ANSIBLE

ANSIBLE

A SCM tool

Software Configuration Management (SCM)

- Software Configuration Management (SCM) is the process of systematically managing changes to software to maintain integrity and traceability throughout the software development lifecycle.

Software Configuration Management (SCM)

- SCM includes:
  - Version control (tracking code changes)
  - Build management (automating compilation and dependencies)
  - Change management (controlling modifications)
  - Release management (packaging and deploying)
  - Configuration management (ensuring consistency across environments)
  - SCM helps teams collaborate efficiently, minimize conflicts, and ensure

What is Ansible?

- Ansible is an open-source IT automation tool used for configuration management, application deployment, and orchestration.
- It automates repetitive tasks like setting up servers, installing packages, managing configurations, and deploying applications.

What is Ansible?

- Thus, Ansible is a suite of software tools that enables infrastructure as code.
- Originally written by Michael DeHaan in 2012, and acquired by Red Hat in 2015, Ansible is designed to configure both Unix-like systems and Microsoft Windows.

What is Ansible?

- Ansible is **agentless**, meaning it does not require agents installed on managed nodes.
- It uses SSH for Linux and WinRM for Windows machines.
- Ansible was initially designed to manage Linux/Unix environments using SSH for communication. This is why its core focus and tooling are naturally Linux-centric.

What is Ansible?

- Ansible does support Windows to some extent, it does so differently.
- Instead of SSH, it relies on **WinRM (Windows Remote Management)** to communicate with Windows systems.
- However, this setup requires some additional configuration

Where is Ansible Used?

- **Infrastructure as Code (IaC):** Automating infrastructure provisioning
- **Application Deployment:** Managing multi-tier applications
- **Configuration Management:** Ensuring systems have the correct settings

Where is Ansible Used?

- **Orchestration:** Managing complex workflows (e.g., scaling applications)
- **Security Compliance:** Enforcing security policies

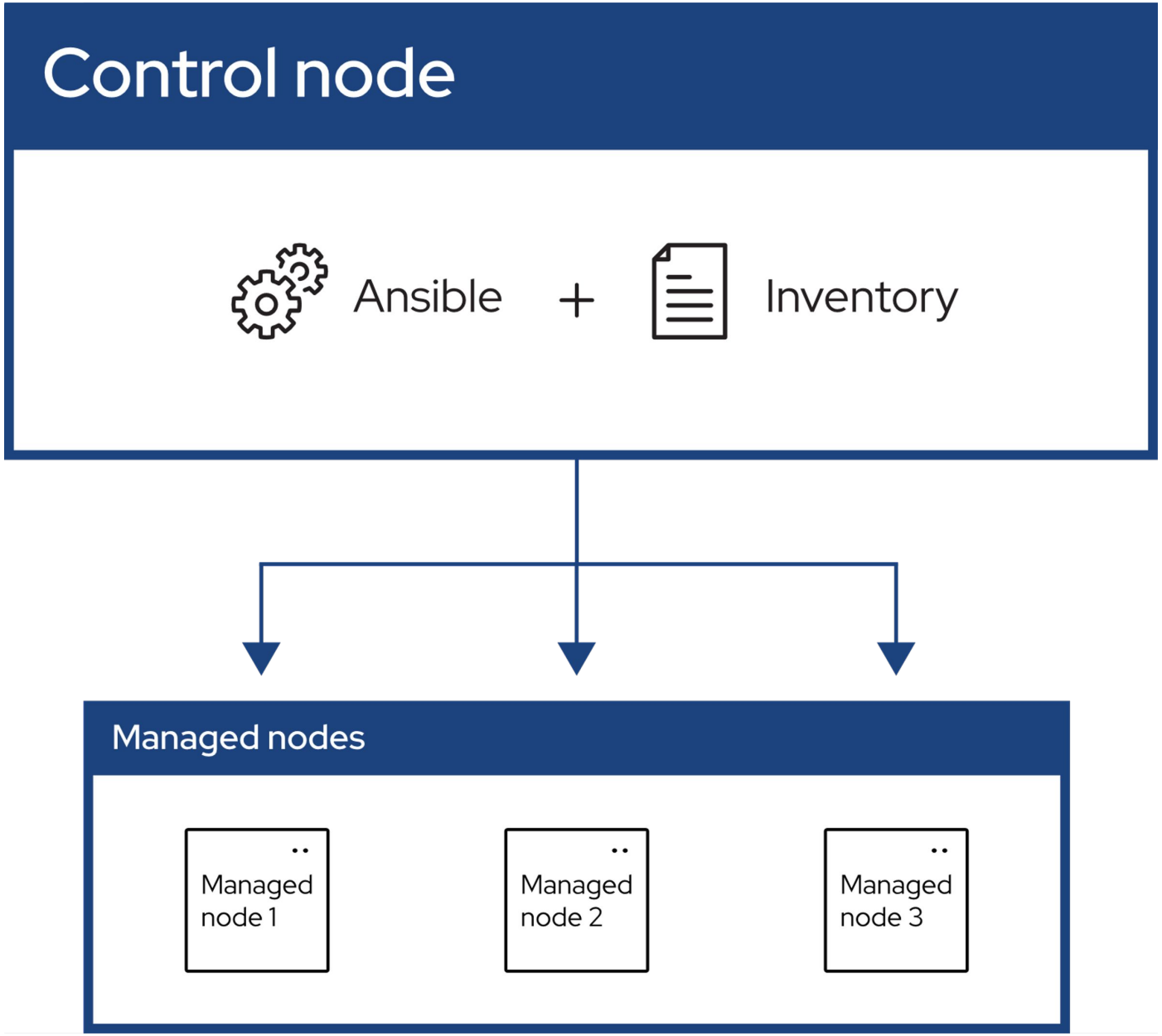
Terms used in Ansible

- **Host:**
  - A remote machine managed by Ansible.
- **Control Node and Managed Nodes**:
  - Ansible operates with a control node (where Ansible is installed) and managed nodes (the systems you want to manage).

Terms used in Ansible

- **Group:**
  - Several hosts grouped together that share a common attribute.
- **Inventory**:
  - The control node uses an inventory file to list and organize the managed nodes.

Terms used in Ansible



Terms used in Ansible

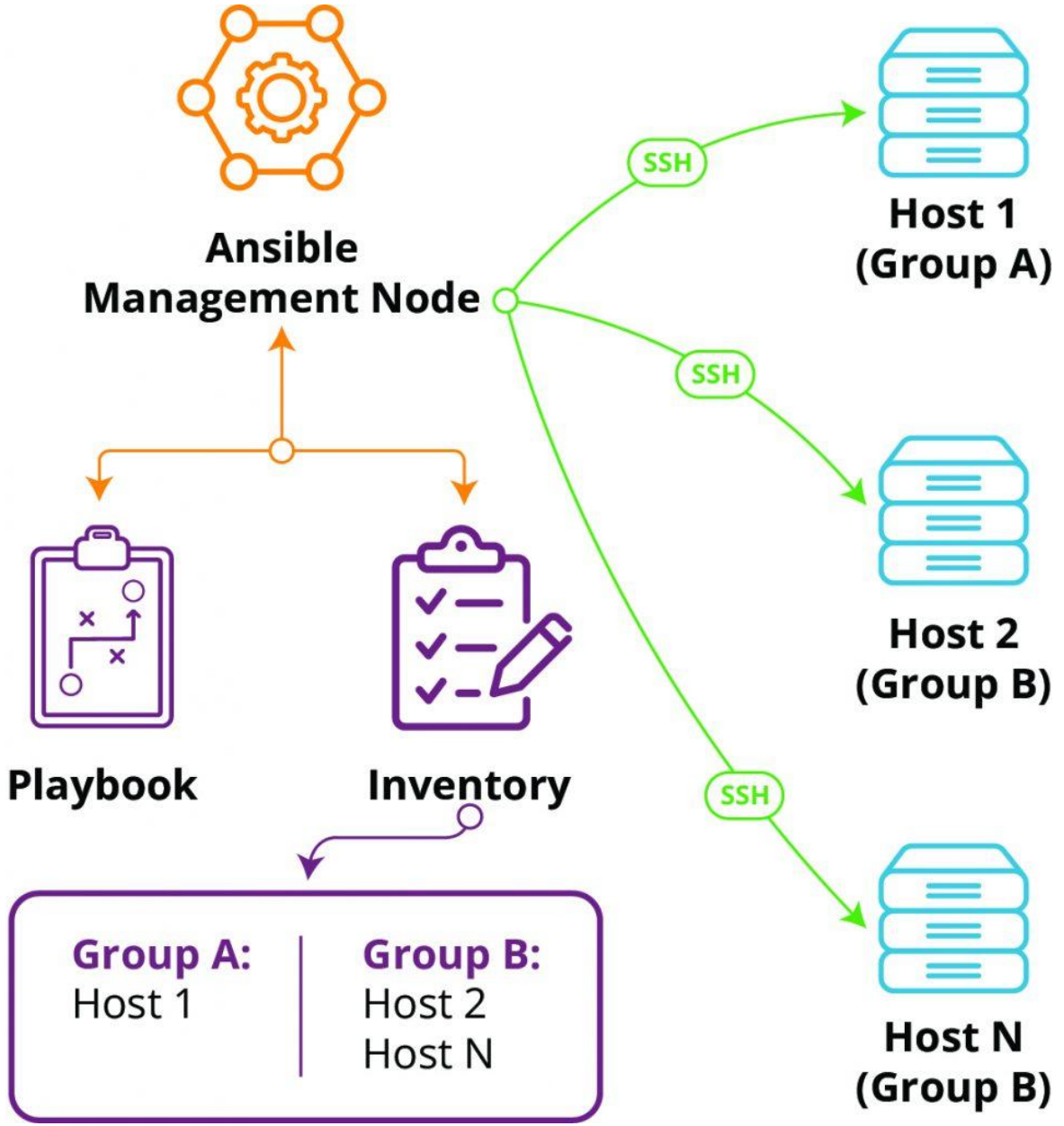


Image curtesy - https://www.liquidweb.com/blog/what-is-ansible/

Terms used in Ansible

- **Modules**:
  - Units of code that Ansible sends to the remote nodes for execution.
  - Ansible uses modules, which are small programs that perform specific actions on the managed nodes.
- **Tasks:**
  - Units of action that combine a module and its arguments along with some other parameters.

Terms used in Ansible

- **Playbooks**:
  - An ordered list of tasks along with its necessary parameters that define a recipe to configure a system. OR
  - A set of complicated IT tasks that can be automated and carried out with little to no human involvement is called an Ansible Playbook.
  - Ansible uses playbooks, which are YAML files that define the tasks to be performed on the managed nodes.

Terms used in Ansible

- **SSH Connection:**
  - Ansible connects to managed nodes using SSH, a secure protocol for remote access.
- **Execution**:
  - The control node sends the modules to the managed nodes, executes them over SSH, and then removes them after completion.

Terms used in Ansible

- **No Agent:**
  - Ansible does not require any agents or daemons to be installed on the managed nodes.
- **Task Execution:**
  - Ansible executes tasks in the order defined in the playbook.

Terms used in Ansible

- Roles:
  - Redistributable units of organization that allow users to share automation code easier.
  - Ansible roles provide a structured way to organize tasks, templates, files, and variables, making it easier to manage complex automation setups.

How does Ansible work?

- Ansible uses the concepts of -
  - control nodes and
  - managed nodes.
- It connects from the control node, any machine with Ansible installed, to the managed nodes sending commands and instructions to them.

How does Ansible work?

- The units of code that Ansible executes on the managed nodes are called modules.
- Each module is invoked by a task, and an ordered list of tasks together forms a playbook.
- Users write playbooks with tasks and modules to define the desired state of the system.

How does Ansible work?

- The managed machines are represented in a simplistic inventory file that groups all the nodes into different categories.
- Ansible leverages a very simple language, YAML, to define playbooks in a human-readable data format that is really easy to understand.

How does Ansible work?

- Ansible doesn’t require the installation of any extra agents on the managed nodes so it’s simple to start using it.
- Typically, the only thing a user needs is a terminal to execute Ansible commands and a text editor to define the configuration files.

What is an Ansible Playbook?

- An **Ansible Playbook** is a YAML file that defines a set of automation tasks for managing and configuring systems.
- Playbooks contain **plays**, which specify **tasks** that are executed on target systems (hosts).

What is an Ansible Playbook?

- Key Components of a Playbook:
  - Hosts: Defines target machines (inventory).
  - Tasks: Actions to be performed (e.g., installing software, modifying files).
  - Modules: Built-in commands like yum, apt, copy, file, etc.
  - Variables: Store reusable values for dynamic configurations.
  - Handlers: Execute tasks only when a change occurs (e.g., restart a service).

Running Ansible on Windows Platform

- To run Ansible, you will need to install it on a control node, this could be your laptop/desktop for example.
- From this control node, Ansible will connect and manage other machines and orchestrate different tasks.
- The managed nodes also need (in the case of Windows) PowerShell 3.0 or later and at least .NET 4.0 installed.

Running Ansible on Windows Platform

- Open the Command Prompt in administrative mode.
- Run the following command to install windows subsystem for Linux -

```
wsl --install
```

Running Ansible on Windows Platform

- The wsl install command installs WSL and the default Linux distribution (usually Ubuntu).
- But if Docker is installed then WSL does not have Ubuntu installed.
- So, to add Ubuntu use follwing command -

```
wsl --install -d Ubuntu-22.04
```

Running Ansible on Windows Platform

- Check Installed Distributions using following command -

wsl -l –v

- It shows "Ubuntu-22.04" and if it is in "STOPPED" state then use following command **to start the Ubuntu** -

wsl **OR** wsl -d Ubuntu-22.04

- To set the default wsl (with * sign in list) use following command -

Running Ansible on Windows Platform

- Now update the Ubuntu system packages using -

```
sudo apt update && sudo apt upgrade -y
```

- Now install the ansible in Ubuntu -

```
sudo apt install ansible -y
```

- Verify Ansible Installation:

```
ansible --version
```

Running Ansible on Windows Platform

- Check the Ansible configuration directory using cd and ls command. If it is missing, then create it manually:

```
sudo mkdir -p /etc/ansible
```

- Now, create the inventory file :

```
sudo vi /etc/ansible/hosts
```

Running Ansible on Windows Platform

- Inside the file, add:

```
[local]

localhost ansible_connection=local
```

- Press Esc to exit insert mode.
- Type :wq (write and quit) and press Enter to save the file and exit.

Running Ansible on Windows Platform

- Now, test if Ansible can communicate with localhost :

ansible local -m ping

- If everything is set up correctly, you should see:

```
devops@DESKTOP-BAFK3FI:/etc/ansible$ ansible local -m ping
localhost | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Running Ansible on Windows Platform

```
devops@DESKTOP-BAFK3FI:/etc/ansible$ ansible local -m ping
localhost | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

- localhost | SUCCESS → Ansible successfully connected to the localhost (your own machine).
- "ping": "pong" → Confirms that the machine is reachable.
- "discovered_interpreter_python": "/usr/bin/python3" → Ansible detected that Python 3 is installed at /usr/bin/python3 (required for Ansible to work).

Ansible Ad-hoc commands

- In Ansible, ad-hoc commands are
  - single, one-line commands
  - used to quickly execute tasks on remote hosts
  - without writing a playbook,
  - providing a quick and easy way to manage infrastructure.

Ansible Ad-hoc commands

- An Ansible ad hoc command uses the */usr/bin/ansible* command-line tool to automate a single task on one or more managed nodes.
- The ad hoc commands are quick and easy, but they are not reusable.

General Syntax of an Ansible Ad-hoc Command

ansible <target> -m <module> -a "<arguments>" [-b]

- Where -
  - **ansible** - The command to invoke Ansible.
  - **<target>** - The inventory group or host on which to run the command (e.g., local, all, webserver).
  - **-m <module>** - The module to execute (e.g., file, user, apt).
  - **-a "<arguments>"** - The arguments for the module (in quotes).
  - **-b** - (Optional) Runs the command with sudo (root/super user) privileges.

Different (-m) Module Types in Ansible

- Command Execution Modules -
  - command - Runs a command (no shell interpretation)
  - shell - Runs a shell command (used when **piping (|), redirection (>, >>), or shell logic (&&, ||)** is required).
- File and Directory Management Modules
  - file - Creates or removes files, directories, and symlinks.
  - copy - Copies files from the control machine to the target.

Different (-m) Module Types in Ansible

- User and Group Management Modules -
  - user -Creates, modifies, or removes users.
  - group - Creates, modifies, or removes groups.
- Package Management Modules
  - apt - Manages APT packages (Debian-based systems).
  - yum - Manages YUM packages (RHEL-based systems).

Different (-m) Module Types in Ansible

- System Information Modules -
  - ping - Checks if a machine is reachable.
  - setup - Gathers system facts.
  - hostname - Changes or gets the system hostname.

Different (-m) Module Types in Ansible

- Service Management Modules -
  - service - Starts, stops, or restarts services.
  - cron - Manages cron jobs.
- Cloud and Virtualization Modules
  - docker_container - Manages Docker containers.
  - docker_image - Manages Docker images.

Running Ad-hoc Ansible commands

- Check Who You Are -

```
devops@DESKTOP-BAFK3FI:/etc/ansible$ ansible local -m shell -a "whoami"
localhost | CHANGED | rc=0 >>
devops
```

- This command runs **whoami** on the local machine and outputs the Ubuntu username.

Running Ad-hoc Ansible commands

- List Files in a Directory -

```
devops@DESKTOP-BAFK3FI:/etc/ansible$ ansible local -m shell -a "ls -la /home/$USER"
localhost | CHANGED | rc=0 >>
total 36
drwxr-x--- 5 devops devops 4096 Mar 17 11:46 .
drwxr-xr-x 3 root   root   4096 Mar 13 12:44 ..
drwxr-xr-x 3 devops devops 4096 Mar 17 11:46 .ansible
-rw------- 1 devops devops  363 Mar 17 14:21 .bash_history
-rw-r--r-- 1 devops devops  220 Mar 13 12:44 .bash_logout
-rw-r--r-- 1 devops devops 3771 Mar 13 12:44 .bashrc
drwx------ 2 devops devops 4096 Mar 13 12:45 .cache
```

- Lists all files (including hidden ones) in your home directory.

Running Ad-hoc Ansible commands

- Add the contents to the file and display the files of a directory -

```
devops@DESKTOP-BAFK3FI:/mnt/e/ansiblePract$ ansible local -m shell -a "echo 'Hello' > /tmp/hello.txt"
localhost | CHANGED | rc=0 >>

devops@DESKTOP-BAFK3FI:/mnt/e/ansiblePract$ ansible local -m shell -a "ls -l /tmp"
localhost | CHANGED | rc=0 >>
total 24
drwx------ 2 devops devops 4096 Mar 18 12:12 ansible_ansible.legacy.command_payload__gs0zc9x
-rw-r--r-- 1 devops devops    6 Mar 18 12:11 hello.txt
drwx------ 2 root   root   4096 Mar 18 10:18 snap-private-tmp
drwx------ 3 root   root   4096 Mar 18 10:18 systemd-private-731d658998eb4c3aa6f1273ce452b63b-systemd-
y0d
drwx------ 3 root   root   4096 Mar 18 10:18 systemd-private-731d658998eb4c3aa6f1273ce452b63b-systemd-
bsyIa
drwx------ 3 root   root   4096 Mar 18 10:18 systemd-private-731d658998eb4c3aa6f1273ce452b63b-systemd-
lRgnsc
```

Running Ad-hoc Ansible commands

- Filters out the file names those contains the word "hello" -

```
devops@DESKTOP-BAFK3FI:/mnt/e/ansiblePract$ ansible local -m shell -a "ls -l /tmp | grep hello"
localhost | CHANGED | rc=0 >>
-rw-r--r-- 1 devops devops    6 Mar 18 12:11 hello.txt
```

- The command "ls -l /tmp", lists files from the /tmp directory using long format, showing file permissions, owner, size, and modification time.
- The command "| grep hello" pipes the output of ls -l /tmp to grep to filter out the lines that contain the word "hello".

Running Ad-hoc Ansible commands

- Check System Uptime -

```
devops@DESKTOP-BAFK3FI:/etc/ansible$ ansible local -m command -a "uptime"
localhost | CHANGED | rc=0 >>
 15:01:16 up  3:58,  1 user,  load average: 0.00, 0.00, 0.00
```

- Displays how long your system has been running with load average.

Running Ad-hoc Ansible commands

- Displays the OS version and other details -

```
devops@DESKTOP-BAFK3FI:/mnt/e/ansiblePract$ ansible local -m command -a "cat /etc/os-release"
localhost | CHANGED | rc=0 >>
PRETTY_NAME="Ubuntu 22.04.5 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.5 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
```

Running Ad-hoc Ansible commands

- Install a Package (e.g., nginx) -

```
devops@DESKTOP-BAFK3FI:/etc/ansible$ ansible local -m apt -a "name=nginx state=present" -b
localhost | FAILED! => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "module_stderr": "sudo: a password is required\n",
    "module_stdout": "",
    "msg": "MODULE FAILURE\nSee stdout/stderr for the exact error",
    "rc": 1
}
```

- If, you are getting password required error then **edit visudo file** by adding **YOUR_USERNAME ALL=(ALL) NOPASSWD: ALL** line at the bottom.

Running Ad-hoc Ansible commands

- Install a Package (e.g., `nginx`) -

```
devops@DESKTOP-BAFK3FI:/etc/ansible$ ansible local -m apt -a "name=nginx state=present" -b
localhost | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1742189643,
    "cache_updated": false,
    "changed": true,
    "stderr": "",
    "stderr_lines": [],
    "stdout": "Reading package lists...\nBuilding dependency tree...\nReading state informa
onal packages will be installed:\n  libgd3 libnginx-mod-http-geoip2 libnginx-mod-http-imag
xslt-filter libnginx-mod-mail libnginx-mod-stream\n  libnginx-mod-stream-geoip2 libxpm4 li
```

- Now, it should install nginx without asking for a password!

- To verify installation use "dpkg -l | grep nginx" command.

Running Ad-hoc Ansible commands

- Remove a Package (e.g., `nginx`) -

```
devops@DESKTOP-BAFK3FI:/mnt/e/ansiblePract$ ansible local -m apt -a "name=nginx state=absent" -b
localhost | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "stderr": "",
    "stderr_lines": [],
    "stdout": "Reading package lists...\nBuilding dependency tree...\nReading state information...\nThe following packag
es were automatically installed and are no longer required:\n  libgd3 libnginx-mod-http-geoip2 libnginx-mod-http-image-f
ilter\n  libnginx-mod-http-xslt-filter libnginx-mod-mail libnginx-mod-stream\n  libnginx-mod-stream-geoip2 libxpm4 libxs
lt1.1 nginx-common nginx-core\nUse 'sudo apt autoremove' to remove them.\nThe following packages will be REMOVED:\n  ngi
```

```
        "Removing nginx (1.18.0-6ubuntu14.6) ..."
    ]
}
```

- This uninstall the nginx from the node.

Running Ad-hoc Ansible commands

- Create a User -

```
devops@DESKTOP-BAFK3FI:/mnt/e/ansiblePract$ ansible local -m user -a "name=testuser state=present" -b
localhost | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "comment": "",
    "create_home": true,
    "group": 1001,
    "home": "/home/testuser",
    "name": "testuser",
    "shell": "/bin/sh",
    "state": "present",
    "system": false,
    "uid": 1001
}
```

- The `-b` flag tells Ansible to use sudo, which is required to create a new user. To verfy created user, use "cat /etc/passwd | grep testuser" command.

Running Ad-hoc Ansible commands

- Delete a User -

```
devops@DESKTOP-BAFK3FI:/mnt/e/ansiblePract$ ansible local -m user -a "name=testuser state=absent" -b
localhost | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "force": false,
    "name": "testuser",
    "remove": false,
    "state": "absent"
}
```

- This command deletes the user created with -b flag that tells Ansible to use sudo, which is required to delete a user.

Running Ad-hoc Ansible commands

- Dealing with a File -
  - Create a File

    ansible local -m file -a "path=/tmp/myfile.txt state=touch"
  - Add Content to the File

    ansible local -m shell -a "echo 'This is a first line.' > /tmp/myfile.txt"
  - Append Content to the File

    ansible local -m shell -a "echo 'This is an additional line.' >> /tmp/myfile.txt"

Running Ad-hoc Ansible commands

- Dealing with a File -
  - Display the File's Contents

    ansible local -m command -a "cat /tmp/myfile.txt"
  - Remove the File

    ansible local -m file -a "path=/tmp/myfile.txt state=absent"

Running Ad-hoc Ansible commands

To

create a

directory

Verify the directory

To remove directory

```
devops@DESKTOP-BAFK3FI:/mnt/e/ansiblePract$ ansible local -m file -a "path=/tmp/mydir state=directory mode=0755"
localhost | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "gid": 1000,
    "group": "devops",
    "mode": "0755",
    "owner": "devops",
    "path": "/tmp/mydir",
    "size": 4096,
    "state": "directory",
    "uid": 1000
}
devops@DESKTOP-BAFK3FI:/mnt/e/ansiblePract$ ls -la /tmp | grep mydir
drwxr-xr-x  2 devops devops 4096 Mar 18 11:09 mydir
devops@DESKTOP-BAFK3FI:/mnt/e/ansiblePract$ ansible local -m file -a "path=/tmp/mydir state=absent"
localhost | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "path": "/tmp/mydir",
    "state": "absent"
}
devops@DESKTOP-BAFK3FI:/mnt/e/ansiblePract$ ls -la /tmp | grep mydir
```

Ansible Playbook – An Overview

- An Ansible Playbook is a YAML-based script that automates complex IT tasks, including configuration management, application deployment, and orchestration of multiple tasks across multiple systems.
- It defines what actions need to be performed on which machines in a structured and reusable way.

Ansible Playbook – An Overview

- An Ansible® Playbook is a blueprint of automation tasks, which are IT actions executed with limited manual effort across an inventory of IT solutions.
- Playbooks tell Ansible what to do to which devices.

How do Ansible Playbooks work?

- Ansible Playbooks consist of a series of tasks that are automatically executed on specified inventory or host groups.
- Multiple tasks can be grouped into a play, which defines an ordered set of actions mapped to specific hosts.
- Tasks within a play are executed sequentially in the order they are defined.

How do Ansible Playbooks work?

- A playbook can contain one or more plays, as well as Ansible Roles, which are reusable collections of tasks and automation components that can be applied across multiple plays or playbooks.
- Each task is executed using a module, which is responsible for performing a specific action within the playbook.

How do Ansible Playbooks work?

- Modules include metadata that determines where, when, and under which user a task is executed.
- Ansible provides thousands of modules designed to automate a wide range of IT tasks.

How do Ansible Playbooks work?

- Workflow of Ansible Playbooks
  - 1 The Control Node Reads the Playbook -
    - The control node (where Ansible is installed) reads the YAML-based playbook.
    - The playbook contains hosts, tasks, and modules to be executed.
  - 2 The Control Node Connects to Managed Nodes
    - Ansible uses SSH (for Linux) or WinRM (for Windows) to connect to managed nodes.
    - It does not require agents on the managed nodes.

How do Ansible Playbooks work?

- Workflow of Ansible Playbooks
  - 3️⃣ Executes Tasks Using Ansible Modules
    - Each task in the playbook calls a module (e.g., apt, file, copy).
    - The module executes commands remotely and enforces the desired state.
  - 4️⃣ Ensures Idempotency
    - If a task is already completed, Ansible skips it (e.g., if Apache is installed, it won’t install again).

How do Ansible Playbooks work?

- Workflow of Ansible Playbooks
  - 5️⃣ Playbook Execution Completes
    - Once all tasks are executed, the playbook finishes, and changes are applied to the system.

Playbook directory structure

- Although Ansible doesn't enforce a specific directory structure, it's recommended that you implement a best-practice directory layout to help organize and scale your Ansible playbook projects.

```
project-name/
├── inventory                # Inventory files
│   ├── production.yml       # Production servers inventory
│   └── staging.yml          # Staging servers inventory
├── group_vars/              # Variables for groups
│   ├── web.yml              # Variables for 'web' group
├── host_vars/               # Variables for specific hosts
│   ├── web1.yml             # Variables for ‘web1’ node
│   └── web2.yml             # Variables for 'web2' node
├── site.yml                 # Playbook site.yml'
└── files/                   # Other files
│   ├──  file1.txt
└── templates/               # Jinja2 templates
│   ├── index.html.j2
```

Playbook directory structure

- However, you can configure a default directory in `/etc/ansible/ansible.cfg` under `roles_path`.
- One can store playbooks anywhere and specify the full path using the command -

```
ansible-playbook /path/to/my-playbook.yml
```

Key Components of Playbooks

Component	Description
 hosts	Specifies **target machines** to apply tasks.
 tasks	List of **actions** to be performed.
 modules	Predefined **commands** for automation (e.g., apt, file).
 handlers	Runs a task **only when notified** (e.g., restarting a service).
 vars	Defines **variables** for dynamic playbooks.
 when	Adds **conditions** to tasks (e.g., execute only if a file exists).

Key Components of Playbooks

- Ansible playbooks consist of one or more plays that perform actions on target nodes.
- Consider an example shown

```yaml
- name: Install and Start Apache
  hosts: local
  become: yes
  tasks:
    - name: Install Apache
      apt:
        name: apache2
        state: present

    - name: Start Apache Service
      service:
        name: apache2
        state: started
        enabled: yes
```

First Demo Playbook

- Use the "vi firstplaybook.yml" command to create a playbook in user directory and add the following contents -

```
C:\Windows\System32>wsl -d Ubuntu-22.04
devops@DESKTOP-BAFK3FI:/mnt/c/Windows/System32$ cd ..
devops@DESKTOP-BAFK3FI:/mnt/c/Windows$ cd ..
devops@DESKTOP-BAFK3FI:/mnt/c$ cd ..
devops@DESKTOP-BAFK3FI:/mnt$ cd ..
devops@DESKTOP-BAFK3FI:/$ cd home/myansibleprj/
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ vi firstplaybook.yml
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ cat firstplaybook.yml
---
- name: First demo Ansible playbook
  hosts: localhost
  gather_facts: no
  tasks:
      - name: print hello world
        debug:
            msg: "Hello World!....."
```

First Demo Playbook

- To run and view the output refer the following -

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ ansible-playbook firstplaybook.yml

PLAY [First demo Ansible playbook] ***************************************************
***************

TASK [print hello world] *************************************************************
***************
ok: [localhost] => {
    "msg": "Hello World!....."
}

PLAY RECAP ***************************************************************************
***************
localhost                  : ok=1    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
```

More about First Demo Playbook

- Here, **gather_facts: no** is used because, **Ansible collects system information** (like OS, IP address, hostname, etc.) before running tasks.
- This information is stored in Ansible facts and can be used in playbooks.
- If your playbook **does not depend on system facts**, like in this "Hello, World!" example, disabling facts speeds up execution.

More about First Demo Playbook

- The **debug** module in Ansible is used to display messages or variable values during playbook execution.
- It helps in **troubleshooting, logging, and debugging** Ansible playbooks.

Updating First Demo Playbook

- Now, update the playbook as follows -

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ vi firstplaybook.yml
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ cat firstplaybook.yml
---
- name: First demo Ansible playbook
  hosts: localhost
  tasks:
      - name: print hello world
        debug:
            msg: "Hello World!....."
      - name: print host name
        debug:
            var: ansible_hostname
      - name: print Ubuntu version
        debug:
            var: ansible_distribution
```

Updating First Demo Playbook

- Output -

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ ansible-playbook firstplaybook.yml

PLAY [First demo Ansible playbook] *********************************************

TASK [Gathering Facts] *********************************************************
ok: [localhost]

TASK [print hello world] *******************************************************
ok: [localhost] => {
    "msg": "Hello World!....."
}

TASK [print host name] *********************************************************
ok: [localhost] => {
    "ansible_hostname": "DESKTOP-BAFK3FI"
}

TASK [print Ubuntu version] ****************************************************
ok: [localhost] => {
    "ansible_distribution": "Ubuntu"
}

PLAY RECAP *********************************************************************
localhost                  : ok=4    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
```

Installing and Running Apache using Playbook

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ cat startapache.yml
---
- name: Install and Start Apache
  hosts: local
  become: yes
  tasks:
    - name: Install Apache
      apt:
        name: apache2
        state: present

    - name: Start Apache Service
      service:
        name: apache2
        state: started
        enabled: yes
```

Installing and Running Apache using Playbook

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ ansible-playbook startapache.yml

PLAY [Install and Start Apache] ************************************************

TASK [Gathering Facts] *********************************************************
ok: [localhost]

TASK [Install Apache] **********************************************************
ok: [localhost]

TASK [Start Apache Service] ****************************************************
ok: [localhost]

PLAY RECAP *********************************************************************
localhost                  : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
```

Installing and Running Apache using Playbook

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2025-03-19 10:28:41 IST; 46min ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 198 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 231 (apache2)
      Tasks: 55 (limit: 9149)
     Memory: 18.4M
     CGroup: /system.slice/apache2.service
             ├─231 /usr/sbin/apache2 -k start
             ├─232 /usr/sbin/apache2 -k start
             └─233 /usr/sbin/apache2 -k start
```

Note - To stop the apache, use following command - sudo systemctl stop apache2

Playbook for Infrastructure as Code

- Infrastructure as Code (IaC) sets up servers, networking, security, & system configurations e.g. Installing Apache, Nginx, or MySQL.
- Ansible Playbook Example for Infrastructure as Code (IaC) consist of following tasks -
    - Install Apache
    - Start the Apache service
    - Deploy a simple index.html file

Playbook for Infrastructure as Code

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ vi infraascode2.yml
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ cat infraascode2.yml
---
- name: Simple Infrastructure Setup
  hosts: localhost
  become: yes  # Run tasks as root

  tasks:
    - name: Install Apache
      apt:
        name: apache2
        state: present
        update_cache: yes

    - name: Start Apache service
      service:
        name: apache2
        state: started
        enabled: yes

    - name: Deploy index.html
      copy:
        content: "<h1>Hello World from Ansible......!</h1>"
        dest: /var/www/html/index.html
```

Playbook for Infrastructure as Code

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ vi infraascode2.yml
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ cat infraascode2.yml
---
- name: Simple Infrastructure Setup
  hosts: localhost
  become: yes  # Run tasks as root

  tasks:
    - name: Install Apache
      apt:
        name: apache2
        state: present
        update_cache: yes

    - name: Start Apache service
      service:
        name: apache2
        state: started
        enabled: yes

    - name: Deploy index.html
      copy:
        content: "<h1>Hello World from Ansible......!</h1>"
        dest: /var/www/html/index.html
```

Playbook for Infrastructure as Code

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ ansible-playbook infraascode2.yml

PLAY [Simple Infrastructure Setup] *********************************************

TASK [Gathering Facts] *********************************************************
ok: [localhost]

TASK [Install Apache] **********************************************************
ok: [localhost]

TASK [Start Apache service] ****************************************************
ok: [localhost]

TASK [Deploy index.html] *******************************************************
changed: [localhost]

PLAY RECAP *********************************************************************
localhost                  : ok=4    changed=1    unreachable=0    failed=0
ignored=0
```

Playbook for Infrastructure as Code

Playbook for Application Deployment

- The Infrastructure as Code (IaC) example can also serve as a basic application deployment example because it:
  - Installs Apache (Web Server)
  - Deploys an index.html file (Simple Web Application)
- However, for a true application deployment, one would typically deploy a dynamic web application such as a JSP, PHP, or Node.js application instead of just a static HTML file.

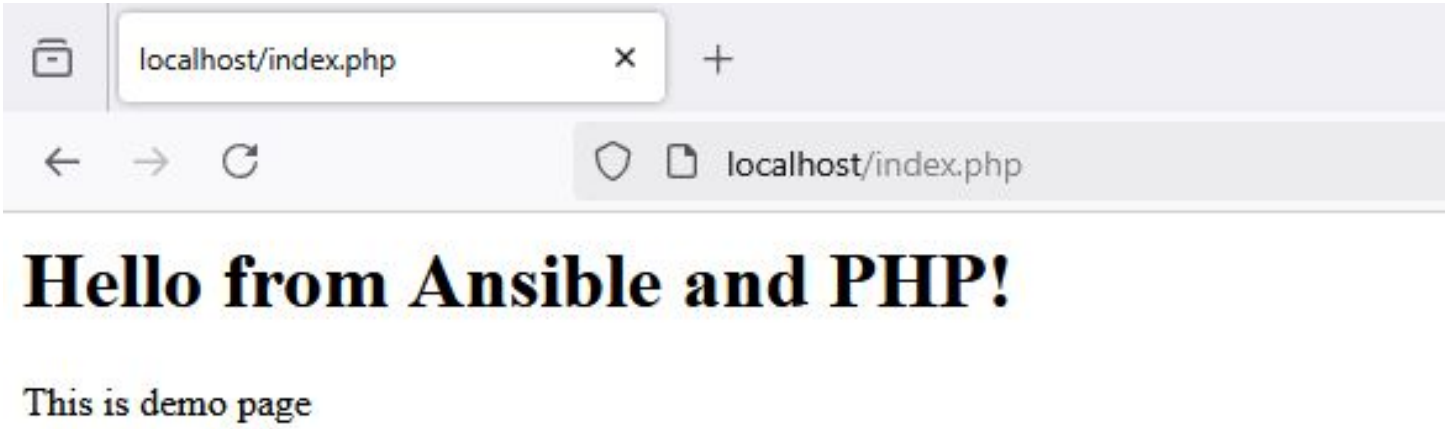
Playbook for Application Deployment

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ cat appdeployplaybook.yml
---
- name: Deploy a PHP Web Application
  hosts: localhost
  become: yes

  tasks:
    - name: Install Apache and PHP
      apt:
        name:
          - apache2
          - php
        state: present

    - name: Deploy a PHP application
      copy:
        content: |
            <?php
            echo "<h1>Hello from Ansible and PHP!</h1>";
            echo "<body>This is demo page</body>";
            ?>
        dest: /var/www/html/index.php

    - name: Restart Apache
      service:
        name: apache2
        state: restarted
```

Playbook for Application Deployment

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ ansible-playbook appdeployplaybook.yml

PLAY [Deploy a PHP Web Application] ********************************************

TASK [Gathering Facts] *********************************************************
ok: [localhost]

TASK [Install Apache and PHP] **************************************************
ok: [localhost]

TASK [Deploy a PHP application] ************************************************
ok: [localhost]

TASK [Restart Apache] **********************************************************
changed: [localhost]

PLAY RECAP *********************************************************************
localhost                  : ok=4    changed=1    unreachable=0    failed=0    ski
ignored=0
```

Playbook for Application Deployment



More About Playbook for Application Deployment

- Here, **become: yes** (or **become: true**) enables privilege escalation (like running sudo in Linux).
- It allows Ansible to run tasks as the root user or another specified user.
- It allows to install apache2 in this case and without become: yes, the task would fail due to permission issues.

More About Playbook for Application Deployment

- The **state**, defines the **desired state** of the resource (package, file, service, etc.).
- Common state values are:

Value	Description
`present`	Ensures the package or file is installed.
`absent`	Ensures the package or file is removed.
`started`	Ensures a service is running.
`stopped`	Ensures a service is stopped.
`latest`	Ensures the latest version of a package is installed.

More About Playbook for Application Deployment

- Here it check whether the apache2 is present or not (as already installed in previous example).
- Also, the pipe (|) allows writing multi-line content inside YAML and it maintains line breaks (preserves formatting).

Playbook for Configuration Management

- Why Playbook for Configuration Management is required:
  - It ensures application (like Apache) is installed (if missing, it installs it).
  - It applies a configuration file (so application (like Apache) always has the correct settings).
  - It ensures application (like Apache) is always running (even after a reboot).
  - If the config file changes, application (like Apache) restarts automatically using notify.

Playbook for Configuration Management

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ vi custom_apache.conf
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ cat custom_apache.conf
<VirtualHost *:80>
    ServerAdmin admin@example.com
    DocumentRoot /var/www/html
    ServerName example.com
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
devops@DESKTOP-BAFK3FI:/home/myansibleprj$
```

Playbook for Configuration Management

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ vi confmgtplaybook.yml
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ cat confmgtplaybook.yml
---
- name: Apache Configuration Management
  hosts: localhost
  become: yes

  tasks:
    - name: Install Apache
      apt:
        name: apache2
        state: present
        update_cache: yes

    - name: Copy custom Apache configuration file
      copy:
        src: /home/myansibleprj/custom_apache.conf
        dest: /etc/apache2/sites-available/custom_apache.conf
        owner: root
        group: root
        mode: '0644'

    - name: Reload Apache to apply changes
      systemd:
        name: apache2
        state: restarted
        enabled: yes
```

Playbook for Configuration Management

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ ansible-playbook confmgtplaybook.yml

PLAY [Apache Configuration Management] ***********************************

TASK [Gathering Facts] ***************************************************
ok: [localhost]

TASK [Install Apache] ****************************************************
ok: [localhost]

TASK [Copy custom Apache configuration file] *****************************
ok: [localhost]

TASK [Reload Apache to apply changes] ************************************
changed: [localhost]

PLAY RECAP ***************************************************************
localhost                  : ok=4    changed=1    unreachable=0    failed=0    sk
```

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2025-03-19 15:02:49 IST; 11min ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 4505 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
```

Playbook for Configuration Management

- If the YML script is not executing properly and apache2 is not starting, then check the port 80.
- If, any process is listing to that port, then kill that process.
- Use following commands for the same -

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ sudo netstat -tulnp | grep ':80\|:443'
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      251/nginx: master p
tcp6       0      0 :::80                   :::*                    LISTEN      251/nginx: master p
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ sudo kill -9 251
```

Playbook for Running HelloWorld.java

- Install jdk and jre in your wsl using following commands -

sudo apt install openjdk-21-jdk-headless

sudo apt install openjdk-21-jre-headless

- Verify the installation using -

java –version

javac -version

Playbook for Running HelloWorld.java

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ vi HelloWorld.java
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ cat HelloWorld.java
class HelloWorld {
        public static void main(String arg[]) {
                System.out.println("Hello World");
        }
}
devops@DESKTOP-BAFK3FI:/home/myansibleprj$
```

Playbook for Running HelloWorld.java

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ vi javaplaybook.yml
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ cat javaplaybook.yml
---
- name: Run Java Program using Ansible
  hosts: localhost
  gather_facts: no
  tasks:
    - name: Compile Java Program
      command: javac /home/myansibleprj/HelloWorld.java
      args:
        chdir: /home/myansibleprj
      register: compile_output

    - name: Run Java Program
      command: java -cp /home/myansibleprj HelloWorld
      args:
        chdir: /home/myansibleprj
      register: run_output

    - name: Display Java Program Output
      debug:
        msg: "{{ run_output.stdout }}"
```

Playbook for Running HelloWorld.java

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ ansible-playbook javaplaybook.yml

PLAY [Run Java Program using Ansible] ******************************************

TASK [Compile Java Program] ****************************************************
changed: [localhost]

TASK [Run Java Program] ********************************************************
changed: [localhost]

TASK [Display Java Program Output] *********************************************
ok: [localhost] => {
    "msg": "Hello World"
}

PLAY RECAP *********************************************************************
localhost                  : ok=3    changed=2    unreachable=0    failed=0    skipped=0
ignored=0
```

More about Java Playbook

- Here, the args keyword allows extra arguments to be passed to a module.
- In this case, it changes the directory to /home/myansibleprj before executing javac Helloworld.java.
- Without args, the command runs in the default home directory.

More about Java Playbook

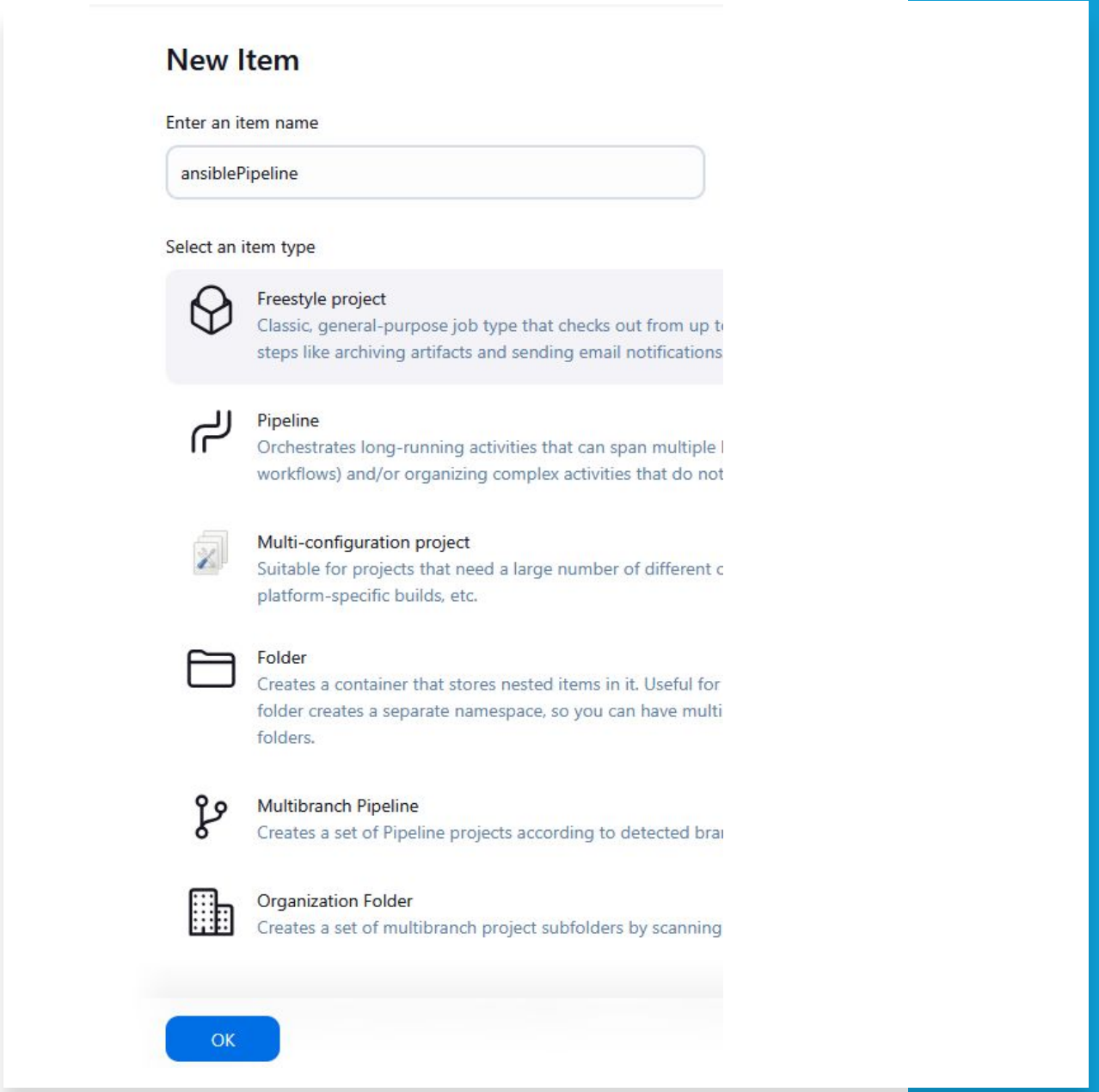
```
- name: Compile Java only if class file does not exist
  command: javac Helloworld.java
  args:
      chdir: /home/myansibleprj
      creates: /home/myansibleprj/Helloworld.class
```

- Here, the creates keyword ensures that javac only runs if Helloworld.class does not exist, avoiding unnecessary recompilation.

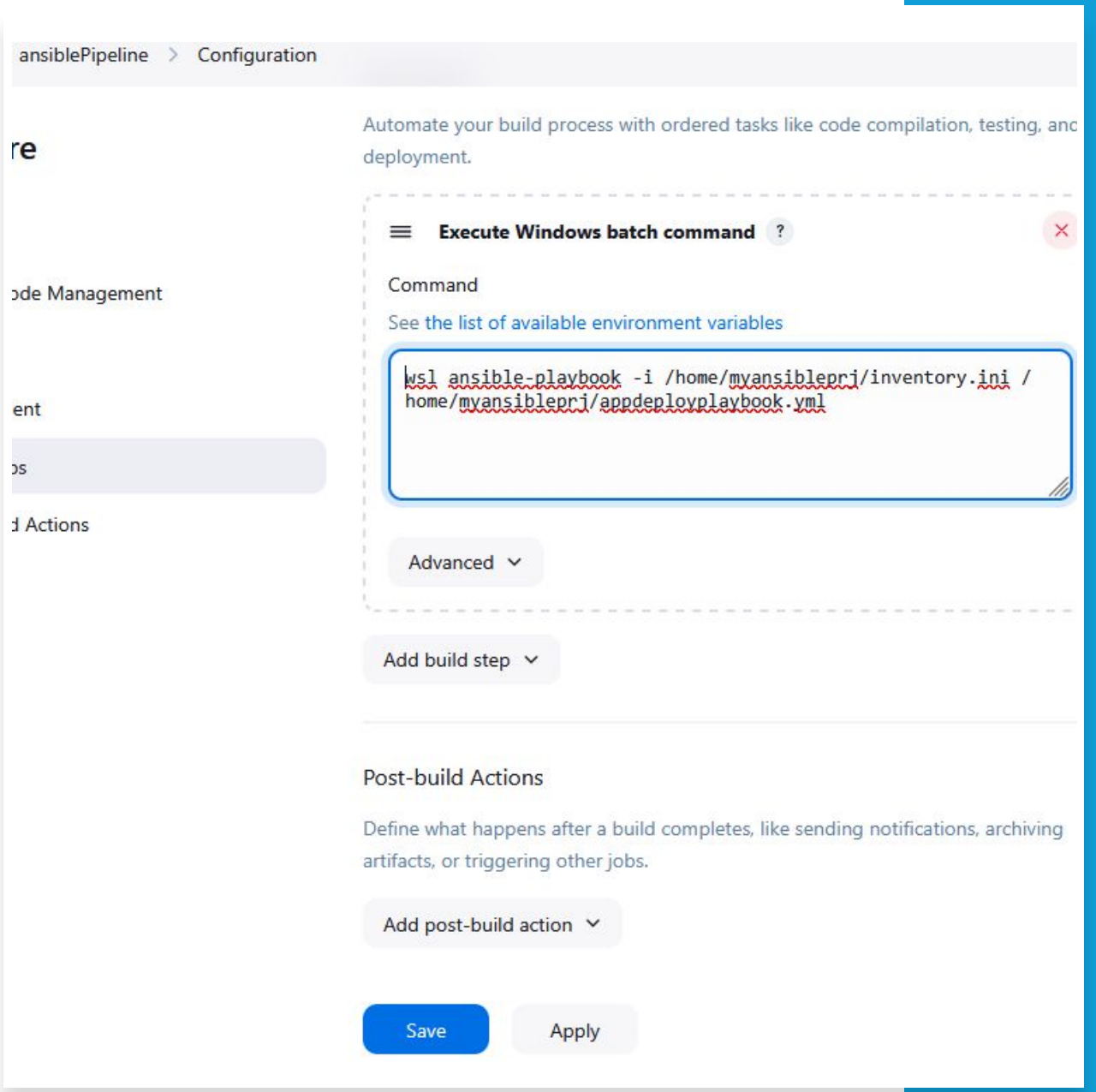
More about Java Playbook

- The **register** keyword stores the output of a command into a variable, which can be used in later tasks.
- The **run_output** stores the output of java Helloworld and debug prints **run_output.stdout**, which contains "Hello World".
- Similary, **stderr** attribute captures the standard error (error messages).

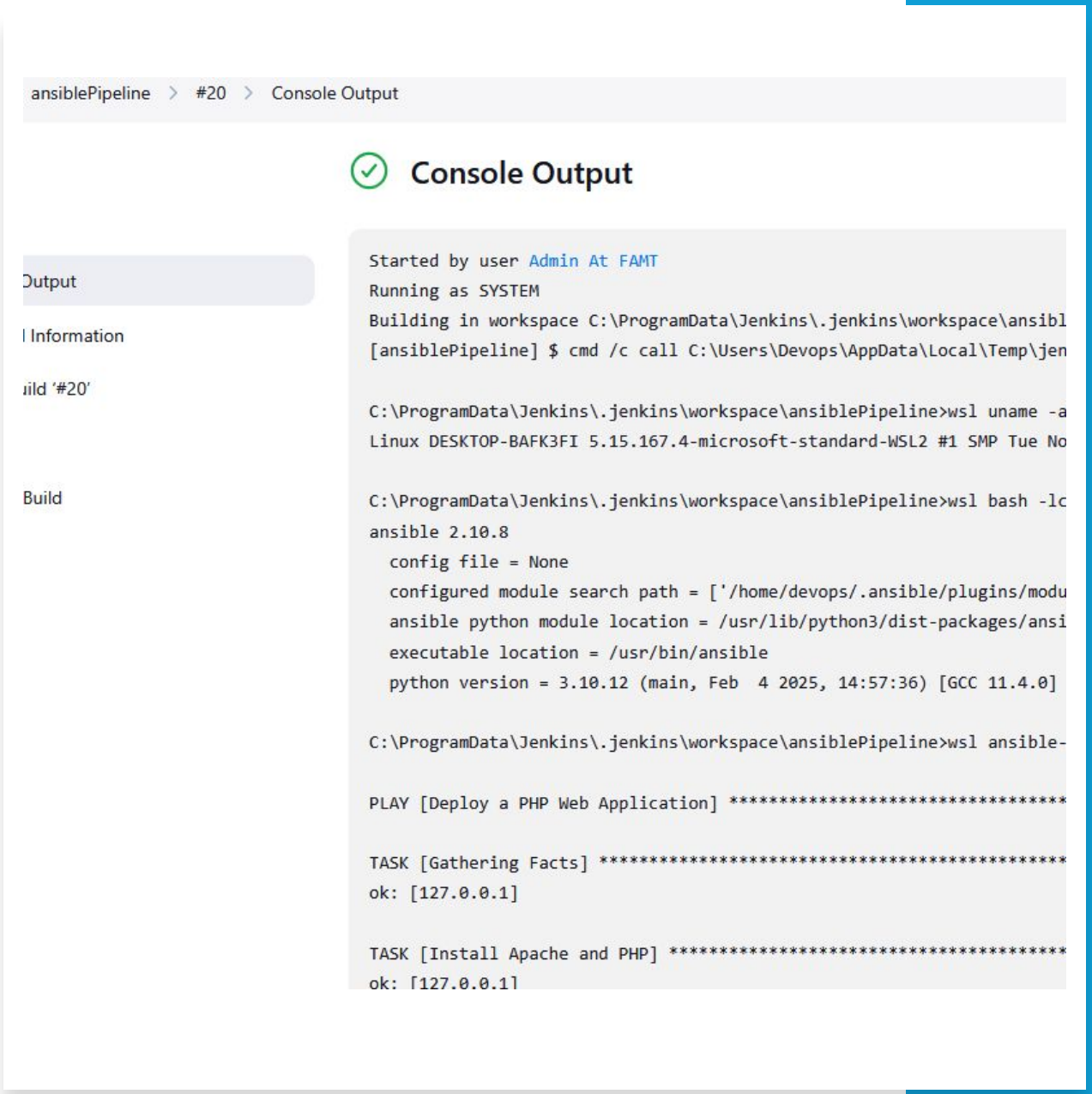
Jenkins Project for the Ansible Playbook



Jenkins Project for the Ansible Playbook



Jenkins Project for the Ansible Playbook



What is Push/Pull Models

- Ansible can operate in two different models when applying configurations to managed nodes:
  - Push models and
  - Pull models.

Push Model (Default in Ansible)

- In the push model, the **Ansible control node** (where Ansible is installed) **directly connects to the managed nodes** (remote servers) over SSH and pushes configurations to them.
- Example: Running an Ansible Playbook -

```
ansible-playbook -i inventory configplaybook.yml
```

Push Model (Default in Ansible)

- How it Works -

1. The control node runs an Ansible Playbook or ad-hoc command.
2. The control node connects to managed nodes using SSH.
3. The required configuration or command is pushed to the target machine.
4. The target machine executes the tasks and returns the result to the control node.

Push Model (Default in Ansible)

- Advantages of Push Model

1. Easier to set up (No agent required on remote machines).
2. Instant control (Runs immediately when executed).
3. Better for ad-hoc commands.

- Disadvantages of Push Model

1. Requires SSH access to all target machines.
2. Not efficient for large-scale deployments.

Pull Model (Ansible-Pull)

- In the pull model, each **managed node pulls its configuration from a central repository** (like Git).
- This is useful for large environments where SSH access from the control node is limited.
- Example -

ansible-pull -U https://github.com/myrepo/ansible-config.git -i localhost, myplaybook.yml

Pull Model (Ansible-Pull)

- In the pull model, each **managed node pulls its configuration from a central repository** (like Git).
- This is useful for large environments where SSH access from the control node is limited.
- Example -

ansible-pull -U https://github.com/myrepo/ansible-config.git -i localhost, myplaybook.yml

Pull Model (Ansible-Pull)

- How it Works:
  - Each managed node periodically pulls the latest configurations from a repository (Git or other sources).
  - The node executes the playbook locally.
  - The playbook applies the necessary changes.

Pull Model (Ansible-Pull)

- Advantages of Pull Model
  - No need for SSH access from the control node
  - Scales better (Each node runs its own update process)
  - Good for distributed environments
- Disadvantages
  - Requires Git or other repository access
  - Not instant (Runs based on a schedule, e.g., cron job)

Pull Model (Ansible-Pull)

- To invert the architecture of Ansible, i.e. a managed nodes check in to a central location, instead of pushing configuration out to them.
- The ansible-pull is a small script that will -
  - checkout a repo of configuration instructions from git and
  - then run ansible-playbook against that content.

Pull Model (Ansible-Pull)

- Assuming you load balance your checkout location, ansible-pull scales essentially infinitely.
- To run pull model use following command -

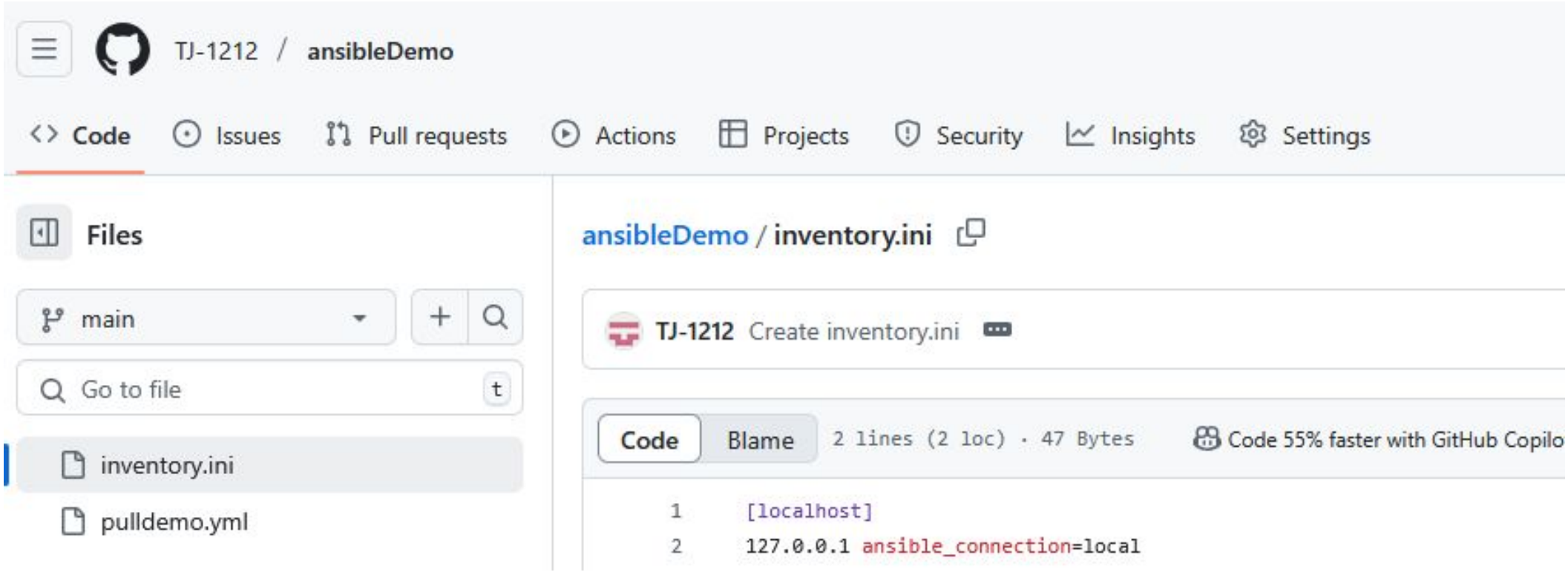
ansible-pull -d <directory name for cloning the repo>
-U <URL of the playbook repository> -i <inventory file name>
<name of playbook to run>

Pull Model Demo

- Login to the github.com and create a repo for ansible pull demo.
- Inside this repo add two files -
  - inventory.ini
  - pulldemo.yml

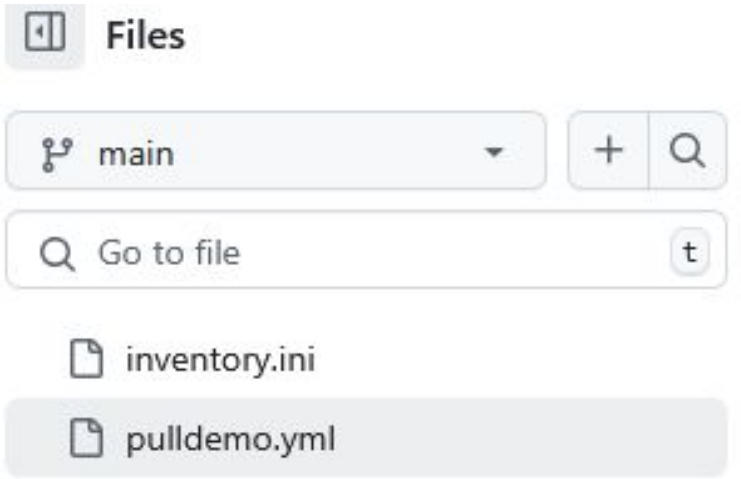
Pull Model Demo

- Contents of inventory.ini -



Pull Model Demo

- Contents of pulldemo.yml



ansibleDemo / pulldemo.yml

TJ-1212 Create pulldemo.yml

Code | Blame | 13 lines (13 loc) · 324 Bytes

```
1   ---
2   - name: First Pull demo Ansible playbook
3     hosts: localhost
4     tasks:
5         - name: print hello world
6           debug:
7               msg: "Hello World!....."
8         - name: print host name
9           debug:
10              var: ansible_hostname
11        - name: print Ubuntu version
12          debug:
13              var: ansible_distribution
```

Pull Model Demo

- In wsl, create a direcory (using mkdir) to download the repo and use the given command.

```
devops@DESKTOP-BAFK3FI:/home/myansibleprj$ ansible-pull -d /home/myansibleprj/ansiblepulldemo -U https://github.com/TJ-1212/ansibleDemo.git -i /home/myansibleprj/ansiblepulldemo/inventory.ini pulldemo.yml
Starting Ansible Pull at 2025-03-20 11:25:09
/usr/bin/ansible-pull -d /home/myansibleprj/ansiblepulldemo -U https://github.com/TJ-1212/ansibleDemo.git -i /home/myansibleprj/ansiblepulldemo/inventory.ini pulldemo.yml
[WARNING]: Could not match supplied host pattern, ignoring: DESKTOP-BAFK3FI
[WARNING]: Could not match supplied host pattern, ignoring: DESKTOP-BAFK3FI.
127.0.0.1 | SUCCESS => {
    "after": "d2df46ed79435f5e4a0d849a6813c52a2635fe8d",
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "before": "d2df46ed79435f5e4a0d849a6813c52a2635fe8d",
    "changed": false,
    "remote_url_changed": false
}
[WARNING]: Could not match supplied host pattern, ignoring: DESKTOP-BAFK3FI
[WARNING]: Could not match supplied host pattern, ignoring: DESKTOP-BAFK3FI.

PLAY [First Pull demo Ansible playbook] ***************************************

TASK [Gathering Facts] ********************************************************
ok: [127.0.0.1]

TASK [print hello world] ******************************************************
ok: [127.0.0.1] => {
```