Ansible on Windows Fundamentals

Controlling Task Execution

Using Variables in Plays

Red Hat

Objectives

- Explain the key places where variables are commonly set.
- Explain the basic rules of variable precedence.
- Create and run a playbook that uses variables.

Red Hat

Introduction to Ansible Variables

- Ansible supports variables that you can use to store values for reuse throughout an Ansible project.
- This simplifies the creation and maintenance of a project and reduce the number of errors.
- Variables provide a convenient way to manage dynamic values.
- Examples of values that variables might contain:
  - Users to create, modify or delete.
  - Software to install or uninstall.
  - Services to stop, start, or restart.
  - Files to create, modify, or remove.
  - Archives to retrieve from the Internet, or to extract.

Red Hat

Naming Variables

- Variable names must start with a letter, and they can only contain letters, numbers, and underscores.

Invalid variable names	Valid Variable names
web server web-server	web_server
remote.file	remote_file
1st file 1st_file	file_1 file1
remoteserver$1	remote_server_1 remote_server1

Red Hat

Variable Scope

- **Global**
  - The value is set for all hosts.
  - Example: extra variables you set in the job template
- **Host**
  - The value is set for a particular host (or group).
  - Examples: variables set for a host in the inventory or a host_vars directory, gathered facts
- **Play**
  - The value is set for all hosts in the context of the current play.
  - Examples: **vars** directives in a play, **include_vars** tasks, and so on

Red Hat

Defining Variables

- If a variable is defined at more than one level, the level with the highest precedence wins.
- A narrow scope generally takes precedence over a wider scope.
- Variables that you define in an inventory are overridden by variables that you define in the playbook.
- Variables defined in a playbook are overridden by “extra variables” defined by the job template.*

Details on exact variable precedence are available at https://docs.ansible.com/ansible/latest/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable

Red Hat

Managing Variables in Playbooks

- Variables can be defined in multiple ways.
- One common method is to place a variable in a **`vars`** block at the beginning of a play:

```
- hosts: all
  vars:
    user_name: joe
    user_state: present
```

- It is also possible to define play variables in external files.
- Use **vars_files** at the start of the play to load variables from a list of files into the play:

```
- hosts: all
  vars_files:
    - vars/users.yml
```


Red Hat

Referencing Variables in Playbooks

- After declaring variables, you can use them in tasks.
- Reference a variable (replace it with its value) by placing the variable name in double braces: **{{ variable_name }}**
- Ansible substitutes the variable with its value when it runs the task.

```
- name: Example play
  hosts: all
  vars:
    user_name: joe

  tasks:
    # This line will read: Creates the user joe
    - name: Creates the user {{ user_name }}
      win_user:
        # This line will create the user named Joe
        name: "{{ user_name }}"
```


Red Hat

Referencing Variables

- When you reference one variable as another variable's value, and the curly braces start the value, you must use quotes around the value.
- This prevents Ansible for interpreting the variable reference as starting a YAML dictionary.
- The following message appears if the quotes are missing:

```
win_service:
     name: {{ service }}
            ^ here
We could be wrong, but this one looks like it might be an issue with
missing quotes.  Always quote template expression brackets when they
start a value. For instance:

    with_items:
      - {{ foo }}

Should be written as:

    with_items:
      - "{{ foo }}"
```


Red Hat

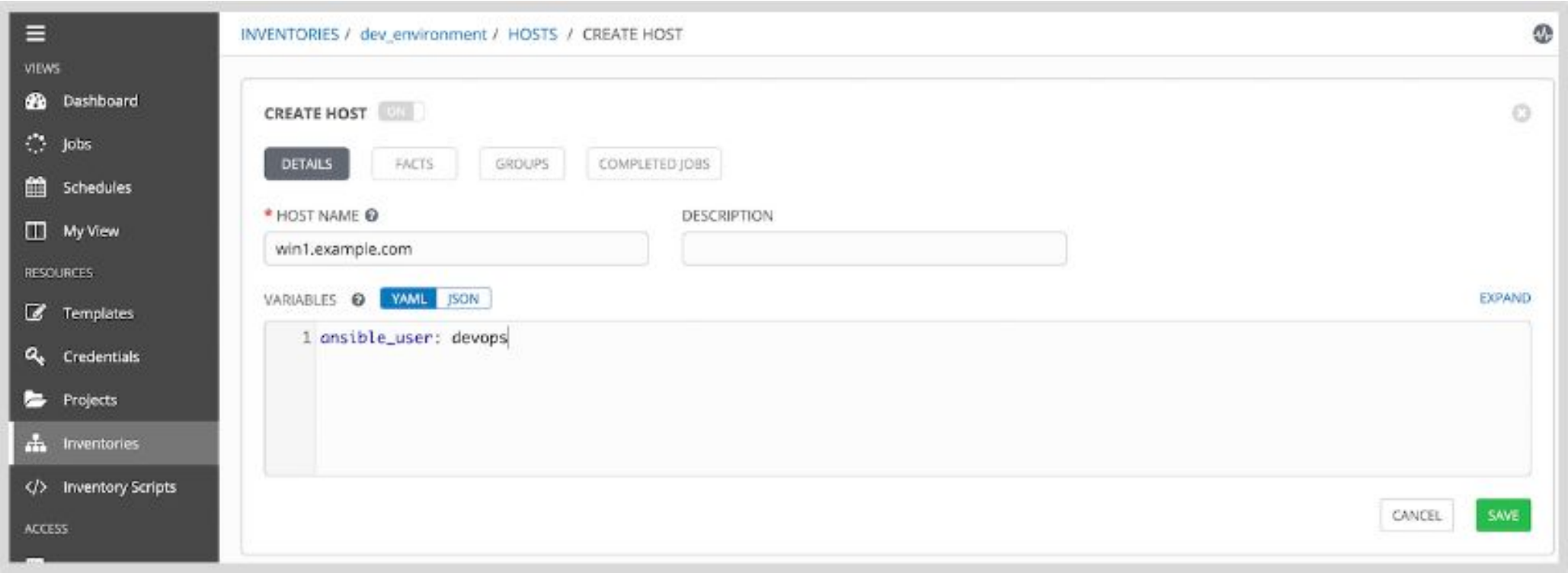
Host Variables and Group Variables

- *Host variables* apply to a specific host.
- *Group variables* apply to all hosts in a host group or in a group of host groups.
- Host variables take precedence over group variables, but variables defined inside a play take precedence over both.
- Host variables and group variables can be defined in the inventory.
- They can also be defined in your Git repository in special directories with your playbook.

Red Hat

Describing Host Variables and Group Variables

- Inventory setting **ansible_user** variable to **devops** for the **win1.example.com** host:



Using Directories to Set Host and Group Variables

- You can set variables for hosts and groups in your Git repository
- In the same directory as your playbook, create two directories, **group_vars** and **host_vars**.
  - To define group variables for the **servers** group, you would create a file named **group_vars\servers**.
  - In that file, set variables to values, using YAML syntax. (The example at right sets ansible_user to a string and newfiles to a list of values.)
- To define host variables for a particular host, create a file with a name matching the host in the **host_vars** directory.

```
ansible_user: devops
newfiles:
  - C:\Temp\a.conf
  - C:\Temp\b.conf
```


Red Hat

Using Directories to Set Host and Group Variables

```
project
├── group_vars
│   ├── all
│   ├── datacenters
│   ├── datacenters1
│   └── datacenters2
├── host_vars
│   ├── demo1.example.com
│   ├── demo2.example.com
│   ├── demo3.example.com
│   └── demo4.example.com
└── playbook.yml
```

- Set host and group variables in **host_vars** and **group_vars** directories in the same place in Git as your project's playbook
- Works just like the host or group variables in your inventory
- They have host scope just like inventory variables
- These "playbook" host and group variables have slightly higher precedence than inventory variables (and override them)
- The files or directories in **host_vars** and **group_vars** have the name of the host or group they apply to
- If you use a directory for the host or group name, it can contain multiple variable files which are all used

Red Hat

Getting Host Information from Facts

- A **fact** is a read-only host-specific variable that contains information about the host itself
- Facts are often gathered automatically when a play starts
- The ansible_facts variable stores the facts as a dictionary of key-value pairs
- For example, ansible_facts["fqdn"] returns the full DNS name of the current host being processed.

Red Hat

Displaying All Variable Values

- You can use the **debug** module to display the value of a variable
- You can use it to display all host-specific variables and facts
- You can also use it to display all variables for all hosts in the current play

```
- name: display DNS hostname
  debug:
    var: ansible_facts["fqdn"]

- name: display all host-specific variables
  debug:
    var: hostvars["inventory_hostname"]

- name: display all variables
  debug:
    var: vars
```


Red Hat

Task Iteration with Loops

Red Hat

Objective

- Review the implementation of loops in Ansible tasks.

Red Hat

Task Iteration with Loops

- Using loops can make it easier to write a sequence of similar tasks that use the same module.
- Instead of writing five tasks to create five users, write one task that iterates over a list of five users.
- Ansible supports iterating a task over a set of items using the **loop** keyword.

Red Hat

Introducing Simple Loops

- Consider the following examples that invoke the **win_chocolatey** module three times to install a set of software packages using Chocolatey. The example on the right uses a loop to do the same thing as the example on the left.

```
- name: Install JRE 8
  win_chocolatey:
    name: jre8
    state: present

- name: Install Java Runtime
  win_chocolatey:
    name: javaruntime-preventasktoolbar
    state: present

- name: Install Dropbox
  win_chocolatey:
    name: dropbox
    state: present
```

```
- name: Install packages
  win_chocolatey:
    name: "{{ item }}"
    state: present
  loop:
    - jre8
    - javaruntime-preventasktoolbar
    - dropbox
```


Red Hat

Introducing Simple Loops

- It is possible to define a variable for the list and pass it to the loop.
- This can make it easier to change the list by defining it in a variable at the start of the play.
- This is also useful with host variables that might differ from server to server.

```
vars:
  packages:
    - jre8
    - javaruntime-preventasktoolbar
    - dropbox
tasks:
  - name: Installing Java and Dropbox
    win_chocolatey:
      name: "{{ item }}"
      state: present
    loop: "{{ packages }}"
```


Red Hat

Loops and Efficiency

- For some modules, using a loop is not the most efficient solution.
  - For example, **win_feature** can take a list of Windows Roles or Features that should be installed
  - If you pass it a list, the task runs once to install all the features (often more efficient)
  - If you loop over the list, the task runs one time for each feature being installed
- Check the documentation for the module you are using to see if you need to use a loop

```
vars:
  features:
    - Web-Server
    - Web-Common-Http
tasks:
  - name: install IIS
    win_feature:
      name: "{{ item }}"
      state: present
    loop: "{{ features }}"
```

```
vars:
  features:
    - Web-Server
    - Web-Common-Http
tasks:
  - name: install IIS
    win_feature:
      name: "{{ features }}"
      state: present
```


Red Hat

Running Conditional Tasks and Handlers

Red Hat

Objective

- Review the implementation of conditionals and handlers in Ansible tasks.

Red Hat

Running Tasks Conditionally

- Ansible can use conditionals to run or skip tasks when certain conditions are met.
- Variables and facts can both be tested by conditionals.
- Operators such as greater than (>) or less than (<) to compare strings, numerical data, or Boolean values can be used.

Red Hat

Some Uses for Conditional Tasks

- Run a task if a fact reporting the available memory on a managed host is lower than a value.
- Run different tasks to create users on a managed host based on which domain it belongs to
- Skip a task if a certain variable is not set or is set to a specific value
- Use the results of a previous task to determine whether to run the task

Red Hat

Writing Conditional Tasks

- This example uses the **when** statement to run a task conditionally.
- One simple test is to check if a Boolean variable is **true** or **false**.

- The first task will run only if the variable **run_my_task** is **true**. It also uses the **register** keyword to store result information for the task in a variable.
- The second task only reboots the system if a variable set by the first task, **feature_output.reboot_required**, is **true**.

```
- name: Simple Boolean Task Demo
  hosts: winhost1
  vars:
    run_my_task: true
  tasks:
  - name: Install IIS Web-Server with sub features and management tools
    win_feature:
      name: Web-Server
      state: present
      include_sub_features: yes
      include_management_tools: yes
    when: run_my_task
    register: feature_output

- name: Reboot if installing Web-Server feature requires it
  win_reboot:
  when: feature_output.reboot_required
```


Red Hat

Writing Conditional Tasks

- This example play installs whatever feature is listed in the **my_service** variable.
- The play assumes that **my_service** will be set somewhere else, perhaps in a host variable or extra variable.
- If **my_service** is not defined, the task will be skipped.
- This helps avoid having the play break because of a syntax error if the variable is not set.

```
- name: Service installation check
  hosts: winhost1
  tasks:
    - name: Install "{{ my_service }}"
      win_feature:
        name: "{{ my_service }}"
        state: present
        include_sub_features: yes
        include_management_tools: yes
      when: my_service is defined
```

Constructing Conditions with Ansible Operators

Operation	Example
Is equal to (with a string)	ansible_facts[‘architecture’] == “64-bit”
Is equal to (with a numerical value)	max_memory == 1024
Is greater than	ansible_facts[‘powershell_version’] > 2
Is less than	ansible_facts[‘powershell_version’] < 6
Is greater than or equal to	ansible_facts[‘processor_vcpus’] >=2
Is less than or equal to	ansible_facts[‘processor_vcpus’] <= 20

Red Hat

Constructing Conditions with Ansible Operators

Operation	Example
Is not equal to	ansible_facts['memtotal_mb'] != 4000
Variable exists	my_service is defined
Variable does not exist	my_service is not defined
A Boolean variable on its own is implicitly a test for whether it is true. The values of 1, True, or yes are evaluated as true. The values of 0, False, or no are evaluated as false.	reboot_required
Boolean variable is false	not reboot_required
First variable's value is present as a value in second variable's list.	ansible_facts['distribution'] in supported_distros

Red Hat

Constructing Conditions with Ansible Operators

- The **ansible_facts['distribution']** variable is a fact set when the play runs, which identifies the operating system of the current managed host.
- The **supported_os** variable contains a list of operating systems supported by the playbook.
- If the value of **ansible_facts['distribution']** is in the **supported_os** list, the conditional passes and the task runs.

```
- name: Testing a condition
  hosts: winhost1
  vars:
    my_service: Web-Server
    supported_os:
      - "Microsoft Windows Server 2016 Datacenter"
      - "Microsoft Windows Server 2016 Core"
      - "Microsoft Windows Server 2012 Datacenter"
      - "Microsoft Windows Server 2012 Core"
  tasks:
    - name: Install "{{ my_service }}"
        name: "{{ my_service }}"
        state: present
        include_sub_features: yes
        include_management_tools: yes
      when: ansible_facts['distribution'] in supported_os
```

Triggering Tasks with Handlers

- *Handlers* are special tasks that run at the end of a play
- Each handler has a unique name
- If a task changes a managed host, it can use a **notify** statement to run a handler on that host
- If multiple tasks notify a handler, it still only runs once
- For example, a handler might be used to reboot a system when multiple tasks in the play might each make a change that needs a reboot to take effect, but the reboot can wait until the play completes

Red Hat

Triggering Tasks with Handlers

- The following example shows how the **reboot server** handler runs when Ansible installs a new package in a task.

```
- name: Testing Handlers
  hosts: winhost1
  tasks:
    - name: Install Dotnet4.6.1 ❶
      win_chocolatey:
        name: dotnet4.6.1
        state: present
      notify: ❷
        - reboot server ❸

  handlers: ❹
    - name: reboot server ❺
      win_reboot: ❻
```

❶ The task that notifies the handler.
❷ The **notify** statement indicates the task needs to trigger a handler.
❸ The name of the handler to run.
❹ The **handlers** keyword indicates the start of the list of handler tasks.
❺ The name of the handler that the task invokes.
❻ The module to use for the handler.

Red Hat