

ANSIBLE

ANSIBLE BEST PRACTICES: THE ESSENTIALS

Magnus Glantz, Senior Solution Architect, Cloud | Infra
Red Hat

About Me

- Worked with Puppet for 5 years but is now starting to regret those years

GitHub/freenode: mglantz

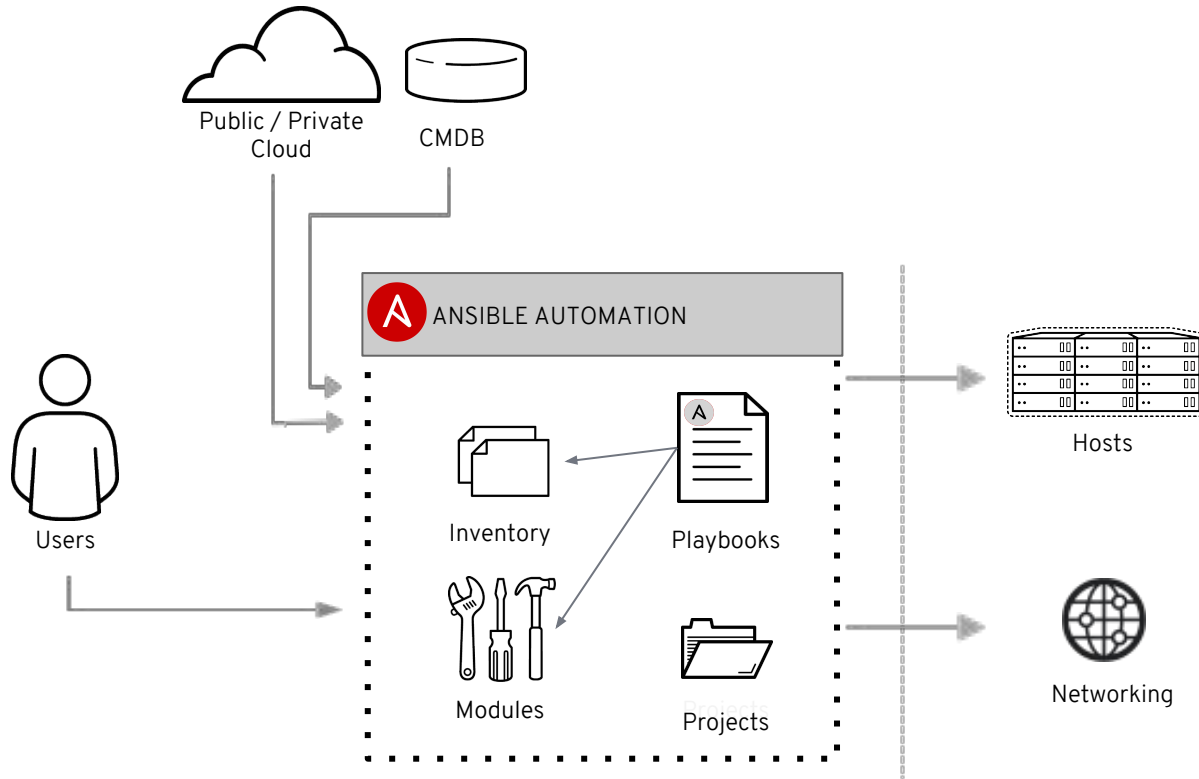
<http://github.com/mglantz/presentations>

About Me

- I've been using Ansible for a while but is now starting to

GitHub/freenode: mglantz

<http://github.com/mglantz/presentations>





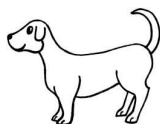
THE ANSIBLE WAY

Principal 1 - COMPLEXITY KILLS PRODUCTIVITY

ANSIBLE

Now

We need to automate



No problem. I'm Phil. I'll have this automated in a bit.



5 years later



Phil wrote that, good luck, he got killed in the great dog-regex war of 2019.



Heey... Do you know what

```
^(?:((?:0?[13578]1[02])(M-|\. )31)\1|((?:0?[13-9]1[0-2])(M-|\.)(?:29|30)\2))((?:1[6-9][2-9]d)?\d{2})$|^((?:0?2(M-|\. )29\3(?:((?:1[6-9][2-9]d)?(?:0[48][2468][048][13579][26])|((?:16[2468][048][3579][26])00))))$|^((?:0?[1-9])|((?:1[0-2])(M-|\.)(?:0?[1-9]1\d|2[0-8])\4(?:1[6-9][2-9]d)?\d{2}))$
```

...does?

Principal 2 - OPTIMIZE FOR READABILITY

```
1  - name: Install Tomcat Application server and deploy sample Java app
2  hosts: all
3  tasks:
4    - name: Ensure tomcat is installed
5      yum:
6        name: tomcat
7        state: present
8    - name: Ensure tomcat service is enabled and started
9      service:
10     name: tomcat
11     enabled: yes
12     state: started
13   - name: Download and deploy Java application
14     get_url:
15       url: https://tomcat.apache.org/tomcat-6.0-doc/appdev/sample/sample.war
16       dest: /var/lib/tomcat/webapps/sample.war
17       mode: 0777
```

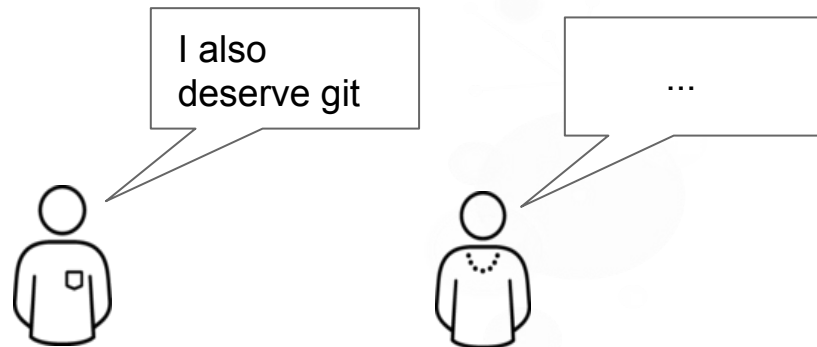
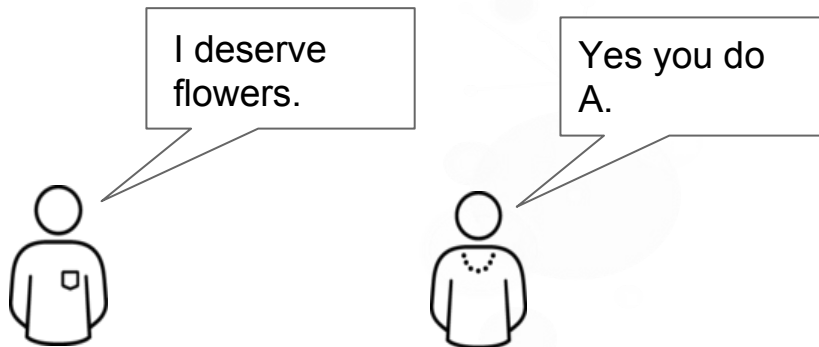
Principal 3 - THINK DECLARATIVELY

ANSIBLE

Ansible is a desired state engine by design. If you're trying to "write code" in your plays and roles, you're setting yourself up for failure. Our YAML-based playbooks were never meant to be for programming.

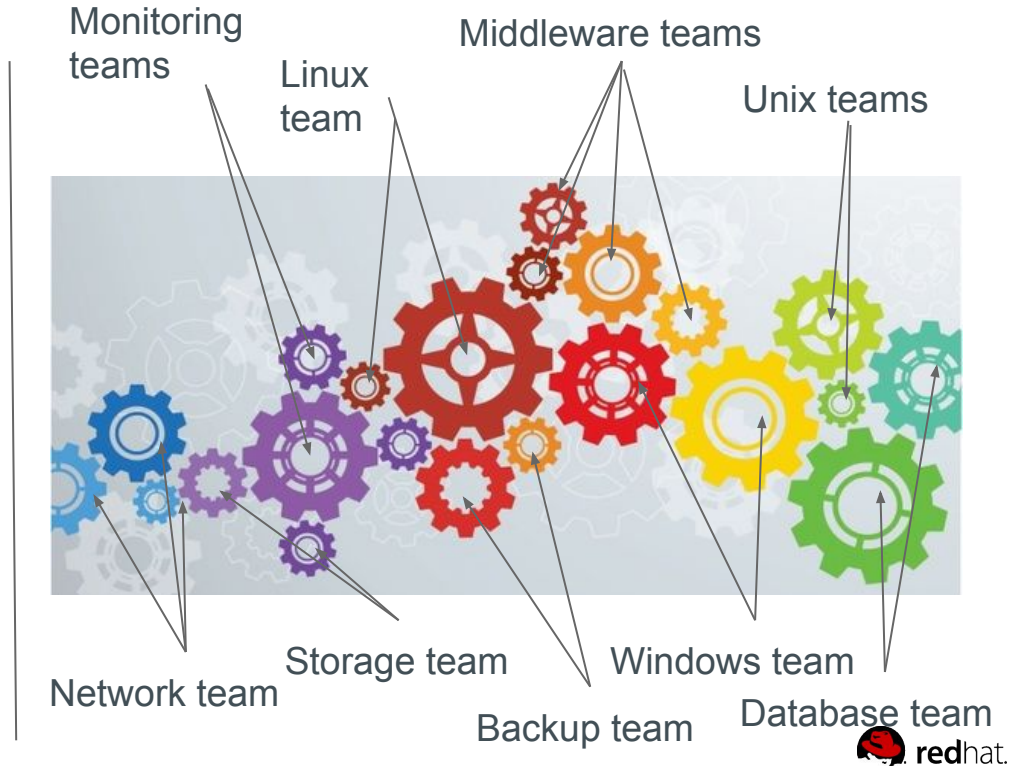
Treat your Ansible content like code

- Version control your Ansible content
- Start as simple as possible and iterate
 - Start with a basic playbook and static inventory
 - Refactor and modularize later



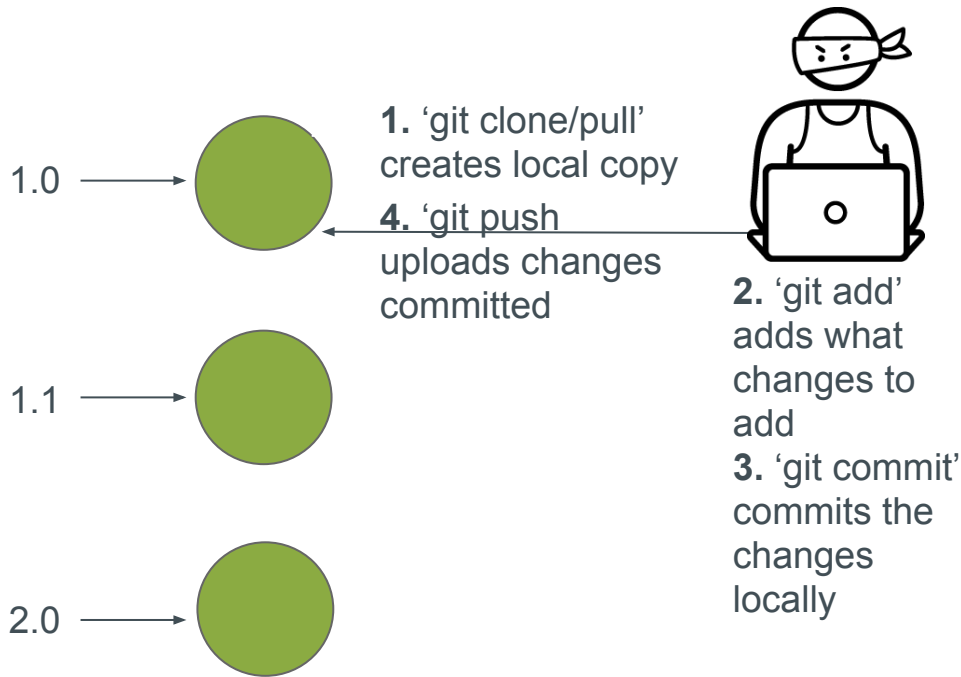
Treat your Ansible content like code

1. **Ansible** doesn't require version control
2. **When** you scale out your Ansible usage (aka. automate all things) you'll have many different teams collaborating
3. **Version control** was invented to solve common collaboration challenges
4. **Git** has earned its worldwide popularity the hard way and is in the core of many of the world's most popular collaboration services and products



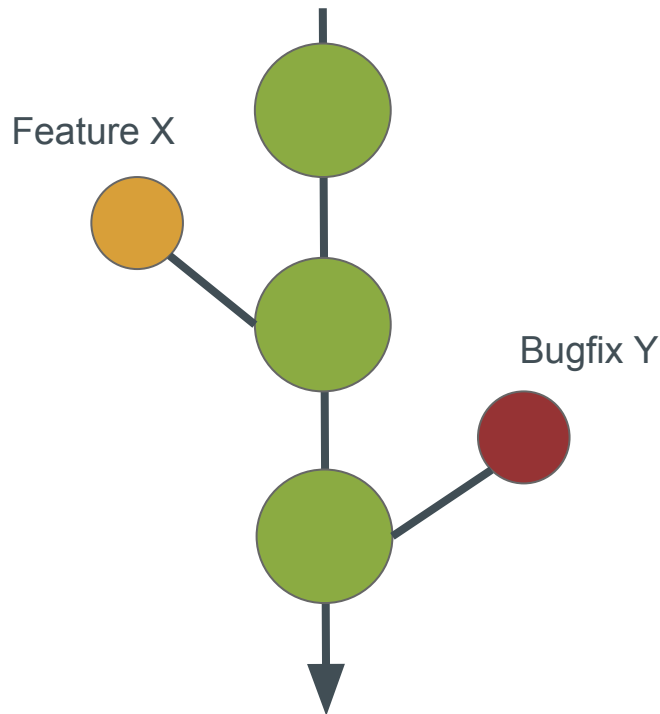
Example: Version control

1. **A** git repository stores files
2. **Access** controls are specific to repositories
3. **All** changes to all files are tracked
4. **When** you want to make a change to a file you first make a local copy of the repository which is stored on your computer, you then change the file locally, commit the change locally and then go ahead and tell git to copy this local change to the repository.

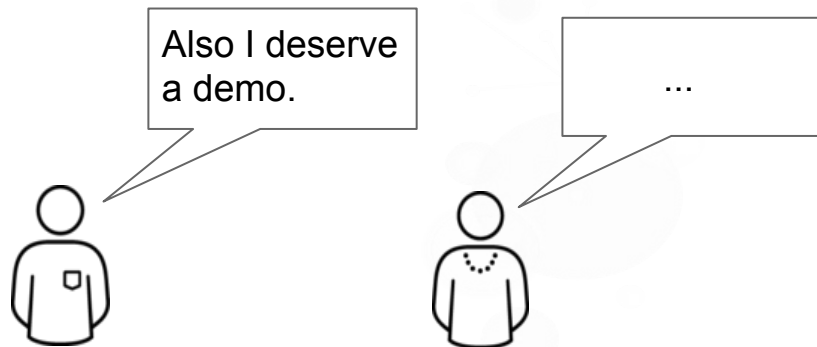


Example: GitHub workflow

1. **Does not** require GitHub, the workflow model is just called that
2. **A** very simple workflow
3. **Master** branch is always possible to release
4. **Branches** are where you develop and test new features and bugfixes.
5. **Yes**, I wrote test. If you do not test your Ansible code you cannot keep the master branch releasable and this all fails.



Treat your Ansible content like code



Do It with Style

- Create a style guide for developers
- Consistency in:
 - Tagging
 - Whitespace
 - Naming of Tasks, Plays, Variables, and Roles
 - Directory Layouts
- Enforce the style
- Check out `ansible-lint` or `molecule`

I deserve
another demo.



Implement a test framework for playbooks

A basic framework for Ansible testing is:

- Verify correct syntax with
 - a. `ansible-playbook --syntax-check your-playbook.yml`
- Verify style for bad practices and behaviour that could potentially be improved
 - a. `ansible-lint your-playbook.yml`
- Do a dry-run of the playbook and ensure it completes without failures
- Run your playbook or role and ensure it completes without failures.
- Run your playbook or role again and ensure that no changes are reported, this ensures playbook idempotency, a key feature of Ansible.
- Query your application's API or do another external test of it's functionality.
- Implement your testing framework into a CI/CD pipeline for your playbooks

Read more

<https://github.com/mglantz/ansible-roadshow/tree/master/labs/lab-9>

```
basic-project
├── inventory
│   ├── group_vars
│   │   └── web.yml
│   ├── host_vars
│   │   └── db1.yml
│   └── hosts
└── site.yml
```



```
myapp
├── roles
│   ├── myapp
│   │   ├── tasks
│   │   │   └── main.yml
│   │   └── ...
│   ├── nginx
│   │   └── ...
│   └── proxy
│       └── ...
└── site.yml
```

myapp

├─ config.yml

├─ provision.yml

├─ roles

│ └─ requirements.yml

└─ site.yml

Give inventory nodes human-meaningful

EXHIBIT A

```
10.1.2.75  
10.1.5.45  
10.1.4.5  
10.1.0.40
```

```
w14301.example.com  
w17802.example.com  
w19203.example.com  
w19304.example.com
```



EXHIBIT B

```
db1 ansible_host=10.1.2.75  
db2 ansible_host=10.1.5.45  
db3 ansible_host=10.1.4.5  
db4 ansible_host=10.1.0.40
```

```
web1 ansible_host=w14301.example.com  
web2 ansible_host=w17802.example.com  
web3 ansible_host=w19203.example.com  
web4 ansible_host=w19203.example.com
```

Group hosts for easier inventory selection and less conditional tasks -- the more groups the better.

WHAT

```
[db]
db[1:4]
```

```
[web]
web[1:4]
```

```
db1 = db, east, dev
```

WHERE

```
[east]
db1
web1
db3
web3
```

```
[west]
db2
web2
db4
web4
```

WHEN

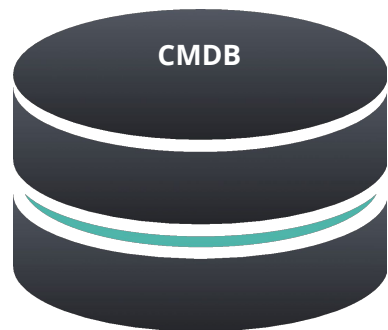
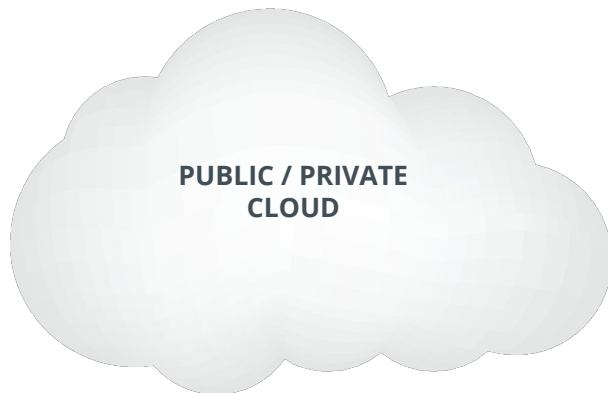
```
[dev]
db1
web1
```

```
[test]
db3
web3
```

```
[prod]
db2
web2
db4
web4
```

Use a single source of truth if you have it -- even if you have multiple sources, Ansible can unify them.

- Stay in sync automatically
- Reduce human error



The world is flat - Proper variable naming can make plays more readable and avoid variable name conflicts

- Use descriptive, unique human-meaningful variable names
- Prefix role variables with its “owner” such as a role name or package

```
apache_max_keepalive: 25
apache_port: 80
tomcat_port: 8080
```

```
- name: Clone student lesson app for a user
host: nodes
tasks:
  - name: Create ssh dir
    file:
      state: directory
      path: /home/{{ username }}/.ssh

  - name: Set Deployment Key
    copy:
      src: files/deploy_key
      dest: /home/{{ username }}/.ssh/id_rsa

  - name: Clone repo
    git:
      accept_hostkey: yes
      clone: yes
      dest: /home/{{ username }}/exampleapp
      key_file: /home/{{ username }}/.ssh/id_rsa
      repo: git@github.com:example/apprepo.git
```

EXHIBIT A

- Embedded parameter values and repetitive home directory value pattern in multiple places
- Works but could be more clearer and setup to be more flexible and maintainable

```
- name: Clone student lesson app for a user
host: nodes
vars:
  user_home_dir: /home/{{ username }}
  user_ssh_dir: "{{ user_home_dir }}/.ssh"
  deploy_key: "{{ user_ssh_dir }}/id_rsa"
  app_dir: "{{ user_home_dir }}/exampleapp"
tasks:
  - name: Create ssh dir
    file:
      state: directory
      path: "{{ user_ssh_dir }}"

  - name: Set Deployment Key
    copy:
      src: files/deploy_key
      dest: "{{ deploy_key }}"

  - name: Clone repo
    git:
      dest: "{{ app_dir }}"
      key_file: "{{ deploy_key }}"
      repo: git@github.com:example/exampleapp.git
      accept_hostkey: yes
      clone: yes
```

EXHIBIT B

- Parameters values are set thru values away from the task and can be overridden.
- Human meaningful variables “document” what’s getting plugged into a task parameter
- More easily refactored into a role

Use native YAML syntax to maximize the readability of your plays

- Vertical reading is easier
- Supports complex parameter values
- Works better with editor syntax highlighting in editors



Noooooooooooooooooooo

- name: install telegraf
yum: name=telegraf-{{ telegraf_version }} state=present update_cache=yes disabled=0
notify: restart telegraf
- name: configure telegraf
template: src=telegraf.conf.j2 dest=/etc/telegraf/telegraf.conf
- name: start telegraf
service: name=telegraf state=started enabled=yes



Better, but no

```
- name: install telegraf
  yum: >
    name=telegraf-{{ telegraf_version }}
    state=present
    update_cache=yes
    disable_gpg_check=yes
    enablerepo=telegraf
  notify: restart telegraf

- name: configure telegraf
  template: src=telegraf.conf.j2 dest=/etc/telegraf/telegraf.conf

- name: start telegraf
  service: name=telegraf state=started enabled=yes
```



Yeeeessss!

```
- name: install telegraf
  yum:
    name: telegraf-{{ telegraf_version }}
    state: present
    update_cache: yes
    disable_gpg_check: yes
    enablerepo: telegraf
  notify: restart telegraf

- name: configure telegraf
  template:
    src: telegraf.conf.j2
    dest: /etc/telegraf/telegraf.conf
  notify: restart telegraf

- name: start telegraf
  service:
    name: telegraf
    state: started
    enabled: yes
```

Names improve readability and user feedback

- Give all your playbooks, tasks and blocks brief, reasonably unique and human-meaningful names

EXHIBIT A

```
- hosts: web
  tasks:
    - yum:
        name: httpd
        state: latest

    - service:
        name: httpd
        state: started
        enabled: yes
```

```
PLAY [web]
*****

TASK [setup]
*****
ok: [web1]

TASK [yum]
*****
ok: [web1]

TASK [service]
*****
ok: [web1]
```

EXHIBIT B

```
- hosts: web
  name: install and start apache
  tasks:
    - name: install apache packages
      yum:
        name: httpd
        state: latest

    - name: start apache service
      service:
        name: httpd
        state: started
        enabled: yes
```

```
PLAY [install and start apache]
*****

TASK [setup]
*****
ok: [web1]

TASK [install apache packages]
*****
ok: [web1]

TASK [start apache service]
*****
ok: [web1]
```

Focus avoids complexity

- Keep plays and playbooks focused. Multiple simple ones are better than having a huge single playbook full of conditionals
- Follow Linux principle of do one thing, and one thing well

Clean up your debugging tasks

- Make them optional with the verbosity parameter so they're only displayed when they are wanted.

```
- debug:  
  msg: "This always displays"  
  
- debug:  
  msg: "This only displays with ansible-playbook -vv+"  
  verbosity: 2
```

Don't just start services -- use smoke tests

```
- name: check for proper response
  uri:
    url: http://localhost/myapp
    return_content: yes
  register: result
  until: '"Hello World" in result.content'
  retries: 10
  delay: 1
```

Use command modules sparingly

- Use the run `command` modules like `shell` and `command` as a last resort
- The `command` module is generally safer it escapes `<> |` a.s.o.
- The `shell` module should only be used for I/O redirect

Always seek out a module first

NO!

- name: add user
 command: useradd appuser
- name: install apache
 command: yum -y install httpd
- name: start apache
 shell: |
 systemctl start httpd && systemctl enable httpd

Yes :-)

- name: add user
 user:
 name: appuser
 state: present
- name: install apache
 yum:
 name: httpd
 state: latest
- name: start apache
 service:
 name: httpd
 state: started
 enabled: yes

Still using command modules a lot?

```
- hosts: all
  vars:
    cert_store: /etc/mycerts
    cert_name: my cert
  tasks:
    - name: check cert
      shell: certify --list --name={{ cert_name }} --cert_store={{ cert_store }} | grep "{{ cert_name }}"
      register: output

    - name: create cert
      command: certify --create --user=chris --name={{ cert_name }} --cert_store={{ cert_store }}
      when: output.stdout.find(cert_name) != -1
      register: output

    - name: sign cert
      command: certify --sign --name={{ cert_name }} --cert_store={{ cert_store }}
      when: output.stdout.find("created") != -1
```

Develop your own module

```
- hosts: all
  vars:
    cert_store: /etc/mycerts
    cert_name: my cert
  tasks:
    - name: create and sign cert
      certify:
        state: present
        sign: yes
        user: chris
        name: "{{ cert_name }}"
        cert_store: "{{ cert_store }}"
```

- Understandable by non-technical people
- CRUD (Create, read, update and delete)

Separate provisioning from deployment and configuration tasks

```
acme_corp/  
├── configure.yml  
├── provision.yml  
└── site.yml
```

```
$ cat site.yml  
---  
- import_playbook: provision.yml  
- import_playbook: configure.yml
```

Jinja2 is powerful but you needn't use all of it

- Templates should be simple:
 - Variable substitution
 - Conditionals
 - Simple control structures/iterations
 - Design your templates for your use case, not the world's
- Things to avoid:
 - Anything that can be done directly in Ansible
 - Managing variables in a template
 - Extensive and intricate conditionals
 - Conditional logic based on embedded hostnames
 - Complex nested iterations

What did we say about complexity?



Careful when mixing manual and automated configuration (Or even different automation frameworks...)

- Label template output files as being generated by Ansible

```
{{ ansible_managed | comment }}
```

Keep in mind

- Like playbooks -- keep roles purpose and function focused
- Use a `roles/` subdirectory for roles developed for organizational clarity in a single project
- Follow the Ansible Galaxy pattern for roles that are to be shared beyond a single project
- Limit role dependencies

Tricks and tips

- Use `ansible-galaxy init` to start your roles...
- ...then remove unneeded directories and stub files
- Use `ansible-galaxy` to install your roles -- even private ones
- Use a roles files (i.e. `requirements.yml`) to manifest any external roles your project is using
- Either peg a role to a specific version such as a tag or commit to help with Life Cycle Management challenges, or don't...

Command line tools have their limitations

- Coordination across a distributed teams & organization...
- Controlling access to credentials...
- Track, audit and report automation and management activity...
- Provide self-service or delegation...
- Integrate automation with enterprise systems...



Complexity kills productivity
Optimize for readability
Think declaratively

Thank you

Complexity

