Linux Automation

Ansible

Overview

- The Problem
- Solutions
- Ansible Concepts
- How it all works
- Demo

The Problem

- SysAdmins spending way too much time on:
  - Configuring
  - Provisioning
  - Troubleshooting
  - Maintaining server operations
- Think 100->1,000->10,000 servers for one admin
- You might have, 100 web servers, 1000 file servers, and so on…

The Solutions -- Bash (Old, Hard Way)

Write shell scripts to telnet, ssh into machines and set them up each.

- Installed everywhere by default

Cons:

- Not multi-threaded, usually.
- Hard to troubleshoot any issues that arise

The Solutions -- Puppet/Chef

Agent/Master architecture -- agent manages node and requests info from master

- Pull changes to servers using agents. Puppet is over https (XML).
- Great analytics and reporting for managing configurations.

Cons:

- Install agents on every server
- Complex (especially puppet, requires understanding of ruby)

The Solutions -- Ansible

Best of both worlds!

- “Push” changes to servers
- No agents, all done through ssh, python (default on most servers)
- Easy to understand, YAML syntax.
- Faster, without master-agent model
- High security with ssh

Cons:

- ssh can have issues scaling
- Ansible Tower (web portal front-end++) not free

Ansible

Basic Concepts:

- Inventory
- Modules
- Ad-Hoc Commands
- Playbooks
  - Tasks
  - Handlers
  - Roles

Inventory

- Define how to classify remote hosts
- Create logical groups for management aid
- Define communication variables (Ex: ssh port) per group if needed
- Default location: /etc/ansible/hosts

Inventory

```
[atlanta]
host1
host2

[raleigh]
host2
host3

[southeast:children]
atlanta
raleigh

[southeast:vars]
some_server=foo.southeast.example.com
halon_system_timeout=30
self_destruct_countdown=60
escape_pods=2

[usa:children]
southeast
northeast
southwest
northwest
```

Modules

- http://docs.ansible.com/ansible/latest/list_of_all_modules.html
- Used for executing tasks
- Keep track of state
  - “Facts”

Ad-Hoc Commands

- Ansible {pattern} -m {module} -a “{options}” {flags}
  - Pattern: which hosts
  - Module: which ansible module
  - Options: module options
  - Flags: command flags
- Examples:
  - File transfer (ansible all -m copy -a “src=/etc/hosts dest=/tmp/hosts”)
  - Ping (ansible all -m ping)
  - Manage services (ansible webservers -m service -a “name=httpd state=started”
- demo

Playbooks

- YAML language
- Any number of “plays” in a list
- Playbook run to provision multi-machine orchestration

Plays - tasks

- In-order execution on all machines matched according to host
  - Sequential on host, parallel on group
- Each uses specific module
- Can incorporate templates (jinja2 syntax)
- Modules used to bring system to desired state
- Can be conditional

```
1 tasks:
2 - name: {task name}
3     {module}: {options}
4
```

```
12 tasks:
13   - name: "shut down Fedora flavored systems"
14     command: /sbin/shutdown -t now
15     when: ansible_os_family == "Fedora"
~
```

Plays - handlers

- Triggered at the end of each block of tasks whenever a “change” has been made on the remote system
  - ONLY on change
- Referenced by name

```
1  tasks:
2  - name: {task name}
3      {module}: {options}
4  notify:
5    - {handler name}
6
7  ...
8  handlers:
9    - name: {handler name}
10     {module}: {options}
```

Plays - templates and variables

- Used for per-group or per-task changes

```
---
- hosts: webservers
  vars:
    http_port: 80
    domain: example.com
  become: true
  tasks:
  - name: install packages
```

```
<VirtualHost localhost:{{ http_port }}>
    ServerAdmin webmaster@{{ domain }}
    ServerName {{ domain }}
    ServerAlias www.{{ domain }}
    DocumentRoot /var/www/{{ domain }}
    ErrorLog /var/www/example.com/error.log
    CustomLog /var/www/example.com/requests.log combined
</VirtualHost>

```

Playbooks - roles

- Using ansible file structure, you can break up plays into roles for use by playbook
- More dynamic

```
14  - hosts: webservers
15    roles:
16      - common
17      - webservers
18
```

```
site.yml
webservers.yml
fooservers.yml
roles/
   common/
      tasks/
      handlers/
      files/
      templates/
      vars/
      defaults/
      meta/
   webservers/
      tasks/
      defaults/
      meta/
```

How it works in Step-by-Step

1. Determine hosts and sort them into groups per Inventory file
2. Create your playbooks
3. Run, ansible gathers “facts”
    a. Facts are used to determine state on a per module basis
4. Ansible executes tasks in parallel on specified hosts
5. Re-gathers “facts” to verify state changes

Demo

... I hope

Questions?