Linux Automation with Ansible 8.0

Contents

Automation and Linux System Administration 1

Infrastructure as Code 2

Mitigating Human Error 2

What is Ansible? 2

Ansible Is Simple 3

Ansible Is Powerful 3

Ansible Is Agentless 3

Ansible: The Language of DevOps 4

Ansible Concepts and Architecture 4

The Ansible Way 6

Use Cases 7

Automation and Linux System Administration

For many years, most system administration and infrastructure management has relied on manual tasks performed through graphical or command-line user interfaces. System administrators often work from checklists, other documentation, or a memorized routine to perform standard tasks.

This approach is error-prone. It is easy for a system administrator to skip a step or perform a step mistakenly. Often there is limited verification that the steps were performed properly or that they result in the expected outcome.

Furthermore, by managing each server manually and independently, it is very easy for many servers that are supposed to be identical in configuration to be different in minor (or major) ways. This can make maintenance more difficult and introduce errors or instability into the IT environment.

Automation can help avoid the problems caused by manual system administration and infrastructure management. As a system administrator, you can use it to ensure that all your systems are quickly and correctly deployed and configured. This allows you to automate the repetitive tasks in your daily schedule, freeing up your time and allowing you to focus on more critical things. For your organization, this means you can more quickly roll out the next version of an application or updates to a service.

Infrastructure as Code

A good automation system allows you to implement *Infrastructure as Code* practices. Infrastructure as Code means that you can use a machine-readable automation language to define and describe the state you want your IT infrastructure to be in. Ideally, this automation language should also be very easy for humans to read, because then you can easily understand what the state is and make changes to it. This code is then applied to your infrastructure to ensure that it is actually in that state.

If the automation language is represented as simple text files, it can easily be managed in a version control system like software code. The advantage of this is that every change can be checked into the version control system, so you have a history of the changes you make over time. If you want to revert to an earlier known-good configuration, you simply can check out that version of the code and apply it to your infrastructure.

This builds a foundation to help you follow best practices in DevOps. Developers can define their desired configuration in the automation language. Operators can review those changes more easily to provide feedback, and use that automation to reproducibly ensure that systems are in the state expected by the developers.

Mitigating Human Error

By reducing the tasks performed manually on servers using automation of tasks and Infrastructure as Code practices, your servers will be in consistent configurations more often. This means that you need to become accustomed to making changes through updating your automation code, rather than manually applying them to your servers. Otherwise, you run the risk of losing manually-applied changes the next time you apply changes through your automation.

Automation allows you to use code review, peer review by multiple subject matter experts, and documentation of the procedure by the automation itself to reduce your operational risks.

Ultimately, you can enforce that changes to your IT infrastructure must be made through automation in order to mitigate human error.

What is Ansible?

Ansible is an open source automation platform. It is a *simple automation language* that can perfectly describe an IT application infrastructure in *Ansible Playbooks*. It is also an *automation engine* that runs Ansible Playbooks.

Linux Automation with Ansible 8.0

Ansible can manage powerful automation tasks and can adapt to many different workflows and environments. At the same time, new users of Ansible can very quickly use it to become productive.

Ansible Is Simple

Ansible Playbooks provide human-readable automation. This means that playbooks are automation tools that are also easy for humans to read, comprehend, and change. No special coding skills are required to write them. Playbooks execute tasks in order. The simplicity of playbook design makes them usable by every team, which allows people new to Ansible to get productive quickly.

Ansible Is Powerful

You can use Ansible to deploy applications, for configuration management, for workflow automation, and for network automation. Ansible can be used to orchestrate the entire application life cycle.

Ansible Is Agentless

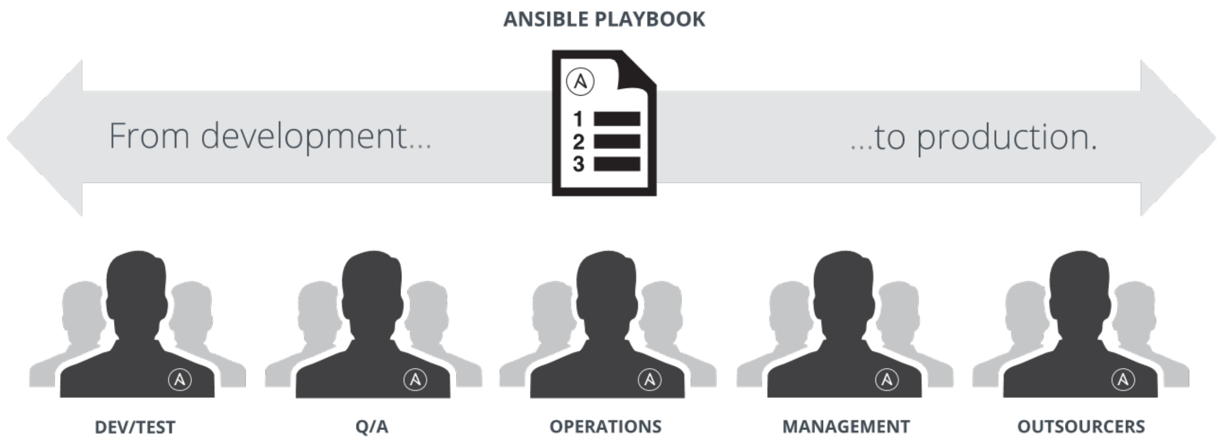
Ansible is built around an *agentless architecture*. Typically, Ansible connects to the hosts it manages using OpenSSH or WinRM and runs tasks, often (but not always) by pushing out small programs called *Ansible modules* to those hosts. These programs are used to put the system in a specific desired state. Any modules that are pushed are removed when Ansible is finished with its tasks. You can start using Ansible almost immediately because no special agents need to be approved for use and then deployed to the managed hosts. Because there are no agents and no additional custom security infrastructure, Ansible is more efficient and more secure than other alternatives.

Ansible has a number of important strengths:

- *Cross platform support*: Ansible provides agentless support for Linux, Windows, UNIX, and network devices, in physical, virtual, cloud, and container environments.
- *Human-readable automation*: Ansible Playbooks, written as YAML text files, are easy to read and help ensure that everyone understands what they will do.
- *Perfect description of applications*: Every change can be made by Ansible Playbooks, and every aspect of your application environment can be described and documented.
- *Easy to manage in version control*: Ansible Playbooks and projects are plain text. They can be treated like source code and placed in your existing version control system.
- *Support for dynamic inventories*: The list of machines that Ansible manages can be dynamically updated from external sources in order to capture the correct, current list of all managed servers all the time, regardless of infrastructure or location.

- *Orchestration that integrates easily with other systems*: HP SA, Puppet, Jenkins, RedHat Satellite, and other systems that exist in your environment can be leveraged and integrated into your Ansible workflow.

Ansible: The Language of DevOps



1.1: Ansible across the application life cycle

Communication is the key to DevOps. Ansible is the first automation language that can be read and written across IT. It is also the only automation engine that can automate the application life cycle and continuous delivery pipeline from start to finish.

Ansible Concepts and Architecture

There are two types of machines in the Ansible architecture: *control nodes* and *managed hosts*. Ansible is installed and run from a control node, and this machine also has copies of your Ansible project files. A control node could be an administrator's laptop, a system shared by a number of administrators, or a server running RedHat AnsibleTower.

Managed hosts are listed in an *inventory*, which also organizes those systems into groups for easier collective management. The inventory can be defined in a static text file, or dynamically determined by scripts that get information from external sources.

Instead of writing complex scripts, Ansible users create high-level *plays* to ensure a host or group of hosts are in a particular state. A play performs a series of *tasks* on the hosts, in the order specified by the play. These plays are expressed in YAML format in a text file. A file that contains one or more plays is called a *playbook*.

Each task runs a *module*, a small piece of code (written in Python, PowerShell, or some other language), with specific arguments. Each module is essentially a tool in your toolkit. Ansible ships with hundreds of useful modules that can perform a wide variety of automation tasks. They can act on system files, install software, or make API calls.

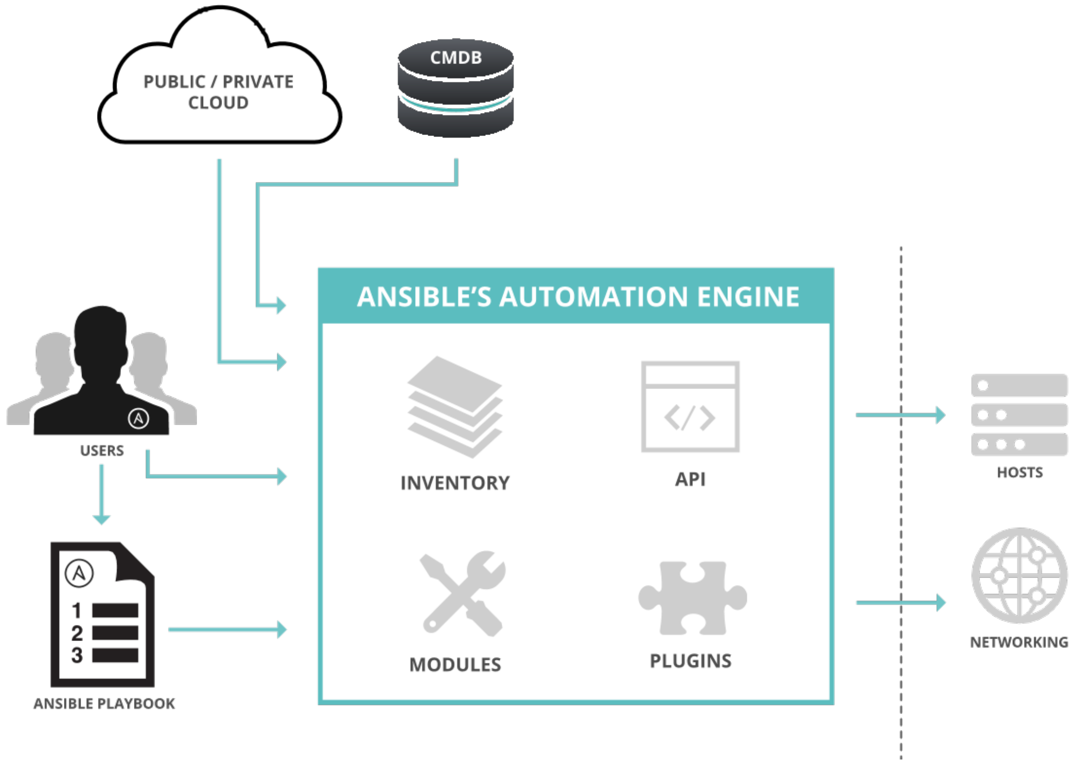
When used in a task, a module generally ensures that some particular aspect of the machine is in a particular state. For example, a task using one module may ensure that a file exists and has particular permissions and contents, while a task using a different module may make certain that a particular file system is mounted. If the system is not in that state, the task should put it in that state. If the system is already in that state, it does nothing. If a task fails, Ansible's default behavior is to abort the rest of the playbook for the hosts that had a failure.

Tasks, plays, and playbooks are designed to be *idempotent*. This means that you can safely run a playbook on the same hosts multiple times. When your systems are in the correct state, the playbook makes no changes when you run it. This means that you should be able to run a playbook on the same hosts multiple times safely. When your systems are in the correct state the playbook should make no changes when you run it. There are a handful of modules that you can use to run arbitrary commands. However, you must use those modules with care to ensure that they run in an idempotent way.

Ansible also uses *plug-ins*. Plug-ins are code that you can add to Ansible to extend it and adapt it to new uses and platforms.

The Ansible architecture is agentless. Typically, when an administrator runs an Ansible Playbook or an ad hoc command, the control node connects to the managed host using SSH (by default) or WinRM. This means that clients do not need to have an Ansible-specific agent installed on managed hosts, and do not need to permit special network traffic to some nonstandard port.

RedHat AnsibleTower is an enterprise framework to help you control, secure, and manage your Ansible automation at scale. You can use it to control who has access to run playbooks on which hosts, share the use of SSH credentials without allowing users to transfer or see their contents, log all of your Ansible jobs, and manage inventory, among many other things. It provides a web-based user interface (web UI) and a RESTful API. It is not a core part of Ansible, but a separate product that helps you use Ansible more effectively with a team or at a large scale.



1.2: Ansible architecture

The Ansible Way

Complexity Kills Productivity

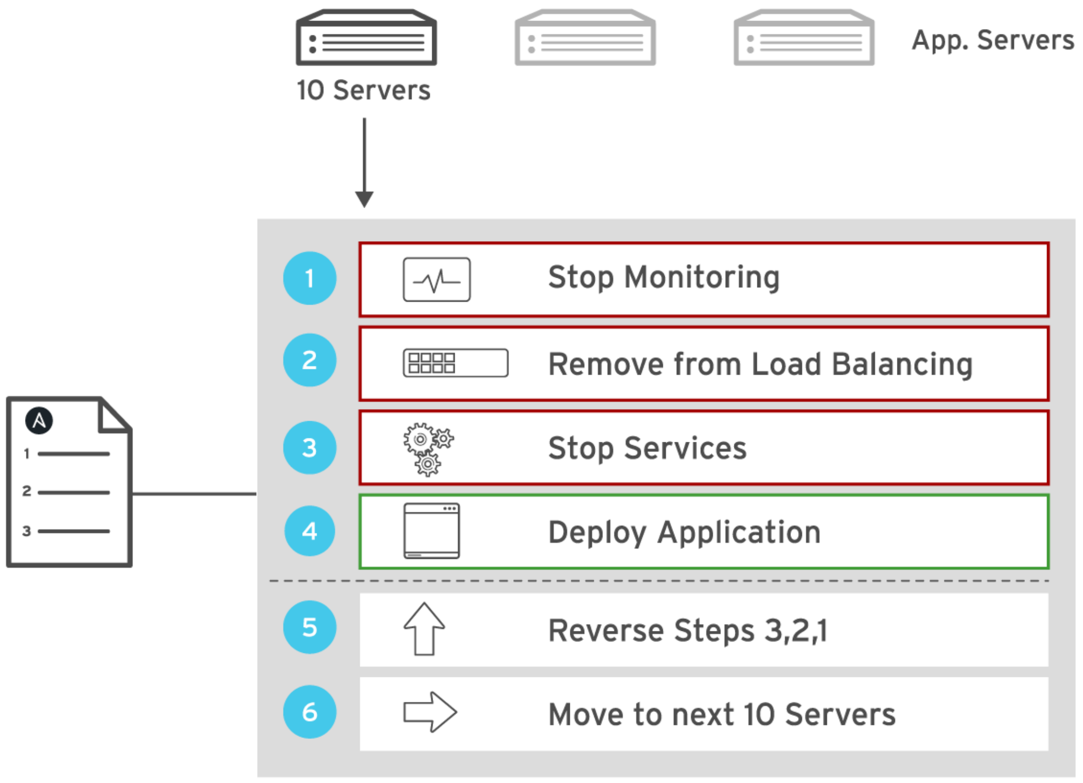
Simpler is better. Ansible is designed so that its tools are simple to use and automation is simple to write and read. You should take advantage of this to strive for simplification in how you create your automation.

Optimize For Readability

The Ansible automation language is built around simple, declarative, text-based files that are easy for humans to read. Written properly, Ansible Playbooks can clearly document your workflow automation.

Think Declaratively

Ansible is a *desired-state engine*. It approaches the problem of how to automate IT deployments by expressing them in terms of the state that you want your systems to be in. Ansible's goal is to put your systems into the desired state, only making changes that are necessary. Trying to treat Ansible like a scripting language is not the right approach.



1.3: Ansible provides complete automation

Use Cases

Unlike some other tools, Ansible combines and unites orchestration with configuration management, provisioning, and application deployment in one easy-to-use platform.

Some use cases for Ansible include:

Configuration Management

> Centralizing configuration file management and deployment is a common use case for Ansible, and it is how many power users are first introduced to the Ansible automation platform.

Application Deployment

> When you define your application with Ansible, and manage the deployment with RedHat Ansible Tower, teams can effectively manage the entire application life cycle from development to production.

Provisioning

Applications have to be deployed or installed on systems. Ansible and RedHat AnsibleTower can help streamline the process of provisioning systems, whether you are PXE booting and kickstarting bare-metal servers or virtual machines, or creating virtual machines or cloud instances from templates. Applications have to be deployed or installed on systems.

Continuous Delivery

Creating a CI/CD pipeline requires coordination and buy-in from numerous teams. You cannot do it without a simple automation platform that everyone in your organization can use. Ansible Playbooks keep your applications properly deployed (and managed) throughout their entire life cycle.

Security and Compliance

When your security policy is defined in Ansible Playbooks, scanning and remediation of site-wide security policies can be integrated into other automated processes. Instead of being an afterthought, it is an integral part of everything that is deployed.

Orchestration

Configurations alone do not define your environment. You need to define how multiple configurations interact, and ensure the disparate pieces can be managed as a whole.