🧪Lab 13 — Advanced Ansible Facts & Fact Filtering

Author: *Sandeep Kumar Sharma*

🕗Learning Objectives

In this lab you will learn: - Advanced usage of the `setup` module and fact filtering - How to gather only required subsets of facts using `gather_subset` - How to cache facts for performance with fact caching - How to create and use custom facts (ansible_local and facts.d) - How to manipulate and persist facts using `set_fact` - How to use facts for conditional logic and complex decision-making

---

👙 Learning Outcomes

After completing this lab you will be able to: - Collect targeted facts instead of the entire fact set - Improve performance by enabling fact caching - Provide custom facts from the hosts to the playbooks - Use `set_fact` to compute and reuse runtime values - Leverage facts in complex conditionals and templates

---

🧰Gathering Targeted Facts with `setup`

The `setup` module can filter returned facts to reduce output and processing time.

Examples:

```
ansible dev -m setup -a "filter=ansible_distribution*"
ansible dev -m setup -a "filter=ansible_default_ipv4"
ansible dev -m setup -a "filter=ansible_mem*"
```

The module supports wildcard patterns. Use filtering to extract only the data you need.

🪙Using `gather_subset` in a Playbook

Instead of gathering all facts, gather a subset for speed.

Example playbook snippet:

```
- hosts: dev
  gather_facts: no

  tasks:
    - name: Gather a minimal set of facts
      setup:
        gather_subset:
          - '!all'
          - 'min'
```

Options include `all`, `min`, `hardware`, `network`, `virtual`, etc.

🪙Fact Caching (Performance Optimization)

For large inventories, enable fact caching to avoid repeated data collection.

Supported cache plugins include: `jsonfile`, `redis`, `memcached`.

Example `ansible.cfg` snippet:

```
[defaults]
fact_caching = jsonfile
fact_caching_connection = /path/to/facts_cache
fact_caching_timeout = 86400
```

After enabling cache, Ansible will store gathered facts and reuse them during subsequent runs (until timeout).

🪙Custom Facts: `ansible_local` and `facts.d`

1. `ansible_local` facts

- If a host exposes a fact under `/etc/ansible/facts.d/` (JSON or executable), the data appears in `ansible_local`.
- Useful for exposing host-level configuration, service metadata, or inventory-specific info.

Example custom fact (JSON file): `/etc/ansible/facts.d/custom.fact`

```
{
  "app_version": "1.2.3",
  "role": "web"
}
```

After running `setup`, the facts are available as `ansible_local.custom.app_version`.

2. Executable facts

- Place an executable script in `facts.d` that prints JSON. Make it executable and owned by root.

🪙Using `set_fact` to Compute & Persist Facts During a Run

`set_fact` sets variables at runtime that persist for the duration of the playbook run.

Example:

```
- name: Compute package manager
  set_fact:
    pkg_mgr: "apt" if ansible_facts['os_family'] == 'Debian' else "yum"
```

`set_fact` values are stored in memory and can be used by later tasks.

🪙Registering and Using Module Output

Many modules return structured output which can be registered and then processed.

Example:

```
- name: Check disk usage
  command: df -h /
  register: disk_out

- debug:
    var: disk_out.stdout_lines
```

Use `failed_when` or `changed_when` combined with registered values to control behavior.

🪙Practical Examples

Example 1 — Gather only network facts and print default IPv4

```
ansible dev -m setup -a "filter=ansible_default_ipv4"
```

Example 2 — Use `gather_subset` to gather only network facts in playbook

```
- hosts: dev
  gather_facts: no

  tasks:
    - name: Gather only network facts
      setup:
        gather_subset:
          - network

    - name: Show primary IP
      debug:
        msg: "Primary IP: {{ ansible_facts['default_ipv4']['address'] }}"
```

Example 3 — Using custom `ansible_local` fact in a playbook

Assume `/etc/ansible/facts.d/custom.fact` exists on host.

```
- hosts: dev
  tasks:
    - name: Print application version from custom fact
      debug:
        msg: "App Version: {{ ansible_local.custom.app_version }}"
```

🪙Fact Filtering Tips

- Use `filter` to reduce noise when debugging.
- Use `gather_subset` for better performance in large environments.
- Use fact caching in CI or frequent runs to save time.
- Avoid excessive `set_fact` usage for large volumes of data; prefer computed vars when possible.

🧪 Hands-On Checklist

- [ ] Use `setup` with `filter` to fetch specific facts
- [ ] Use `gather_subset` in a playbook
- [ ] Enable and verify fact caching with `jsonfile`
- [ ] Create a custom JSON fact in `/etc/ansible/facts.d/` and read it
- [ ] Use `set_fact` to compute runtime variables
- [ ] Register module output and use it in conditionals

👙 Lab Summary

This lab explained advanced facts usage, performance tuning with subsets and caching, and how to expose custom host-level data to Ansible playbooks.

Author: Sandeep Kumar Sharma