



What is Ansible?

What is Ansible?

- Ansible is a simple agentless idempotent **task automation tool**.
- By default, tasks are executed in-order but we can change that if we want
- **Tasks** are performed via **modules**
- **Tasks are grouped together via plays**
 - Also via **roles**, but more on that later
 - A **play** operates on a set of hosts
- **Playbooks** can contain one or many plays



What is Ansible?

- It's a **simple automation language** that can perfectly describe an IT application infrastructure in **Ansible Playbooks**.
- It's an **automation engine** that runs Ansible Playbooks.
- Ansible Tower is an **enterprise framework** for controlling, securing and managing your Ansible automation with a **UI and RESTful API**.

Why Ansible?



SIMPLE

- Human readable automation
- No special coding skills needed
- Tasks executed in order
- Get productive quickly**



POWERFUL

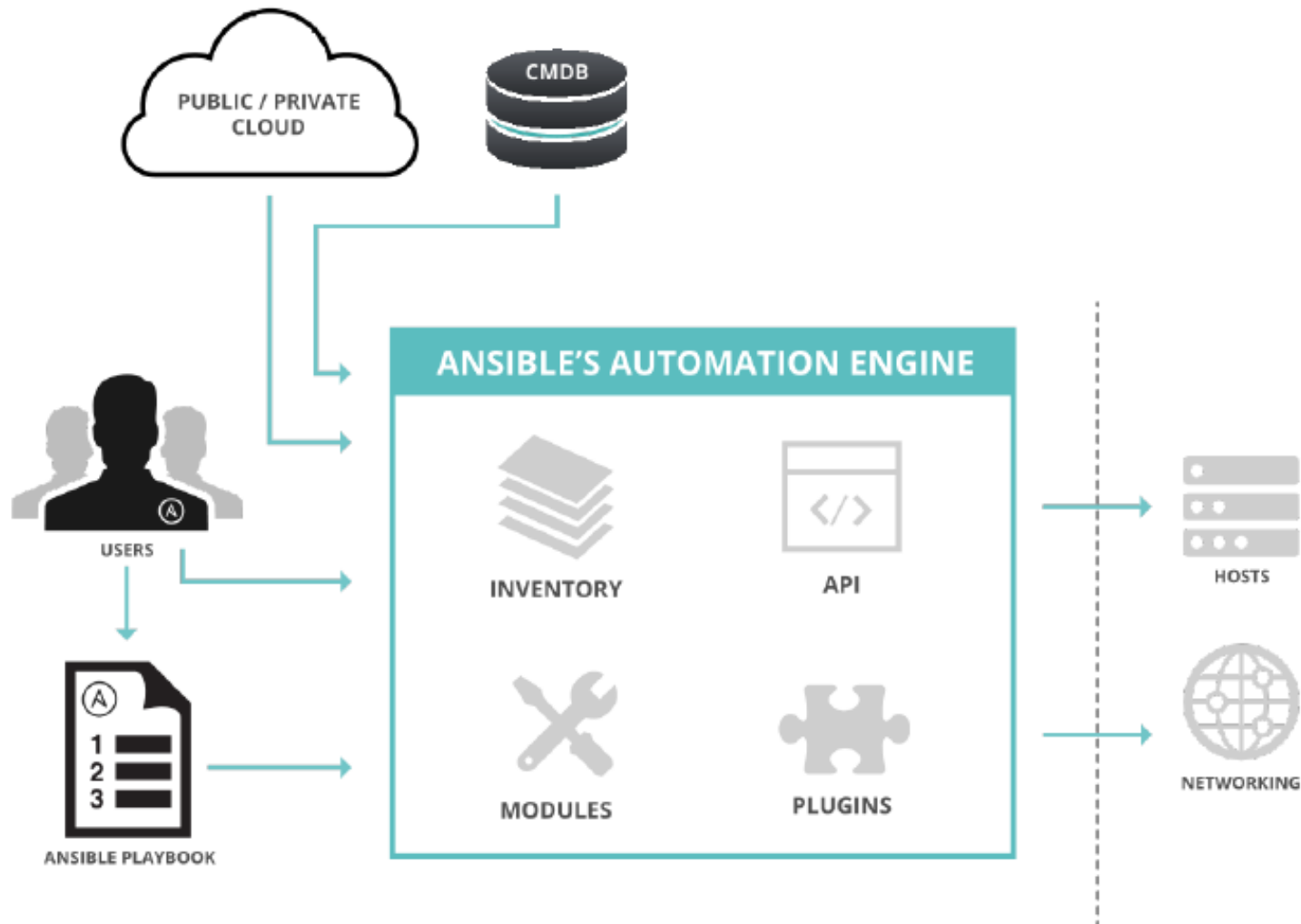
- App deployment
- Configuration management
- Workflow orchestration
- Orchestrate the app lifecycle**



AGENTLESS

- Agentless architecture
- Uses OpenSSH & WinRM
- No agents to exploit or update
- More efficient & more secure**

How Ansible Works?



Ansible Ways

- **Complexity Kills Productivity**

Ansible is designed so that its tools are simple to use and automation is simple to write and read.

- **Optimize For Readability**

The Ansible automation language is built around simple, declarative, text-based files that are easy for humans to read

- **Think Declaratively**

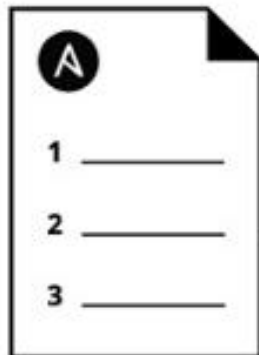
Ansible is a desired-state engine. It approaches the problem of how to automate IT deployments by expressing them in terms of the state that you want your systems to be in.










10 SERVERS



APP SERVERS



- 1   STOP MONITORING
- 2   REMOVE FROM LOAD BALANCING
- 3   STOP SERVICES
- 4   DEPLOY APPLICATION
- 5  REPEAT STEPS 3, 2, 1
- 6  MOVE TO NEXT 10 SERVERS



Ansible Nodes

Control Nodes

The Ansible software only needs to be installed on the control node (or nodes) from which Ansible will be run.

The control node should be a Linux or UNIX system. Microsoft Windows is not supported as a control node, although Windows systems can be managed hosts.

Managed Hosts

One of the benefits of Ansible is that managed hosts do not need to have a special agent installed. The Ansible control node connects to managed hosts using a standard network protocol to ensure that the systems are in the specified state.

Inventory

Inventory is a collection of hosts (nodes) with associated data and groupings that Ansible can connect and manage.

- Hosts (nodes)
- Groups
- Inventory-specific data (variables)
- Static or dynamic sources
- 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, 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.



Inventory

Static Inventory

A static inventory file is an INI-like text file that specifies the managed hosts that Ansible targets.

#basic ini file

```
web1.example.com
web2.example.com
db1.example.com
db2.example.com
```

#host group file

```
[webservers]
web1.example.com
web2.example.com
```

```
[db-servers]
db1.example.com
db2.example.com
```

Ansible Modules

Modules are bits of code transferred to the target system and executed to satisfy the task declaration.

All Ansible modules technically return JSON format data.

- copy
- user
- File
- apt/yum
- get_url
- ping
- debug

Modules : Run Commands

If Ansible doesn't have a module that suits your needs there are the “**run command**” modules:

command: Takes the command and executes it on the host. The most secure and predictable.

shell: Executes through a shell like /bin/sh so you can use pipes etc. Be careful.

script: Runs a local script on a remote node after transferring it.

raw: Executes a command without going through the Ansible module subsystem.

NOTE: Unlike standard modules, run commands have no concept of desired state and should only be used as a last resort.



Ansible Playbooks

A playbook is a text file written in YAML format, and is normally saved with the extension `.yaml`.

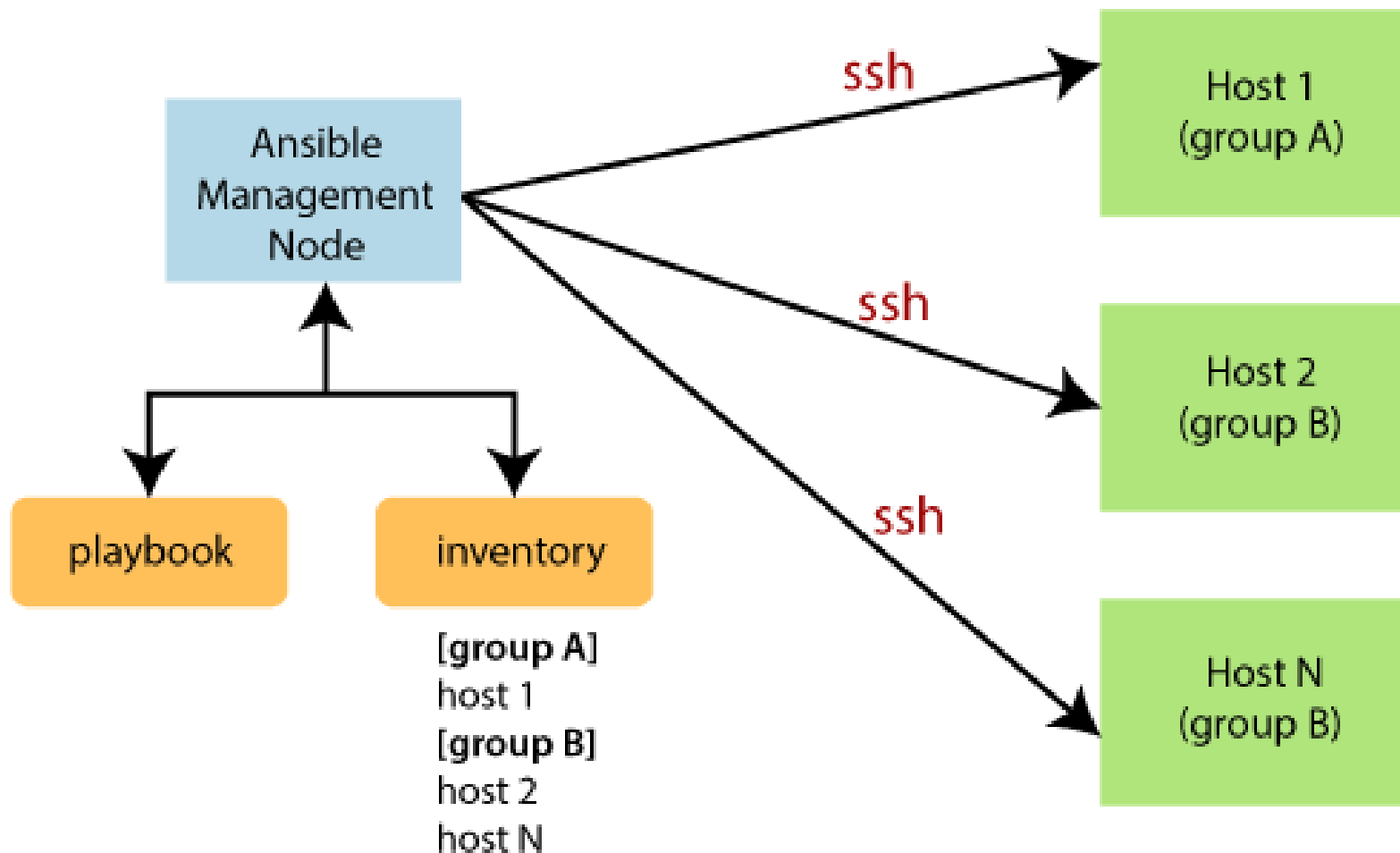
Running Playbooks

The `ansible-playbook` command is used to run playbooks

A **play** is an ordered set of tasks which should be run against hosts selected from your inventory.

A **playbook** is a text file that contains a list of one or more plays to run in order.

Ansible Working





Ansible Roles

- Instead of requiring you to explicitly include certain files and playbooks in a role, Ansible automatically includes any `main.yml` files inside specific directories that make up the role.
- There are only two directories required to make a working Ansible role:
 - `rolename/`
 - `meta/`
 - `tasks/`
- If you create a directory structure like the one shown above, with a *`main.yml`* file in each directory, Ansible will run all the tasks defined in *`tasks/main.yml`* if you call the role from your playbook using the following syntax:

- hosts: all
 roles:
 - role_name

Ansible Roles

- Another simple way to build the scaffolding for a role is to use the command:
`$ ansible-galaxy init role_name`
- Running this command creates an example role in the current working directory, which you can modify to suit your needs.
- Using the **init** command also ensures the role is structured correctly in case you want to someday contribute the role to Ansible Galaxy.

Variable Precedence

- **Variable Precedence**

It should be rare that you would need to dig into the details of which variable is used when you define the same variable in five different places, but since there are odd occasions where this is the case, Ansible's documentation provides the following ranking:

1. **--extra-vars** passed in via the command line
2. **Task-level vars** (in a task block).
3. **Block-level vars** (for all tasks in a block).
4. **Role vars** (e.g. [role]/vars/main.yml) and vars from include_vars module.
5. Vars set via **set_facts** modules.
6. Vars set via **register** in a task.
7. Individual play-level vars: 1. vars_files 2. vars_prompt 3. vars
8. Host facts.
9. Playbook host_vars.
10. Playbook group_vars.
11. Inventory: 1. host_vars 2. group_vars 3. vars
12. **Role default vars** (e.g. [role]/defaults/main.yml).

Ansible Terminologies

- **Controller Machine:** The Controller machine is used to provisioning the servers, which is managed. This is the machine where Ansible is installed.
- **Inventory:** An inventory is an initialization file which has details about the different servers you are managing.
- **Playbook:** It is a code file that is written in the YAML format. A playbook contains the tasks that need to be automated or executed.
- **Task:** Every task represents a single procedure that needs to be executed.
- **Module:** A module is the set of tasks that can be executed. Ansible has 100s of built-in modules, and also you can create custom ones.
- **Role:** The role is a pre-defined way for organizing playbooks and other files to facilitate sharing and reusing portions of provisioning.
- **Play:** The task executed from start to finish, or the execution of a playbook is called the play.
- **Facts:** Facts are global variables which are store details about the system, such as network interfaces or operating system.
- **Handlers:** Handlers are used to trigger the status of a service, such as restarting or stopping a service.

