Ansible in Operation
Learning Goals

● Manage inventory
● Ansible ad-hoc commands
● Write & run Playbooks
● Understanding of variables
● Loops and conditions
Inventories

- A list of hosts, groups and aspects of hosts
- Can be dynamic or static
- Groups defined by brackets \[\] and by name
  - Describe systems
  - Decide what systems you are controlling at what times and for what purpose (roles)
  - Groups can be nested with :children
- Hosts can be in more than one group
  - server could be both a webserver and a dbserver.
  - variables will come from all of the groups they are a member of
Static Inventories

- Static inventory: simplest, most common form
  - localhost

- Add a group
  - [CentOS]
    - localhost

- Add host variables
  - Localhost ansible_ssh_host=127.0.0.1
    - [CentOS]
    - localhost
Dynamic Inventories

- Static inventories negate the environment of the cloud
- Can use almost data source to generate dynamic inventories
Lesson 1: Run an empty play

1. `git init`
2. Configure an inventory file
3. Create at least one group (by OS)
4. Start a playbook
5. Run the empty playbook against all hosts
Host selection

• Host selection can be done by including or excluding groups and single hosts

• Selection can be done by passing:

  • all / *
  • Groups names
  • Exclusion (all:!CentOS)
  • Intersection (webservers:&staging)
  • Regex
Executing - Tasks

• Ad-Hoc: commands which execute single tasks
• Tasks: leverage an Ansible module, which is executed on the target host
• Modules:
  • (Mostly) written in Python
  • Shipped via SSH to the target host
  • Return JSON, interpreted by Ansible for outcome
  • Removed once executed
Executing - Modules

- Modules are the "Batteries included" of Ansible
- Core modules provided by Ansible and "extras" by the community
- Well-documented
  - Web: [http://docs.ansible.com/ansible/modules_by_category.html](http://docs.ansible.com/ansible/modules_by_category.html)
  - CLI: `ansible-doc -l`
Hands-on session
Ansible ad-hoc commands
Ad-hoc actions

1. Check facts on all hosts
   1. Ansible all -i inventory -m setup
2. Copy a file
3. Install nginx and add a user
4. Clone a git repo to a path
5. Ensure that httpd is present and started
6. Background operations, with polling
Orchestration

- The true power of ansible comes from abstraction and orchestration, using *playbooks*
- Playbook is a set of ordered tasks, combined with selected targets
- Playbooks provide *ready-made* strategies for bringing (groups of) hosts to a desired state
Roles

- Roles provide a way to encapsulate and re-use code
- Instead of writing lots of tasks, your playbook can be more readable and understandable to someone else:

```
- hosts: dirac
  user: ansible
  sudo: true
  roles:
    - bootstrap
    - common
    - certificates
    - dirac
```

Roles are applied *in order*

Roles may have *dependencies*
Roles and filesystem structure

- Roles are usually placed in a "library" in a sub-directory.
- Each role has a standard structure.
- Roles can be scaffolded using `ansible-galaxy`.

```yaml
site.yml
roles/
  role1/
    files/
    templates/
    tasks/
    handlers/
    vars/
    meta/
```
Creating new roles with Galaxy

- A new role can be created using `ansible-galaxy init <rolename>`
- Ensure that you create the role in the "roles" directory, or you won't be able to simply call them by name in the playbooks.
- Ansible Galaxy creates all the files you need to get started, including a README and a meta file
- Roles can be shared and discovered via `http://galaxy.ansible.com`
Variables

- While automation exists to make it easier to make things *repeatable*, all of your systems are likely not exactly *alike*.
- The behaviour or state of configured machines may change and impact the desired state of other services, dynamically.
- Certain configuration files may exist as templates, which need instantiation, based on their context.
- Variables in Ansible are how we deal with differences between systems and states.
- Variables allow you to "program" with *conditions* and *loops*. 
Setting Variables

• Variables in Ansible help you to contextualise and abstract roles.

• Variables can be defined in several areas
  • Inventory
  • Playbook
  • Files and Roles
  • Command Line
  • Facts
Variable Hierarchy

1) Command line variables have the highest precedence. -e

2) 'most everything else' come next.
   1) Role vars
   2) Task and play variables

3) Variables defined in inventory.
   1) Host and group vars

4) Next comes facts discovered about a system.

5) Default vars defined in roles have the lowest priority.
Host Variables

http://docs.ansible.com/ansible/intro_inventory.html#host-variables

- Host variables are assigned in the inventory
- Arbitrary variables can be assigned to individual hosts
- There are also variables which change the way Ansible behaves when managing hosts e.g

90.147.156.175  \
ansible_ssh_private_key_file=~/.ssh/ansible-default.key  \
ansible_ssh_user=centos
Group Variables

- Hosts are grouped according to aspects, or any desired grouping
- Ansible allows you to define group variables which are available for any host in a group
- Group variables can be defined in the inventory:
  ```
  [webservers:vars]
  http_port=80
  ```
- Or in separate files under group_vars
  ```
  group_vars/webservers → ___
  http_port=80
  ```
Facts

• Facts are discovered about the play hosts at the start of each play
  • Unless turned off with `gather_facts=false`
  • Facts can be cached

• Facts uses the setup module, which uses various tools such as `facter` and `ohai` to obtain facts about hosts

• Facts are useful in determining the state of the machines in the play
Registering and using variables

- Variables can be statically set in the inventory, roles or plays, but can also be picked up based on the events of the play.
- Use register to set transient variables:
  `register: newvar`
- Call variables using `{{ newvar }}`
Example – Ensure that EPEL is available only on RedHat machines

• Vars set in role/x/vars:

```yaml
---
epel_package:
  '6':
  http://ftp.fau.de/epel/6/x86_64/epel-release-6-8.noarch.rpm

  '7':
  https://ftp.fau.de/epel/7/x86_64/e/epel-release-7-5.noarch.rpm

base_packages:
  - httpd
```
Example – Ensure that EPEL is available only on RedHat machines

• Use the facts and role variables in a task
  
  • `ansible_distribution_major_version`: discovered fact
  • `epel_package`: role variable
  • `epelinstall`: registered variable

```yaml
- name: Ensure that EPEL is present and configure yum:
  name: "{{ epel_package[ansible_distribution_major_version] }}"
  state: present
  register: epelinstall
- name: Re-generate metadata
  yum:
    name: '*'
    state: latest
    when: epelinstall.changed
```
Magic Variables

• Some variables are automatically created and filled by Ansible:

  • inventory_dir
  • inventory_hostname
  • inventory_hostname_short
  • inventory_file
  • playbook_dir

  • play_hosts
  • hostvars
  • groups
  • group_names
  • ansible_ssh_user
Variable from ansible_facts

"ansible_facts": {
    "ansible_all_ipv4_addresses": [
        "192.168.2.22",
        "172.17.42.1"
    ],
    "ansible_default_ipv4": {
        "address": "192.168.2.22",
        "alias": "wlan0",
        "gateway": "192.168.2.1",
        "interface": "wlan0",
        "macaddress": "3c:a9:f4:0d:74:c8",
        "mtu": 1500,
        "netmask": "255.255.255.0",
        "network": "192.168.2.0",
        "type": "ether"
    }
},
Calling complex variables

- Ansible uses mostly JSON to manage variables.
- Variables can have arbitrary complexity.
- Variables can be dereferenced using two different syntaxes:
  - `{{ ansible_eth0["ipv4"]["address"] }}`
  - `{{ ansible_eth0.ipv4.address }}`
Conditions

• Ansible provides a means to apply boolean or other conditions on variables

• Usually used in tasks or templates with the Jinja `when` statement – e.g.

  - name: "shutdown Debian flavored systems"
    command: /sbin/shutdown -t now
    when: ansible_os_family == "Debian"

• Use parentheses () to group conditions:

  when: ansible_distribution == "CentOS" and
        (ansible_distribution_major_version == "6" or
         ansible_distribution_major_version == "7")
Loops

http://docs.ansible.com/ansible/playbooks_loops.html

• Ansible loops are useful for writing cleaner playbooks and templates.

• Ansible provides several ways to loop:
  
  • Standard Loops
  • Nested Loops
  • Looping over Hashes
  • Looping over Fileglobs
  • Looping over Parallel Sets of Data
  • Looping over Subelements
  • Looping over Integer Sequences
  • Random Choices
  • Do-Until Loops
  • Finding First Matched Files
  • Iterating Over The Results of a Program Execution
Example: Loops in templates

• An easy way to generate an `/etc/hosts` file

{% for host in groups['head-nodes'] %}
{{ hostvars[host]['ansible_eth0']['ipv4']['address'] }} {{ host }}
{% endfor %}

{% endfor %}
Example: Loop over a list

- A list variable can be used in a task to perform several similar actions using the same module:
  - name: Install base packages
    yum:
      name: "{{ item }}"
    state: present
    with_items:
      - this_package
      - that_package
      - another_package
Recap

• We have written our first inventory and started to manage our machines with Ansible

• Ad-hoc commands are once-off ways to perform tasks on sets of hosts

• Playbooks are more complex groupings of tasks which define the desired states of our managed hosts

• Playbooks depend on variables, which have a hierarchical precedence and allow proper contextualisation of the tasks

• Ansible has the powerful feature of variables, including the possibility to have conditional statements and loops.
Hands-on session
Starting our Ansible playbooks