Managing complex infrastructure

using Ansible

@dagwieers

The case at hand

The business case at hand

An automated solution saves $$$,
helps customers better and faster

LAB infrastructure

POD

- A set of core switches
  (e.g. Nexus 3K or 9K)
- 3 or more spines/leafs
- 6 or more UCS systems
  (e.g. C220M4 or C240M5)
- Optional chassis + blades

Shared

- VMware vCenter
  (for virtual machines)
- Cobbler setup
  (for deploying systems)
- Microsoft Deployment Toolkit
  (for Windows)

So we build a solution

CCLA: Awarded* web-based solution

Front-end

- NodeJs
- AngularJS

- Register POD
- Perform scenario
- Report

API

- Flask
- Python

- REST API
- Database-driven
- Builds inventories
- Manages runs
- Keeps logs
- Systemd integration

Back-end

- Ansible
- Python libraries

- Playbooks
- Roles
- Modules
- Filters

And we build communities

Ansible Working Groups

```
---

- Core WG
- Network WG
- Windows WG
- VMware WG
- Container WG
- AWS WG
- Azure WG
- ACI community
```

https://github.com/ansible/community/wiki

Example automated scenarios

- Power-on, power-off, reset devices, clear terminals
- Building an ACI fabric
  - Clear existing APICs, Spines and Leafs
  - Load ACI software
  - Prepare manufacturing
  - Provision ACI fabric
  - Deploy ACI/NXOS topologies
- Building AD + MSSQL + SCVMM on VMware/Win2012R2
  - Provision and register physical HyperV (MDT)
  - Build SCVMM cluster
- Deploy VCSA, ESX, Candid, OpenStack, ACI MultiSite, ...

Everything upstream

During this project:

- 13 core proposals (+implementation)
- +600 commits, +60k new lines
- Contributed to +45 new modules
  - incl. wait_for_connection, cobbler_*, win_psexec, aci_*, msc_*, reboot, win_reboot
- Changes to +50 modules
- A lot of core changes
- Windows integration
  - Idempotency, new modules, …
  - WinRM and PSRP connection
  - Become support
- Custom VMware changes/modules
- Cobbler modules
- ACI and ACI MultiSite modules
- Module documentation changes
- Doc website improvements
- **Future:** contribute Cisco roles

Some “special” examples

Using **shell** with **expect** to interact with a serial console

```
- name: Wait for APIC install to finish and configure APIC
  shell: |
    log_file {{ '~/logs/task%03d-pod%s-%s-job%06d.%s.explog'|format(task|int, pod, name, job|int, inventory_hostname) }}
    set send_human {.1 .3 1 .05 2}
    set timeout 120

    spawn ssh -oStrictHostKeyChecking=no {{ cimc_username }}@{{ cimc_host }}

    expect {
        " password:" { send -h "{{ cimc_password }}\n" }
        timeout { exit 2 }
    }

    expect {
        "# " { send -h "connect host\n" }
        timeout { exit 3 }
    }
    ...
  args:
    executable: /usr/bin/expect
```

Select **all APIC nodes** from cluster **except this one**

— — —

Inventory contains:

```
apic:
  hosts:
    bdsol-aci51-apic1:
      ansible_host: 10.48.31.177
    bdsol-aci51-apic2:
      ansible_host: 10.48.31.178
    bdsol-aci51-apic3:
      ansible_host: 10.48.31.179
```

Jinja2 statement:

```
apic_hostnames: "{{ hostvars|dictsort|selectattr('1.group_names', 'issuperset', ['apic'])|
                    map(attribute='1.ansible_host')|reject('match', '^'~ansible_host~'$')|list }}"
```

Statement returns (on bdsol-aci51-apic1):

```
apic_hostnames: [ 10.48.31.178, 10.48.31.179 ]
```

Perform **form-urlencoded** HTTP requests using **uri**

```
- name: Get CALO connection id
  uri:
    url: https://cxlabs-ot6api.cisco.com/ui/get_instance_info
    use_proxy: no
    return_content: yes
    method: POST
    body_format: form-urlencoded
    body:
      api_key: '{{ calo_api_key }}'
      client_connection_id:
      uri_base: caloapi
  register: info
  failed_when: info.json is not defined
  delegate_to: localhost
```

Using **cobbler_system** and **imc_rest**

```
- name: Update system in Cobbler and enable netboot
  cobbler_system:
    host: '{{ groups["cobbler"]|first }}'
    username: '{{ cobbler_username }}'
    password: '{{ cobbler_password }}'
    name: '{{ cobbler_name }}'
    properties:
      name_servers: '{{ global_dns_ip }}'
      name_servers_search: '{{ global_domain }}'
      netboot_enabled: yes
      profile: '{{ cobbler_profile }}'
    interfaces:
      eth0:
        ipaddress: '{{ ansible_host }}'
        macaddress: '{{ device_mac }}'
  delegate_to: localhost
```

```
- name: Reconfigure IMC and start PXE boot
  imc_rest:
    content: |
        <!-- POWER DOWN SERVER -->
        <configConfMo><inConfig><computeRackUnit
dn="sys/rack-unit-1" adminPower="down" usrLbl="ACI Lab - POD{{
pod_id }} - {{ inventory_hostname_short
}}"/></inConfig></configConfMo>

        <!-- CONFIGURE PXE BOOT -->
        <configConfMo><inConfig><lsbootLan
dn="sys/rack-unit-1/boot-policy/lan-read-only" access="read-only"
order="1" prot="pxe" type="lan"/></inConfig></configConfMo>

        <!-- POWER UP SERVER -->
        <configConfMo><inConfig><computeRackUnit
dn="sys/rack-unit-1" adminPower="up"/></inConfig></configConfMo>
    delegate_to: localhost
```

SCVMM server installation

A textbook procedure from Microsoft written in YAML

- Creating required users/groups (local/AD)
- Install some tools (ADK?, MSSQL cli + utilities)
- Reboot
- Template config response file
- Mount System-Center ISO image
- Run installer from virtual drive (Async+Become)
- Unmount ISO
- Create new System-Center cloud (PowerShell)

Windows AD join

Anyone interested in this ?