this causes many errors and mistakes but also because the configuration drift from the desired to the actual state was considerable.

You have two options to implement all of this. First, you can connect up these services by, for example, manually spinning up the servers, installing the necessary packages, and SSHing to each one, or you can go down the path of automation, in particular, automation with Ansible.

So, with Ansible deployment architecture, we have the Automation Engine, the CLI, and Ansible Tower, which is more of an automation platform for enterprise-grade automation. This post focuses on Ansible Engine.
As a quick note, if you have environments with more than a few teams automating, I recommend Ansible Tower or the open-source version of AWX. This is because Ansible Tower has a 60-day trial license, while AWX is fully open-sourced and does not require a license. The open-source version of AWX could be a valuable tool for your open networking journey.

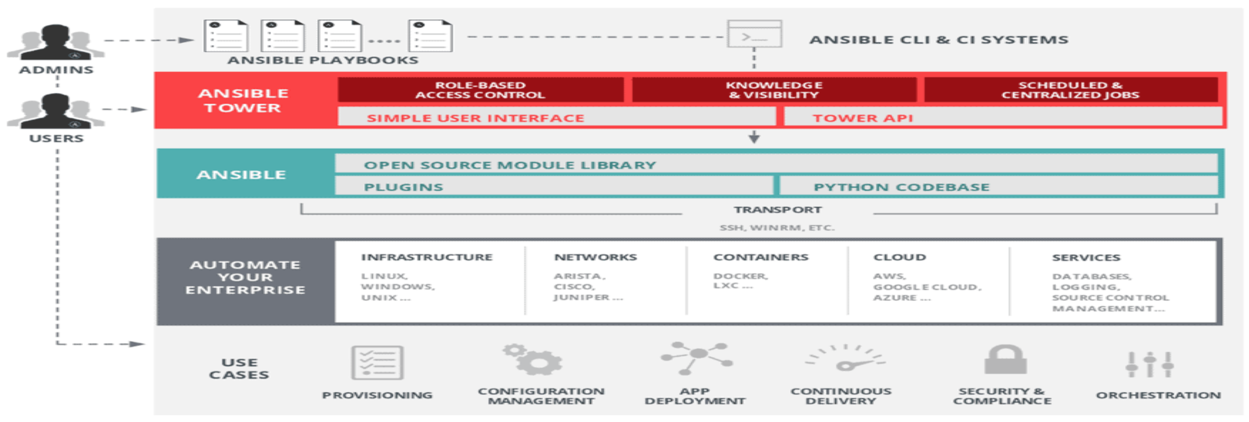


Diagram: Red Hat Ansible Tower. Source RedHat.

- **A Key Point: Risky: The Manual Way.**

Let me put it this way. If you are configuring manually, you will likely maintain all the settings manually. Or, more importantly, what about mitigating vulnerabilities and determining what patches or packages are installed in a large environment?

How can you ensure all your servers are patched and secured manually? Manipulating configuration files by hand is a tedious and error-prone task. Not to mention time-consuming. Equally, performing pattern matching to make changes to existing files is risky.

- **A Key Point: The issue of Configuration Drift**

The manual approach will result in configuration drift, where some servers will drift from the desired state. Configuration drift is caused by inconsistent configuration items across devices, usually due to manual changes and updates and not following the automation path. Ansible is all about maintaining the desired state and eliminating configuration drift.

Ansible Workflow:

Ansible operates on a push-based model, where the control node pushes configurations and commands to the managed nodes. The workflow involves the following steps:

1. Inventory:

The inventory is a file that contains a list of managed nodes. It provides Ansible with information such as IP addresses, hostnames, and connection details required to establish communication.

2. Playbooks:

Playbooks are YAML files that define the desired state of the managed nodes. They consist of a series of tasks or plays, where each task represents a specific action to be executed on the managed nodes. Playbooks can be as simple as a single task or as complex as a multi-step deployment process.

3. Execution:

Ansible executes playbooks on the control node and communicates with managed nodes to perform the defined tasks. It uses modules, which are small programs written in Python or other scripting languages, to interact with the managed nodes and carry out the required operations.

4. Reporting:

Ansible provides detailed reports on the execution status of tasks, allowing administrators to monitor and troubleshoot any issues that arise. This helps maintain visibility and ensure the desired configurations are applied consistently across the infrastructure.

- **A key point: Video on Ansible Job Template**

In this product demonstration, we will go through the key components of Ansible Tower and its use of Job Templates. We will look at the different Job Template parameters that you can use to form an automation job that you can deploy to your managed assets.

Advantages of Ansible Architecture:

The Ansible architecture offers several advantages, making it a preferred choice for automation:

1. Simplicity:

Ansible's architecture is designed to be simple and easy to understand. Using YAML playbooks and declarative language allows administrators to define configurations and tasks in a human-readable format.

2. Agentless:

Unlike traditional configuration management tools, Ansible requires no agent software installed on managed nodes. This reduces complexity and eliminates the need for additional overhead.

3. Scalability:

Ansible's architecture is highly scalable, enabling administrators to manage thousands of nodes simultaneously. SSH and WinRM protocols allow for efficient communication and coordination across large infrastructures.

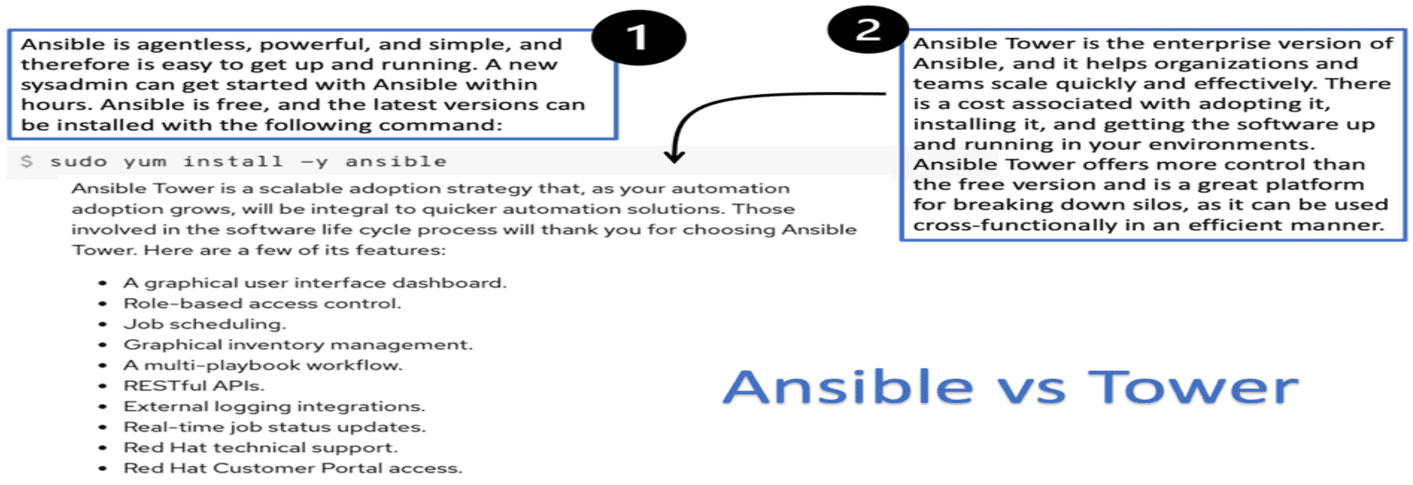
Components of Ansible Deployment Architecture

- ### Configuration management

The Ansible architecture is based on a **configuration management tool** that can help alleviate these challenges. Ansible replaces the need for an operator to tune configuration files manually and does an excellent job in application deployment and orchestrating multi-deployment scenarios. It can also be integrated into CI/CD pipelines.

In reality, Ansible is relatively easy to install and operate. However, it is not a single entity. Instead, it comprises tools, modules, and software-defined infrastructure that form the ansible toolset configured from a single host that can manage multiple hosts.

We will discuss the value of idempotency with Ansible modules later in mind. Even with the idempotency nature of modules, you can still have users of Ansible automate over each other. Ansible Tower or AWS is the recommended solution for multi-team automation efforts.


1

Ansible is agentless, powerful, and simple, and therefore is easy to get up and running. A new sysadmin can get started with Ansible within hours. Ansible is free, and the latest versions can be installed with the following command:

```
$ sudo yum install -y ansible
```

Ansible Tower is a scalable adoption strategy that, as your automation adoption grows, will be integral to quicker automation solutions. Those involved in the software life cycle process will thank you for choosing Ansible Tower. Here are a few of its features:

- A graphical user interface dashboard.
- Role-based access control.
- Job scheduling.
- Graphical inventory management.
- A multi-playbook workflow.
- RESTful APIs.
- External logging integrations.
- Real-time job status updates.
- Red Hat technical support.
- Red Hat Customer Portal access.

2

Ansible Tower is the enterprise version of Ansible, and it helps organizations and teams scale quickly and effectively. There is a cost associated with adopting it, installing it, and getting the software up and running in your environments. Ansible Tower offers more control than the free version and is a great platform for breaking down silos, as it can be used cross-functionally in an efficient manner.

Ansible vs Tower