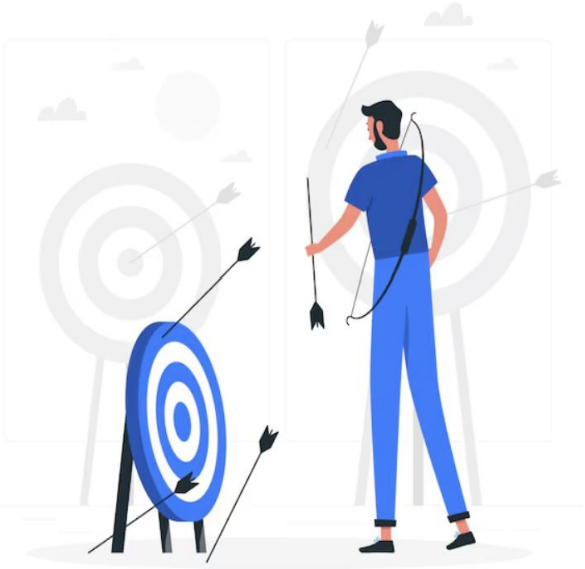
PW SKILLS | DevOps and Cloud Computing

Getting Started with Ansible

Objective

- Understand Ansible’s agentless architecture, push-based execution model, and YAML-based syntax.
- Differentiate between declarative and imperative configuration models.
- Install Ansible on various operating systems.
- Learn Ansible’s core components: control node, managed nodes, inventory, modules, tasks, and playbooks.
- Use SSH-based communication and run ad-hoc commands to manage remote systems.



PW SKILLS | DevOps and Cloud Computing

Explaining Ansible's agentless model (uses SSH instead of installing agents on managed nodes)

Let’s see

Ansible's agentless model is a key feature that distinguishes it from other configuration management tools. Here's how it works:

Key Components of Ansible's Agentless Model:

- No Agent Installation Required
- Use of Existing Protocols
- Push-Based Model

How It Works:

- Control Node
- Managed Nodes
- Modules

Discuss its push-based execution (configuration is pushed from the control node to managed nodes)

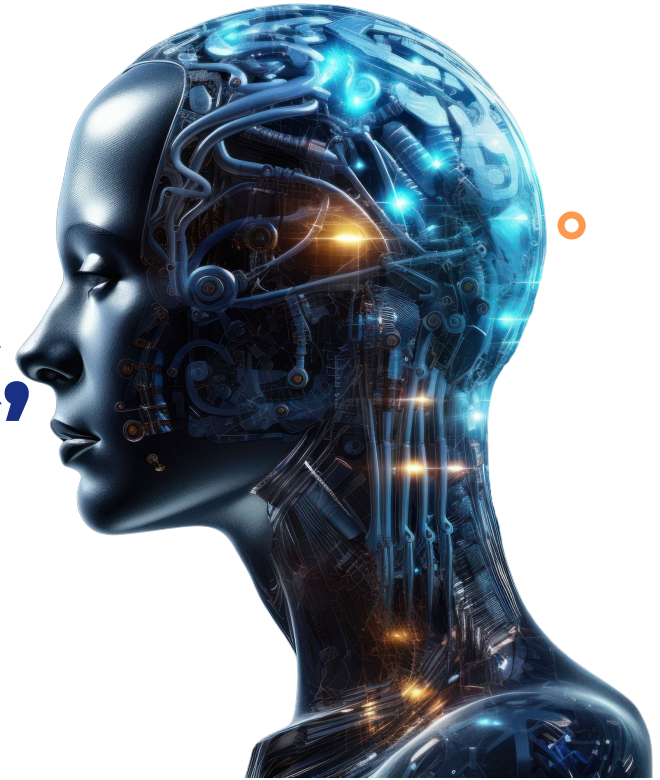
Let’s discuss

Ansible's push-based execution involves a control node that pushes configurations or tasks directly to managed nodes. This model allows for real-time updates and easy monitoring of changes across the infrastructure. Here's a brief overview:

- Control Node
- Managed Nodes
- Push-Based Model
- Advantages: Simplifies deployment, reduces maintenance overhead, and provides immediate updates across the infrastructure.

PW SKILLS | DevOps and Cloud Computing

Highlighting its YAML-based declarative syntax, making it easy to read and manage



YAML-based declarative syntax

Ansible uses a YAML-based declarative syntax that makes playbooks easy to read and manage. Here are the key highlights:

- Declarative Syntax: Ansible playbooks describe the desired state of the system, not how to achieve it. This makes them straightforward to understand and maintain.
- YAML Format: Playbooks are written in YAML, which is simple and human-readable. YAML uses indentation to denote structure, making it easy to visualize the configuration.

Key features:

- Lists and Dictionaries
- Variables and Templates
- Consistent Indentation

PW SKILLS | DevOps and Cloud Computing

Comparing Ansible with other CM tools (Puppet, Chef) in terms of architecture

Let’s compare

Here's a comparison of Ansible with Puppet and Chef in terms of architecture:

Ansible:

- Agentless Architecture: Uses SSH or WinRM to connect to nodes without installing any agents.
- Decentralized Control: No central server required; control nodes can be any machine with Ansible installed.
- Scalability: Highly scalable due to its lightweight nature.

Let’s compare

Puppet:

- Master-Agent Architecture: Requires a central Puppet Master server and agents on managed nodes.
- Multi-Master Support: Allows multiple master servers for redundancy and scalability.
- Centralized Control: Configurations are stored centrally and pulled by agents.

Chef:

- Master-Agent Architecture: Similar to Puppet, with a central Chef Server and agents on nodes.
- Workstation Component: Additional workstation layer for testing configurations before deployment.
- Centralized Control: Configurations are stored centrally and managed through the Chef Server.

PW SKILLS | DevOps and Cloud Computing

Explaining the imperative model (step-by-step execution, e.g., Bash scripts)

Imperative model

The imperative model involves a programming paradigm where tasks are executed through step-by-step instructions. Here's a brief overview:

- Step-by-Step Instructions: Imperative programming requires explicit commands that dictate how a task should be performed. This is similar to following a recipe where each step is detailed and must be executed in sequence.

- Control Flow: The control flow is explicit, meaning the programmer specifies the exact order in which instructions should be executed. This provides direct control over the program's execution but can make the code more complex and harder to maintain.

- Examples: Bash scripts are a common example of imperative programming. They consist of a series of commands that are executed one after another to achieve a specific outcome.

SKILLS | DevOps and Cloud Computing

Explaining the declarative model (describe the desired end state, e.g., Ansible playbooks)

Declarative model

The declarative model is a programming paradigm that focuses on specifying the desired end state or outcome of a computation without detailing the exact steps required to achieve it.

Key Features of Declarative Programming:

1. Focus on Desired Outcome:

- In declarative programming, the focus is on defining what the system should look like after the execution, rather than how to achieve that state.

2. Examples and Applications:

- Ansible playbooks are a prime example of declarative programming in configuration management. They describe the desired state of the system, and Ansible determines the necessary steps to achieve that state.
- Other examples include SQL for database queries and HTML for web page layout.

PW SKILLS | DevOps and Cloud Computing

Real-world examples comparing imperative scripts vs. Ansible playbooks

Real world examples

Imperative Script Example (Bash):

- Purpose: Execute a series of commands on multiple servers to check system status.
- Approach: Write a Bash script that loops through a list of servers, executes commands like ‘free –h’ for memory usage, and ‘uptime’ for CPU load.

- Example Code:

```
#!/bin/bash
for server in $(cat servers.txt); do
  ssh $server "free -h; uptime"
done
```

Real world examples

Ansible Playbook Example:

- Purpose: Configure multiple web servers with Nginx.
- Approach: Write an Ansible playbook that defines the desired state (Nginx installed and configured) and executes it on a group of servers.

- Example Code:

```
---
- name: Install and Configure Nginx
  hosts: webservers
  become: yes
  tasks:
    - name: Install Nginx
      apt:
        name: nginx
        state: present
    - name: Start Nginx
      service:
        name: nginx
        state: started
```

PW SKILLS | DevOps and Cloud Computing

Demonstrating installation on Linux (Ubuntu, CentOS)

Let’s do it

Here's a concise guide to installing Ansible on Ubuntu and CentOS:

Installing Ansible on Ubuntu:

1. Update the System:

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

2. Install Ansible:

First, add the Ansible PPA repository:

```
sudo add-apt-repository --yes --update ppa:ansible/ansible
```

Then, install Ansible:

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

3. Validate Installation:

```
ansible --version
```

Installing Ansible on CentOS:

1. Update the System:

```
sudo yum update -y
```

2. Install EPEL Repository:

```
sudo yum install epel-release
```

3. Install Ansible:

```
sudo yum install ansible
```

4. Validate Installation:

```
ansible --version
```

Pop Quiz

Q. How can you verify that Ansible has been installed successfully on Ubuntu?

A

Run the command: ansible --version

B

Restart the system and check for updates

Pop Quiz

Q. How can you verify that Ansible has been installed successfully on Ubuntu?

A

Run the command: ansible --version

B

Restart the system and check for updates

SKILLS | DevOps and Cloud Computing

Explaining installation via pip

Installation via pip

To install Ansible using pip, follow these steps:

1. Ensure Python and pip are installed on your system.
2. Open a terminal or command prompt.
3. Run the following command:

```
pip install ansible
```

4. Verify the installation by checking the Ansible version:

```
ansible --version
```

PW SKILLS | DevOps and Cloud Computing

Demonstrating setting up Ansible on macOS using Homebrew

Let’s do it

To set up Ansible on macOS using Homebrew, follow these steps:

1. Install Homebrew (if not already installed):
Run the following command in your terminal to install Homebrew:

```
 /bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. Update Homebrew:

```
brew update
```

Let’s do it

3. Install Ansible:

```
brew install ansible
```

4. Verify Installation:
Check the installed version of Ansible to confirm the installation:

```
ansible --version
```

PW SKILLS | DevOps and Cloud Computing

Providing Windows alternatives (using WSL or Ansible controllers like AWX)

Windows alternatives

1. Using Windows Subsystem for Linux (WSL)

- Install WSL and a Linux distribution (e.g., Ubuntu):

```
wsl --install
```

- Launch the Linux distribution and install Ansible:

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

- This method allows you to run Ansible natively in a Linux environment on Windows.

Windows alternatives

2. Using AWX (Ansible Controller):

- AWX provides a web-based interface for managing Ansible tasks.
- Install AWX using Docker, Kubernetes (e.g., Minikube), or directly on a Windows system with virtualization tools like VMware or Hyper-V36.
- AWX is ideal for teams needing a centralized, GUI-based Ansible control system.

Take A 5-Minute Break!

- Stretch and relax
- Hydrate
- Clear your mind
- Be back in 5 minutes

PW SKILLS | DevOps and Cloud Computing

Defining the Control Node: The machine running Ansible

Control Node

The Control Node in Ansible is the machine where Ansible is installed and from which all automation tasks are executed. It serves as the central point for running Ansible commands, playbooks, and managing configurations.

Key Features:

- Execution Point
- Supported Systems
- Agentless Management

PW SKILLS | DevOps and Cloud Computing

Defining Managed Nodes: Machines configured and managed via Ansible

Managed nodes

Managed Nodes, also referred to as "hosts," are the target devices that Ansible configures and manages. These can include:

- **Physical or Virtual Machines**
- **Containers**
- **Network Devices**
- **Applications or Software Instances**

PW SKILLS | DevOps and Cloud Computing

Explaining the Inventory File

Inventory file

The Inventory File in Ansible is a configuration file that defines the managed nodes (hosts) and organizes them into groups for automation tasks.

Key Points:

- Default Location: /etc/ansible/hosts is the default inventory file location, but custom files can be specified using the '-i' option or configured in Ansible settings.
- Grouping of Servers: Hosts can be grouped based on roles (e.g., [webservers], [dbservers], [load balancers]) to run tasks on specific sets of servers.
- Formats Supported: Inventory files can be written in formats like INI or YAML for flexibility.
- Dynamic Inventory: Ansible supports dynamic inventory plugins to fetch hosts from cloud providers or other sources.

Pop Quiz

Q. What is the default location of the Ansible inventory file??

A

/var/ansible/inventory

B

/etc/ansible/hosts

Pop Quiz

Q. What is the default location of the Ansible inventory file??

A

/var/ansible/inventory

B

/etc/ansible/hosts

PW SKILLS | DevOps and Cloud Computing

Introducing Ansible Modules (pre-built scripts to perform specific tasks, e.g., yum, apt, service)

Ansible modules

Ansible Modules are pre-built scripts that perform specific tasks on managed nodes. They are the building blocks of Ansible automation and can be executed directly from the command line or included in playbooks. Key points about Ansible Modules:

1. Purpose: Modules handle various IT functions like package management, service control, and file manipulation.
2. Language: Typically written in Python, but can be in other languages.
3. Example:
   - yum: Manage packages on RHEL-based systems
   - apt: Manage packages on Debian-based systems
   - service: Control system services
4. Idempotency: Modules are designed to be idempotent, meaning they can be run multiple times without changing the result beyond the initial application.

Pop Quiz

Q. Which module is used to manage services on remote hosts?

A

service

B

apt

Pop Quiz

Q. Which module is used to manage services on remote hosts?

A

service

B

apt

PW SKILLS | DevOps and Cloud Computing

Defining Tasks (single operations like installing a package)

Tasks

In Ansible, a Task is the smallest unit of automation that performs a single operation on a managed node. Tasks are defined within playbooks and are executed sequentially.

Key Features:

- Purpose: Each task performs one specific action, such as installing a package, modifying a file, or starting a service.
- Modules: Tasks use Ansible modules (e.g., yum, apt, service) to execute their actions.
- Structure: Tasks are written in YAML under the tasks section of a playbook and include parameters for the module being used.
- Example:

```
tasks:
  - name: Install Apache
    apt:
      name: apache2
      state: present
```

PW SKILLS | DevOps and Cloud Computing

Explain Playbooks (YAML files containing multiple tasks and roles)

Playbooks

Ansible Playbooks are YAML files that define automation tasks and configurations for managed nodes. They are the core component of Ansible automation.

Key Features:

- Structure: Playbooks consist of one or more plays, each containing tasks, modules, and roles.
- Plays: Map tasks to specific hosts or groups defined in the inventory file.
- Tasks: Execute specific operations, such as installing packages or managing files.
- Roles: Bundles of tasks and associated resources for reuse across playbooks.
- Execution: Run with the ansible-playbook command.

Playbooks

Example:

```
- name: Configure web servers
  hosts: webservers
  tasks:
    - name: Install Apache
      apt:
        name: apache2
        state: present
    - name: Start Apache service
      service:
        name: apache2
        state: started
```

Pop Quiz

Q. What is the primary format used for writing Ansible Playbooks?

A

YAML

B

JSON

Pop Quiz

Q. What is the primary format used for writing Ansible Playbooks?

A

YAML

B

JSON

PW SKILLS | DevOps and Cloud Computing

Demonstrating running a basic playbook to install a package on a remote node

Let’s do it

To demonstrate running a basic Ansible playbook to install a package on a remote node, follow these steps:

1. Create the Playbook
Write the following YAML content into a file, e.g., ‘install package.yml’:

```
---
- name: Install a package on remote nodes
  hosts: all
  become: yes
  tasks:
    - name: Install Apache
      apt:
        name: apache2
        state: present
```

This playbook:

- Targets all hosts in the inventory (hosts: all).
- Uses elevated privileges (become: yes).
- Installs the Apache package using the ‘apt’ module.

2. Run the Playbook

Execute the playbook with the following command:

```
ansible-playbook -i inventory_file_path install_package.yml
```

Replace inventory file path with the path to your inventory file.

3. Verify Installation

Check if Apache is installed and running on the remote node.

PW SKILLS | DevOps and Cloud Computing

Explaining how Ansible uses SSH to connect to remote hosts

Let’s see

Ansible uses SSH to connect to remote hosts for executing tasks and managing configurations. Here's how it works:

1. Default Connection Method: Ansible uses the native OpenSSH client by default for secure communication with remote nodes. If OpenSSH is unavailable, it can fall back to the ‘paramiko’ library.

2. Authentication:

- SSH Keys: Ansible prefers SSH key-based authentication for security and convenience. Public keys are installed on remote nodes, and private keys are stored on the control node.
- Password Authentication: It can also use passwords with the --ask-pass option, though this is less secure.

Let’s see

3. Configuration:
Define connection details (e.g., username, private key) in the inventory file:

```
[webservers]
host1 ansible_user=admin ansible_ssh_private_key_file=~/.ssh/id_rsa
```

- Alternatively, set global defaults in ansible.cfg or use --private-key during playbook execution.

4. Optimizations:
- Uses ControlPersist to maintain persistent SSH sessions, improving performance for repeated connections.

Pop Quiz

Q. Which command checks connectivity to all hosts in your inventory using SSH?

A

```
$ ansible -m ping all -vvvv
```

B

```
$ ansible -i inventory.yaml
--check-connection all
```

Pop Quiz

Q. Which command checks connectivity to all hosts in your inventory using SSH?

A

```
$ ansible -m ping all -vvvv
```

B

```
$ ansible -i inventory.yaml
--check-connection all
```

PW SKILLS | DevOps and Cloud Computing

Demonstrating setting up passwordless SSH authentication (ssh-keygen and ssh-copy-id)

Let’s do it

To set up passwordless SSH authentication, follow these steps:

1. Generate SSH Key Pair
On the local machine, generate a key pair:

```
ssh-keygen -t rsa -b 2048
```

- Press Enter to save the keys in the default location (~/.ssh/id rsa).
- Optionally, add a passphrase (or press Enter to skip).

Let’s do it

2. Copy the Public Key to the Remote Host
Use ssh-copy-id to transfer the public key to the remote host:

```
ssh-copy-id user@remote_host
```

- Replace ‘user’ with your username and remote host with the IP or hostname of the remote machine.
- Enter the password when prompted.

Let’s do it

3. Verify Passwordless Login
Test the connection:

```
ssh user@remote_host
```

If successful, you will log in without being prompted for a password.

PW SKILLS | DevOps and Cloud Computing

Introducing Ansible command structure

Ansible command structure

The basic structure of an Ansible command is as follows:

```
ansible <host_group> -m <module> -a "<arguments>"
```

Explanation:
1. <host group>: Specifies the target hosts or groups from the inventory file (e.g., all, webservers).
2. -m <module>: Indicates the module to use (e.g., ping, apt, command).
3. -a "<arguments>": Provides arguments for the module (e.g., package name, command to run).

Ansible command structure

Example:

- Ping all hosts:

```
ansible all -m ping
```

This checks connectivity to all hosts in the inventory.

- Run a command:

```
ansible all -m command -a "uname -a"
```

Executes uname -a on all hosts.

- This structure enables ad-hoc management of remote systems using Ansible.

PW SKILLS | DevOps and Cloud Computing

Demonstrating ad-hoc commands

Ad-hoc commands

Here's how to demonstrate running basic Ansible ad-hoc commands:

1. Check Connectivity
Use the 'ping' module to verify connectivity to all hosts:

```
ansible all -m ping
```

2. Retrieve System Information
Use the 'setup' module to gather facts about all hosts:

```
ansible all -m setup
```

Ad-hoc commands

3. Example Output
For ‘ping’, you should see a success message like:

```
host1 | SUCCESS => {
    "ping": "pong"
}
```

- For ‘setup’, detailed system information (facts) will be displayed.

These commands are quick, one-liner solutions for immediate tasks without requiring a playbook.

Time for case study!

Important

- **Complete the post-class assessment**
- **Complete assignments (if any)**
- **Practice the concepts and techniques taught in this session**
- **Review your lecture notes**
- **Note down questions and queries regarding this session and consult the teaching assistants**

Thanks !