Introduction to Ansible

Ansible Fundamentals

Agenda

- What is Ansible?
- What Ansible Automates
- A short comparison of Ansible, Chef and Puppet
- Installing Ansible
- Ansible Architecture

What is Ansible?

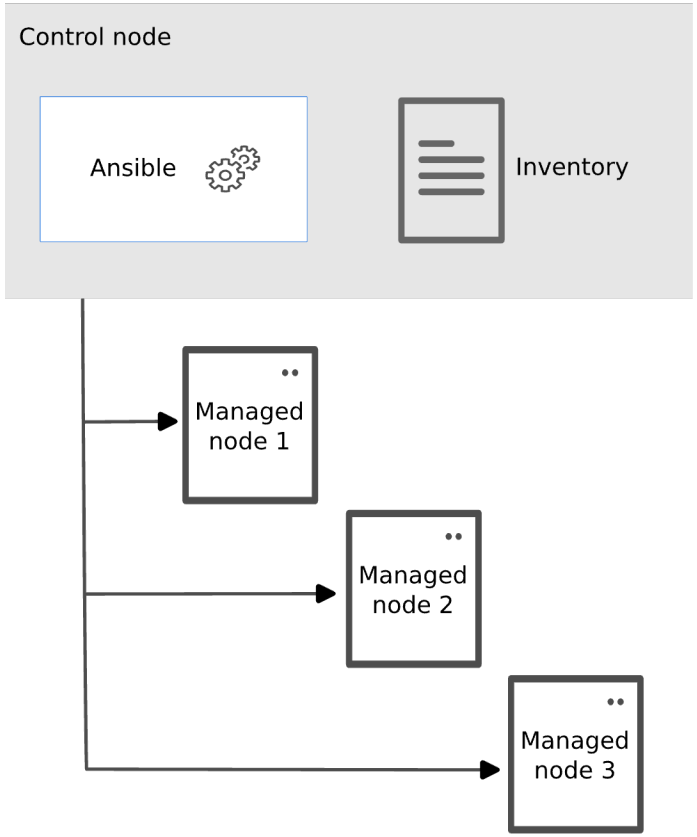
Introduction to Ansible

What is Ansible?

- Open-Source Automation Tool: Ansible is a free and open-source tool designed for automation tasks such as configuration management, application deployment, and infrastructure provisioning.
- Agentless Design: Unlike many other automation tools, Ansible operates in an agentless manner, relying on SSH (or WinRM for Windows) to communicate with target machines, eliminating the need to install additional software on those machines.
- Playbook-Driven: Ansible uses playbooks written in YAML (Yet Another Markup Language) to define and describe automation tasks in a human-readable format.
- Extensibility: With a vast library of built-in modules and the ability to create custom ones, Ansible can manage and automate diverse systems, from servers and databases to networking devices and cloud resources.

What is Ansible?

- **Control node**: A system on which Ansible is installed. You run Ansible commands such on a control node.
- **Managed node**: A remote system, or host, that Ansible controls.
- **Inventory**: A list of managed nodes that are logically organized. You create an inventory on the control node to describe host deployments to Ansible.



What Ansible Automates

Introduction to Ansible

What Ansible Automates

- Configuration Management
  - Ensure consistent configurations across servers.
  - Apply system updates and patches.
  - Manage users and groups.
- Application Deployment
  - Deploy applications to various environments.
  - Coordinate multi-tier rollouts.
  - Handle database migrations and updates.
- Infrastructure Provisioning
  - Spin up and tear down virtual machines or cloud instances.
  - Manage cloud resources, such as storage, networking, and databases.
  - Work with popular cloud providers like AWS, Azure, and Google Cloud.
- Orchestration
  - Coordinate tasks on multiple machines.
  - Handle complex deployments and workflows.
  - Manage rolling updates without downtime.

What Ansible Automates

- Network Automation
  - Configure network devices from vendors like Cisco, Juniper, and Arista.
  - Automate network setups, VLAN configurations, and firewall rules.
- Security and Compliance
  - Harden systems based on security benchmarks.
  - Ensure compliance with policies and standards.
  - Rotate passwords and manage SSH keys.
- Continuous Delivery and Integration
  - Integrate with CI/CD pipelines.
  - Automate testing and validation processes.
  - Handle artifact delivery and staging.
- Task Automation
  - Run ad-hoc commands across a group of servers.
  - Schedule routine tasks like backups and cleanups.
- Storage Management
  - Automate storage provisioning and management.
  - Handle tasks related to file systems, LVM, and RAID configurations.

What Ansible Automates

- Monitoring and Logging
  - Integrate with monitoring solutions.
  - Automate the setup of logging solutions and forwarders.
- Backup and Recovery
  - Automate backup processes for databases, applications, and systems.
  - Handle restoration processes.
- Cloud Services Integration
  - Manage services like AWS Lambda, S3, RDS, and more.
  - Automate scaling actions and resource adjustments.
- Custom Extensions
  - With its modular design, users can write custom modules to automate tasks specific to their needs.

A short comparison of Ansible, Chef and Puppet

Introduction to Ansible

Ansible vs Chef vs Puppet

	Ansible	Chef	Puppet
Design Philosophy	Agentless; relies on SSH for communication	Uses an agent-server model; requires a Chef workstation, server, and client	Agent-master model; uses a declarative approach to define the desired state
Configuration Language	Uses YAML for its playbooks	Uses a Domain Specific Language (DSL) based on Ruby called "recipes" and "cookbooks"	Puppet's own DSL
Primary Use Cases	Configuration management, application deployment, and task automation	Infrastructure automation by treating infrastructure as code	Configuration management and system automation
Learning Curve	Known for its simplicity and human-readable playbooks	Steeper due to its Ruby-based DSL, but offers a lot of flexibility and power	Requires understanding of its unique DSL; offers detailed reporting and a visual management dashboard
Community & Support	Strong open-source community; backed by Red Hat	Vibrant community with a marketplace for cookbooks; backed by Progress Software	Well-established user base; Puppet Enterprise offers commercial support

Installing Ansible

Introduction to Ansible

Control Node Requirements

- For your control node (the machine that runs Ansible), you can use nearly any UNIX-like machine with Python 3.9 or newer installed
- This includes Red Hat, Debian, Ubuntu, macOS, BSDs, and Windows under a Windows Subsystem for Linux (WSL) distribution
- Windows without WSL is not natively supported as a control node; see (more details here)

Managed Node Requirements

- The Managed node (the machine that Ansible is managing) does not require Ansible to be installed, but requires Python 2.7, or Python 3.5 - 3.11 to run Ansible-generated Python code
- The managed node also needs a user account that can connect through SSH to the node with an interactive POSIX shell.

ansible vs. ansible-core

- ansible-core
  - Minimalist language and runtime package
- ansible
  - Much larger “batteries included” package
  - Adds a community-curated selection of Ansible Collections for automating a wide variety of devices
  - Includes ansible-core
- **ansible** package is the preferred choice. **ansible-core** must be used only in limited scenarios

Instalation using Python package manager (pip)

- Install Ansible

```
$ python3 -m pip install --user ansible
```

- Upgrade Ansible

```
$ python3 -m pip install --upgrade --user ansible
```

- Confirm your installation

```
$ ansible --version
```

Instalation using OS prefered way

- You can install ansible using a direct way supported by your OS
- Installing Ansible on Fedora Linux
- Installing Ansible from EPEL
- Installing Ansible on OpenSUSE Tumbleweed/Leap
- Installing Ansible on Ubuntu
- Installing Ansible on Debian
- Installing Ansible on Arch Linux
- Installing Ansible on Windows

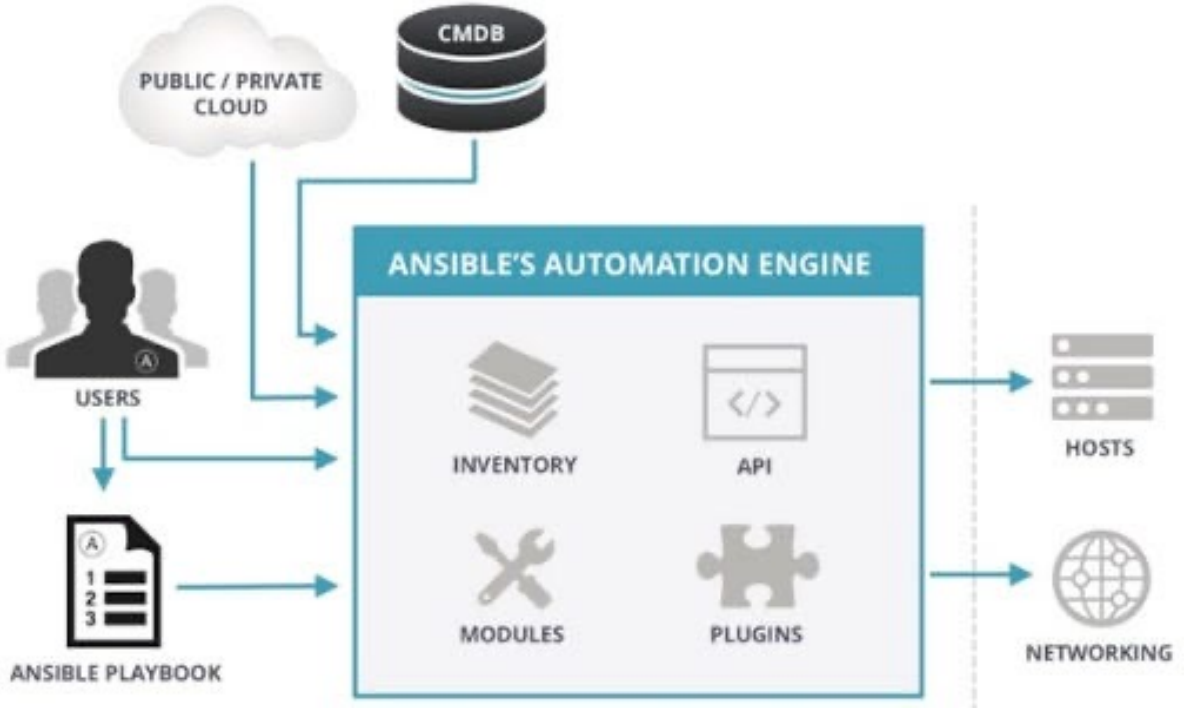
Check Ansible Running

Demo

Ansible Architecture

Running Ad Hoc Commands

Ansible Architecture



Automation Engine: Inventory

- By default, Ansible represents what machines it manages using a very simple INI file creating a static inventory, using groups of your own choosing
- Other Ansible option, is to use a dynamic inventory to pull your inventory from data sources like AWS, Azure, vSphere, etc.
- Once inventory hosts are listed, variables can be assigned to them in simple text files (in a subdirectory called 'group_vars/' or 'host_vars/' or directly in the inventory file.

Automation Engine: Inventory

- Example of a inventory file

```
---
[webservers]
www1.example.com
www2.example.com

[dbservers]
db0.example.com
db1.example.com
```

Automation Engine: Modules

- Building Blocks of Tasks: Modules are the fundamental units in Ansible used to perform individual tasks. When you define a task in a playbook, you are essentially calling a module with specific parameters.
- Wide Range of Functions: Ansible comes with a vast library of built-in modules that can manage system resources, software packages, files, cloud infrastructure, network devices, and much more.
- Idempotent Operations: Like playbooks, most modules are designed to be idempotent. This means that they will only make changes if the current state of the system doesn't match the desired state.
- Extensibility: While Ansible provides a plethora of built-in modules, users can also write custom modules to cater to specific needs or to manage unique systems.
- Language Agnostic: Although Ansible is written in Python, modules can be written in any language. As long as the language can return JSON, it can be used to write an Ansible module.

Modules: yum package manager

- Used for managing packages with the yum package manager, which is common on Red Hat-based systems.

Modules: file

- Manages file properties, such as permissions, ownership, and links

```yaml
tasks:
  - name: "Change ownership of /etc/foo.conf to user 'john' and group 'john'"
    file:
      path: /etc/foo.conf
      owner: john
      group: john
```

Modules: community.vmware.vmware_guest

- modify an existing VM using the vmware_guest module. In this scenario, we'll adjust the CPU and memory configuration of an existing VM.

```yaml
tasks:
  - name: Adjust VM hardware configuration
    community.vmware.vmware_guest:
      hostname: 'vcenter.example.com'
      username: 'administrator@vsphere.local'
      password: 'your_password'
      validate_certs: no
      cluster: 'ClusterName'
      name: 'existing_vm_name'
      hardware:
        memory_mb: 4096
        num_cpus: 4
        hotadd_cpu: yes
        hotadd_memory: yes
      state: 'present'
```

Automation Engine: Plugins

- Extensions to Core Features: Plugins augment Ansible's core functionality. They are small pieces of code that enhance built-in capabilities, allowing Ansible to interact with various systems, software, and APIs.
- Diverse Categories: There are many types of plugins in Ansible, including inventory, action, callback, connection, filter, lookup, and more, each serving a specific purpose.
- Customization: While Ansible comes with a wide range of built-in plugins, users can also develop custom plugins to meet specific needs or to integrate with unique systems.
- Execution: Unlike modules, which are executed on the target machine, plugins typically run on the control machine, enhancing the functionality of the Ansible core engine.
- Configuration: Plugins can be configured within the ansible.cfg file, allowing users to define specific settings or choose which plugins to use for specific tasks.

Plugins: Examples

- Inventory Plugins
  - Purpose: Dynamically generate inventory from external data sources.
  - Example: The aws_ec2 inventory plugin fetches inventory from AWS EC2.
- Callback Plugins
  - Purpose: Enable users to add additional actions to respond to Ansible events.
  - Example: The profile_tasks callback plugin provides a summary of how much time each task took.
- Lookup Plugins
  - Purpose: Retrieve data or variables from external sources.
  - Example: The password lookup plugin can be used to generate random passwords for use in playbooks.

Automation Engine: Modules vs Plugins

- Ansible Modules
  - Purpose: Modules are the units of work in Ansible. They are the commands or set of similar commands executed on the target machine(s).
  - Execution Location: Modules are executed on the target machine(s). The control machine sends the module to the target, where it's executed.
  - Language: While most built-in modules are written in Python, they can technically be written in any language. The key is that they accept and return JSON.
  - Usage: Used within tasks in playbooks to perform actions, manage state, or gather data from target systems.
- Ansible Plugins
  - Purpose: Plugins augment or extend Ansible's core functionality. They modify or enhance the behavior of Ansible but don't directly perform tasks on target machines.
  - Execution Location: Plugins are executed on the control machine, not on the target systems.
  - Language: Plugins are primarily written in Python.
  - Usage: Enhance the functionality of playbooks, configurations, and runtime operations. They can be used to customize Ansible's behavior, integrate with external systems, or process data.

Automation Engine: Playbooks

- YAML Scripts: Playbooks are written in YAML (Yet Another Markup Language), which is both human-readable and machine-parsable, ensuring clarity in defining tasks.
- Ordered Set of Tasks: Playbooks define an ordered set of tasks to be executed against host machines, ensuring configurations are applied in the correct sequence.
- Idempotent Operations: Tasks in playbooks are designed to be idempotent, meaning they can be executed multiple times without changing the result after the first successful run.
- Multi-Tier Orchestration: Playbooks can manage configurations across multi-tier setups, allowing for complex deployments involving databases, application servers, web servers, etc.
- Variables & Templates: Playbooks support the use of variables and templates, enabling dynamic configurations based on specific conditions or inputs.

Automation Engine: Playbooks

- **`hosts: webservers`**: This playbook targets hosts grouped under "webservers" in the Ansible inventory.
- **`vars`**: Defines variables (http_port and max_clients) that can be used later in the playbook or in templates.
- **`tasks`**: Lists the tasks to be executed in order. Here, it ensures Apache is installed, running, and a custom configuration file is uploaded.
- **`yum and service modules`**: These are used to manage packages and services on the target hosts, respectively.
- **`template module`**: This is used to push a templated configuration file to the target hosts. The .j2 extension indicates it's a Jinja2 templated file, which can utilize the defined variables for dynamic content.
- **`handlers`**: These are special tasks that run once, at the end of the playbook, if notified by another task. In this case, if the configuration file changes, Apache will be restarted.

```
---
- name: Install and Start Apache Web Server
  hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  tasks:
    - name: Ensure Apache is installed
      yum:
        name: httpd
        state: present

    - name: Ensure Apache is running and enabled at boot
      service:
        name: httpd
        state: started
        enabled: yes

    - name: Upload custom configuration file
      template:
        src: /path/to/local/template.cfg.j2
        dest: /etc/httpd/conf/custom.cfg
      notify: Restart Apache

  handlers:
    - name: Restart Apache
      service:
        name: httpd
        state: restarted
```