Templating using Jinja2

Ansible Advanced

Agenda

- What are templates?
- Template module
- Template file
- Filters

What are templates?

Ansible Advanced

What are templates?

- Templates are files that contain placeholders, which get replaced with actual values when the template is rendered
- This allows for dynamic generation of files based on variables
- In the context of Ansible, templates are used to generate host-specific files dynamically
- Instead of manually creating individual files for each host or group, you can have a single template, and Ansible will generate the required files for each host based on that template.

Jinja2 Template file

- Expressive Syntax
  - Jinja2 offers a clear and expressive syntax that allows for both simple variable substitutions and more complex operations like loops and conditionals.
- Widely Adopted
  - Jinja2 is a popular templating engine in the Python ecosystem
  - It's used by many projects, including Flask and Django, which means there's a large community and a lot of resources for learning and troubleshooting
- Extensible
  - Jinja2 is extensible, allowing developers to add custom filters, tests, and extensions

Jinja2 Template file

- Secure
  - Jinja2 templates are sandboxed, meaning they run in a restricted environment
  - This helps prevent the execution of arbitrary code
- Integrated with Ansible
  - Since Ansible is written in Python, the integration with Jinja2 is seamless
  - This allows variables, facts, and other data from Ansible to be easily used within templates
- Filters
  - Jinja2 provides a wide range of filters that can be used to modify variables
  - Ansible also adds its own set of filters, further enhancing the templating capabilities

Use Cases for Templates

- Web Server Configuration
- Application Environment Files
- Dynamic Firewall Rules
- User SSH Configuration
- Database Replication Configuration
- …

Template module

Ansible Advanced

How to use templates in Ansible

- Together your playbooks, you may create a **`templates`** folder
- Create template files using Jinja2 templating placeholders
- Add extension **`.j2`** to your file
- Then you use **`template`** module to generate the file

Template Module

- Source and Destination
  - The template module primarily requires two parameters: **`src`** (the source template file) and **`dest`** (the destination path on the target host where the rendered file should be placed).
- Jinja2 Templating
  - The module uses the Jinja2 templating engine to process the source template
  - This means you can use Jinja2 syntax in your templates for variable substitution, loops, conditionals, filters, and more
- Variable Replacement
  - During the templating process, any variables, expressions, or placeholders within the template are replaced with their corresponding values from Ansible's context (e.g., playbook variables, host variables, group variables, facts).

Template Module Parameters

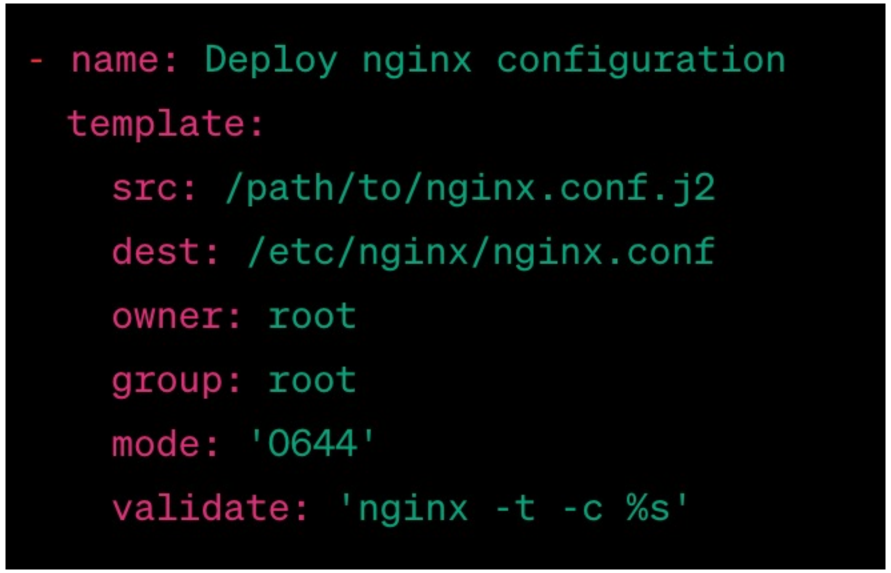
- `src`: The path to the source template file on the control machine. This file should contain the content you want to deploy, with placeholders for dynamic content.
- `dest`: The path on the target host where the rendered file should be placed.
- `mode`: (Optional) The permissions to set on the destination file. For example, 0644.
- `owner` and `group`: (Optional) The name of the user and group that should own the file, respectively.
- `backup`: (Optional) If set to yes, a backup of the destination file will be created if it already exists and is different from the rendered file.
- `validate`: (Optional) A validation command to run before copying the file to the destination. This is useful for configuration files to ensure they are syntactically correct.

How it works

- Read the Template
  - Ansible reads the source template file specified in the src parameter
- Render the Template
  - Using the Jinja2 engine, Ansible processes the template, replacing any Jinja2 expressions with their corresponding values from the available variables
- Copy to Destination
  - The rendered file is then copied to the target host at the specified dest location
  - If the destination file already exists and its content differs from the rendered file, the module will replace it (and optionally back it up if the backup parameter is set)
- Set Permissions
  - If specified, the module will set the file permissions, owner, and group as per the mode, owner, and group parameters
- Validation
  - If the validate parameter is provided, Ansible will run the validation command on the rendered file before copying it to the destination. If validation fails, the task will fail.

Example

- The **`template`** module will take the **`nginx.conf.j2`** template, render it, validate the rendered configuration using **`nginx -t`**, and then deploy it to the target host's **`/etc/nginx/nginx.conf`** path



```yaml
- name: Deploy nginx configuration
  template:
    src: /path/to/nginx.conf.j2
    dest: /etc/nginx/nginx.conf
    owner: root
    group: root
    mode: '0644'
    validate: 'nginx -t -c %s'
```

Template file

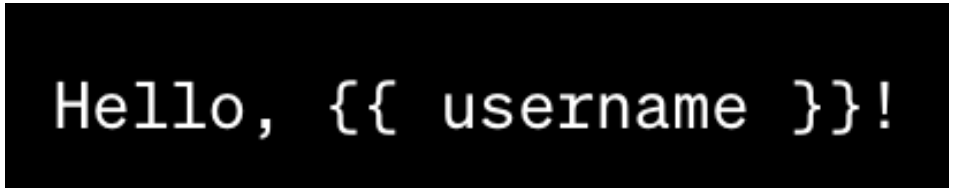
Ansible Advanced

Jinja2 Template Engine

- Jinja2 is a modern and designer-friendly templating engine for Python programming
- It's used by Ansible to transform data inside a template expression and expose it to a template
- Exist 3 different types of template expressions
  - **`{% ... %}`** for Statements
  - **`{{ ... }}`** for Expressions to print to the template output
  - **`{# ... #}`** for Comments not included in the template output

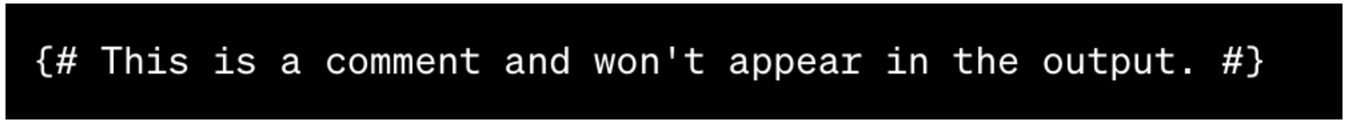
Basic Syntax

- Variables: Use **`{{ variable_name }}`** to print the value of a variable.



```
Hello, {{ username }}!
```

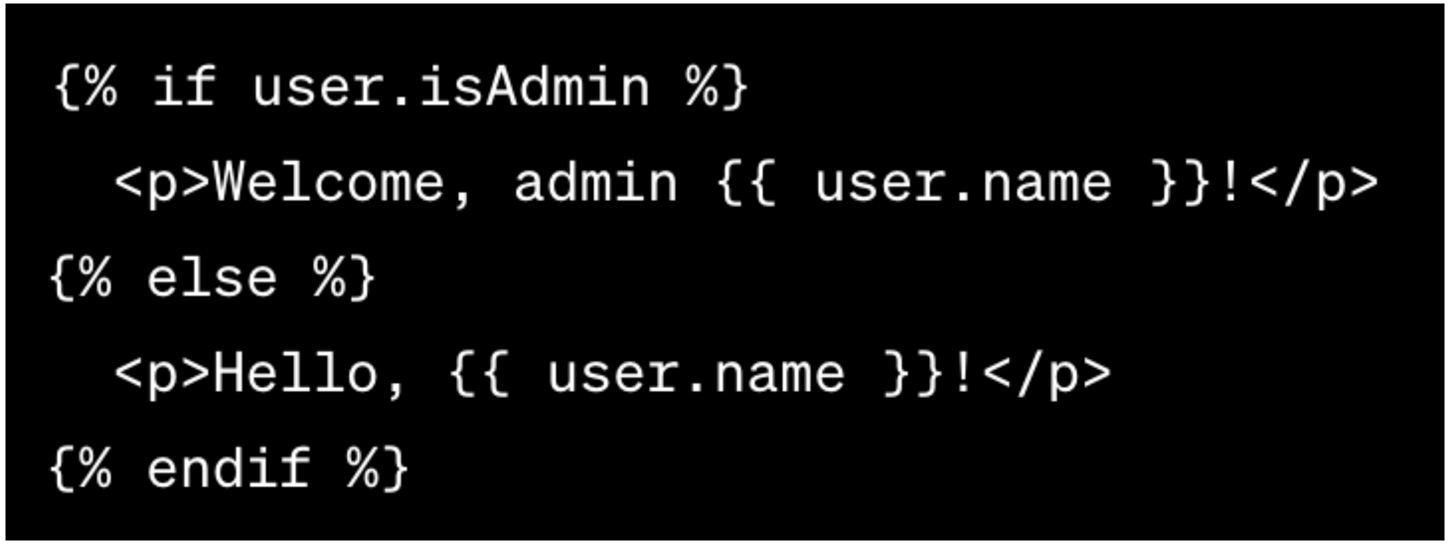
- Comments: Anything between **`{# and #}`** is a comment and will not be output in the final render.



```
{# This is a comment and won't appear in the output. #}
```

Control Structures

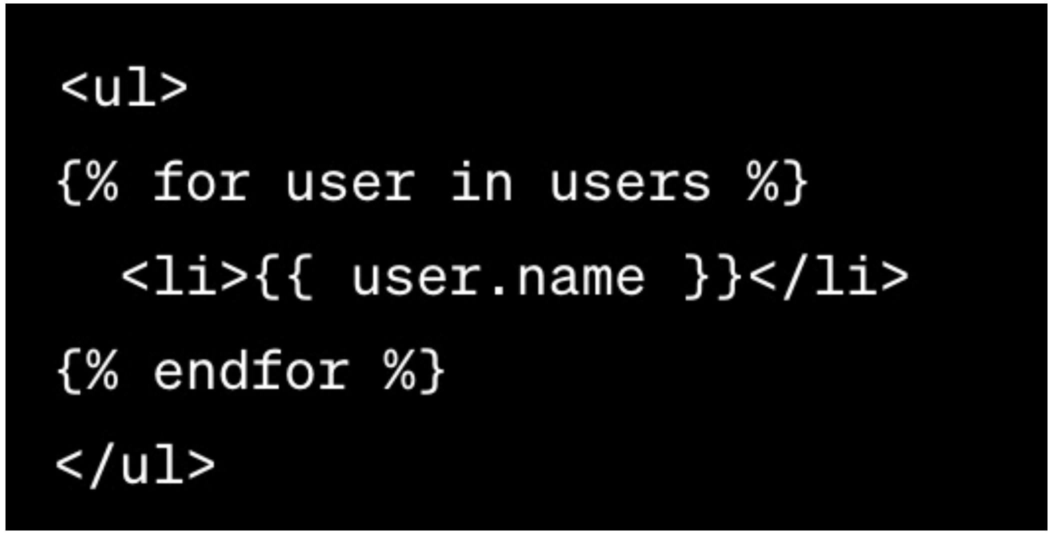
- Use **`{% if %}`**, **`{% elif %}`**, and **`{% else %}`** to control the flow based on conditions.



```
{% if user.isAdmin %}
  <p>Welcome, admin {{ user.name }}!</p>
{% else %}
  <p>Hello, {{ user.name }}!</p>
{% endif %}
```

Loops

- Use **`{% for %}`** to iterate over sequences.



```
<ul>
{% for user in users %}
  <li>{{ user.name }}</li>
{% endfor %}
</ul>
```

Using On Ansible

- Host Variables & Facts: Access host-specific data in your templates
- Playbook Variables: Access variables defined on playbooks
- Group Variables: Access variables defined for specific inventory groups.

Filters

Ansible Advanced

Filters

- Filters in Ansible templates are a way to transform data inside a template expression
- They're used to modify or format variables before they're rendered in the template
- Filters are applied to variables using the **|** character, followed by the filter name and any arguments

Filters

- **`default`**: Provides a default value for a variable if it's undefined or empty.

```
{{ username | default('Guest') }}
```

- **`upper`** and **`lower`**: Converts a string to uppercase or lowercase

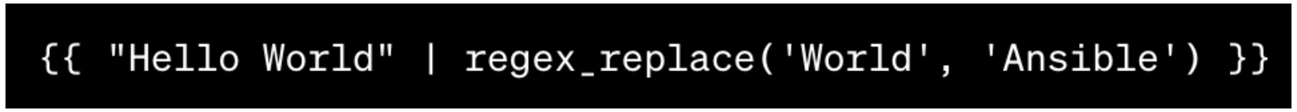
```
{{ "hello" | upper }}  {# Outputs "HELLO" #}
{{ "HELLO" | lower }}  {# Outputs "hello" #}
```

- **`length`**: Returns the length of a string, list, or dictionary

```
{{ "hello" | length }}  {# Outputs "5" #}
```

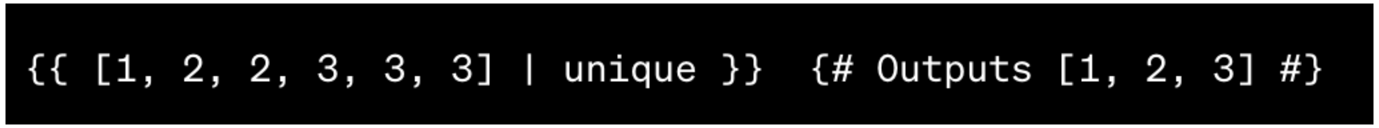
Filters

- **`regex_replace`:** Replaces occurrences in a string that match a regular expression.



```
{{ "Hello World" | regex_replace('World', 'Ansible') }}
```

- **`unique`**: Use Case: Returns a list with duplicate items removed.



```
{{ [1, 2, 2, 3, 3, 3] | unique }}  {# Outputs [1, 2, 3] #}
```

Builtin Filter

- Full list: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/index.html#filter-plugins

- Additional filters can be added using plugins

Demo: Templates

Ansible Advanced