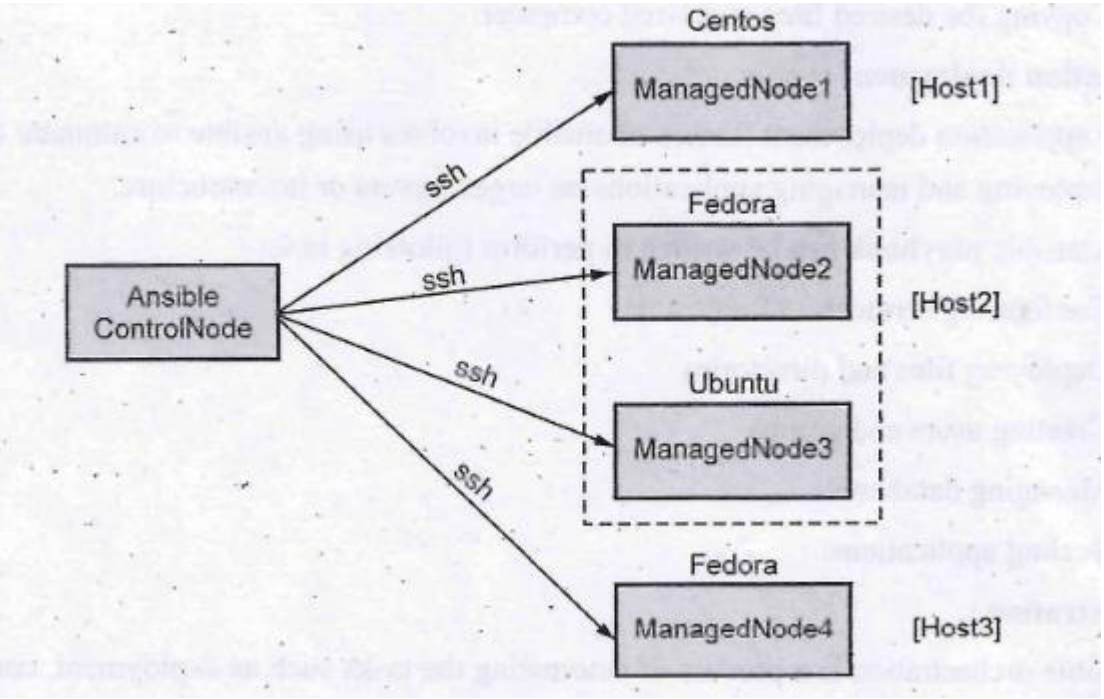
4.3 ANSIBLE MASTER SLAVE CONFIGURATION

Ansible functions on a master-slave architecture, where a central control machine (the "master") manages and automates tasks on remote servers (the "slaves" or "managed nodes").

Architecture:



(i) Ansible Control Machine (Master): This is the machine where Ansible is installed and from where all automation tasks are initiated. It contains:

- **Ansible Engine**: The core Ansible software.
- **Inventory**: A file (or dynamic source) listing all managed nodes, their IP addresses or hostnames, and any group assignments.
- **Playbooks**: YAML files defining the automation tasks to be executed on the managed nodes.
- **Modules**: Pre-built units of code that perform specific tasks (e.g., installing software, managing files, starting services).
- **Configuration Files**: ansible.cfg for global settings and other configuration details.

(ii) Managed Nodes (Slaves):

These are the remote servers or devices that Ansible manages. They do not require any Ansible software or agents installed on them. They only need:

• **SSH Server**: For secure communication with the Ansible control machine.

• **Python**: Most Ansible modules require Python to be present on the managed node to execute tasks.

Working:

• Inventory Definition: The Ansible control machine is configured with an inventory file that lists all the managed nodes it needs to control.

```
[webservers]
web1.example.com
web2.example.com
[databases]
db1.example.com
```

• **SSH Connection**: Ansible connects to the managed nodes using SSH (Secure Shell). Passwordless authentication using SSH keys is typically set up for convenience and security.

• **Module Execution**: When a playbook is executed, Ansible pushes small programs called "modules" to the managed nodes over SSH. These modules are executed on the managed nodes to perform the specified tasks.

• **Task Execution and State Management**: The modules execute the tasks on the managed nodes, ensuring they reach the desired state defined in the playbooks. Ansible is idempotent, meaning it will only make changes if the system is not already in the desired state.

• **Result Collection**: After module execution, Ansible collects the results and returns them to the control machine, providing feedback on the success or failure of the tasks.

This agentless architecture simplifies setup and maintenance, as there's no need to install and manage agents on each managed node.

4.4 YAML BASICS

Ansible heavily relies on YAML (YAML Ain't Markup Language) for defining its automation workflows, particularly through Ansible Playbooks. YAML is chosen for its human-readable and straightforward syntax, making it easier to write, understand, and maintain configuration files and automation scripts compared to formats like XML or JSON.

Key aspects of YAML in Ansible:

• **Playbooks**: Ansible Playbooks, which are the core of Ansible automation, are written entirely in YAML. They describe the tasks to be executed on managed hosts.

• **Structure:** YAML uses indentation to define the structure and hierarchy of data. This is crucial for correctly representing plays, tasks, modules, and their parameters within a playbook.

• **Data Representation**: YAML represents data using key-value pairs, lists (sequences), and dictionaries (mappings).

• **Key-value pairs**: Simple data representation, e.g., name: "Install Nginx".

• **Lists**: Ordered collections of items, typically indicated by a dash (-) at the same indentation level, e.g., a list of tasks in a play.

• **Dictionaries**: Unordered collections of key-value pairs, used to group related properties, e.g., parameters for an Ansible module.

• **Indentation**: Correct indentation is paramount in YAML. It defines the relationships between different elements in a playbook. Incorrect indentation will lead to syntax errors.

• **Comments**: Lines starting with a hash symbol (#) are treated as comments and are ignored during execution, allowing for documentation within the playbook.

• **Anchors and Aliases**: YAML features like anchors (&) and aliases (*) allow for defining and reusing variable values or complex data structures within a playbook, promoting reusability and reducing redundancy.

Understanding YAML syntax and its specific application within Ansible is fundamental for anyone working with Ansible automation. Tools like yamllint can be used to validate the syntax of YAML files.

List :

All members of a list are lines beginning at the same indentation level starting with a "- " (a dash and a space):

```
---
# A list of tasty fruits
- Apple
- Orange
- Strawberry
- Mango
```

dictionary : A dictionary is represented in a simple key: value form (the colon must be followed by a space):

```
# An employee record
martin:
  name: Martin D'vloper
  job: Developer
  skill: Elite
```

- More complicated data structures are possible, such as lists of dictionaries, dictionaries whose values are lists or a mix of both:

```
# Employee records
- martin:
    name: Martin D'vloper
    job: Developer
    skills:
      - python
      - perl
      - pascal
```

```
- tabitha:
    name: Tabitha Bitumen
    job: Developer
    skills:
      - lisp
      - fortran
      - erlang
```

- Dictionaries and lists can also be represented in an abbreviated form if you really want to:

```
---
martin: {name: Martin D'vloper, job: Developer, skill: Elite}
fruits: ['Apple', 'Orange', 'Strawberry', 'Mango']
```

These are called “Flow collections”.

- Ansible doesn’t really use these too much, but you can also specify a boolean value (true/false) in several forms:

```
create_key: true
needs_agent: false
knows_oop: True
likes_emacs: TRUE
uses_cvs: false
```

Use lowercase ‘true’ or ‘false’ for boolean values in dictionaries if you want to be compatible with default yamllint options.

- Values can span multiple lines using | or >. Spanning multiple lines using a “Literal Block Scalar” | will include the newlines and any trailing spaces. Using a “Folded Block Scalar” > will fold newlines to spaces; it is used to make what would otherwise be a very long line easier to read and edit. In either case the indentation will be ignored. Examples are:

```
include_newlines: |
        exactly as you see
        will appear these three
```

```
        lines of poetry
fold_newlines: >
        this is really a
        single line of text
        despite appearances
```

Example YAML file (Ansible Playbook)

```
- name: Install Nginx
  hosts: webservers
  become: yes
  tasks:
    - name: Install nginx
      apt:
        name: nginx
        state: present
```