Get the best out of Live Sessions

HOW?

e!

Check your Internet Connection

Log in 10 mins before, and check your internet connection to avoid any network issues during the LIVE session.

Speak with the Instructor

By default, you will be on mute to avoid any background noise. However, if required you will be **unmuted by instructor**.

Clear Your Doubts

Feel free to clear your doubts. Use the "**Questions**" tab on your webinar tool to interact with the instructor at any point during the class.

Let us know if you liked our content

Please share feedback after each class. It will help us to enhance your learning experience.

edureka!

edureka!

DevOps Certification Training

COURSE OUTLINE
MODULE 05

1. Introduction to DevOps
2. Version Control with Git
3. Git and Jenkins
4. Continuous Integration with Jenkins
5. Configuration Management using Ansible
6. Containerization using Docker Part - I
7. Containerization using Docker Part - II
8. Container Orchestration Using Kubernetes Part-I
9. Container Orchestration Using Kubernetes Part-II
10. Monitoring Using Prometheus and Grafana
11. Provisioning Infrastructure using Terraform Part - I
12. Provisioning Infrastructure using Terraform Part - II

edureka!

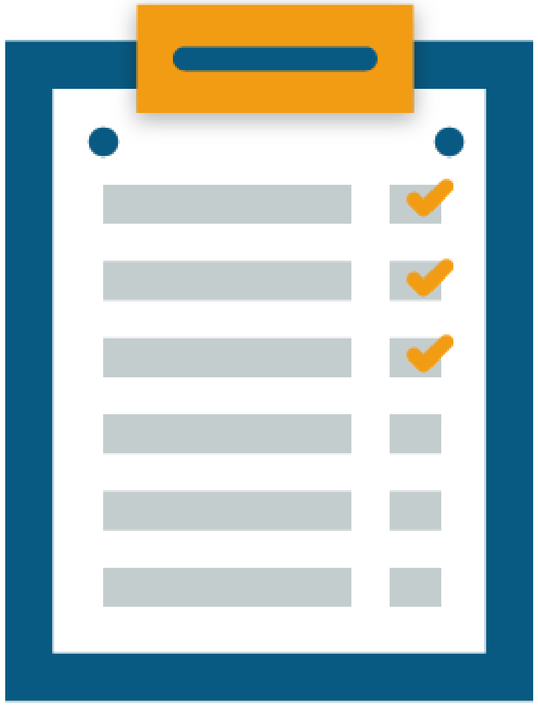
Module 5 – Configuration Management using Ansible

edureka!

Topics

Following are the topics covered in this module:

- Configuration Management
- Ansible Architecture
- Ansible CLI
- Ansible Ad-Hoc Commands
- Ansible Playbooks
- Playbooks Variables
- Playbook Handlers



Objectives

After completing this module, you should be able to:

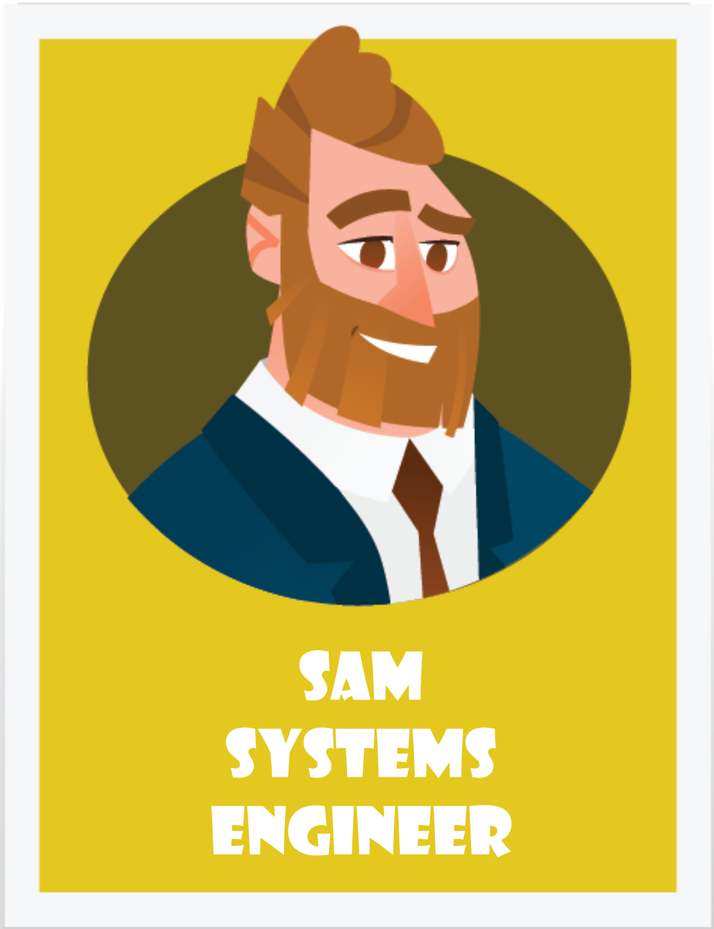
- Utilize Ansible CLI
- Execute Ansible Ad-Hoc Commands for one-off tasks
- Automate host servers using Ansible Playbooks
- Use Variables in Playbooks
- Using Handlers

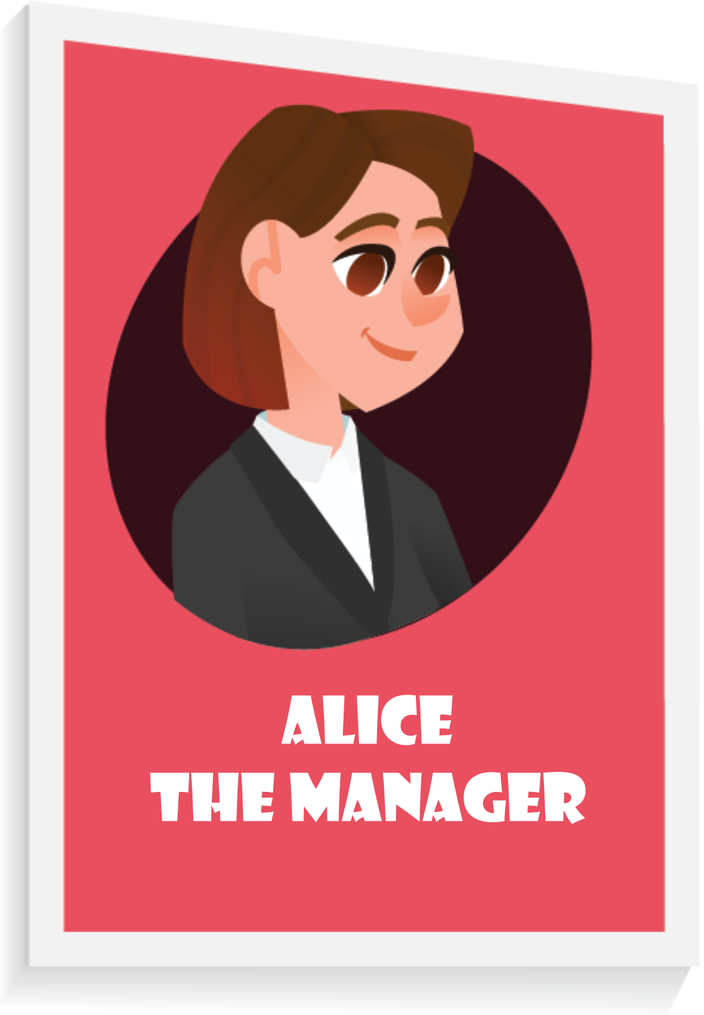
edureka!

Scenario of an IT Company

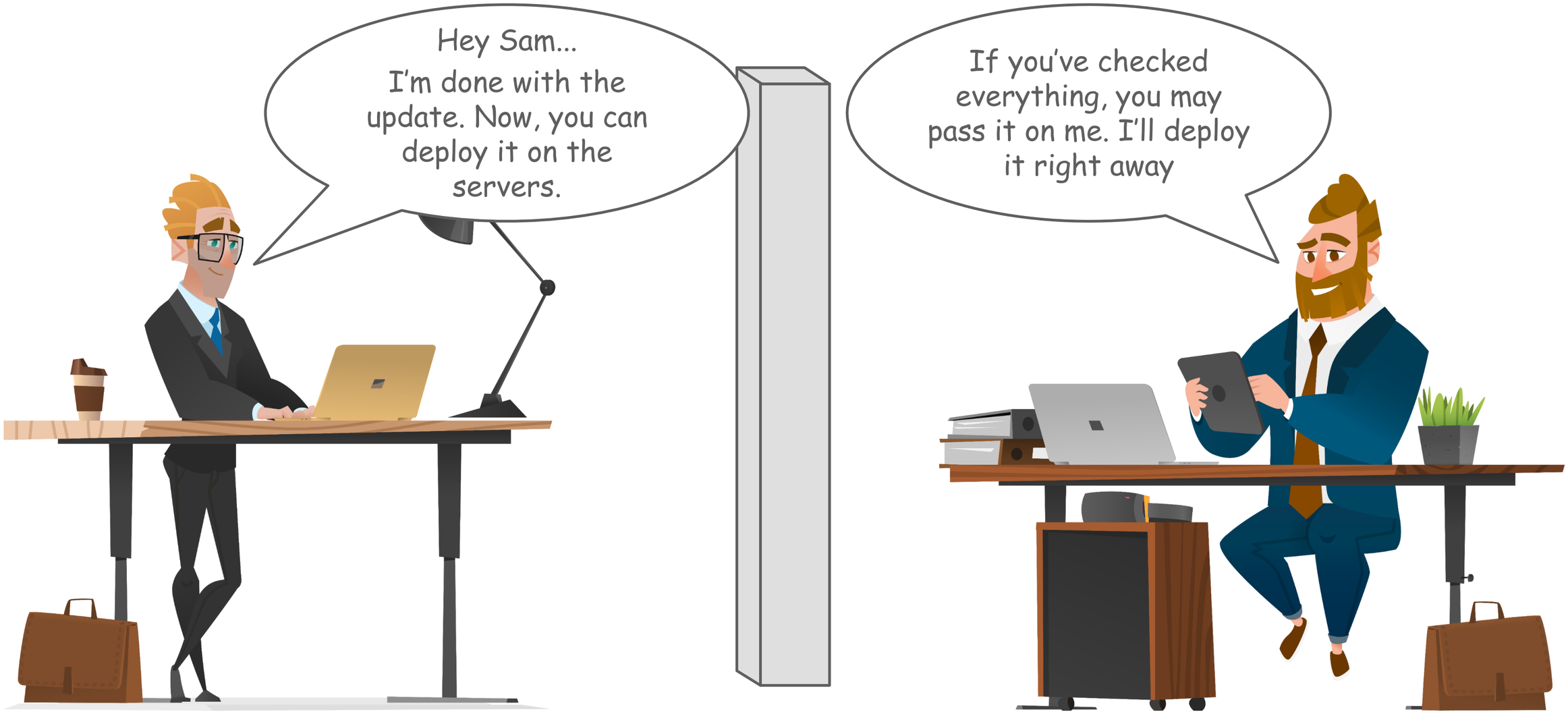
edureka!

Meet The Team





Dave Passes Update to Operations



edureka!

Sam Starts Deploying the Update on Each Server

Manager's New Request

Seems like we're getting a lot of traffic on our servers. I want you to add 200 more servers to our system.

But I haven't even updated the current 20 servers with Dave's changes. How am I going to do that for 200 more servers.

Issues Without a Configuration Management Tool

Configuring a large number of Servers together becomes a daunting task

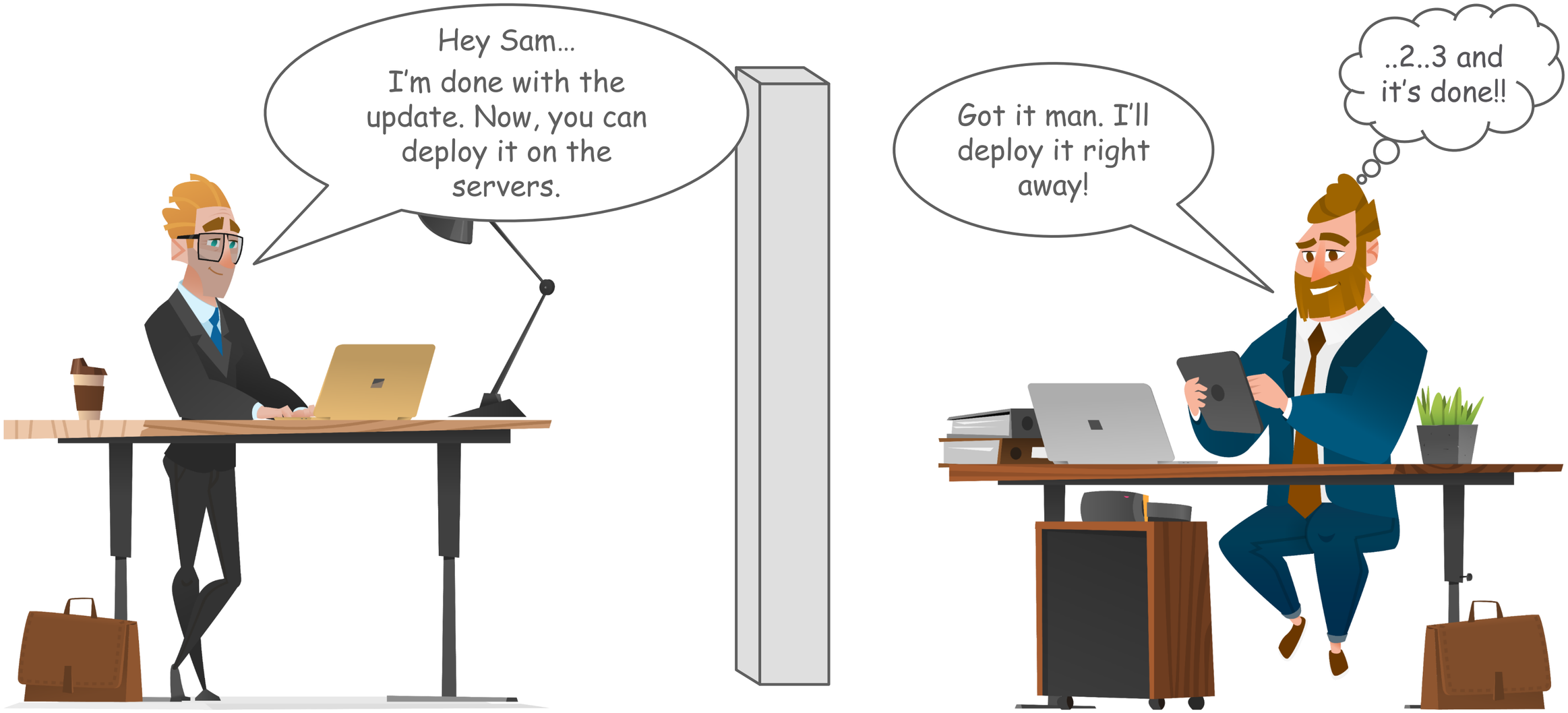
Scaling new servers without a Configuration Management tool is difficult

Development and **Deployment** environment mismatch halts work

With a Configuration Management Tool

edureka!

Using CM Tool: Dave Passes Update to Operations



edureka!

Using CM Tool: Scaling

Seems like we're getting a lot of traffic on our servers. I want you to add 200 more servers to our system.

No problem. With the help of this CM tool, I can easily scale, deploy and configure as many servers as you want.

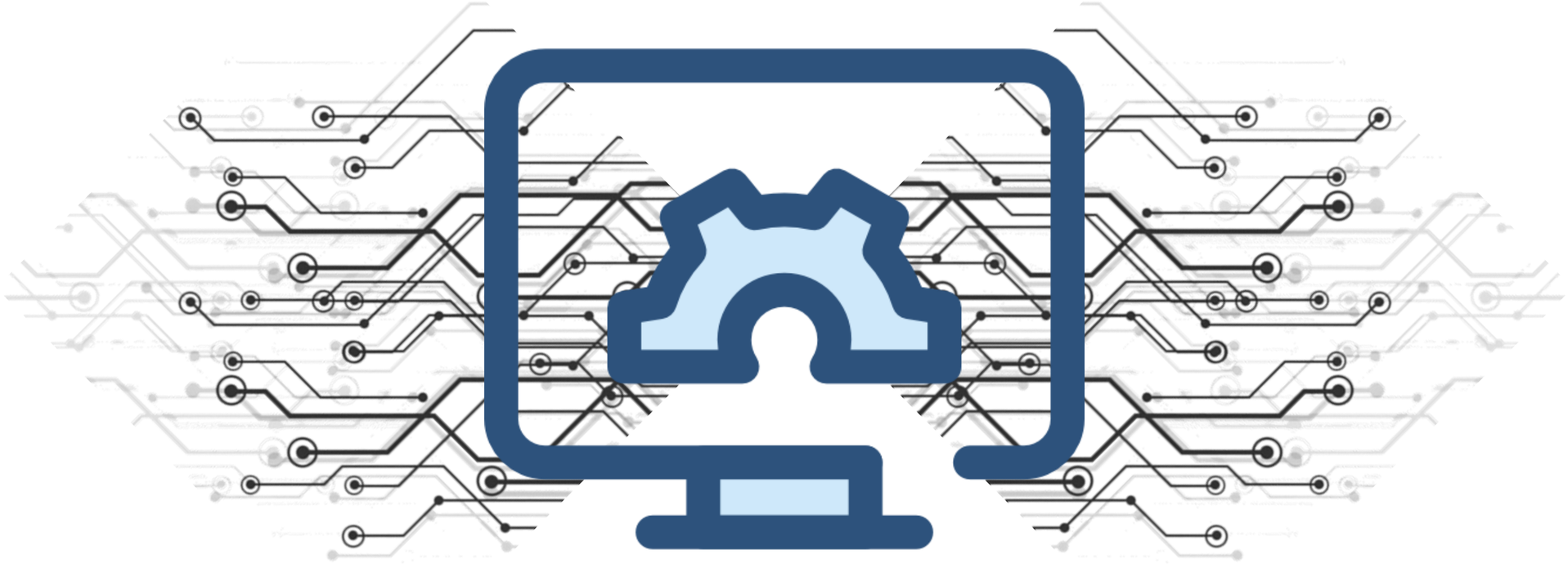
Configuration Management

edureka!

Configuration Management

Configuration Management is all about bringing in consistency in the infrastructure. This is done by ensuring that the current design system, state and environment is known, trusted and agreed upon by everyone.

Configuration Management also helps record all the changes made in the system.



Configuration Management

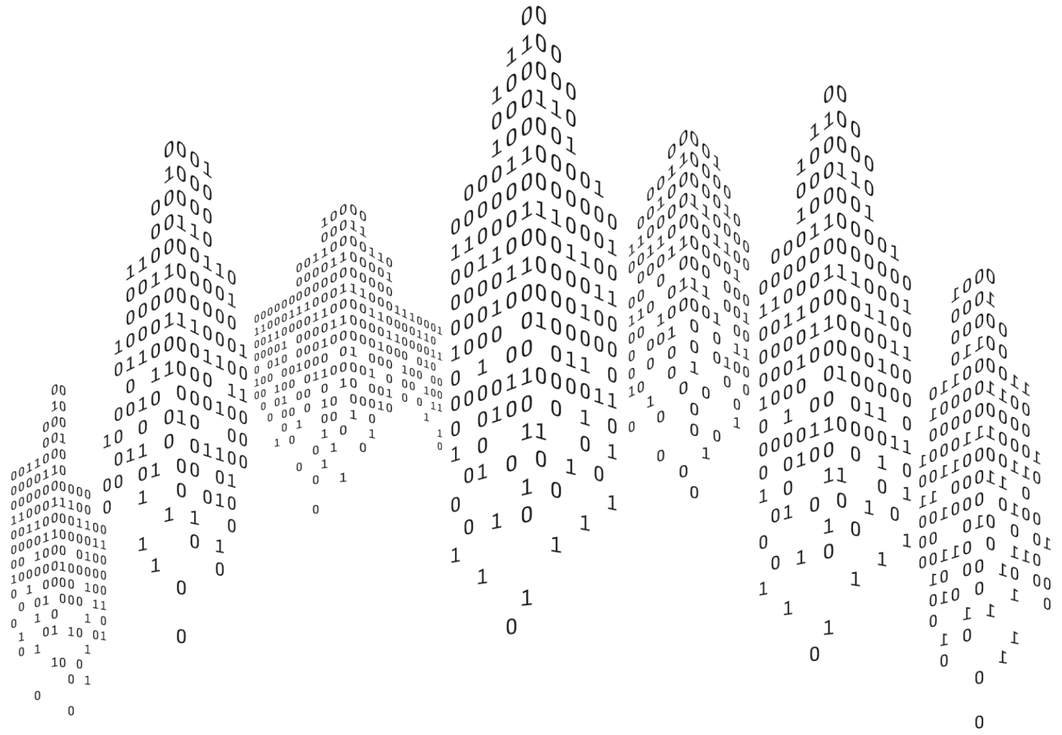
- Configuration Management Tools provide an easier approach to manage and configure servers
- Writing individual scripts for server's time and again just to get a few things done has become obsolete
- Configuration Management enables users to manage and configure the entire infrastructure and its environment

Infrastructure as Code

edureka!

Infrastructure as Code (IaC)

Infrastructure as Code enables automation of IT Operations(build, deploy, manage) by provisioning of code, rather than manually handling each phase for different environments. It bridges the environment differences system admins encountered every time they had to either deploy new code or setup new servers

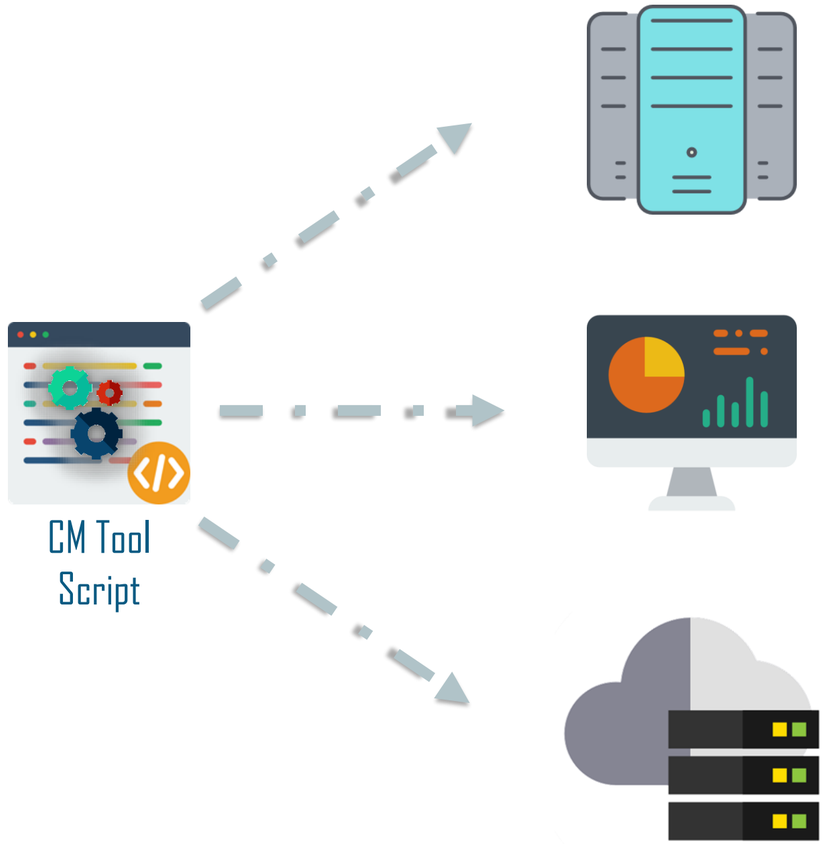


edureka!

Infrastructure as Code

- Provisioning servers using Infrastructure as Code is easier than writing shell scripts

- Shell scripts require workflow definitions whereas CM tool scripts have pre-defined workflows



Infrastructure as Code (IaC)

Below is a Shell Script and a Configuration Management Tool Script. Both are adding a user to a host, but while the CM tool script is **easy to understand** and **write**, the shell script is hard to understand, and you'll have to learn how to write Shell Scripts on you own

Shell Script

```
echo
"spock:*:1010:1010:Spock:/hom
e/spock:/bin/sh" \ >>
/etc/passwd
(the user spock is added to
passwd file)
```

CM Tool Script

```
user { "spock":
ensure => present,
gid => "science",
home => "/home/spock",
shell => "/bin/sh"
}
```

Schwarz Group: Automating Growth

edureka!

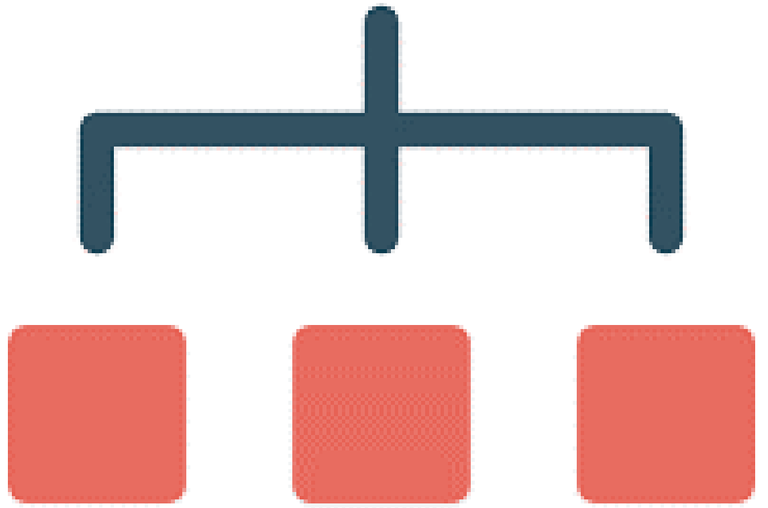
About Schwarz Group

- The Schwarz Group is the largest European retailer
- It employs more than 458,000 employees in its 12,500 stores across 33 countries
- It comprises of two retail divisions: Lidl and Kaufland

The Infrastructure Issue

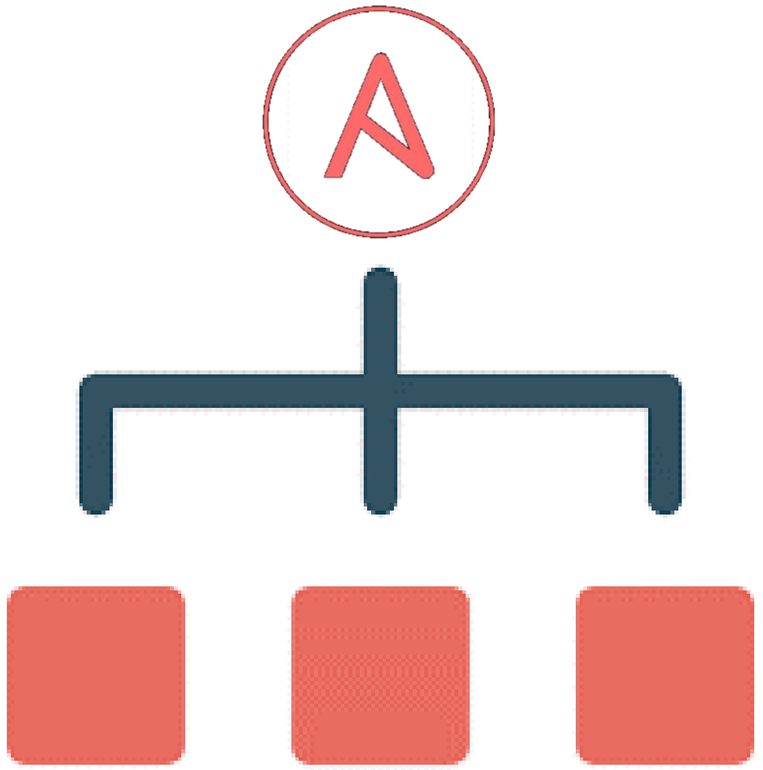
- Managing the ever-growing number of stores was becoming difficult
- Delivery times needed to be shorter to stay competitive in the market
- Flexibility required to adapt to the changing local markets

SCHWARZ GRUPPE



The Solution: Ansible

- Ansible enabled Schwarz Group to manage their stores more easily and reliably
- With the help of Ansible, more stable digital features are pushed to the stores at a higher frequency
- Enhanced risk management using the role-based system access



Introduction to Ansible

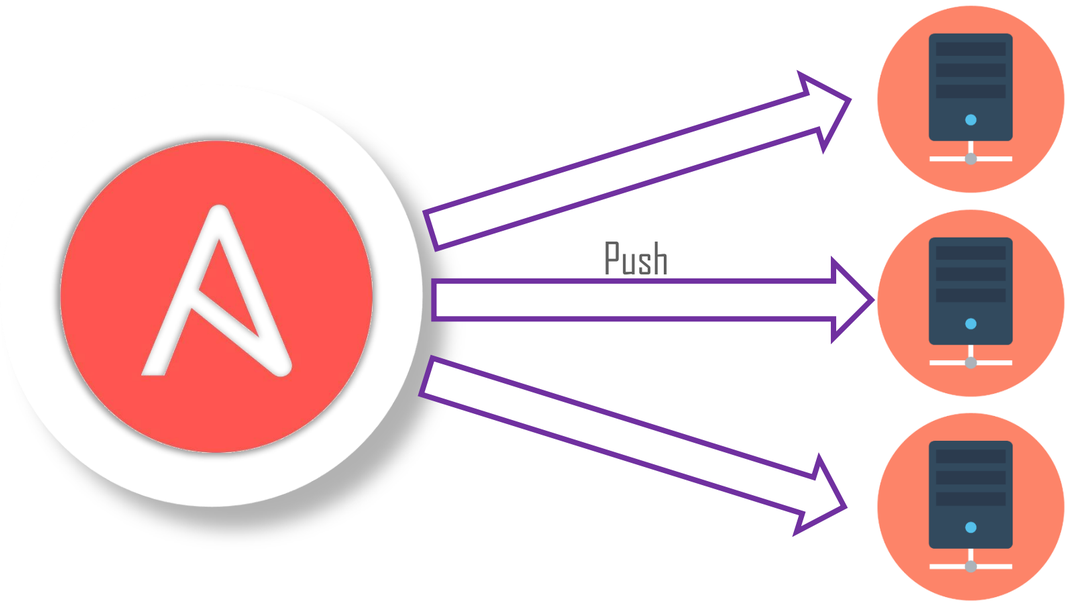
edureka!

What is Ansible?

Ansible is a deployment automation tool which traditionally uses push approach to achieve its objectives, by managing all the servers through one single machine running the Ansible Configuration Management Tool

What is Ansible?

Ansible clients do not have any agents installed on them, therefore there is no concept of polling with central server. Ansible uses the **Push** approach instead. Still, Ansible is flexible enough to let the user implement Pull architecture as well



Push vs Pull

PUSH	PULL
Push approach does not require agents set up on individual nodes like pull does	Pull works in a master slave architecture which requires agents set up on all slave nodes
Push based systems are completely Synchronous as you can see the changes made instantaneously and can fix the system if changes cause problems	Systems using pull architecture can scale quite easily which is not the case with push model

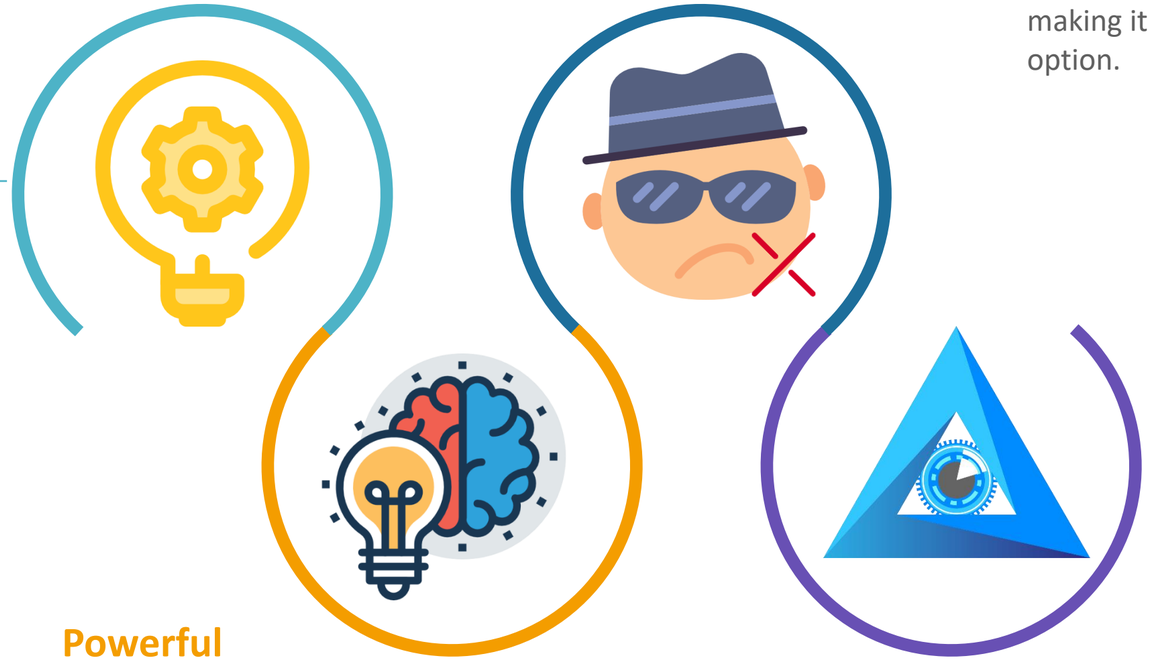
Why Ansible?

Agentless

Ansible uses OpenSSH and WinRM as transport to achieve automation. It is completely agentless, therefore making it a more efficient and secure option.

Simple

Ansible uses YAML(Yet Another Mark-up Language) which is very simple and readable. No special coding skills are required to write in YAML



Powerful

Ansible can be used to manage entire infrastructure of an application. It can orchestrate the entire application and environment lifecycle no matter where it is deployed.

Accessible

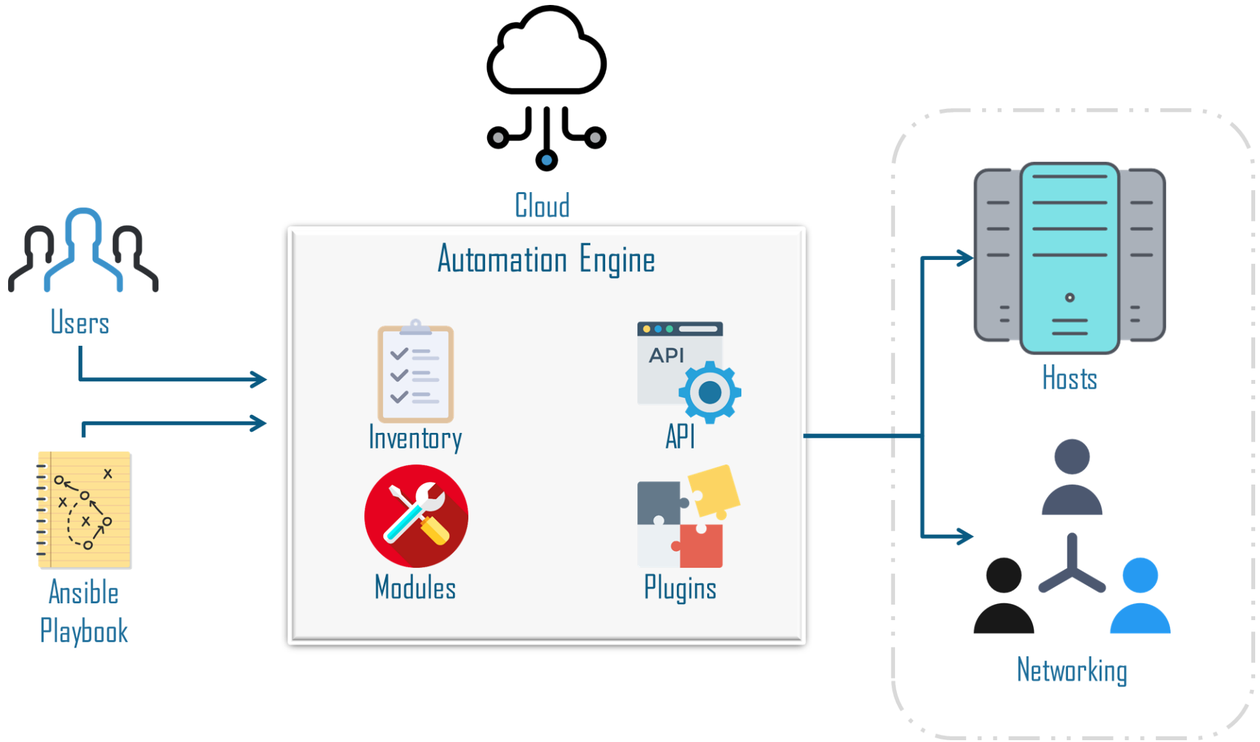
If someone automates something in Ansible, everyone on the team now knows how it is done.

Ansible Architecture

edureka!

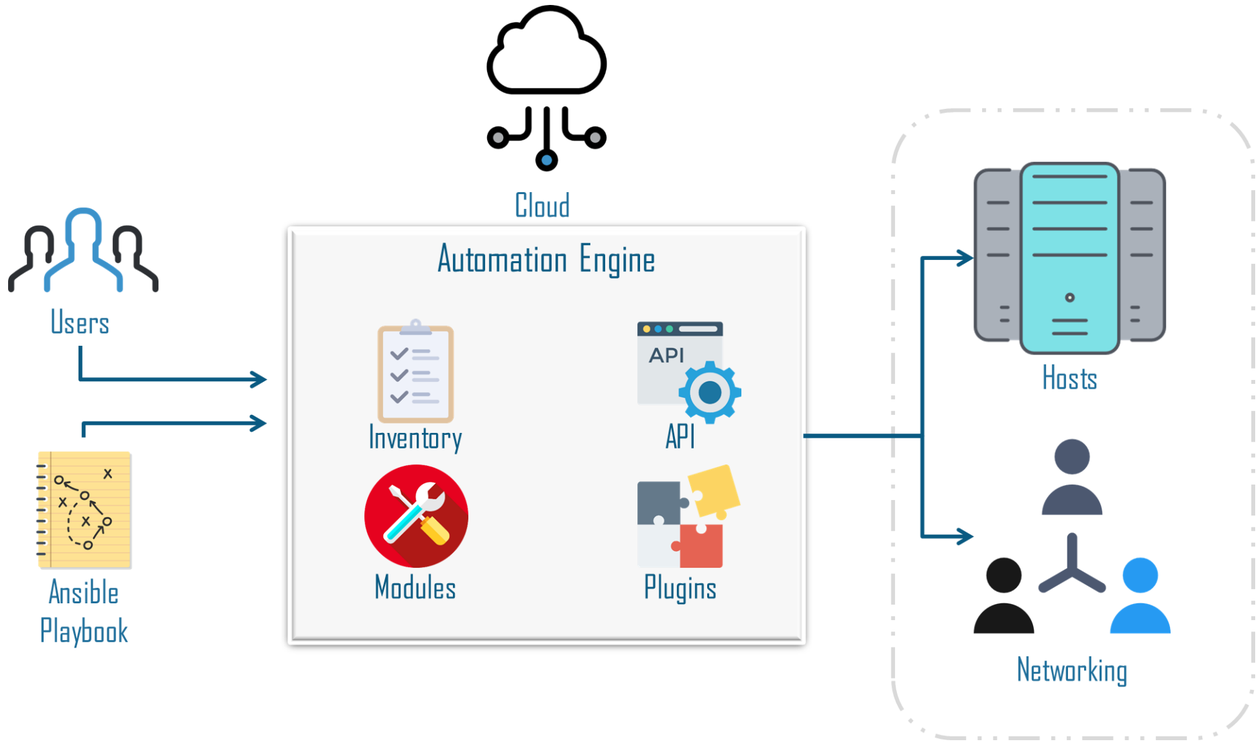
Ansible Architecture

Ansible connects to the nodes and pushes out small programs called '**Modules**'. These Modules bring the systems to the desired state. Ansible uses SSH to execute these modules and then removes them when finished



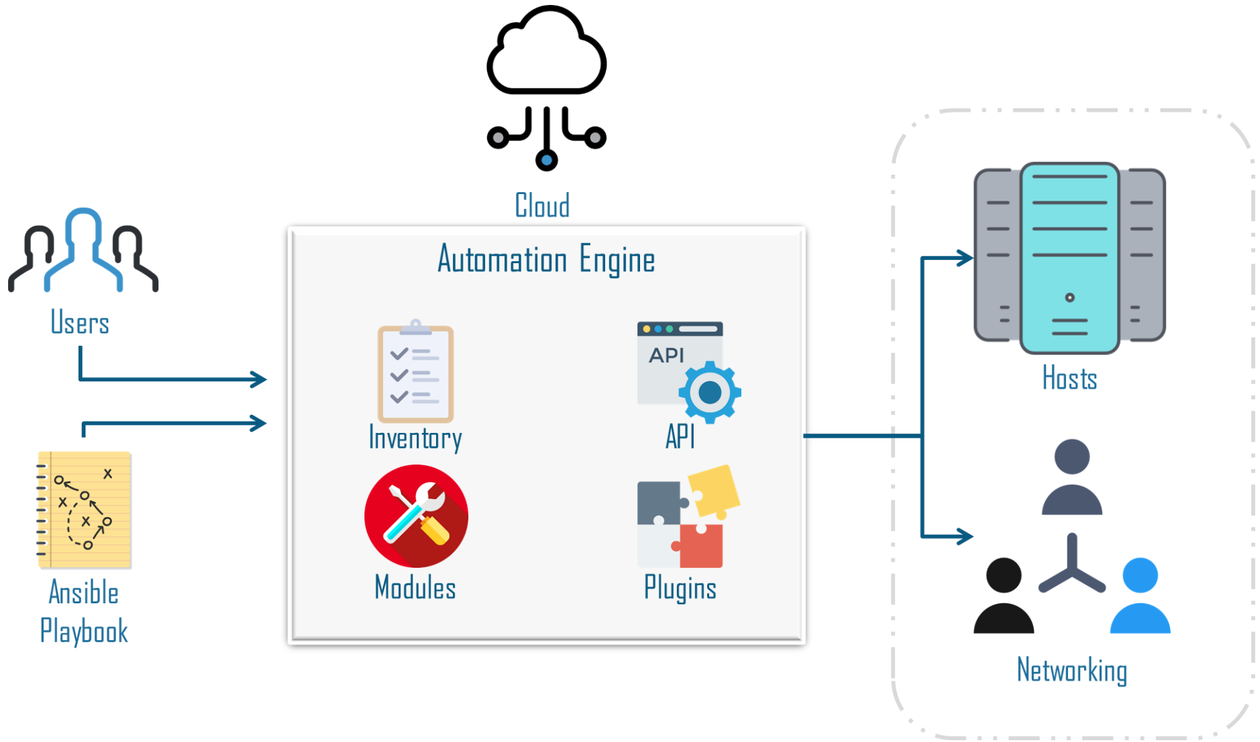
Ansible Architecture: Playbooks

- Ansible uses playbooks to implement the changes desired by the users
- Playbooks contain plays, plays have tasks and tasks call modules
- Modules are the units which execute on the servers



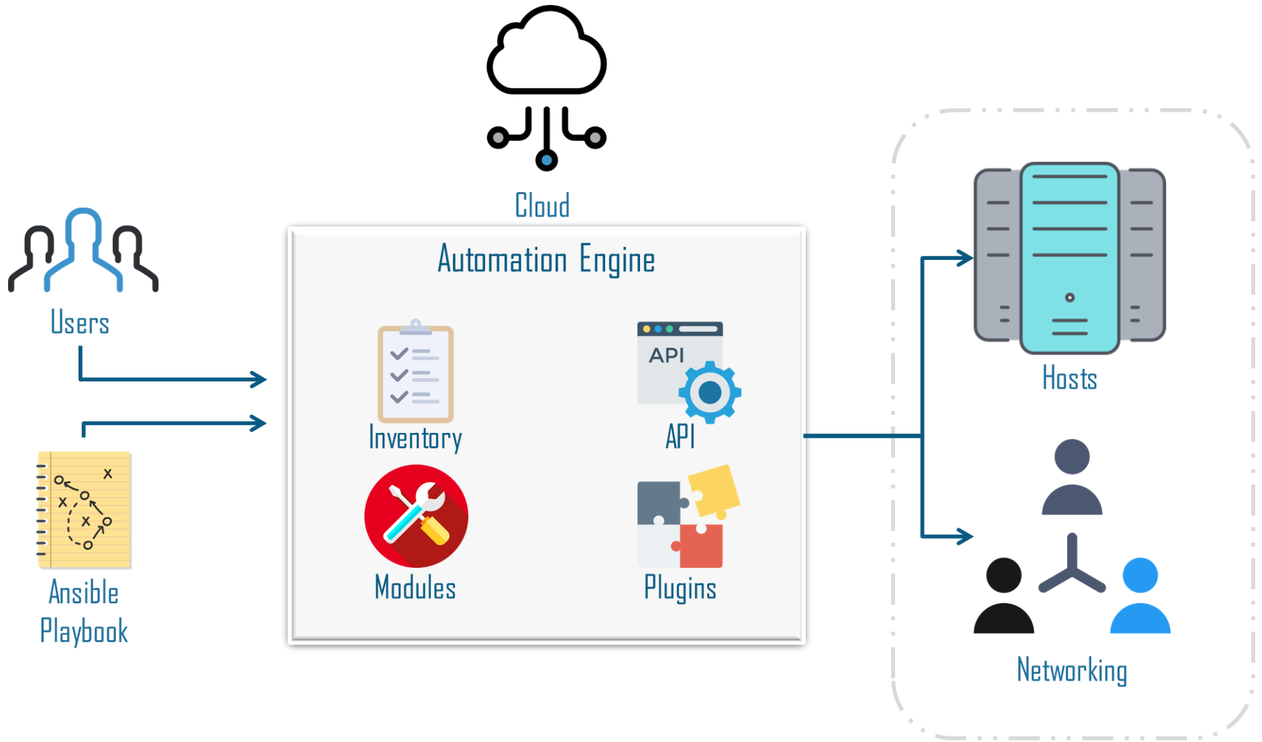
Ansible Architecture: Inventory

- Ansible uses the inventory file to represent all the machines it is managing
- Users can then group the machines to their own liking



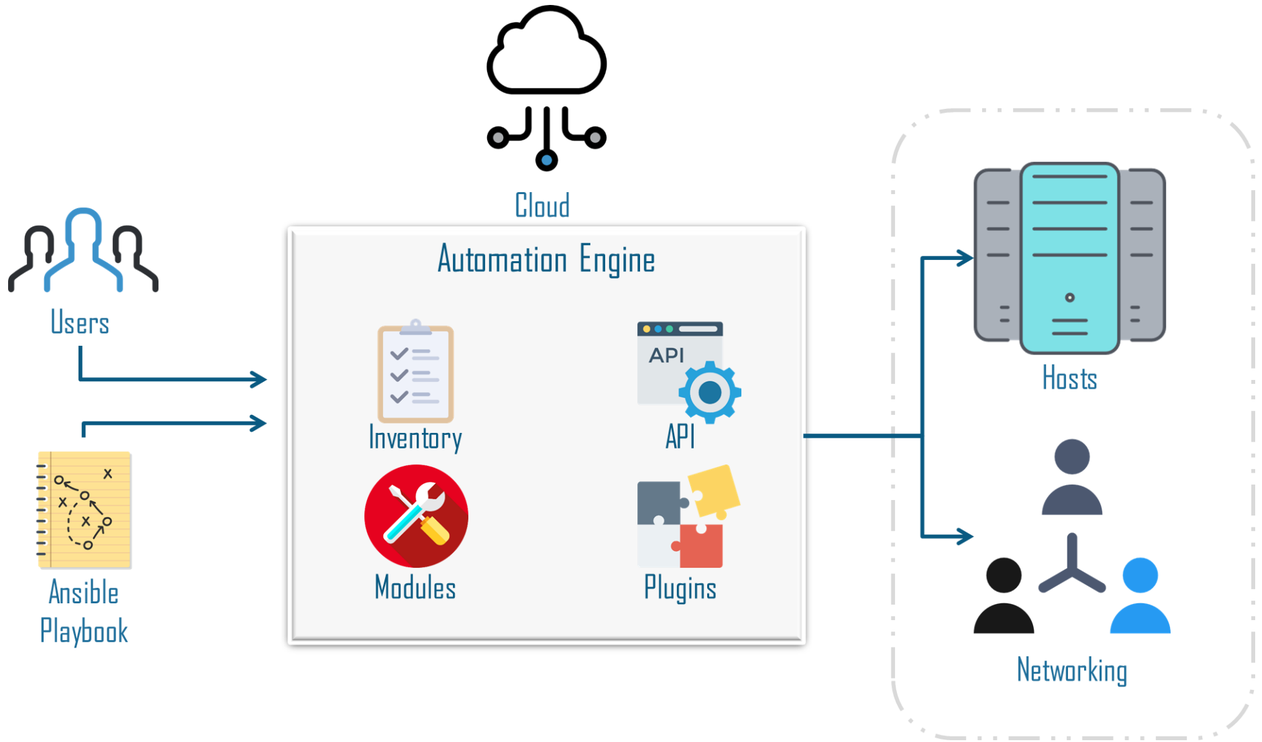
Ansible Architecture: API

- The API's in Ansible are there to support services like cloud and CLI
- Eg. Python API is used for Command Line Interface



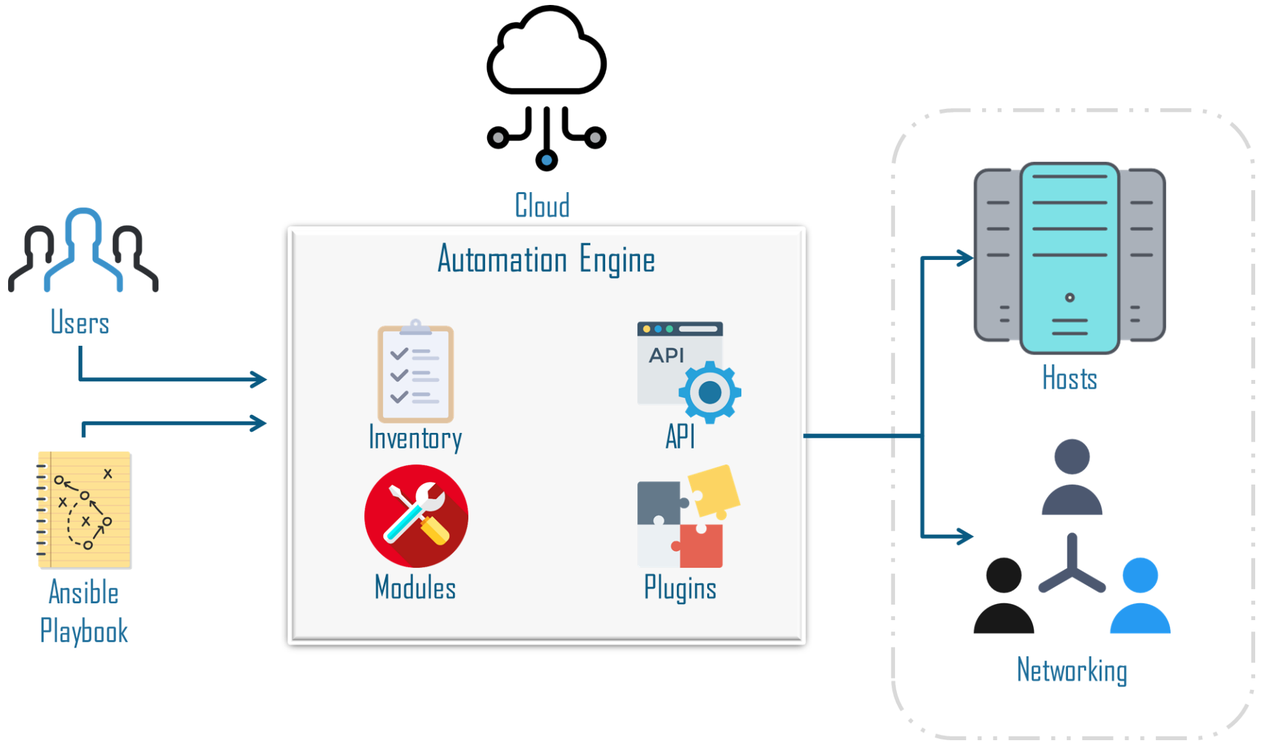
Ansible Architecture: Plugins

- Plugins act as extension to Ansible
- Eg. Action plugin lets you perform tasks on your Ansible machine before you execute a playbook



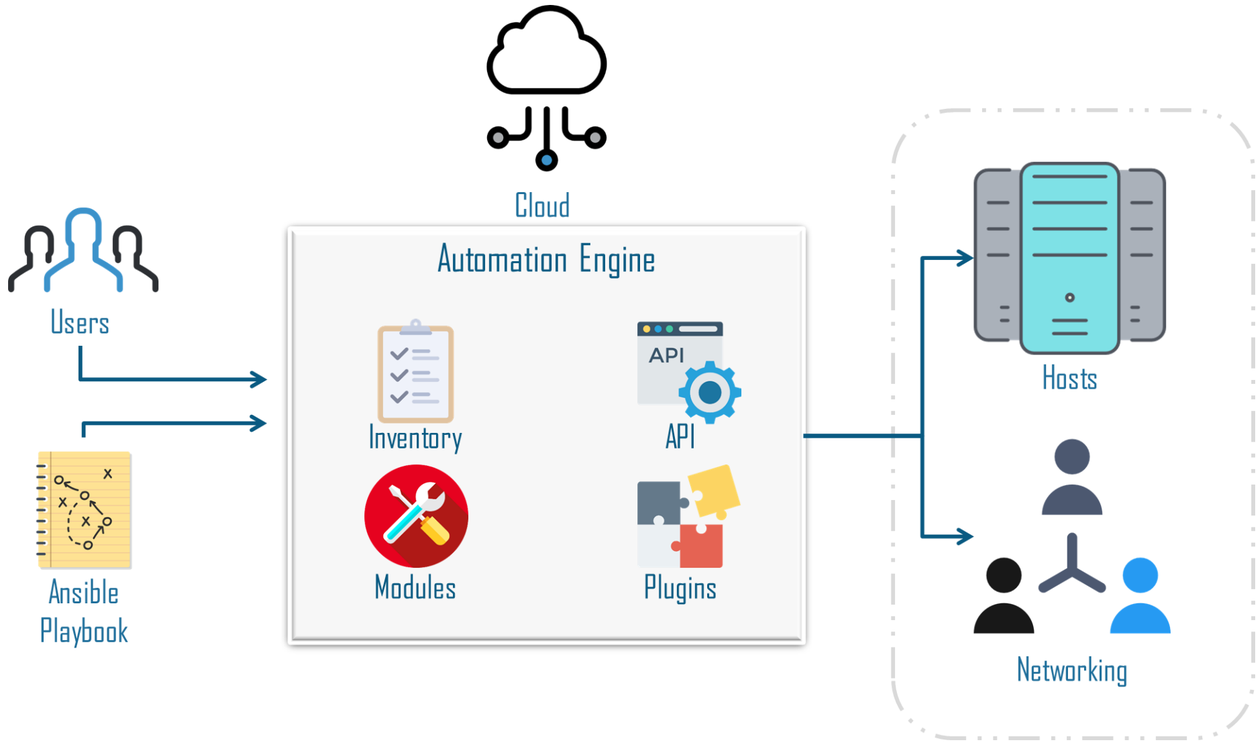
Ansible Architecture: Cloud

- Ansible has a huge support for numerous cloud applications
- These cloud applications can seamlessly be integrated with Ansible
- Ansible makes provisioning cloud infrastructures easy



Ansible Architecture: Hosts and Network

- The hosts are connected to the Ansible system via secure SSH connection
- Different networks can be managed together giving each network separate access rights

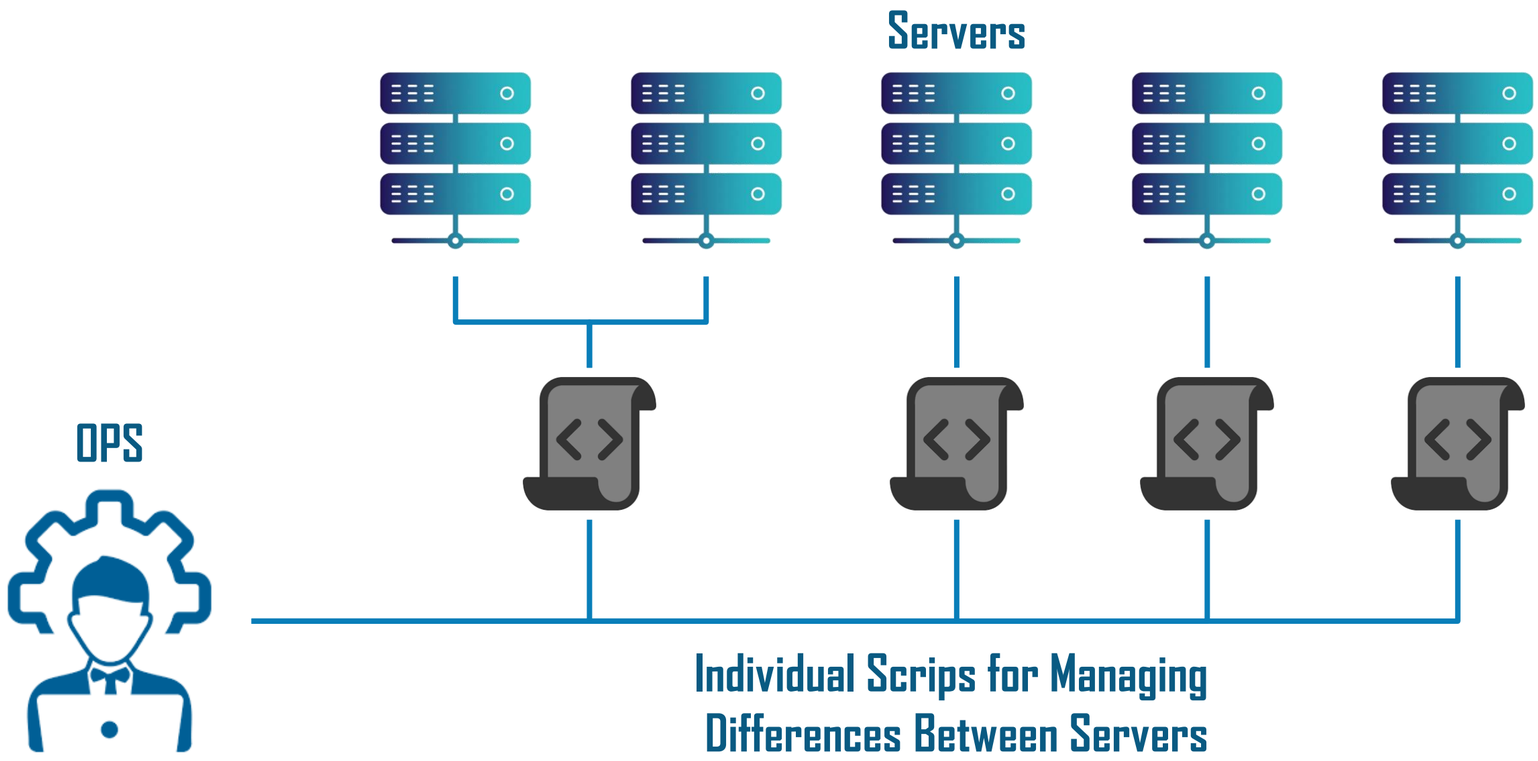


Automation Using Ansible

edureka!

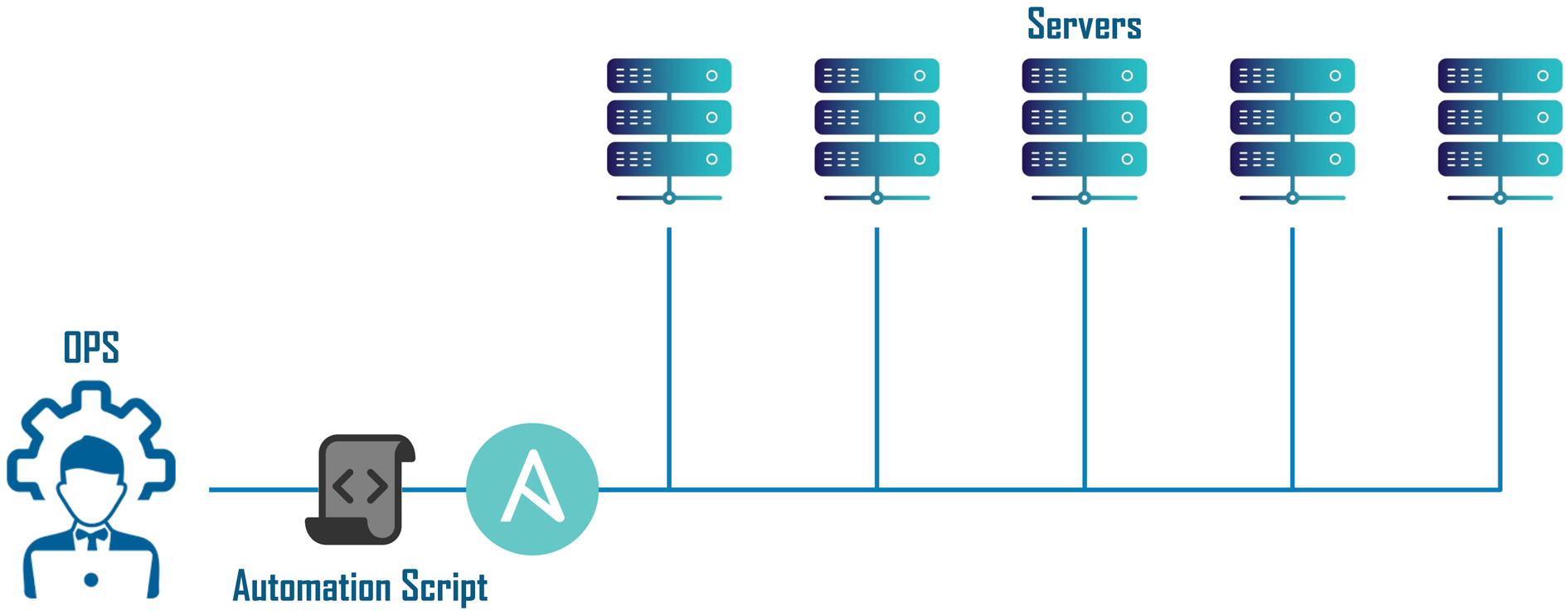
Centralizing Configuration Management

Configuration Management without an automation tool like Ansible requires a lot of manual labour



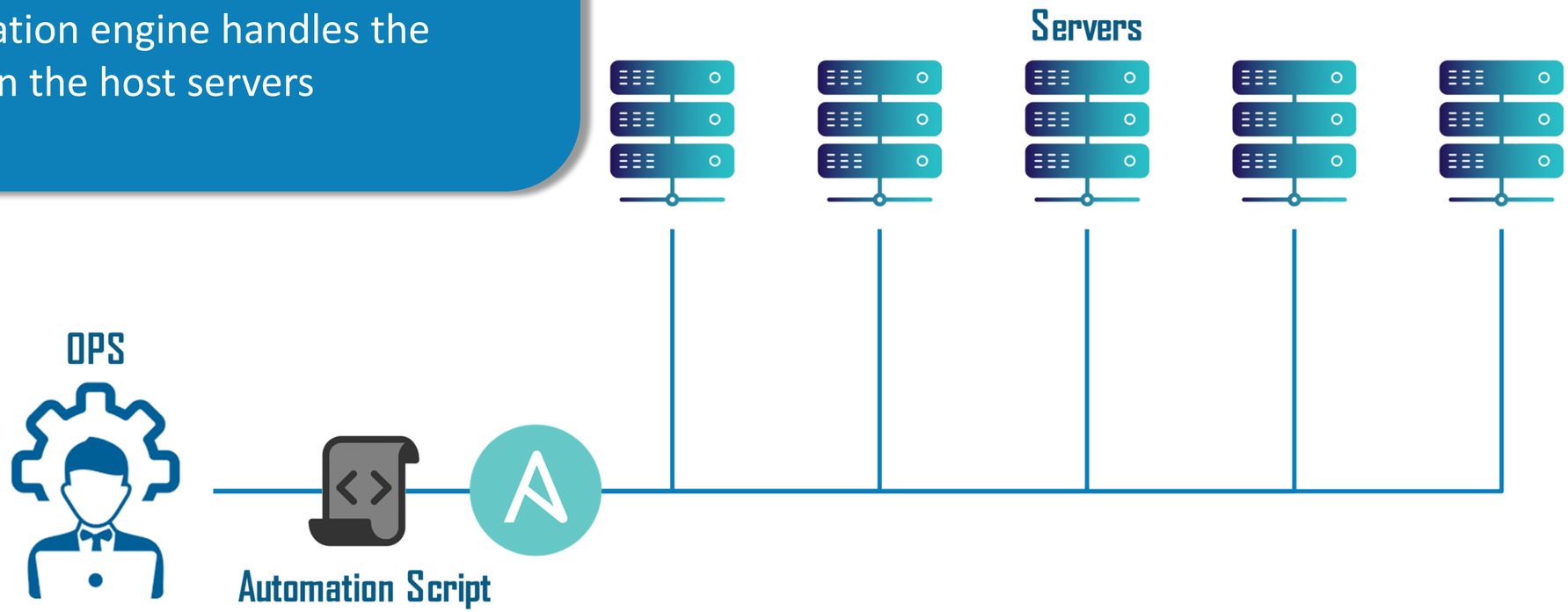
Centralizing Configuration Management

With Ansible users can automate and bring the servers to the desired state with a single automation script



Automation Using Ansible

- The desired state of the server is mentioned in the automation script
- The Ansible automation engine handles the differences between the host servers



edureka!

Automation Using Ansible

Ansible allows users to automate their environment using two different ways:

Ad-Hoc Commands

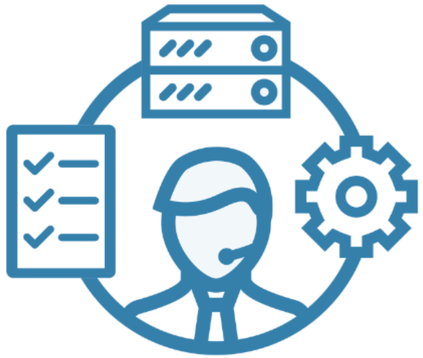
Playbooks

Ad-Hoc Commands

edureka!

Ad-Hoc Commands

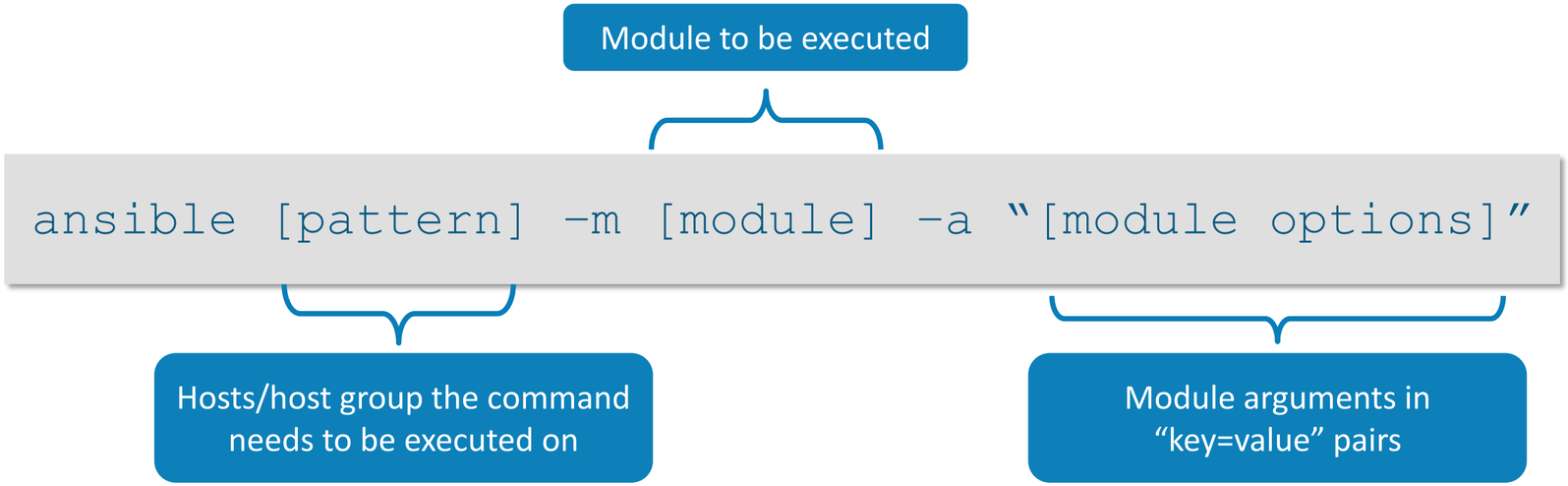
- Ansible ad-hoc commands are used to accomplish tasks quickly
- It uses the `ansible` command line tool to execute tasks
- These commands can be executed on single or multiple hosts using any inventory file
- Any Ansible module can be used to run as an ad-hoc task



Why Use Ad-Hoc Commands?

- Used to execute one-off tasks
- Quick and easy to execute
- Non-reusable
- For Example:
  - Tasks like rebooting servers, copying files, gathering facts, etc. can be easily done using a single ad-hoc command

Ad-Hoc Command Syntax




```
#Example
 ansible frontend –m service –a “name=nginx state=restarted”
```

Demo: Ad-Hoc Commands

edureka!

Ansible Playbooks

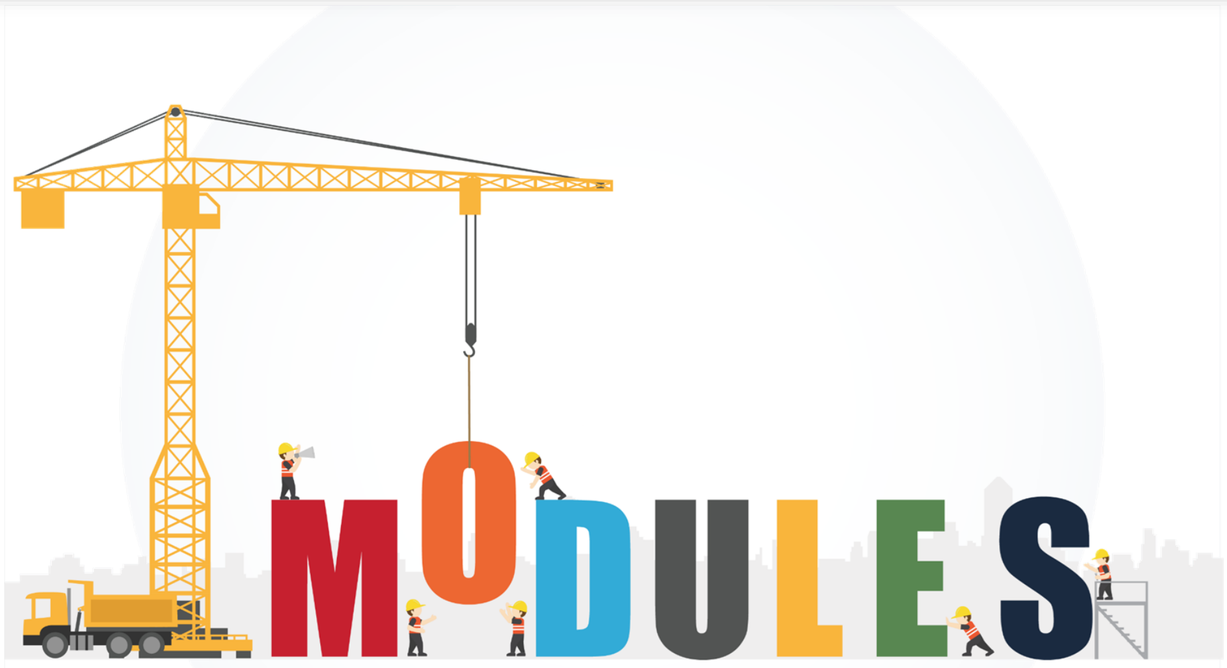
edureka!

What is a Playbook?

- Playbooks are the access point to Ansible provisioning.
- It is the Ansible's way of deploying and configuring different remote servers and environments
- It is written in **YAML(YAML Ain't Mark-up Language)**
- On an advanced level playbooks can be used to
  - Handle multi-tier rollouts
  - Load balancing tasks for the servers

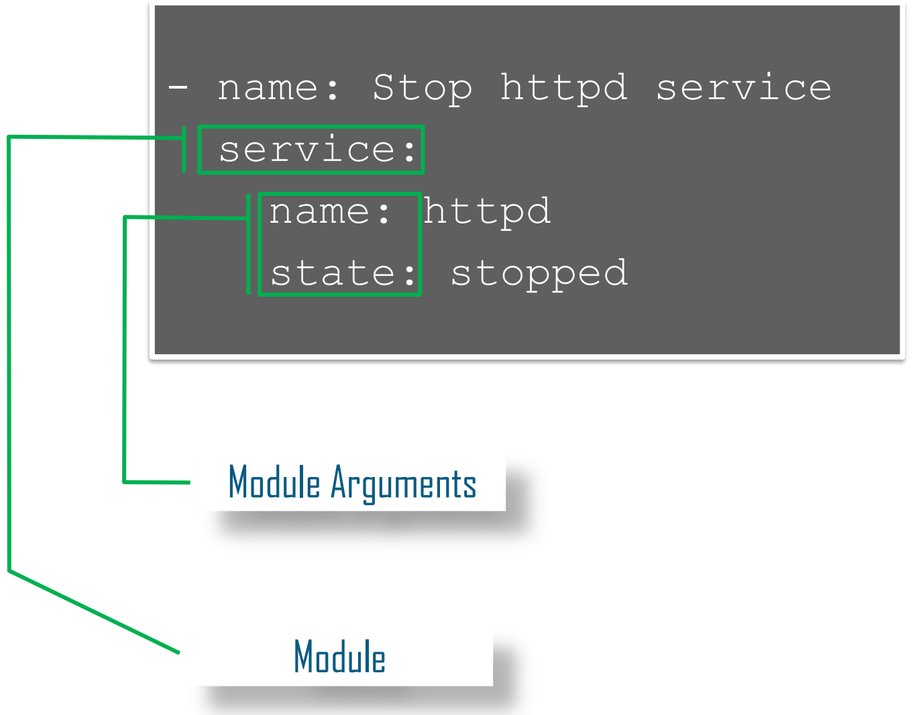
Playbook: Modules

Modules are units of code which get the work done in Ansible. They are used to control system resources like services, packages, files or execute system commands



Playbook: Modules (Contd.)

- Modules abstract system tasks, like dealing with packages or handling files from Ansible servers
- Users can choose to either use a module from Ansible's in-built library or create their own custom modules
- Almost all the modules support taking arguments in key=value format



Playbook Structure

edureka!

Playbook Structure

Besides the basic structure of a YAML file there are a few things to be kept in mind before writing a playbook

```
---
- hosts: all
  become_user: desmond
  become_method: su
  vars:
    region: northeast
  tasks:
  - name: add docker repo
    apt_repository:
      repo: deb [arch=amd64] https://...
      state: present
    notify:
      - update registry
  handlers:
  - name: update registry
    apt:
      update_cache: yes
```

Note: This is a sample playbook

Playbook Structure: ---

All playbooks with three hyphens '---'at the top.

They signify the beginning of a playbook

```
---
- hosts: all
  become_user: desmond
  become_method: su
  vars:
    region: northeast
  tasks:
  - name: add docker repo
    apt_repository:
      repo: deb [arch=amd64] https://...
      state: present
    notify:
      - update registry
  handlers:
  - name: update registry
    apt:
      update_cache: yes
```

Playbook Structure: Hosts

- Each play in a playbook starts with the host(s) the play is supposed to be executed on
- There may be multiple plays in a single playbook

```
---
- hosts: all
  become_user: desmond
  become_method: su
  vars:
    region: northeast
  tasks:
  - name: add docker repo
    apt_repository:
      repo: deb [arch=amd64] https://...
      state: present
    notify:
      - update registry
  handlers:
  - name: update registry
    apt:
      update_cache: yes
```

Playbook Structure: Become

- Become is a privilege escalation directive in Ansible
- Here, the three directives tells ansible to become the user desmond and run the tasks with sudo privileges

```
---
- hosts: frontend
  become: yes
  become_user: desmond
  become_method: su
  vars:
    region: northeast
  tasks:
  - name: add docker repo
    apt_repository:
      repo: deb [arch=amd64] https://...
      state: present
    notify:
      - update registry
  handlers:
  - name: update registry
    apt:
      update_cache: yes
```

Playbook Structure: Vars

- Variables are defined by using the vars keyword
- Playbooks are executed line by line hence variables need to be defined before their use in playbooks

```
---
- hosts: frontend
  become: yes
  become_user: desmond
  become_method: su
  vars:
    region: northeast
  tasks:
  - name: add docker repo
    apt_repository:
      repo: deb [arch=amd64] https://...
      state: present
    notify:
      - update registry
  handlers:
  - name: update registry
    apt:
      update_cache: yes
```

Playbook Structure: Tasks

- All the tasks that are to be executed on remote systems are defined under the tasks section
- A single play can consist multiple tasks
- Tasks are executed in order

```
---
- hosts: frontend
  become: yes
  become_user: desmond
  become_method: su
  vars:
    region: northeast
  tasks:
  - name: add docker repo
    apt_repository:
      repo: deb [arch=amd64] https://...
      state: present
    notify:
      - update registry
  handlers:
  - name: update registry
    apt:
      update_cache: yes
```

Playbook Structure: Handlers

- Tasks may contain notify actions which are triggered at the end of a play
- Notify triggers call upon tasks in the handler section

```
---
- hosts: frontend
  become: yes
  become_user: desmond
  become_method: su
  vars:
    region: northeast
  tasks:
  - name: add docker repo
    apt_repository:
      repo: deb [arch=amd64] https://...
      state: present
    notify:
      - update registry
  handlers:
  - name: update registry
    apt:
      update_cache: yes
```

Playbook Structure: Blocks

- Blocks can be used to create groups of tasks within a playbook
- All directives applied to a block apply to all the tasks in the block as well

```
tasks:
  block:
    - name: add docker repo
      apt_repository:
        repo: deb [arch=amd64] https://...
        state: present
      notify:
        - update registry
```

Demo: Running a Simple Playbook

edureka!

Variables

edureka!

Variables in Playbooks

- Variables are Ansibles way to deal with differences between different systems and environments
- They can handle major/minor differences between systems with same configuration
- Users can also collect facts from other remote systems and store them in variables
- Variables can be defined and used anywhere in Ansible:
  - Playbooks
  - Ad-Hoc command
  - Inventory

Defining and Referencing Simple Variables

```
#Defining a simple variable
 region: asia

#Referencing a simple variable
 server: "{{region}}"
```

Defining and Referencing a List of Variables

```
#List variables can be defined in YAML list format or in '[]' brackets
 region:
   - asia
   - europe
   - northamerica
   - southamerica
#OR
 region: ['asia', 'europe', 'northamerica', 'southamerica']

#Referencing a list of variables
 server: "{{region[2]}}"
```

Defining and Referencing Dictionary Variables

```
#Defining variables as key:value dictionaries
 sample:
   arg1: yes
   arg2: no

#Referring key:value dictionary variables
 sample['arg1']
#OR
 sample.arg2
```

Registering Variables

Outputs from other tasks can be stored and used as variables for other variables.
The process is known as registering variables

```
#Example
- tasks:
  - name: execute shell script
    shell: somescript.sh
    register: result

  - name: restart Docker if result < 5
    service: “name=docker state=restarted”
    when: result.rc < 5
```

Utilizing Handlers

edureka!

Handlers

- Handlers are tasks that execute only if the system configuration changes because of a task
- They only execute if a task notifies it for execution
- Handlers require a globally unique name for identification

Handlers Syntax

```
#Syntax
 - name: Install ntp
   apt:
     pkg: ntp
     state: present
   notify:
     - run update

 handlers:
 - name: run update
   apt:
     update_cache: yes
```

Controlling Handlers

- Handlers always run after a play has completed its tasks execution
- This ensures that handlers run only once even on multiple callings

```
#A Flush handlers task allows users to execute
#handlers that have been notified before the play ends
- name: Flush hanlders
  meta: flush_handlers
```

Demo: Using Variables and Handlers

edureka!

Ansible Roles

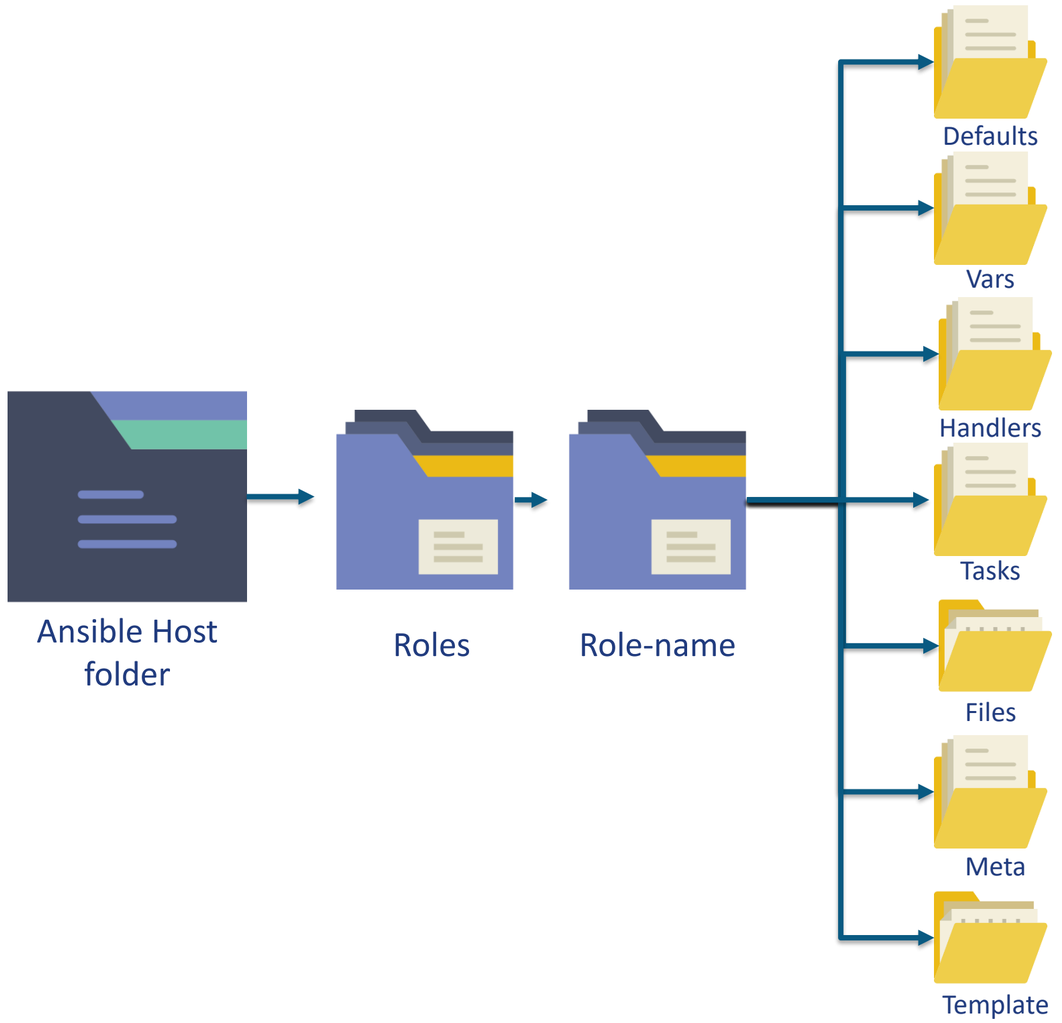
edureka!

Ansible Roles

Overtime working with Ansible a user may create hundreds of playbooks, variables, templates, defaults etc. Roles allow users to group this logic into an organized manner making reusability and sharing of Ansible structure easier

Ansible Roles

- Roles uses directories to structure and group all the playbooks, variables, templates, tasks, handlers, files, and defaults
- This collected logic can be grouped in any way the user wants, for example you can group server specific roles together
- These roles can then be used inside playbooks and even as in-line commands



Demo: Ansible Roles

edureka!

Summary

Configuration Management

Configuration Management is all about bringing in consistency in the infrastructure. This is done by ensuring that the current design system, state and environment is known, trusted and agreed upon by everyone. Configuration Management also helps record all the changes made in the system.

edureka!

What is Ansible?

Ansible is a deployment automation tool which traditionally uses push approach to achieve its objectives, by managing all the servers through one single machine running the Ansible Configuration Management Tool

ANSIBLE

edureka!

Ansible Architecture

Ansible connects to the nodes and pushes out small programs called **'Modules'**. These Modules bring the systems to the desired state. Ansible uses SSH to execute these modules and then removes them when finished

edureka!

Automation Using Ansible

- The desired state of the server is mentioned in the automation script
- The Ansible automation engine handles the differences between the host servers

edureka!

What is a Playbook?

- Playbooks are the access point to Ansible provisioning.
- It is the Ansible's way of deploying and configuring different remote servers and environments
- It is written in **YAML(YAML Ain't Mark-up Language)**
- On an advanced level playbooks can be used to
  - Handle multi-tier rollouts
  - Load balancing tasks for the servers

PLAYBOOK

edureka!

Ansible Roles

Overtime working with Ansible a user may create hundreds of playbooks, variables, templates, defaults etc. Roles allow users to group this logic into an organized manner making reusability and sharing of Ansible structure easier

edureka!

edureka!

edureka!

Thank You

For more information please visit our website
www.edureka.co