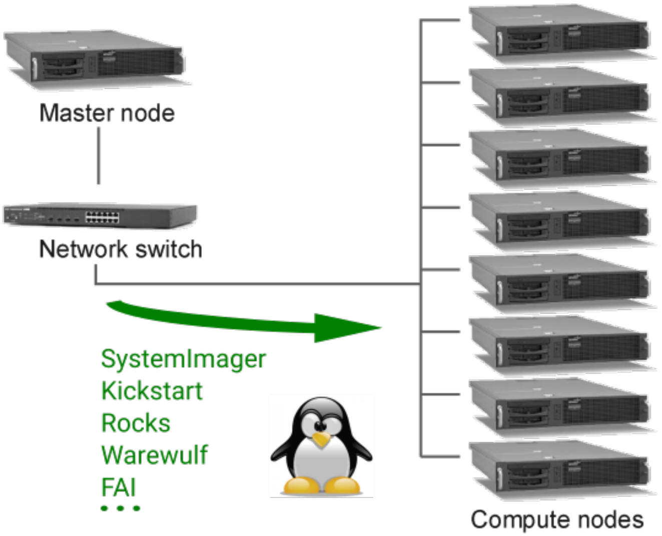
Configuration Management Systems: Ansible

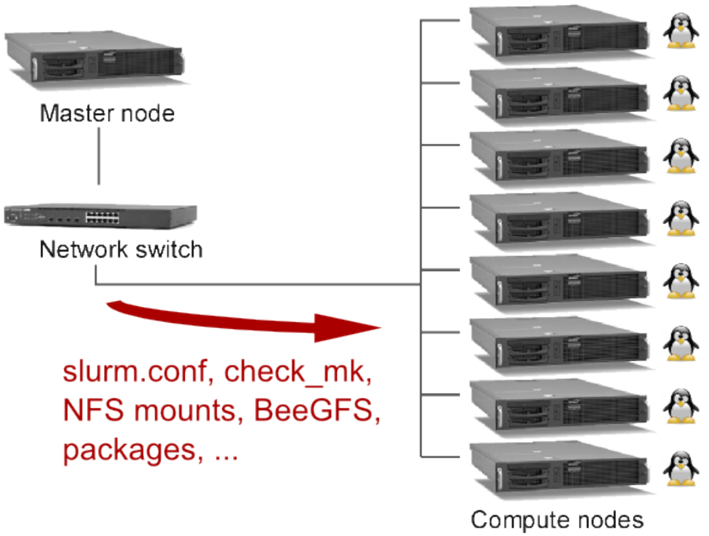
Iñigo Aldazabal Mensa – Centro de Física de Materiales (CSIC-UPV/EHU)

NASERTIC, Pamplona, 9th April 2018

Cluster deploy



Cluster evolution



Configuration Management

As we incorporate changes, servers configuration drifts away from its initial state.

How to deal with configuration changes:

- New “master” images
- `pdcp`, `fabric` et. al. for configuration file deploy
- ad hoc scripts
- systems management solutions as Spacewalk ...

This is a common situation happening not only in HPC systems; hasn’t this problem been tackled before in a general way?

Configuration Management

As we incorporate changes, servers configuration drifts away from its initial state.

How to deal with configuration changes:

- New “master” images
- `pdcp`, `fabric` et. al. for configuration file deploy
- ad hoc scripts
- systems management solutions as Spacewalk ...

This is a common situation happening not only in HPC systems; hasn’t this problem been tackled before in a general way?

Configuration Management Systems!

What is a CMS?

Configuration Management System (CMS) (ITILv2): A software tool that provides support for Configuration, Change and Release Management.

Configuration management systems provide an **automated** solution for **remotely managing** all aspects of systems administration such as:

- configuration (and other) files deployment `(pdcp, scp)`
- configuration files in place modification `(sed)`
- packages install / removal `(yum, apt)`
- system services configuration `(service, chkconfig)`
- users / groups / keys add / removal `(useradd, ssh-copy-id)`
- mountpoints configuration `(mount, fstab)`

...

CMS Features

Pros

- Specifically designed for the task.
- Configuration is **self-documented** by means of the own configuration files.
- Changes in a server/node configuration just requires re-running the configuration manager.
- Recipes idempotency (no changes are made if the defined state is already reached).
- For some of them, **very** easy to use comparing eg. against shell scripting.

Cons

- Learning yet another tool configuration internals and syntax.
- Not everything can be easily done.

Examples

CFEngine®

ANSIBLE

puppet labs®

SALTSTACK

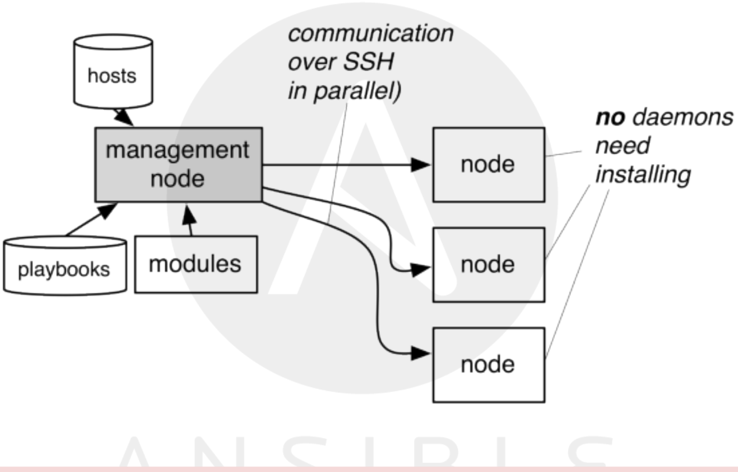
CHEF™

Ansible

“Ansible is the simplest solution for operating system configuration management available. It’s designed to be minimal in nature, consistent, secure, and highly reliable, with an extremely low learning curve for administrators, developers, and IT managers”

“Ansible requires nothing more than a password or SSH key in order to start managing systems and can start managing them without installing any agent software, avoiding the problem of “managing the management” common in many automation systems.”

Ansible


hosts
management node
playbooks
modules
communication over SSH in parallel)
node
node
node
no daemons need installing
ANSIBLE

Perfectly fits the typical HPC system / mindset

Ansible model

Agentless, push model

“Ansible uses no agents and no additional custom security infrastructure, so it’s easy to deploy.”

Ansible modules

“Ansible works by connecting to your nodes and pushing out small programs, called “Ansible Modules” to them. These programs are written to be resource models of the desired state of the system. Ansible then executes these modules (over SSH by default), and removes them when finished.”

Ansible YAML *playbooks*

“Ansible uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English.”

Requirements / Installation

Control machine requirements

- Python >=2.6

Managed node requirements

- Python >=2.4
- *If using SELinux, lib-selinux-python*

Installation

- RHEL, SL, CentOS, Fedora, Debian, or Ubuntu: OS package manager.
- other: `pip`

Getting started

Getting started with Ansible:

- Choose a machine as your management system and install Ansible (EPEL, apt-get, pip, ...)
- Ensure you have an SSH key for the nodes you want to manage and that your management system can log onto those nodes.
- Create a hosts file containing an inventory of your nodes.
- Start using Ansible.

Ansible inventory

ansible/hosts

```
[headnode]
headnode ansible_ssh_host=10.100.100.1 ansible_ssh_user=vagrant

[computing]
node1 ansible_ssh_host=10.100.101.1  ansible_ssh_user=vagrant
node2 ansible_ssh_host=10.100.101.2  ansible_ssh_user=vagrant
node3 ansible_ssh_host=10.100.101.3  ansible_ssh_user=vagrant
```

Ansible modules

```
[vagrant@headnode ~]$ ansible-doc -l * | wc -l
1378   # 242 on 2015!
[vagrant@headnode ~]$ ansible-doc ping
> PING

  A trivial test module, this module always returns ‘pong’ on
  successful contact. It does not make sense in playbooks, but
  it is useful from ‘/usr/bin/ansible’

# Test ’webservers’ status
ansible webservers -m ping
```

test run

ansible/hosts

```
[headnode]
headnode ansible_ssh_host=10.100.100.1 ansible_ssh_user=vagrant

[computing]
node1 ansible_ssh_host=10.100.101.1  ansible_ssh_user=vagrant
node2 ansible_ssh_host=10.100.101.2  ansible_ssh_user=vagrant
node3 ansible_ssh_host=10.100.101.3  ansible_ssh_user=vagrant
```

```
[vagrant@headnode ~]$ ansible computing -i ansible/hosts -m ping

node3 | success >> {
    "changed": false,
    "ping": "pong"
}
node1 | success >> {
    "changed": false,
    "ping": "pong"
}
node2 | success >> {
    "changed": false,
```

Ansible playbooks

“Playbooks are Ansible’s configuration, deployment, and orchestration language. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process.

If Ansible modules are the tools in your workshop, playbooks are your design plans.”

playbook.yml

```
---
- hosts: headnode
  become: yes

  tasks:
    - name: setup epel repo
      yum: pkg=yum-conf-epel state=present

    - name: Disable EPEL repo by defult
      replace: dest=/etc/yum.repos.d/epel.repo
               regexp='^enabled=1'
```

Ansible playbooks

playbook.yml

```
---
- hosts: headnode
  become:yes

  tasks:
    - name: setup epel repo
      yum: pkg=epel-release state=present

    - name: Disable epel repo by defult
      replace: dest=/etc/yum.repos.d/epel.repo
               regexp='^enabled=1'
               replace='enabled=0'

    - name: Install common epel packages
      yum: pkg={{ item }} enablerepo=epel state=present
      with_items:
        - bash-completion
        - htop
        - tmux
        - ansible
```

Ansible playbooks – test

```
[vagrant@headnode ~]$ ansible-playbook  \
      playbook.yml -i ansible/hosts


...
PLAY RECAP ********************************************
headnode: ok=14   changed=0    unreachable=0    failed=0
```

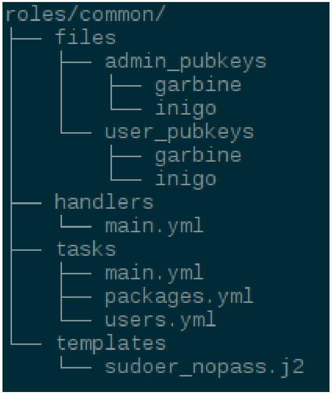
playbook.yml

```
---
- hosts: headnode
  become: yes
  tasks:
    - name: setup epel repo
      yum: pkg=epel-release state=present

    - name: Disable EPEL repo by defult
      replace: dest=/etc/yum.repos.d/epel.repo
               regexp='^enabled=1'
               replace='enabled=0'
```

Ansible roles

computing.yml

```
---
# file: computing.yml
- hosts: computing
  become: yes

  roles:
    - common
    - check_mk
```



```
roles/common/
├── files
│   ├── admin_pubkeys
│   │   ├── garbine
│   │   └── inigo
│   └── user_pubkeys
│       ├── garbine
│       └── inigo
├── handlers
│   └── main.yml
├── tasks
│   ├── main.yml
│   ├── packages.yml
│   └── users.yml
└── templates
    └── sudoer_nopass.j2
```

Ansible variables

host_vars/headnode

```
# users with ssh access to this specific machine
sshusers:
  - inigo
  - hal
  - dave
  - mycroft
```

roles/common/tasks/main.yml

```
/common/tasks/main.yml
- include: packages.yml
- include: users.yml
```

roles/common/tasks/users.yml

```
- name: Add this host defined regular users
    user: name={{ item }} state=present
    with_items: sshusers
```

Ansible gathered variables (*facts*)

```
[vagrant@headnode ~]$ ansible-doc setup
  > SETUP

  This module is automatically called by playbooks to gather useful
  variables about remote hosts that can be used in playbooks. It can
  also be executed directly by '/usr/bin/ansible' to check what
  variables are available to a host. Ansible provides many 'facts'
  about the system, automatically.
```

```
 [vagrant@headnode ~]$ ansible headnode -i ansible/hosts -m setup

headnode | success >> {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "10.0.2.15",
            "192.168.100.100",
            "10.100.100.1"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::a00:27ff:fe07:b86",
            "fe80::a00:27ff:fe63:d8eb",
            "fe80::a00:27ff:fe95:e962"
```

Ansible templates

roles/common/templates/sudoer_nopass.j2

```
{{ item }} ALL=(ALL) NOPASSWD: ALL
```

roles/common/vars/main.yml

```
---
sudoers:
  -     inigo
  - hal
```

roles/common/tasks/users.yml

```
# Sudoers: add host specific superusers  to nopassword-sudoers
- name: Set sudo permissions to (local to this host) superusers
  template: src=sudoer_nopass.j2
            dest=/etc/sudoers.d/{{ item }}_conf
            owner=root
            group=root mode=0440
  with_items: sudoers
```

Ansible

Hands on time!