OPITZ CONSULTING

PLAYBOOK DETAILS

VARIABLES IN PLAYBOOKS

- Variables are integrated into playbooks in this way:

```yaml
---
- hosts: all
  vars:
    user_name: example
  tasks:
    - name: Create user
      ansible.builtin.user: name="{{ user_name }}"
```

- Naming convention
  - Lowercase letters and underscores for separation

VARIABLES IN PLAYBOOKS

- Variables can also be stored in separate files

```yaml
---
- hosts: all
  vars_files:
    - myvars.yml
  tasks:
    - name: Create user
      ansible.builtin.user: name="{{ user_name }}"
```

Playbook

```yaml
---
user_name: example
simple_webservice:
  endpoint: "http://example.org/ws"
  timeout_ms: 30000
```

myvars.yml

VARIABLES IN ROLES

- **Directories for variables**
  - `vars` **for variables used internally in roles**
  - `defaults` **for variables that can be overridden from outside**
- **Variables in** `vars` **override those in** `defaults`**.**
- **Variable names are globally visible**
  - **Best practice: Let variables start with role names**
  - **Example:**

```
httpd/
  meta/
    main.yml
  tasks/
    main.yml
  handlers/
  vars/
  defaults/
    main.yml
  templates/
    httpd.conf.j2
  files/
    vhost.conf
```

`httpd/defaults/main.yml`

```
---
httpd_context_path: /
httpd_port: 80
```

VARIABLES IN PLAYBOOKS: HOST OR GROUP VARIABLES

- Group and host-specific variables in **`host_vars`** and **`group_vars`** directory
- Must be **relative to the inventory**

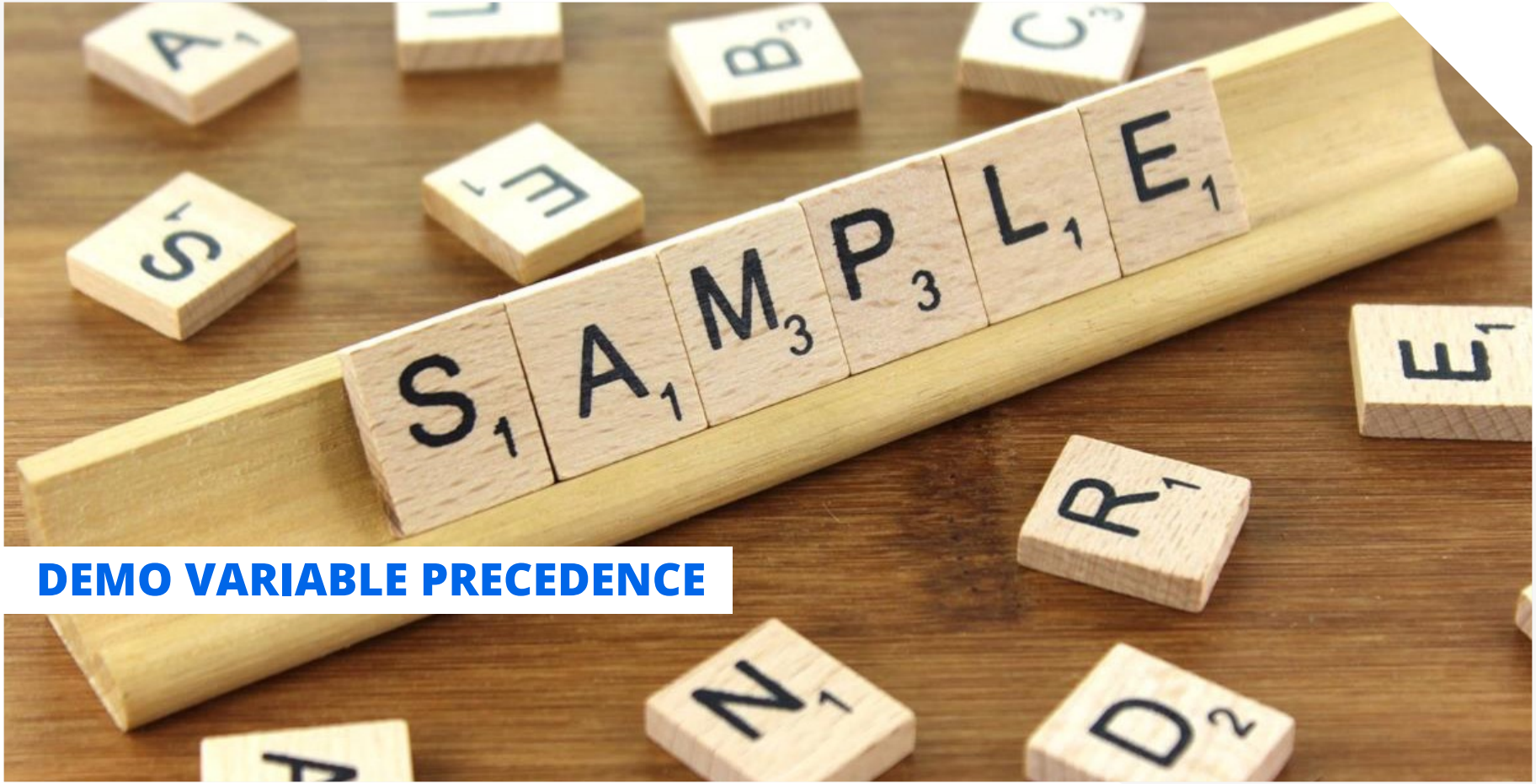
```
inventory
group_vars/
      all.yml
      database.yml
      webservers.yml
host_vars/
      web1.yml
      web2.yml
```

VARIABLE PRECEDENCE

Understanding variable precedence

Ansible does apply variable precedence, and you might have a use for it. Here is the order of precedence from least to greatest (the last listed variables override all other variables):

1. command line values (for example, `-u my_user` , these are not variables)
2. role defaults (defined in role/defaults/main.yml) [1]
3. inventory file or script group vars [2]
4. inventory group_vars/all [3]
5. playbook group_vars/all [3]
6. inventory group_vars/* [3]
7. playbook group_vars/* [3]
8. inventory file or script host vars [2]
9. inventory host_vars/* [3]
10. playbook host_vars/* [3]
11. host facts / cached set_facts [4]
12. play vars
13. play vars_prompt
14. play vars_files
15. role vars (defined in role/vars/main.yml)
16. block vars (only for tasks in block)
17. task vars (only for the task)
18. include_vars
19. set_facts / registered vars
20. role (and include_role) params
21. include params
22. extra vars (for example, `-e "user=my_user"`)(always win precedence)


SAMPLE
DEMO VARIABLE PRECEDENCE

DIFFERENT TYPES OF VARIABLES

- Simple variable:
  - `test_variable: "String Variable"`
  - `test_variable: 123`

Access:

```
"{{ test_variable }}"
```

- List

```
test_list:
  - 123
  - 456
  - 789
```

```
test_list:
  - "String 1"
  - "String 2"
  - "String 3"
```

Access:

```
"{{ test_list | first }}"
"{{ test_list[0] }}"
```

- Dictionary

```
test_dict:
  - key: test1
    value: test1-value
  - key: test2
    value: test2-value
  - key: test3
    value: test3-value
```

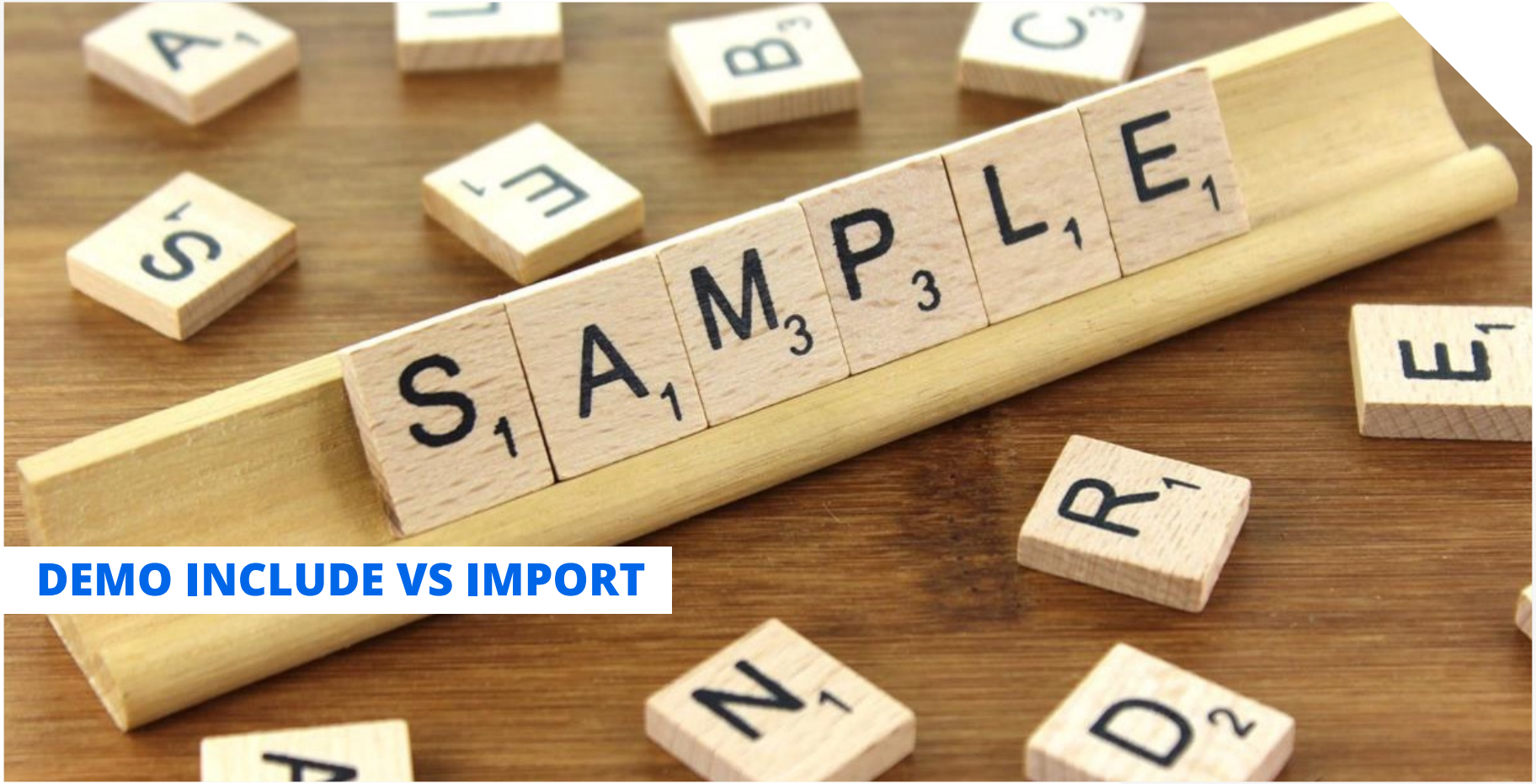
Access:

```
"{{ test_dict.key }}"
```

DEMO VARIABLES / ACCESS - DICTS

DIFFERENCE BETWEEN INCLUDE VS. IMPORT

- Code (tasks) can be outsourced to other files
  - For structuring
  - As a loop, for multiple execution of a sequence of tasks
- Various options for reusing tasks
  - ansible.builtin.import_playbook Imports a playbook
  - ansible.builtin.import_role Imports a role into a Play
  - ansible.builtin.import_tasks Imports a list of tasks
  - ansible.builtin.include_role Dynamic include of a role (at runtime - loop, etc.)
  - ansible.builtin.include_tasks Dynamic include of a task list (at runtime - loop, etc.)



DEMO INCLUDE VS IMPORT

EXAMPLE FOR META `MAIN.YML`

- Meta can be used to define additional roles as a **dependency of a role.**
- These roles are then executed first.

`roles/docker/meta/main.yml`

```yaml
---
dependencies:
  - os_base_config #including curl
```

TEMPLATES

- Ansible supports the template language Jinja2 to create text files dynamically, for example with variable values.

/message.j2

```
Today is {{ ansible_date_time.date }} on {{ ansible_hostname }}.
```

- Use the template module
  - The template file contains the same variables as the task

```
- name: Generate Message
  template:
    src: /message.j2
    dest: /message.txt
```

JINJA2 FILTERS AND VARIABLES

- Jinja2 filters can be used to modify the use of variables **in templates and playbooks.**
  - Wherever variables can be placed
- Examples:
  - {{ variable | mandatory }} # Throws error if variable not defined
  - {{ variable | default(5) }} # Returns 5 if variable not defined
  - {{ some_list | join(" ") }} # Joins list elements together
  - {{ 'mysecret' | password_hash('sha512') }} # SHA-512 hashed password
  - {{ filepath | basename }} # /some/path/to/foo.txt → foo.txt
  - {{ filepath | dirname }} # /some/path/to/foo.txt → /some/path/to
- Further examples under:
  - https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html

TAGGING IN PLAYBOOKS

- Tagging tasks allows the playbook execution to be filtered to specific tasks
- Notes
  - One day on several tasks possible
  - Multiple tags possible on one task
- Tag inheritance
  - A tag on a Play, Block, Role is inherited by all contained tasks

```
---
- hosts: db
  tasks:
    - name: Install MySQL
      ansible.builtin.apt: name=mysql-server
      tags:
        - install
        - update
    - name: Start Service MySQL
      ansible.builtin.service: name=mysql-server
      tags:
        - service
```

Use

```
ansible-playbook site.yml --tags install
ansible-playbook site.yml --skip-tags service
```

SPECIAL TAGS

- The special tags always and never have a special meaning
  - always is always executed, even if the tag was not specifically called
  - never is never executed unless it is explicitly called with the tag
- Special calls
  - all
  - tagged
  - untagged

```
---
- name: Install DB
  hosts: db
  tasks:
    - name: Install MySQL
      ansible.builtin.apt: name=mysql-server
      tags:
        - always
    - name: Start Service MySQL
      ansible.builtin.service: name=mysql-server
      tags:
        - never
        - serverStart
```

PLAYBOOK TEST RUN

- A dry run can be carried out to check a playbook without execution

- No changes are made to the hosts with the --check parameter

Use `ansible-playbook site.yml --tags install --check`

CONDITIONALS

- `when` to attach conditions to tasks

```
---
- hosts: all
  tasks:
    - name: Install Apache (Debian)
      ansible.builtin.apt: name=apache2
      when: ansible_os_family == 'Debian'

    - name: Install Apache (RedHat)
      ansible.builtin.yum: name=httpd
      when: ansible_os_family == 'RedHat'
```

Example:
Different handling
of distributions

HANDLERS

- Handlers are tasks that are to be executed once, when changes have been made
- Features
  - Are executed **once** after a block of tasks
  - Condition: At least one task with changes notifies the handler
- Use cases
  - Restarting a service after config files have been updated

HANDLERS: EXAMPLE

```yaml
---
- hosts: webservers
  tasks:
    - name: Enable Apache rewrite module.
      apache2_module: name=rewrite state=present
      notify: Restart apache

  handlers:
    - name: Restart apache
      ansible.builtin.service:
        name: apache2
        state: restarted
```

ERROR HANDLING IN ANSIBLE

- Standard: Abort at first faulty task
- `failed_when:` Change termination criterion of a task
- `ignore_errors:` do not cancel in the event of an error

```yaml
---
- hosts: localhost
  tasks:
    - name: run non existing command
      command: cat idontexist
      ignore_errors: true
    - debug:
        msg: "After failing task"
```

TRY / CATCH / FINALLY

- block: Only block is exited in the event of an error
- rescue: Executed in case of error
- always: Always executed

```
---
- hosts: localhost
  tasks:
    - name: Do with error handling
      block:
        - name: Run risky script
          command: "/usr/bin/myscript.sh"
      rescue:
        - name: Run less risky script
          command: "/usr/bin/fallback.sh"
      always:
        - debug: msg="Keep smiling!"
```

HEALTH-CHECK / READY-CHECK

- Health-Check / Ready-Check
  - Module `wait_for`:
    - Port is available or drained
    - File is available, file contains text (e.g. success message in the log)
  - Module `uri`:
    - Accessibility of an HTTP endpoint

```yaml
- name: trying to reach from master
  delegate_to: localhost
  uri:
    url: "http://{{ inventory_hostname }}/internal/health"
  retries: 10
  delay: 2
  register: response
  until: "response.status == 200"
```

LOOPS

- With loops, tasks can be called up several times with different parameters

```
---
- hosts: all
  tasks:
    - name: Install NodeJs and NPM
      ansible.builtin.apt: name="{{ item }}"
      loop:
        - nodejs
        - npm
    - name: Install required NPM modules
      ansible.builtin.npm:
        name: "{{ item.name }}"
        version: "{{ item.version }}"
      loop:
        - { name: 'grunt', version: '0.4.5' }
        - { name: 'bower', version: '1.4.1' }
        - { name: 'angular', version: '1.3.15' }
```