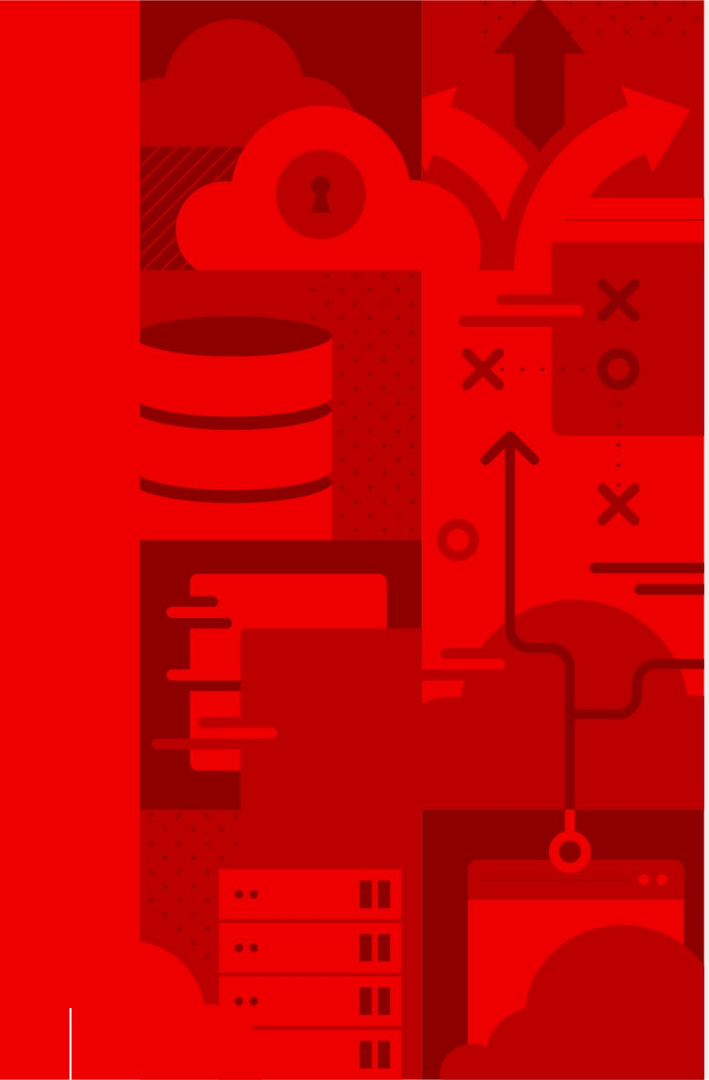


Holistic System Automation with Ansible

Tim Quinlan

Ohio LinuxFest 2019

Red Hat

Approaches to Administration

Traditional

Monolithic systems

Generalist sysadmins

Individual hardware and software selections

Individual tooling

Snowflakes, edge cases

Not scalable

Enterprise

Resilient yet complex (load balancing, clustering, databases, storage)

Specialist system admins

Company wide hardware and software pre-selections

Functional tooling

Fewer snowflakes, fewer edge cases, load-bearing technical debt

Scalable, until it isn’t

Cloud Native

Resilient, cohesive components

Underlying complexity abstracted

Specialist engineers, generalized operations

Commodity hardware, public clouds, everything “as code” or “as a service”

Functional tooling abstracted with cross-functional pipelines

Truly scalable

Cloud Native

Highly Resilient

Underlying complexity abstracted

Specialist engineers, generalized operations

Commodity hardware, public clouds, everything “as code” or “as a service”

Functional tooling **abstracted with cross-functional pipelines**

Truly scalable

Devops

...a set of practices that combines software development (*Dev*) and information-technology operations (*Ops*) which aims to shorten the systems development life cycle and provide continuous delivery with high software quality

...intended to be a cross-functional mode of working, those that practice the methodology use different sets of tools—referred to as "toolchains"—rather than a single one

https://en.wikipedia.org/wiki/DevOps

Devops

...a set of practices that combines software development (*Dev*) and information-technology operations (*Ops*) which aims to shorten the systems development life cycle and provide continuous delivery with high software quality

...intended to be a cross-functional mode of working, those that practice the methodology use **different sets of tools—referred to as "toolchains"**—rather than a single one

https://en.wikipedia.org/wiki/DevOps

Agile

...comprises various approaches to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s)

https://en.wikipedia.org/wiki/Agile_software_development

Time Out!

We’re sysadmins, not developers

We’ve never used a development methodology before

Time Out!

We’re sysadmins, not developers

We’ve never used a development methodology before

Betcha have

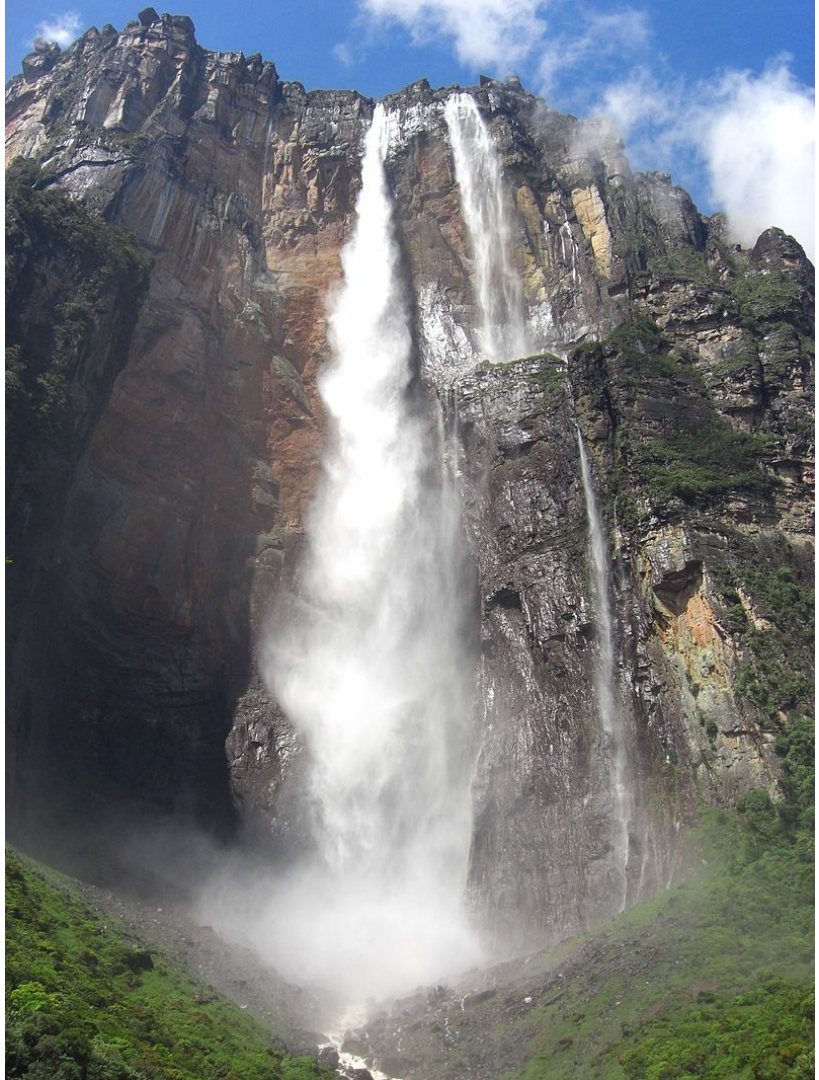
Time Out!

We’re sysadmins, not developers

We’ve never used a development methodology before

You have

Any guesses?



https://en.wikipedia.org/wiki/Waterfall

Waterfall

… a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one and corresponds to a specialisation of tasks

… tends to be among the less iterative and flexible approaches, as progress flows in largely one direction

https://en.wikipedia.org/wiki/Waterfall

Agile Basics

Scrum

… an agile process framework for managing complex knowledge work … designed for teams of ten or fewer members, who break their work into goals that can be completed within timeboxed iterations, called *sprints*, no longer than one month and most commonly two weeks, then track progress and re-plan in 15-minute time-boxed stand-up meetings, called *daily scrums*.

https://en.wikipedia.org/wiki/Scrum_(software_development)

Sprint

… a repeatable fixed time-box during which a "Done" product of the highest possible value is created.

… daily meetings are held to discuss the progress of the project undertaken and any difficulty faced by any team member of the team while implementing the project. The outcome of the sprint is a deliverable, albeit with some increments.

https://en.wikipedia.org/wiki/Scrum_Sprint

Pair Programming

… an agile software development technique in which two programmers work together at one workstation. One, the *driver*, writes code while the other, the *observer* or *navigator*,[1] reviews each line of code as it is typed in. The two programmers switch roles frequently.

While reviewing, the observer also considers the "strategic" direction of the work, coming up with ideas for improvements and likely future problems to address. This is intended to free the driver to focus all of their attention on the "tactical" aspects of completing the current task, using the observer as a safety net and guide.

https://en.wikipedia.org/wiki/Pair_programming

Tooling

Version Control

Make sure there are native clients for **EVERY** user's platform

Training, training, training

Use as conduit

Training

https://xkcd.com/1597/

Process Building

Get buy in

Accept input

No easy buttons (aka black boxes)

No single points of failure

Ansible

Integrated, no glue scripts

Agentless, just need ssh and a recent version of Python

Auto fact gathering

Idempotent

YAML reduces complexity

Step by step error checking and reporting built in

Built in templating

Parallel execution by default (system level)

Ansible Playbooks

```
- name: Enable Webservers
  hosts: webservers
  become: yes
  tasks:
    - name: Deploy the web page
      copy:
        src: ./index.html
        dest: /var/www/html
        owner: apache
        group: apache
        mode: 0644
```

Ansible Inventory Files

```
$ cat prod_inventory
[webservers]
prod1 ansible_host=192.168.122.181
prod2 ansible_host=192.168.122.249
```

Ansible Command Line

```
$ ansible-playbook -i dev_inventory playbook.yaml
```

```
$ ansible-playbook -i prod_inventory playbook.yaml
```

Ansible Output

```
$ ansible-playbook -i prod_inventory all_tasks.yml

… SNIP …
TASK [Enable httpd] ************************************************************************************
ok: [prod1]
changed: [prod2]

PLAY [Enable Webservers] *******************************************************************************

TASK [Gathering Facts] *********************************************************************************
ok: [prod2]
ok: [prod1]

TASK [Deploy the web page] *****************************************************************************
ok: [prod1]
changed: [prod2]

PLAY RECAP *********************************************************************************************
prod1          : ok=11   changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
prod2          : ok=11   changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Fact Gathering

```
$ ansible -i dev_inventory dev1 -m setup
dev1 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "192.168.122.82"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::5054:ff:fecb:e31e"
        ],
        "ansible_apparmor": {
            "status": "disabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "04/01/2014",
        "ansible_bios_version": "1.12.0-2.fc30",
        "ansible_cmdline": {
            "BOOT_IMAGE": "(hd0,msdos1)/vmlinuz-4.18.0-80.11.2.el8_0.x86_64",
            "crashkernel": "auto",
            "quiet": true,
            "rd.lvm.lv": "rhel/swap",
            "resume": "/dev/mapper/rhel-swap",
            "rhgb": true,
```

Where to Start?

OS Administration

Focused hardware selection

Industry and corporate standards

Scripted builds

Plain-text config management

Fleet-wide system administration and operations

Glue code still needed to integrate with other functions

Modern OS Administration

Focused hardware selection

Industry and corporate standards

Scripted builds

Plain-text config management

Fleet-wide system administration and operations

Glue code still needed to integrate with other functions

Server Deployment

Pre-build: storage, networking

OS install: automated

Post-build: application install, UAT, backups, scheduling

Deployment: production push, networking, monitoring

Server Deployment

Pre-build: storage, networking

OS install: **automated**

Post-build: **application install, UAT,** backups, scheduling

Deployment: **production push,** networking, monitoring

Demo: Team Roles

User access and permissions are still handled at the OS level

- tech: Provisions vanilla servers but does not have rights customize to them.
- dev: Provides the app and makes sure it runs in development. Has full access to the development servers, but no access to the production servers.
- ops: Has full rights to all the servers, but is primarily concerned with deploying the production servers.

Demo: Team Roles

- tech:
  - dev_hosts: Inventory of development servers
  - prod_hosts: Inventory of production servers
- dev:
  - website.yml: Playbook to deploy the web page
  - index.html: Web page to deploy
- ops:
  - server_config.yml: Playbook to setup Apache, Firewalld and SELinux
  - haproxy.yml, haproxy.cfg: Playbook and config file for load balancer
  - all_tasks.yml: Playbook that runs server_config.yml then website.yml

Demo Scenarios

Web Update

New production server

Working with Config Files

Maintaining a copy in version control

Built-in modules

Galaxy modules

Go For It

Objection Handling

This seems like overkill

- For very small examples, yes, but scales very well

We already do this with bash

- Have you ever used expect on a network switch interactive shell?
- Parallel exec, idempotency, error checking built in
- Ansible modules https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.html
- Galaxy modules https://galaxy.ansible.com/

My teammates/boss/whoever won't want to do it

- Agile
- Value Stream Mapping https://en.wikipedia.org/wiki/Value-stream_mapping
- Critical Path Method https://en.wikipedia.org/wiki/Critical_path_method

First steps

Start using VCS for existing scripts and configs

Get different working groups on the same page (Pair Programming method)

Thank You

Tim Quinlan

tquinlan@redhat.com

@trquacker

https://github.com/timquinlan