BSD Systems Management with Ansible

Transforming your Sysadmin Shell Scripts to Ansible

Benedict Reuschling
bcr@FreeBSD.org

September 9, 2017

vBSDcon 2017

Table of contents

Introduction

About me

. . . and why this is relevant to my talk:

- Big Data Lab Engineer (sysadmin) without a Lab at a University is my dayjob

About me

. . . and why this is relevant to my talk:

- Big Data Lab Engineer (sysadmin) without a Lab at a University is my dayjob
- Main purpose: Labs, research, and thesis work with and around NoSQL databases and Big Data

About me

. . . and why this is relevant to my talk:

- Big Data Lab Engineer (sysadmin) without a Lab at a University is my dayjob
- Main purpose: Labs, research, and thesis work with and around NoSQL databases and Big Data
- Took over 28 machines (Dell C6220) plus central file server from a colleague, all running Ubuntu

About me

. . . and why this is relevant to my talk:

- Big Data Lab Engineer (sysadmin) without a Lab at a University is my dayjob
- Main purpose: Labs, research, and thesis work with and around NoSQL databases and Big Data
- Took over 28 machines (Dell C6220) plus central file server from a colleague, all running Ubuntu
- Central fileserver with NFS mounts for `/home` dirs of the nodes

About me

. . . and why this is relevant to my talk:

- Big Data Lab Engineer (sysadmin) without a Lab at a University is my dayjob
- Main purpose: Labs, research, and thesis work with and around NoSQL databases and Big Data
- Took over 28 machines (Dell C6220) plus central file server from a colleague, all running Ubuntu
- Central fileserver with NFS mounts for `/home` dirs of the nodes
- 40 nodes now, mixed FreeBSD/Ubuntu environment (gradually ~~subverting~~ migrating to FreeBSD)

How did I get involved with Ansible?

Cluster machines needed too long to install (25 min). Which year is this again?

How did I get involved with Ansible?

Cluster machines needed too long to install (25 min). Which year is this again?

It turns out, it's the same image for regular lab machines provided by our IT department:

- graphics drivers, CUPS, GUI tools
- Preinstalled NoSQL databases in my case

How did I get involved with Ansible?

Cluster machines needed too long to install (25 min). Which year is this again?

It turns out, it's the same image for regular lab machines provided by our IT department:

- graphics drivers, CUPS, GUI tools
- Preinstalled NoSQL databases in my case

. . . for a server?

How did I get involved with Ansible?

Cluster machines needed too long to install (25 min). Which year is this again?

It turns out, it's the same image for regular lab machines provided by our IT department:

- graphics drivers, CUPS, GUI tools
- Preinstalled NoSQL databases in my case

. . . for a server?

Stripped it down to bare minimum with custom Ubuntu/FreeBSD ISO, only doing:

- Partitioning
- Installing the base OS
- Setup Networking
- Reboot

Result of that: Managed to drop install time by less than half.

For FreeBSD, custom base install shell script takes barely 3 minutes.

Refining, tweaking, and optimizing is an ongoing process

Ansible for all the rest

Once a node is in multi-user mode, Ansible scripts run for final setup tasks depending on usage

Allows us a much more flexible machine use, quicker provisioning for end-users

Previously, shell scripts were doing the work of setting up NoSQL databases:

- MongoDB
- Couchbase
- Hadoop (Java)
- Pig

Generally works well, encourages further use

Before, when using shell scripts, we ran into some interesting scenarios...

Problems with Shell Scripts...


OOOPS...
I DID IT AGAIN.
imgflip.com

Oops, I did it again...

Have you ever accidentally ran your shell setup script twice? Rolled out into production?

- Config files have the same lines in it twice, where each one must not appear more than once
- Now I need a lot of extra code to check for and prevent duplication
- A simple script becomes a big if-this-then-that-take-care-of-all-eventualities mess
- Undoing is even worse, probably resulting in yet another script...
- Different scripts for different environments, OSes, package managers, heterogenous networks
- Sometimes not even clear what operating system is running on the (reinstalled) remote machine

Introduction to Ansible

Ansible is a lightweight way of automating, provisioning and configuration management

It can run in ad-hoc mode (commandline) and more complex scripts called playbooks

Defines a control machine to send commands to multiple target machines in parallel

Target machines retrieve them via SSH and executes these instructions in idempotent fashion

No need for a client running on target host other than SSH and python environment

Abstracts concrete commands/OS specifics away into modules like `package`, `copy`, `fetch`, `lineinfile`, etc.

Bootstrapping Ansible: BSD specifics

- Install the `sysutils/ansible` port/package on the control machine
- Create an inventory file with hosts to manage:

```
[freebsd:vars]
ansible_python_interpreter=/usr/local/bin/python2.7

[mygroup]
bsdhost1
```

- Install Python 2.7 on the remote machines:

```
ansible -m raw -a "pkg install -y python27" bsdhost1
```

Example Playbook

```
$ cat ssh-access.yml
- name: "Allow {{user_id}} to log in via SSH"
  gather_facts: false
  hosts: '{{ host }}'
  tasks:
    - name: Adding the user {{user_id}} to the AllowUsers line in sshd_config
      replace:
        backup: no
        dest: /etc/ssh/sshd_config
        regexp: '^(AllowUsers(?!.*\b{{ user_id }}\b).*)$'
        replace: '\1 {{ user_id }}'

    - name: Restarting SSH service
      service:
        name: sshd
        state: restarted
```

Example Playbook

```
$ cat ssh-access.yml
- name: "Allow {{user_id}} to log in via SSH"
  gather_facts: false
  hosts: '{{ host }}'
  tasks:
    - name: Adding the user {{user_id}} to the AllowUsers line in sshd_config
      replace:
        backup: no
        dest: /etc/ssh/sshd_config
        regexp: '^(AllowUsers(?!.*\b{{ user_id }}\b).*)$'
        replace: '\1 {{ user_id }}'

    - name: Restarting SSH service
      service:
        name: sshd
        state: restarted
```

Execute with:

```
$ ansible-playbook -Kb ssh-access.yml -e 'host=bsdhost1 user_id=joe'
```

Things Ansible does for you

Things Ansible does for you

- Takes away tedious, repetitive tasks, so you can focus on the important bits.
- Repetitions, iterations, and loops are comparably simple (`with_items`)
- Obscure options are abstracted away
- Do multiple things at once: create/fetch/copy a file, change owner, and permissions
- Newer versions suggest modules instead of you using `command`/`shell`/`sed` everywhere:

```
[WARNING]: Consider using template or lineinfile module rather than running
sed
```

- A `setup` step collects a lot of info for you and presents it as ready-to-use variables

Setup & Variables

- A problem in shell scripts is getting certain machine information other than the one in $ENV
- No central registry or information repository to query in the OS
- Often results in a lot of `greping` and `awking` to get what you want
- Ansible can run a `setup` task that gathers so called **facts** from a target machine
- Facts: Information stored in JSON format and available as `{{variables}}` in the playbook
- Includes: Network information, hardware, date & time, partitions, OS distribution and kernel, environment variables, SSH hostkeys, etc.

Do multiple things in one command

In Ansible:

```
- name: "Create User foo"
  user:
    name: foo
    home: /usr/home/foo
    shell: /bin/csh
    generate_ssh_key: yes
    ssh_key_type: 25519
    createhome: yes
```

In Shell:

```
$ sudo pw useradd -n foo -s /bin/csh -m
$ sudo -u foo ssh-keygen -t ed25519 \
 -f /home/foo/.ssh/ed25519
```

Do multiple things in one command

In Ansible:

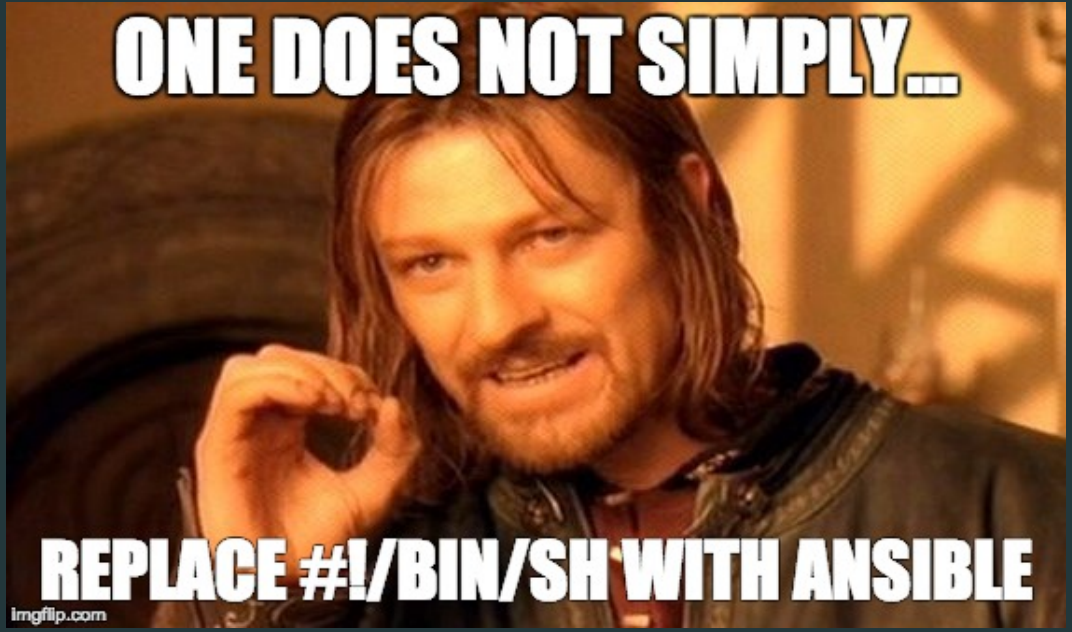
```
- name: "Create User foo"
  user:
    name: foo
    home: /usr/home/foo
    shell: /bin/csh
    generate_ssh_key: yes
    ssh_key_type: 25519
    createhome: yes
```

In Shell:

```
$ sudo pw useradd -n foo -s /bin/csh -m
$ sudo -u foo ssh-keygen -t ed25519 \
 -f /home/foo/.ssh/ed25519
```

Other module options:

- ssh_key_bits
- ssh_key_comment
- ssh_key_file
- ssh_key_passphrase


ONE DOES NOT SIMPLY...
REPLACE #!/BIN/SH WITH ANSIBLE
imgflip.com

Making Playbooks work more like shell scripts

Normal invocation:

```
$ ansible-playbook myplaybook.yml
```

Should work more like a shell script with the interpreter in the first line:

```
$ head -n 1 myplaybook.yml
#!/usr/local/bin/ansible-playbook
$ chmod +x myplaybook.yml
./myplaybook.yml
```

What does FreeBSD need to work well with Ansible?

- Better setup results (compare to Linux)
- Continued vendor awareness about it's existence and integration
- Modules for jails, bhyve, geom/geli, other BSD specifics
- The usual: patches, bugfixes, suggestions
- Works surprisingly well despite the above

Where FreeBSD Scores in Relation to Automation Software like Ansible

- Reliable network interface names
- Ubuntu 14.04 and 16.04 gave us these on Dell C6320/C6220: `enf0s0`, `eno1`, `em1`, `p1p1`
  - Can be disabled via symlink
  - But why this naming in the first place?
  - FreeBSD uses driver/manufacturer name + number and does not change it!
- Stable API/ABIs between major versions for programs to attach to and use
- Powerful and easy to use ports/packages system

Ansible will not. . .

- Reduce the number of lines you have in a playbook vs. a shell script
- Make writing playbooks easy, you'll likely swear a lot about YAML syntax when starting out
- Run as fast without some config tuning for hundreds of hosts
  - increase `forks` setting in `ansible.cfg`
  - experiment with `strategy: free` to not have hosts wait for each other
- magically convert your scripts into playbooks, rewriting from scratch is more likely
- copy files from target machine to somewhere else on target machine
  - Always starts from the control machine, no client on target
  - Advantage of Ansible, but also it's greatest weakness
  - The `synchronize` module helps sometimes

Tips for converting your existing shell scripts

- Go over it line by line and look for equivalents in Ansible's list of all modules[1]
- Divide it up into functional parts (create users, install software, configuration, etc)
- Start with hard-coded values and replace with variables when working
- Test playbooks repeatedly to see that idempotency is actually enforced and working
- `name` each step to recognize which one currently executes
- Use loops and `with_items` often, just like in the shell script
- Make use of the `template` module to provision your own config files
- Check return codes, just like `$?`, don't assume everything magically works
- Don't gather facts if you don't use variables in your (mostly short) playbooks

[1] `docs.ansible.com/ansible/latest/list_of_all_modules.html`

Conclusion

Summary

- Ansible is relatively easy to get started with
- The more tasks you repeatedly execute, the more likely it is you will benefit from Ansible
- No silver bullet, writing playbooks takes time, YAML isn't that user-friendly
- Once you have them thought, huge time-saver through parallel execution
- Keep in mind the clientless architecture and central deployment machine

Special Thanks

Thanks to Verisign, especially Chris Gordon & Vincent (Rick) Miller

Special Thanks

Thanks to Verisign, especially Chris Gordon & Vincent (Rick) Miller
The whole organizing team and Program Committee for vBSDcon!

Special Thanks

Thanks to Verisign, especially Chris Gordon & Vincent (Rick) Miller
The whole organizing team and Program Committee for vBSDcon!
Allan Jude for motivating me to submit in the first place

Special Thanks

Thanks to Verisign, especially Chris Gordon & Vincent (Rick) Miller
The whole organizing team and Program Committee for vBSDcon!
Allan Jude for motivating me to submit in the first place

My university for letting me ~~play~~ work with expensive hardware and trusting me not to break things

Special Thanks

Thanks to Verisign, especially Chris Gordon & Vincent (Rick) Miller
The whole organizing team and Program Committee for vBSDcon!
Allan Jude for motivating me to submit in the first place

My university for letting me ~~play~~ work with expensive hardware and trusting me not to break things

You for listening!

Questions?