Ansible

Automating Cloud Orchestration and Management

Kash Karimi. October 2016

Concepts

Infrastructure As Code (IaC)

- The process of managing and provisioning computing infrastructure and their configuration through code (instead of physical or interactive configuration)

Recipe for Desire

- Compares current state to `desired state` defined by task -> Make all changes
- Ideally manage both `Machine state` and `Cloud state` same way

Problem

- Modern infrastructure is too hard to manage without IaC

Other Tools

Puppet, Chef, Saltstack

- Master-Agent model always pulls
- Custom protocols (vs SSH) are harder to troubleshoot
- Need manual setting up or another automation tool

General language tools

- Non-standard and lots of choice - Fabric, Paver, Shell scripts etc
- Imperative rather than declarative - can be powerful but can get complicated
- Reinventing the wheel, lacks standards/best practices

Ansible

- Syntax for both machine and human friendly
- Manage Everything with built-in modules
- Big community
- Powerful YAML syntax
- Does not require agents or root access
- Checks many of devops toolchain list: `Automation, Cloud modules, Provisioning, Testing & Verification` ...
- Can manage hundreds (if not thousands) servers in parallel
- Verifies and fails fast
- Can also write tests (asserts module) though other tools might be better (RSpec / Test Kitchen)

Ansible Concepts

- Declarative `desired state` vs `how to get there`
- Idempotency `only apply when need to`
- Push by default but also supports pull method
- Tasks synchronous actions (but can be async too)
- Inventory (static vs dynamic) and Hosts
- Roles encourage reuse
- Playbooks map roles or tasks to hosts
- Ansible Galaxy thousands of community developed roles
- Others: Jinja2 templates, rolling updates, notify, lookups, facts

Playbook Example

```
---

- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at boot
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

Tasks can be dynamic

```
- ecs_service:
    state: present
    name: "popsss-{{ dc }}-{{ cluster_name }}-{{ item }}"
    cluster: "popsss-{{ dc }}-{{ cluster_name }}"
    desired_count: "{{ cluster_app_node_count }}"
    task_definition: "{{ item_{{ task_definition }} }}"
  with_items:
    - app-main
    - plugin-server
    - http-proxy
    - celery-master
    - celery-worker
    - consul
    - consul-registrator
    - rabbitmq
    - redis
  ignore_errors: no
```

Variables override each other through priority precedence, allowing defaults for group of roles or hosts.

default.yml

```
#nested
aws:
  region: "eu-west-1"
  ec2:
    type: "m3.medium"
    ami: "ami-241ac045"
# dynamic
docker_registry_user: "{{ lookup('env','DOCKER_USER') }}"
postgres_Server: "{{ lookup('ini', 'url section=database
```

CLI wins every time:

```
ansible-playbook "-e 'NODE_ENV=staging GIT_BRANCH=develop
```

Roles are recommended directory structure for managing all the resources for a role

```
roles/
   webserver/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
```

..but it can be simplified a lot.

Gotchas

- YAML can be strange
- Variables are global
- Pull method might be for scaling to updating for thousands of nodes
- Tasks run in order (not parallel)

Questions?

Thank you