Need of Automation?

> Accomplish processes that is difficult to do manually

- IT automation tools allow you to do a wide variety of things that simply can't be done manually. For example, provisioning and deploying environments manually can be time-consuming and requires highly skilled, hands-on knowledge.

> Control your delivery chain while maintaining compliance

- By adopting a CI/CD pipeline that applies automation throughout your application life cycle—including integration, testing, delivery and deployment—your teams can produce tested and verified applications more quickly and with zero downtime.

> Eliminate errors and risks associated with routine tasks

- It improves consistency, repeatability and verifiability throughout your operations. By reducing the risk of human error in common daily tasks, automation gives you predictable and repeatable processes for managing configurations to improve consistency, speed changes and increase uptime.
- The goal is to increase the speed and reliability of software delivery by reducing manual intervention and promoting alignment between development and operations teams. DevOps automation tools can speed up delivery cycles by reducing human error and bottlenecks, which can result in shorter feedback loops. This increased efficiency can apply to the entire process, from the most recent code committed to a repository to the final release and delivery of an application or service upgrade.
- DevOps automation can also augment DevOps monitoring principles, such as logging, alerting, tracing, auditing, and monitoring.

Ansible

- Open-source IT automation tool that simplifies the process of
  - Install Software / Orchestrating workflows
  - Configure Server / Configuring systems
  - Deploy App / Deploying software

> If we do not have Ansible?

- Then we will use bash script and, use ssh method to connect to server and run the script to install required software.

> If we have Ansible

- Here instead of running command like bash script, we can have description of it, and it is easy to understand.

> Advantages of Ansible

- Simple to Learn
- Integration with DevOps Practices
- Agentless Architecture
- Powerful Automation Capabilities
- Use of Python and YAML

Why it is Agent-less ?

- Ansible is installed on Developer's machine and not on server.
- When we execute it, it uses ssh to connect to remote machine/server and execute all the required commands.
- To put it simply `Ansible is called agentless because it doesn't require any software to be installed on the machines it manages. Instead, it connects to these machines over SSH or WinRM and executes tasks directly, eliminating the need for managing agents on each target system.`

> How Ansible works?

- Need to write `YAML` file and it is called Ansible playbook.

> Install Ansible

- For windows:
  - Install wsl
  - `sudo apt-get update`
  - `sudo apt-get upgrade`
  - `sudo apt-get python3-pip git libffi-dev libssl-dev python-is-python3 -y`
  - `sudo apt-get install ansible`

Project Structure of Ansible

- Ansible_Project_Folder
  - *Inventory* Folder
    - *vm-setup-playbook* Folder
      - *hosts*: file containing IP / Domain name of all the host.
  - *roles* Folder // it contains many roles, each role has a task to be performed
    - *role1* Folder
      - *tasks* Folder
        - *main.yaml* File
    - *playbook.yaml* Core File of our project.
      - Include remote host to connect from hosts file
      - Define remote user
      - Include Roles to run

-Cmd to execute playbook:

```
ansible-playbook --inventory inventory/folder-name/hosts main-playbook-file.yaml
```

Setting Private and Public key for Ansible

- `ssh-keygen` creates this pair of keys.
- Private for us, Public for Server.
- `ssh-copy-id -i path/to/public/key userName@IP_OF_SERVER` Use this command to copy public key to the remote server.
- Now associates private key with playbook.
    - In host file alongside IP address set variable `ansible_ssh_private_key_file` with value of `path/to/private_key`

Ansible Handlers

- What are They ?
    - Ansible handlers are special tasks that execute only when notified by another task using the notify directive.
- When to use them?
    - They are designed to perform actions like restarting services when a configuration change occurs, ensuring idempotency by running only when there's a change in state.
- Handlers are defined under the handlers: section in a playbook and are executed at the end of the play, after all tasks have completed.

Variables in Ansible

- Types
    - String

```
main_playbook_variable: "Nisarg"
```

    - Boolean

```
is_Nisarg_Good: true
```

    - List

```
favorite_anime:
    - Naruto
```

```
      - One Piece
      - Horimiya
      - Yourname
      - Black Clover
      - Pokemon

      # Accessing a value
  debug:
      var: favorite_anime[0]
```

- Map/Dictionary

```
  price_of_manga:
      naruto: "RS100"
      one-piece: "RS100"

      # Accessing a value
  debug:
      var: price_of_manga.naruto
```

- You can declare a variable in ansible using `vars` property as:

```
---
- name: some_name
  vars:
    variable_name: variable_value
```

- To Print Something while executing playbook, write `debug`

```
debug:
    msg: "Message to be printed"
    var: var_name #variable name to be printed. # Optional
    when: boolean condition when to print # Optional
```

Registering the value of old var to new var

```
- name: some_name
  command: echo "{{price_of_manga['one-piece']}}"
  register: one_piece_manga_price
- name: debug new var value
  debug:
    var: one_piece_manga_price.stdout
```

Nested Vars

```
watch_list:
    - name: shounen
      anime:
        - naruto
        - one-Piece
    - name: slice-of-life
      anime:
        - Your name
        - Hello World

# Accessing above vars.
    - name: Get price of naruto by accessing it's value from above var and using
it in price_of_manga var.
      debug:
        var: price_of_manga[watch_list[0].anime[0]]
```

Jinja 2 Filters on variables

```
- name: Using Jinja 2 Filters
  debug:
    var: price_of_manga.keys() | list | map('upper') | list
```

Importing vars file

- We can also declare only variables in a file and import that file in to our playbook

```
vars_files:
  - path-to-vars-file.yaml
```

- After this we can use variable present in file as we use to do when declared in playbook.

Passing Variable at runtime

- Execute command as

```
ansible-playbook --inventory path/to/hosts path/to/ansible-playbook.yaml --extra-
vars '{"var1":"This is Extra Var"}'
```

- In Playbook

```
- name: Var at runtime
  debug:
    var: var1
```

Passing Vars file at runtime

- Execute command as

```
ansible-playbook --inventory path/to/hosts path/to/ansible-playbook.yaml --extra-vars "@path/to/vars-file.yaml"
```

- In Playbook we can access variable normally
- Can also pass JSON as we passed yaml

Precedence of Variables

- The Variables entered at runtime i.e. passed as command line argument is given preference.

Environment variables

- We can declare it using, environment property.

```
# below is variable at playbook level
environment:
  EXAMPLE_VAR_NAME: "value"

tasks:
  - name: Environment var at playbook level
    ansible.builtin.command: "echo $EXAMPLE_VAR_NAME"

  - name: ENV var at task level
    environment:
      TASK_LEVEL_VARIABLE: "Value"
    ansible.builtin.command: "echo $TASK_LEVEL_VARIABLE"
```

Jinja2

- It is a template engine, which is used to implement dynamic configuration and dynamic script.
- To achieve dynamic config we need to use interpolation syntax, i.e.

```
{{var_name}}
```

- to use this kind of syntax.

How to Call Jinja2 template

- Use the keyword template and specify source and destination file path
- Declare variable that is used in the template.
- Playbook

```
#Our Playbook

- name: Generate Config File
  template:
    src: config_file.j2
    dest : /path/name.conf
    owner: root
    group: root
    mode: '0644'
 vars:
    server_port: 80
```

- Jinja2 Template

```
# config_file.j2
server.modules = (
        "mod_indexfile",
        "mod_access",
        "mod_alias",
        "mod_redirect",
    )

    server.document-root        = "/var/www/html"
    server.upload-dirs          = ( "/var/cache/lighttpd/uploads" )
    server.errorlog             = "/var/log/lighttpd/error.log"
    server.pid-file             = "/run/lighttpd.pid"
    server.username             = "www-data"
    server.groupname            = "www-data"
    server.port                 = {{ server_port }}

server.http-parseopts = (
  "header-strict"           => "enable",# default
"host-strict"             => "enable",# default
"host-normalize"          => "enable",# default
"url-normalize-unreserved"=> "enable",# recommended highly
```

```
"url-normalize-required"  => "enable",# recommended
"url-ctrls-reject"        => "enable",# recommended
"url-path-2f-decode"      => "enable",# recommended highly (unless breaks app)
"url-path-dotseg-remove"  => "enable",# recommended highly (unless breaks app)
)

index-file.names            = ( "index.php", "index.html" )
url.access-deny             = ( "~", ".inc" )
static-file.exclude-extensions = ( ".php", ".pl", ".fcgi" )

compress.cache-dir          = "/var/cache/lighttpd/compress/"
compress.filetype           = ( "application/javascript", "text/css", "text/html",
"text/plain" )

include_shell "/usr/share/lighttpd/create-mime.conf.pl"
include "/etc/lighttpd/conf-enabled/*.conf"

server.modules += (
    "mod_compress",
    "mod_dirlisting",
    "mod_staticfile",
)
```

Deployment Strategies

- Single Server
  - Only Single Host in **hosts** file.
  - Simple command for execution

```
ansible-playbook --inventory path/to/hosts playbook-name.yaml
```

- Multi Server
  - Multiple Hosts in **hosts** file.

```
[blue]
ip1
ip2

[green]
ip3
ip4
```

  - same command as above.

Deploy on All Servers

Deploy on Selected Server

Blue Green Deployment Strategy

- When deploying app on multiple servers it is difficult to remember IP of all, so we divide it into 2 main groups: *BLUE* and *GREEN* servers.
- Green(Active):
  - Actively working servers in production environment, which actually servers users.
- Blue(Passive):
  - It is also production server, but not actively serving users.
  - New features are deployed here first.
- So once our deployment of new features is successful on **blue** servers, then we will cutoff traffic from **green** server and redirect it to blue one.
- So no downtime in production.
- Need to add `--limit` flag in ansible command to limit deployment to a particular group.

```
ansible-playbook --inventory path/to/hosts playbook.yaml --limit
group_name_in_hosts_file
```

Ansible Vault

- Encrypts important data that is in plain text format when in playbook.
- Ansible Vault is a built-in tool that lets you encrypt sensitive information (passwords, API keys, etc.) within your Ansible playbooks and variable files. This ensures that this data remains hidden from unauthorized users who might have access to the playbook files.
- When you run the playbook using ansible-playbook main.yml, Ansible will prompt you for the vault password to decrypt the variables on the fly.
- For more refer this Article

- Create vault

```
ansible-vault create path/to/vault.yaml
```

- Edit

```
ansible-vault edit path/to/vault.yaml
```

- View

```
ansible view path/to/vault.yaml
```

- Change Password

```
ansible-vault rekey path/to/vault.yaml
```