

Inventories

Ansible Fundamentals

Agenda

- Inventory Basics
- Adding Vars to Hosts
- Dynamic Inventories

Inventory Basics

Inventory

What is inventory?

- Inventory is a list of machines that can be used on your ansible commands
- Inside inventory you must define groups that allow you to identify your servers on a easy way
- Inventory may be referenced in different ways following this precedence list
 - Command Line **-i** Option
 - **ANSIBLE_INVENTORY** Environment Variable
 - **ansible.cfg** Configuration File:
 - Default Inventory Path: **/etc/ansible/hosts**
- Some best practices: [Ansible Best Practices](#)

Inventory Format

- You may use two formats: INI and YAML

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

INI

```
ungrouped:
  hosts:
    mail.example.com:
webservers:
  hosts:
    foo.example.com:
    bar.example.com:
dbservers:
  hosts:
    one.example.com:
    two.example.com:
    three.example.com:
```

YAML

Default groups

- Even if you do not define any groups in your inventory file, Ansible creates two default groups: **all** and **ungrouped**
- The **all** group contains every host
- The **ungrouped** group contains all hosts that don't have another group aside from **all**
- Every host will always belong to at least 2 groups (all and ungrouped or all and some other group)

Hosts in multiple groups

- You can (and probably will) put each host in more than one group.
- For example a production webserver in a datacenter in Atlanta might be included in groups called **[prod]** and **[atlanta]** and **[webservers]**.
- You can create groups that track:
 - **What** - An application, stack or microservice (for example, database servers, web servers, and so on).
 - **Where** - A datacenter or region, to talk to local DNS, storage, and so on (for example, east, west).
 - **When** - The development stage, to avoid testing on production resources (for example, prod, test).

Hosts in multiple groups

```
ungrouped:
  hosts:
    mail.example.com:
webservers:
  hosts:
    foo.example.com:
    bar.example.com:
dbservers:
  hosts:
    one.example.com:
    two.example.com:
    three.example.com:
east:
  hosts:
    foo.example.com:
    one.example.com:
    two.example.com:
west:
  hosts:
    bar.example.com:
    three.example.com:
prod:
  hosts:
    foo.example.com:
    one.example.com:
    two.example.com:
test:
  hosts:
    bar.example.com:
    three.example.com:
```


Parent/child group relationships

- You can create parent/child relationships among groups
- Parent groups are also known as nested groups or groups of groups
- For example, if all your production hosts are already in groups such as **atlanta_prod** and **denver_prod**, you can create a production group that includes those smaller groups.
- This approach reduces maintenance because you can add or remove hosts from the parent group by editing the child groups.
- Child groups have a couple of properties to note:
 - Any host that is member of a child group is automatically a member of the parent group.
 - Groups can have multiple parents and children, but not circular relationships.
 - Hosts can also be in multiple groups, but there will only be one instance of a host at runtime. Ansible merges the data from the multiple groups.

Parent/child group relationships

```
ungrouped:
  hosts:
    mail.example.com:
webservers:
  hosts:
    foo.example.com:
    bar.example.com:
dbservers:
  hosts:
    one.example.com:
    two.example.com:
    three.example.com:
east:
  hosts:
    foo.example.com:
    one.example.com:
    two.example.com:
west:
  hosts:
    bar.example.com:
    three.example.com:
prod:
  children:
    east:
test:
  children:
    west:
```

Adding Ranges of Hosts

- If you have a lot of hosts with a similar pattern, you can add them as a range rather than listing each hostname separately:

```
[webservers]  
www[01:50].example.com
```

```
webservers:  
hosts:  
  www[01:50].example.com:
```

- You can specify a stride (increments between sequence numbers) when defining a numeric range of hosts:

```
[webservers]  
www[01:50:2].example.com
```

```
webservers:  
hosts:  
  www[01:50:2].example.com:
```

- You can also define alphabetic ranges:

```
[databases]  
db-[a:f].example.com
```

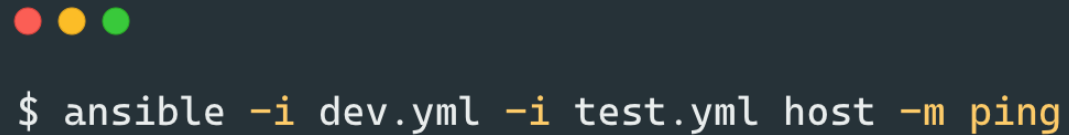
Organizing inventory in a directory

- You can consolidate multiple inventory sources in a single director
- Simplest version of this is a directory with multiple files instead of a single inventory file
- The following inventory directory combines an inventory plugin source, a dynamic inventory script, and a file with static hosts:

```
inventory/  
  openstack.yml      # configure inventory plugin to get hosts from OpenStack cloud  
  dynamic-inventory.py # add additional hosts with dynamic inventory script  
  on-prem            # add static hosts and groups  
  parent-groups      # add static hosts and groups
```


Passing multiple inventory sources

- You can target multiple inventory at the same time by giving multiple inventory parameters from the command line



```
$ ansible -i dev.yml -i test.yml host -m ping
```

- Or you can target this inventory directory as follows:



```
$ ansible -i inventory all -m ping
```

Adding Vars to Hosts

Inventory

Adding variables to inventory

- You can store variable values that relate to a specific host or group in inventory
- To start with, you may add variables directly to the hosts and groups in your main inventory file
- However, storing variables in separate host and group variable files is a more robust approach to describing your system policy
- Setting variables in the main inventory file is only a shorthand

Adding variables to inventory

- INI Format with key=value

```
[targets]

localhost          ansible_connection=local
other1.example.com ansible_connection=ssh      ansible_user=myuser
other2.example.com ansible_connection=ssh      ansible_user=myotheruser
```

- YAML Format

```
atlanta:
  hosts:
    host1:
      http_port: 80
      maxRequestsPerChild: 808
    host2:
      http_port: 303
      maxRequestsPerChild: 909
```


Inventory Aliases

- You can also define aliases in your inventory using host variables
- INI Format

```
jumper ansible_port=5555 ansible_host=192.0.2.50
```

- YAML Format

```
hosts:  
  jumper:  
    ansible_port: 5555  
    ansible_host: 192.0.2.50
```

Group Variables

- If all hosts in a group share a variable value, you can apply that variable to an entire group at once
- INI Format

```
[atlanta]
host1
host2

[atlanta:vars]
ntp_server=ntp.atlanta.example.com
proxy=proxy.atlanta.example.com
```

YAML Format

```
atlanta:
  hosts:
    host1:
    host2:
  vars:
    ntp_server: ntp.atlanta.example.com
    proxy: proxy.atlanta.example.com
```

Organizing Host and Group Variables

- **Directory Structure:** Host and group variables should be organized in separate directories named **host_vars** and **group_vars** respectively, located at the root level of your Ansible project.
- **File Naming:** Inside the **host_vars** directory, create a file for each host named after its hostname. Similarly, inside the **group_vars** directory, create a file for each group named after its group name.
- **YAML Format:** Variables inside these files should be defined in YAML format. Each variable is defined with a name followed by its value.
- **Variable Precedence:** Variables defined in **host_vars** have higher precedence. This means that if a variable is defined both in a host's file in **host_vars** and in a group file in **group_vars**, the value from the **host_vars** file will be used for that host.

Organizing Host and Group Variables: Example

- Let's say you have a group called **web_servers** and a host called **web01.example.com**.
- **group_vars/web_servers.yml**

```
nginx_version: 1.18.0  
document_root: /var/www/html
```

- **host_vars/web01.example.com.yml**

```
document_root: /var/www/web01
```

Organizing Host and Group Variables: Example

- In this example, all hosts in the **web_servers** group will have the **nginx_version** variable set to **1.18.0**.
- However, while the **document_root** for most hosts in the group will be **/var/www/html**, for the **web01.example.com** host, it will be **/var/www/web01** due to the higher precedence of the **host_vars** definition.

Best practices for Ansible project Folders

- On Ansible Best Practices you may find two approaches

```
production
staging

group_vars/
  group1.yml
  group2.yml
host_vars/
  hostname1.yml
  hostname2.yml
```

```
inventories/
  production/
    hosts
    group_vars/
      group1.yml
      group2.yml
    host_vars/
      hostname1.yml
      hostname2.yml

  staging/
    hosts
    group_vars/
      group1.yml
      group2.yml
    host_vars/
      stagehost1.yml
      stagehost2.yml
```

Review Complex Inventory

Demo

Dynamic Inventories

Inventories

Dynamic Inventories

- **Definition:** Dynamic inventories provide a way for Ansible to fetch a list of hosts from external sources at runtime, rather than relying on a statically defined inventory file.
- **Flexibility:** They are especially useful in cloud environments where infrastructure can be ephemeral, with hosts being created and destroyed dynamically.
- **Scripts & Plugins:** Dynamic inventories can be implemented using scripts (legacy method) or plugins (recommended). These scripts or plugins fetch host information and output it in a JSON format that Ansible understands.
- **Grouping:** Dynamic inventory plugins can automatically group hosts based on common attributes, such as tags in cloud environments.

Dynamic Inventories

- **Caching:** To improve performance, Ansible can cache the results of dynamic inventories, ensuring faster execution times.
- **Configuration:** Dynamic inventories often require configuration files where credentials, filters, and other settings specific to the external source are defined.
- **Multiple Sources:** You can use multiple dynamic inventory sources together, allowing for a mix of static and dynamic hosts.
- **Built-in Support:** Ansible has built-in support for many popular cloud providers and platforms for dynamic inventories, including AWS, Azure, VMware, and more.

Dynamic Inventories: VMWare

- Needs a Python plugin called **pyvmomi** (<https://github.com/vmware/pyvmomi>)
- Then you can create your inventory file using VMWare credentials and some host filtering

```
plugin: vmware_vm_inventory
hostname: your_vcenter_hostname
username: your_vcenter_username
password: your_vcenter_password
validate_certs: no

# Filtering VMs based on power state
host_filters:
  - runtime.powerState == "poweredOn"

# Fetching properties for auto-grouping
properties:
  - 'name'
  - 'config.uuid'
  - 'guest.guestId'

# Auto-grouping VMs based on guest OS
keyed_groups:
  - key: 'guest.guestId'
    prefix: 'os'
```

Static vs Dynamic Inventories

Type	Pros	Cons
Static	Simplicity Stability No Dependencies Fine-grained Control	Manual Updates Scalability Issues Duplication
Dynamic	Automation Scalability Flexibility Auto Grouping	Complexity Dependencies Potential Delays Security Concerns

Dynamic Inventory

Demo

