Orchestration

Ansible and more!

Summary of Orchestration

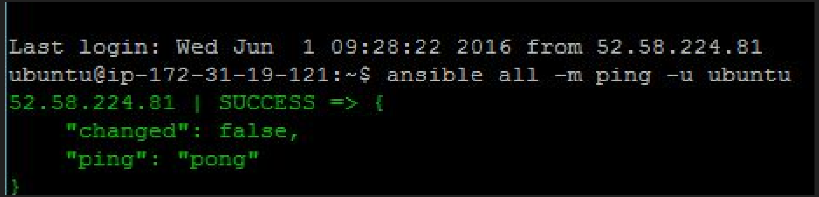
	Puppet	Chef	Ansible	Salt Stack
Pros	• Ruby • Push and Pull Based • Mature • Support for Windows and Linux	• Ruby or yaml • Pull Based • Mature • Support for Windows and Linux • Very large toolset, e.g. Kitchen • Extensible with Ruby	• Agentless, everything over SSH • Good integration with cloud services • Simple CLI • Simple yml based files • Extensible with Python	• Geared towards scalability and speed • Simple configuration files
Cons	• Steep learning curve • Requires agent on machine (for pull) • More geared towards ops teams • Not as easy to extend as others	• Steep learning curve • Requires agent on machine	• Windows cannot be control machine • Somethings can be a little more complex than they should be • Not fantastic windows support	• Requires agents on machines • More of a curve than Ansible, but less than Chef / Puppet

More on Ansible

- Terminology:
  - Hosts are servers
  - Playbooks describe all steps needed and which hosts to affect
  - Plays describe individual sets of related tasks
  - Tasks are a separate step to do something
  - Roles are playbooks that other playbooks can call
- Hosts can be classified together (say by client, product and server role)
- Playbooks will identify servers to affect and describe the steps to follow
- Steps are followed linearly in playbook, but describe target states, each step will work out what commands it needs to run to complete

Simple to Run ad-hoc

```
ansible all -m ping
```



```
Last login: Wed Jun  1 09:28:22 2016 from 52.58.224.81
ubuntu@ip-172-31-19-121:~$ ansible all -m ping -u ubuntu
52.58.224.81 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

```
ansible-playbook deploy.yml
```

```
- name: Ensure server dependencies are correct
    hosts: tag_Name_nen_c_proto_server
    gather_facts: True
    remote_user: ubuntu
    become: yes
    become_user: root
    become_method: sudo
```

Install / Deploy

```
- name: install dependencies
      apt: name={{item}}
      with_items:
      - openjdk-8-jdk
      - elasticsearch
      - git
      - postgresql-9.4
      - nginx
```

```
- name: stop website service
      service: name=nen-connect-proto state=stopped
```

```
- name: copy elasticsearch config
      copy: src=files/elasticsearch/elasticsearch.yml dest=/etc/elasticsearch/elasticsearch.yml
      notify: restart elasticsearch
```

```
handlers:
      - name: restart nginx
      service: name=nginx state=restarted

      - name: restart elasticsearch
      service: name=elasticsearch state=restarted
```

Provisioning an EC2 Instance

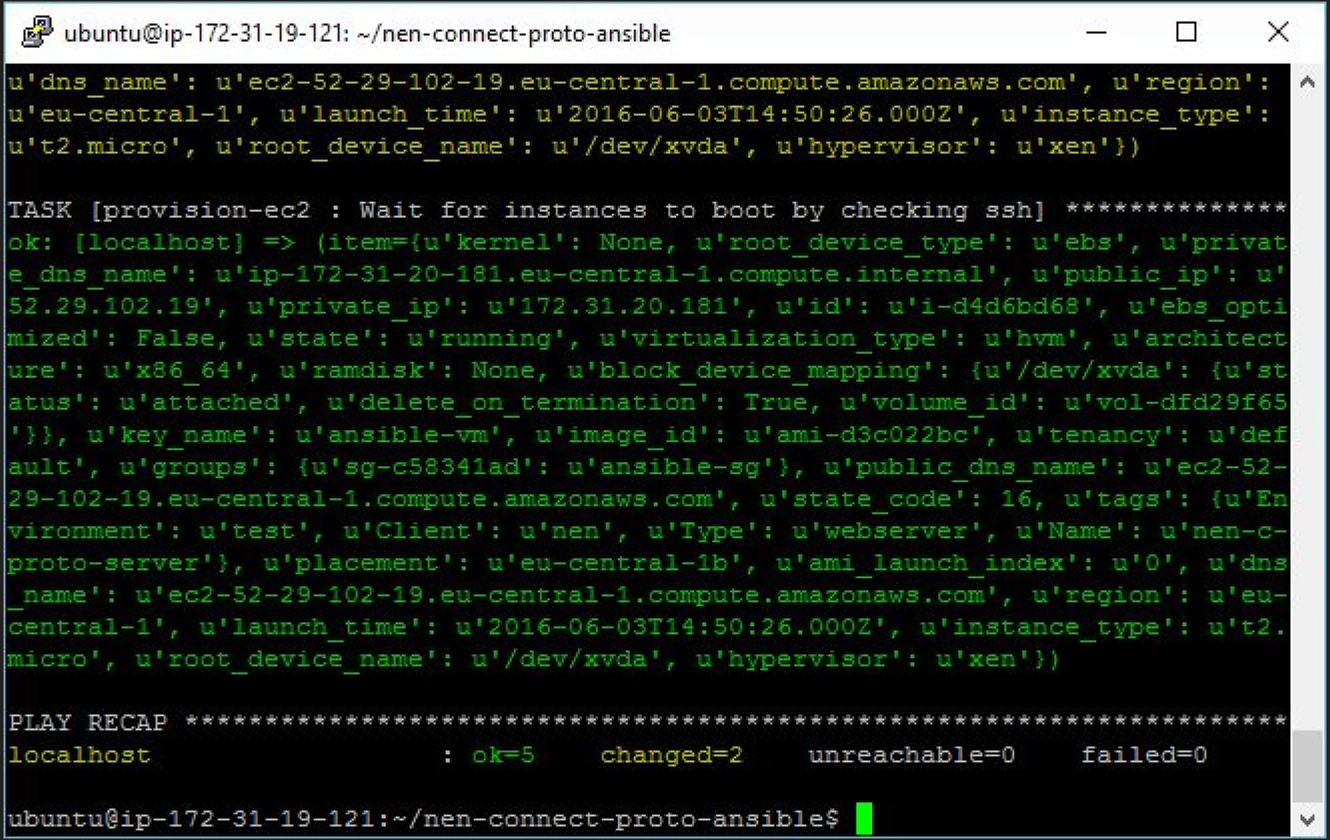
```
 - name: Provision EC2 Server
        local_action:
        module: ec2
        key_name: ansible-vm
        group_id: sg-xxxxxx
        instance_type: t2.micro
        image: "{{ ec2_image }}"
        region: "{{ ec2_region }}"
        vpc_subnet_id: "{{ ec2_subnet_id }}"
        instance_tags: '{"Name":"{{ ec2_tag_Name }}","Type":"{{ ec2_tag_Type}}","Environment":"{{ ec2_tag_Environment }}","Client":"{{ ec2_tag_Client }}"}'
        assign_public_ip: yes
        wait: true
        count: 1
        volumes:
        - device_name: /dev/xvda
          device_type: gp2
          volume_size: "{{ ec2_volume_size }}"
          delete_on_termination: true
        register: ec2
```

Provisioning an EC2 Instance (cont)

```
 - add_host: name={{ item.public_ip }}
                 groups=tag_Type_{{ec2_tag_Type}},tag_Environment_{{ec2_tag_Environment}},tag_Client_
{{ec2_tag_Client}}
                 ec2_region={{ec2_region}}
                 ec2_tag_Name={{ec2_tag_Name}}
                 ec2_tag_Type={{ec2_tag_Type}}
                 ec2_tag_Environment={{ec2_tag_Environment}}
                 ec2_ip_address={{item.public_ip}}
         with_items: "{{ec2.instances}}"

  - name: Wait for instances to boot by checking ssh
         wait_for: host={{ item.private_ip }} port=22 delay=60 timeout=320 state=started
         with_items: "{{ec2.instances}}"
```

Provisioning an EC2 Instance (cont)



```
u'dns_name': u'ec2-52-29-102-19.eu-central-1.compute.amazonaws.com', u'region':
u'eu-central-1', u'launch_time': u'2016-06-03T14:50:26.000Z', u'instance_type':
u't2.micro', u'root_device_name': u'/dev/xvda', u'hypervisor': u'xen'})

TASK [provision-ec2 : Wait for instances to boot by checking ssh] **************
ok: [localhost] => (item={u'kernel': None, u'root_device_type': u'ebs', u'privat
e_dns_name': u'ip-172-31-20-181.eu-central-1.compute.internal', u'public_ip': u'
52.29.102.19', u'private_ip': u'172.31.20.181', u'id': u'i-d4d6bd68', u'ebs_opti
mized': False, u'state': u'running', u'virtualization_type': u'hvm', u'architect
ure': u'x86_64', u'ramdisk': None, u'block_device_mapping': {u'/dev/xvda': {u'st
atus': u'attached', u'delete_on_termination': True, u'volume_id': u'vol-dfd29f65
'}}, u'key_name': u'ansible-vm', u'image_id': u'ami-d3c022bc', u'tenancy': u'def
ault', u'groups': {u'sg-c58341ad': u'ansible-sg'}, u'public_dns_name': u'ec2-52-
29-102-19.eu-central-1.compute.amazonaws.com', u'state_code': 16, u'tags': {u'En
vironment': u'test', u'Client': u'nen', u'Type': u'webserver', u'Name': u'nen-c-
proto-server'}, u'placement': u'eu-central-1b', u'ami_launch_index': u'0', u'dns
_name': u'ec2-52-29-102-19.eu-central-1.compute.amazonaws.com', u'region': u'eu-
central-1', u'launch_time': u'2016-06-03T14:50:26.000Z', u'instance_type': u't2.
micro', u'root_device_name': u'/dev/xvda', u'hypervisor': u'xen'})

PLAY RECAP *********************************************************************
localhost                  : ok=5    changed=2    unreachable=0    failed=0

ubuntu@ip-172-31-19-121:~/nen-connect-proto-ansible$
```

Best Cases to Use

- Multiple servers (say server farm)
  - Parallel deployments on many machines that must be the same
  - Can control parallelism, so only some servers are down during deployments
- Multiple deployment environments
  - Additional arguments can be passed in through the command line
- Lots of dependencies to manage
  - Have to truly understand what needs to be deployed on a server for an application to work
  - Self documenting too thanks to the friendly YML syntax
- Deploying similar things multiple times
  - Ansible Galaxy helps with providing reusable roles

Problems

Some modules require additional installs on the host being controlled which you have to manually handle (postgres_db)

Amazon Linux doesn’t work very well

Some task types are OS dependant (apt vs yum)

Ansible Galaxy has some useful, but sometimes questionably designed roles