Ansible Cheat Sheet with Real-Time Use Cases - Part 2 🚀

16. Error Handling

```yaml
---
- name: Controlling failure
  hosts: all
  gather_facts: no
  tasks:
  - name: Verifying presence of apache
    shell: rpm -q httpd
    ignore_errors: yes
    register: result
  - name: Printing Verification result
    debug: var=result.stdout
  - name: package installation
    yum: name=httpd state=latest
  - name: Clearing cache
    shell: yum clean all
    failed_when: result.rc == 0
    ignore_errors: yes
```

- **Command:** `failed_when` and `changed_when`
- **Use Case:** Fine-tune task outcomes based on specific conditions.

```
tasks:
  - name: Handle errors gracefully
    command: /path/to/failing_script.sh
    failed_when: "'FAILED' in result.stdout"
```

17. Dynamic Playbook Inclusion

```yaml
- name: Include tasks based on OS
  include_tasks: "{{ ansible_os_family }}_tasks.yaml"
```

- **Command:** `include_tasks` and `import_tasks`
- **Use Case:** Include or import tasks dynamically based on conditions.

```
- name: Include tasks based on OS
  include_tasks: "{{ ansible_os_family }}_tasks.yaml"
```

18. Vaulted Files

- **Command:** `ansible-vault create vaulted_file.yaml`
- **Use Case:** Encrypt entire files containing sensitive data.

- `ansible-vault create vaulted_file.yaml`
- `ansible-vault create hosts # helps to encrypt the inventory file`

19. Rollback Strategies

- **Command:** `--start-at-task` and `--step`
- **Use Case:** Implement rollback strategies by starting from specific tasks or enabling step-by-step execution.

```
ansible-playbook deploy.yaml --start-at-task="rollback_task"
```

20. Remote Facts Gathering

- **Command:** `gather_facts`
- **Use Case:** Gather facts about remote hosts for dynamic playbook behavior.

```
- name: Gather facts
  gather_facts: yes
```

21. Delegation and Local Actions

- **Command:** `delegate_to` and `local_action`
- **Use Case:** Delegate a task to a different host or execute a task locally.

```
- name: Copy files to remote host
  copy:
    src: /path/to/local/files
    dest: /remote/path/
  delegate_to: "{{ inventory_hostname }}"
```

22. Asynchronous Actions with Polling

- **Command:** `async` and `poll`
- **Use Case:** Execute a task asynchronously with a defined poll interval.

```
- name: Run a long-running task asynchronously
  command: /path/to/long_running_script.sh
  async: 300
  poll: 60
  register: async_result
```

23. Dynamic Variable Assignment

- **Command:** `set_fact`
- **Use Case:** Dynamically assign variables based on conditions.

```
- name: Set variable based on OS
  set_fact:
    my_variable: "{{ 'Linux' if ansible_distribution == 'Ubuntu' else
'Other' }}"
```

24. Callbacks

- **Command:** `ANSIBLE_STDOUT_CALLBACK`
- **Use Case:** Customize output callbacks for better visualization.

```
export ANSIBLE_STDOUT_CALLBACK=actionable
```

25. Custom Facts

- **Command:** `ansible_facts`
- **Use Case:** Define custom facts for specific hosts.

```
- name: Set custom facts
  set_fact:
    my_custom_fact: "some_value"
```

26. Integration with Source Control (Git)

- **Command:** `git` module
- **Use Case:** Clone and update Git repositories as part of your automation.

```
- name: Clone a Git repository
  git:
    repo: "https://github.com/example/repo.git"
    dest: "/path/to/local/repo"
```

27. Handling Timeouts

- **Command:** `ansible.builtin.async_status` and `ansible.builtin.fail`
- **Use Case:** Implement a timeout mechanism and handle accordingly.

```
- name: Run a task with timeout
  command: /path/to/timeout_script.sh
  async: 300
  poll: 0
  register: result

- name: Handle timeout
  async_status:
    jid: "{{ result.ansible_job_id }}"
  register: job_result
  until: job_result.finished
  retries: 3
  failed_when: "'Timeout' in job_result.failed"
```

28. Vaulted Variable Files

- **Command:** `ansible-vault create vars/vaulted_file.yaml`
- **Use Case:** Encrypt variable files containing sensitive data.

```
ansible-vault create vars/vaulted_file.yaml
```

29. Inventory Plugins

- **Command:** Various inventory plugins
- **Use Case:** Use dynamic inventory plugins for fetching inventory information from external sources (e.g., AWS, Azure).

```
ansible-inventory -i aws_ec2.yaml --list
```

30. Custom Callback Plugins

- **Command:** `ANSIBLE_STDOUT_CALLBACK`
- **Use Case:** Develop and use custom callback plugins for tailored output.

```
export ANSIBLE_STDOUT_CALLBACK=my_custom_callback
```

31. Dynamic Subgroups

- **Command:** `groups`
- **Use Case:** Dynamically create subgroups in the inventory based on specific conditions.

```
[web_servers]
server1
server2

[app_servers]
server3

[all:children]
web_and_app_servers
```

32. Advanced Looping

- **Command:** `loop`, `loop_control`
- **Use Case:** Use advanced looping with control statements for more complex scenarios.

```
- name: Loop with condition
  debug:
    msg: "Item is {{ item }}"
  loop: "{{ range(10) }}"
  loop_control:
    loop_var: item
    continue_loop: "{{ item % 2 == 0 }}"
```

33. Jenkins Integration

- **Command:** Use Ansible playbooks in Jenkins pipelines.
- **Use Case:** Integrate Ansible into Jenkins jobs for seamless automation in CI/CD pipelines.

34. Parallel Execution Control

- **Command:** `serial`
- **Use Case:** Control the number of hosts Ansible manages in parallel to avoid overwhelming resources.

```
- name: Execute tasks serially
  command: /path/to/task_script.sh
  serial: 2
```

35. Ansible Vault File Edit

- **Command:** `ansible-vault edit`
- **Use Case:** Edit an encrypted file directly without decrypting it first.

```
ansible-vault edit encrypted_file.yaml
```

36. Selective Variable Loading

- **Command:** `--extra-vars`
- **Use Case:** Load specific variables dynamically during playbook execution.

```
ansible-playbook deploy.yaml --extra-vars "env=production"
```

37. Inventory Filters

- **Command:** `--limit`, `--exclude`, `--inventory`
- **Use Case:** Limit or exclude hosts dynamically during playbook execution.

```
ansible-playbook deploy.yaml --limit web_servers
```

38. Custom Dynamic Inventory Script

- **Command:** Custom Python script
- **Use Case:** Develop a custom dynamic inventory script for fetching host information from non-standard sources.

```
ansible-inventory -i custom_inventory.py --list
```

39. Fork Control

- **Command:** `--forks`
- **Use Case:** Control the number of parallel processes Ansible uses for running tasks.

```
ansible-playbook deploy.yaml --forks 5
```

40. Managing Role Dependencies

- **Command:** `ansible-galaxy`
- **Use Case:** Manage dependencies of Ansible roles, ensuring smooth integration.

```
ansible-galaxy install -r requirements.yaml
```

Real-Time Use Cases (Continued)

- **Blue-Green Deployments:**
  - Implement a blue-green deployment strategy using Ansible to minimize downtime during application updates.
- **Secrets Rotation:**
  - Use Ansible to automate the rotation of secrets and credentials across your infrastructure securely.
- **Continuous Compliance:**
  - Integrate Ansible with compliance tools to ensure continuous adherence to security policies and regulatory requirements.
- **Ephemeral Environments:**
  - Automate the creation and teardown of ephemeral environments for testing and development purposes.
- **Auto-Scaling Infrastructure:**
  - Implement auto-scaling by dynamically adjusting the number of servers based on workload using Ansible.
- **Multi-Tier Application Deployment:**
  - Automate the deployment of multi-tier applications, including frontend, backend, and database components.
- **Multi-Cloud Deployment:**
  - Use Ansible to deploy and manage resources across multiple cloud providers (e.g., AWS, Azure, GCP) for a hybrid or multi-cloud infrastructure.
- **Automated Disaster Recovery:**
  - Develop Ansible playbooks for automating disaster recovery procedures, ensuring quick and reliable recovery in case of system failures.
- **Environment Drift Detection:**
  - Implement Ansible to detect and remedy configuration drift across servers, ensuring consistency in large-scale environments.
- **Integration with Service Discovery Tools:**
  - Seamlessly integrate Ansible with service discovery tools (e.g., Consul, etcd) for dynamic inventory updates and configuration management.
- **Custom Credential Management:**
  - Use Ansible Vault for encrypting credentials and implement custom credential management practices for enhanced security.
- **Custom Module Development for API Interaction:**
  - Develop custom Ansible modules to interact with external APIs, expanding automation capabilities beyond built-in modules.
- **Container Orchestration Integration (Docker, Kubernetes):**
  - Use Ansible to automate tasks related to container orchestration, such as deploying Docker containers or managing Kubernetes resources.

- **Network Automation:**
    - Extend Ansible capabilities to automate network device configurations and monitoring tasks.
- **Continuous Monitoring Setup:**
    - Automate the deployment and configuration of monitoring solutions for continuous infrastructure and application monitoring.
- **Application Configuration Templating:**
    - Utilize Jinja2 templating to dynamically generate application configurations based on environment variables or other parameters.
- **Cross-Platform Management:**
    - Manage and configure a diverse set of servers with different operating systems using Ansible's cross-platform capabilities.
- **Environment-specific Parameterization:**
- Use Ansible to parameterize environment-specific configurations, ensuring consistency across various deployment environments.

Conclusion

By exploring these advanced Ansible commands and real-time use cases, you can further refine your automation strategies. Ansible's extensive capabilities enable you to address intricate challenges in IT operations, making it a valuable tool for orchestrating complex environments. 🔄 🛠️ With these advanced Ansible commands and real-time use cases, you can address complex scenarios and build robust automation solutions. Ansible's flexibility and extensibility make it a valuable tool for managing and orchestrating infrastructure at scale. 🌐 This Ansible cheat sheet, along with real-time use cases, serves as a handy reference for both beginners and experienced DevOps engineers. With its simplicity and flexibility, Ansible empowers teams to automate a wide range of tasks, enhancing efficiency and reliability in IT operations. 💻