

vCenter - Linux Templates

To deploy multiple VMs with different hostnames and IP addresses while utilizing the customization capabilities provided by the `vmware_guest` module in Ansible, you can use VMware's customization specifications. This approach allows for more advanced customization options, such as setting the domain, hostname, and network settings directly within the playbook. Below is an example of how to modify the playbook to use VMware's customization feature for deploying 3 VMs with distinct configurations:

Inventory

To create a separate inventory file with all the variables used in the provided playbook, you'll need to organize these variables in a structured way. Ansible inventory files can be in INI or YAML format, but for complex configurations like this, YAML is more suitable due to its support for hierarchical data.

Below is an example of how to create an Ansible inventory file in YAML format (`inventory.yml`) that defines all the variables required by your playbook. This example demonstrates setting up variables for deploying three VMs, but you can adjust the quantities and details as needed:

```
all:
  vars:
    vcenter_hostname: vcenter.example.com
    vcenter_username: admin@vsphere.local
    vcenter_password: securepassword
    vcenter_datacenter: DC1
    vcenter_folder: /DC1/vm/ansible_managed_vms
    vcenter_cluster: Cluster1
    vm_template: CentOS_Template
    vm_network: VM_Network
    vm_netmask: 255.255.255.0
    vm_gateway: 192.168.1.1
    dns01: 8.8.8.8
    dns02: 8.8.4.4
  hosts:
    vm01:
      vm_name: vm01
```

vm_ip: 192.168.1.101
vm_ram: 2048
vm_cores: 2
vm_sockets: 1
vm_notes: "VM01 Notes"
vm_department: "department1"
vm_application: "Application1"
vm_role: "Role1"
vm_env: "Development"
vm_buildcode: "Build01"
vm_lifecycle: "Lifecycle1"
vm_contact: "Contact1"

vm02:

vm_name: vm02
vm_ip: 192.168.1.102
vm_ram: 4096
vm_cores: 4
vm_sockets: 2
vm_notes: "VM02 Notes"
vm_department: "department2"
vm_application: "Application2"
vm_role: "Role2"
vm_env: "Testing"
vm_buildcode: "Build02"
vm_lifecycle: "Lifecycle2"
vm_contact: "Contact2"

vm03:

vm_name: vm03
vm_ip: 192.168.1.103
vm_ram: 8192
vm_cores: 4
vm_sockets: 2
vm_notes: "VM03 Notes"
vm_department: "department3"
vm_application: "Application3"
vm_role: "Role3"
vm_env: "Production"

```
vm_buildcode: "Build03"
vm_lifecycle: "Lifecycle3"
vm_contact: "Contact3"
```

Adjusting the Inventory

- **Hosts and Variables:** The example above assumes you are deploying three VMs (`vm01`, `vm02`, and `vm03`). Each VM has its set of variables defined under `hosts`. You can add more VMs or adjust the existing definitions as needed.
- **Global Variables:** Variables that are common across all VMs are defined under `all: vars`. This includes vCenter connection details, network configuration, and Infoblox provider details. These can be overridden at the host level if necessary.
- **Customization:** Tailor the inventory to match your environment's specifics, including vCenter details, template names, network settings, and VM specifications.

This approach allows you to manage your infrastructure as code, making deployments repeatable and reducing the likelihood of human error.

Playbook: `deploy_vms.yml`

```
---
- name: Deploy Multiple VMs on vCenter
  hosts: all
  gather_facts: false

  tasks:
    - name: Setting Facts
      set_fact:
        vm_guest_name: "{{ vm_name | upper }}"
        vm_hostname: "{{ vm_name | lower }}"

    - name: Deploy or Clone Linux VM
      vmware_guest:
        hostname: "{{ vcenter_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: no
        datacenter: "{{ vcenter_datacenter }}"
        folder: "{{ vcenter_folder }}"
```

```
name: "{{ vm_guest_name }}"
cluster: "{{ vcenter_cluster }}"
state: poweredon
template: "{{ vm_template }}"
annotation: "{{ vm_notes }}"
hardware:
  memory_mb: "{{ vm_ram }}"
  num_cpus: "{{ vm_cores }}"
  num_cpu_cores_per_socket: "{{ vm_sockets }}"
networks:
  - name: "{{ vm_network }}"
    ip: "{{ vm_ip }}"
    netmask: "{{ vm_netmask }}"
    gateway: "{{ vm_gateway }}"
wait_for_ip_address: yes
wait_for_customization: yes
cdrom:
  type: none
customization:
  hostname: "{{ vm_hostname }}"
  domain: "example.com"
  timezone: "America/New_York"
  dns_servers:
    - "{{ dns01 }}"
    - "{{ dns02 }}"
delegate_to: localhost
register: vmcreate
```

- name: Add Custom Attributes to the VM

```
vmware_guest_custom_attributes:
  hostname: "{{ vcenter_hostname }}"
  username: "{{ vcenter_username }}"
  password: "{{ vcenter_password }}"
  validate_certs: no
  name: "{{ vm_guest_name }}"
  attributes:
    - name: Department
      value: "{{ vm_department | default('') }}"
```

```
- name: Application
  value: "{{ vm_application | default('') }}"

- name: Role
  value: "{{ vm_role | default('') }}"

- name: Environment
  value: "{{ vm_env | default('') }}"

- name: Automation
  value: "Baseline"

- name: buildcode
  value: "{{ vm_buildcode | default('') }}"

- name: lifecycle
  value: "{{ vm_lifecycle | default('') }}"

- name: Contact
  value: "{{ vm_contact | default('') }}"
```

Explanation of Each Task

1. **Setting Facts:** Converts the VM name to uppercase and lowercase versions for different uses, such as the display name in vCenter (`vm_guest_name`) and the internal hostname of the VM (`vm_hostname`).
2. **Deploy or Clone Linux VM:** Uses the `vmware_guest` module to either deploy a new VM or clone an existing one from a template specified in the inventory. This task includes configuring the VM's hardware specifications, network settings, and customization specifications like the hostname and DNS settings. It waits for the IP address to be assigned and customization to complete before proceeding.
3. **Add Custom Attributes to the VM:** Adds custom attributes to the newly created VM in vCenter. These attributes can include metadata such as the department, application, role, and environment the VM is associated with. This helps in organizing and managing VMs based on these metadata.

Running the Playbook

To run this playbook, use the following command, ensuring you specify the inventory file:

```
ansible-playbook -i inventory.yml deploy_vms.yml
```

This command tells Ansible to deploy VMs as configured in `inventory.yml`, applying the settings and customizations specified for each VM.

Notes:

- **Template Requirements:** The template you use must be prepared for customization. For Linux VMs, ensure VMware Tools is installed, and the Perl scripting language is available for the customization scripts to run.
- **Customization Script:** VMware's customization mechanism uses a script that runs on the first boot. If the customization does not apply, troubleshooting may involve checking that VMware Tools is correctly installed and that the template is properly prepared for cloning and customization.
- **Ansible and VMware Versions:** Ensure you are using recent versions of Ansible and the VMware modules, as improvements and bug fixes are regularly added.

This method leverages VMware's powerful customization engine, allowing for a wide range of customization options beyond what was demonstrated here.

Revision #5

Created 1 February 2024 14:31:06 by Michael Cleary

Updated 6 February 2024 13:25:52 by Michael Cleary