I come to bury Ansible, not to praise it!

Daryl Tester
Handcrafted Computers

https://github.com/handcraftedcomputers/eo2025

eo2025@handcraftedcomputers.com.au

Warning!

This talk may contain:

- Large doses of opinion
- Trace amounts of fact

So What Is Ansible?

So, What? Is Ansible!

- Set of tools for “Infrastructure as Code”
- Domain Specific Language
- Agentless
- Ideally, idempotent.

Ansible model of operation

- Inventories: list of hosts to work ~~over~~ on
- Playbooks contain 1 or more plays
  - Plays contain 1 or more tasks
- Facts
- State based

So what’s my beef with Ansible?

- “Agentless”
- Cognitive overload

“It is Infrastructure as Code,
not Infrastructure as YAML!”

- Loic Tosser

Variable precedence, from least to greatest

- command line values (for example, -u my_user)
- role defaults (as defined in Role directory structure)
- inventory file or script group vars
- inventory group_vars/all
- playbook group_vars/all
- inventory group_vars/*
- playbook group_vars/*
- inventory file or script host vars
- inventory host_vars/*
- playbook host_vars/*
- host facts / cached set_facts
- play vars
- play vars_prompt
- play vars_files
- role vars (as defined in Role directory structure)
- block vars (only for tasks in block)
- task vars (only for the task)
- include_vars
- set_facts / registered vars
- role (and include_role) params
- include params
- extra vars (for example, -e "user=my_user")

Further beefs

- Global vars. Global vars everywhere
- Unless they’re vars - and then they’re constants
- Any data structure more complicated than a list
- Nested looping - inner vs outer var, zip, product
- blocks - kinda, sorta

So can we do better?

pyinfra

- Inventories and deploys (playbooks) are python
- Facts are loaded on demand
- “Requires” a posix-ish shell at the remote end
- “connectors”: how we talk to the managed node
- can run facts/operations from the command line
- Facts/operations are easily written/extended

pyinfra model of operation

- “Run” the inventory
- Connect to the inventory hosts
- Phase 1: For each host (in parallel):
  - Run the deploy (facts as necessary)
  - Gather operations
- Phase 2: For each host (in parallel):
  - Execute operations in lockstep

```
##  file: inventory.py

web_servers = ["web-01", "web-02", "web-03"]
db_server = ["db-01"]
```

```
##  file: deploy.py

from pyinfra import host, operations

operations.apt.packages(
    name="Install debugging packages",
    packages=["python3-bpfcc", "tcpdump"],
    update=True,
)

if "db_server" in host.groups:
    operations.apt.packages(
        name="Install postgres server",
        packages=["postgresql-server"],
    )

if "web_servers" in host.groups:
    operations.apt.packages(
        name="Install nginx",
        packages=["nginx"],
    )
```

How do I run thee?

```
$ pyinfra -y inventory.py deploy.py
```

A real world example

Certificate update process

- Archive the existing certificates (in case of rollback)
- Update the certificates in an atomic fashion
- Reload any dependent services using those certificates

And now a brief word from our atomic overlords

```
##  Inventory file, EO2025

inventory = (
    [
        # Host name, local data
        ('host1', {'ssh_hostname': 'host1.example.com.' }),
        ('host2', {'ssh_hostname': 'host2.example.com.' }),
        ('host3', {'ssh_hostname': 'host3.example.com.' }),
    ],

    # Group data
    {
        'ssh_user':              'ansible',
        'ssh_key':               '/home/XXXXX/.ssh/ansible',
        'ssh_known_hosts_file': '/etc/ssh/ssh_known_hosts',
        'acme_path':             '/home/XXXXX/.acme.sh',
        'acme_domains':          'example.com,goodcorp.org',        # Y U no list?
        # https://github.com/paramiko/paramiko/issues/1984
        ##  'ssh_paramiko_connect_kwargs': {
        ##      'disabled_algorithms': {
        ##          'pubkeys': ['rsa-sha2-512', 'rsa-sha2-256']
        ##      }
        ##  },
    },
)
```

```
$ pyinfra inventory.py debug-inventory
--> Loading config...
--> Loading inventory...

[host1]
--> Groups: inventory
--> Data:
{
    "ssh_user": "ansible",
    "ssh_key": "/home/XXXXX/.ssh/ansible",
    "ssh_known_hosts_file": "/etc/ssh/ssh_known_hosts",
    "acme_path": "/home/XXXXX/.acme.sh",
    "acme_domains": "example.com,goodcorp.org",
    "ssh_hostname": "host1.example.com."
}

[host2]
--> Groups: inventory
--> Data:
{
    "ssh_user": "ansible",
    "ssh_key": "/home/XXXXX/.ssh/ansible",
    "ssh_known_hosts_file": "/etc/ssh/ssh_known_hosts",
    "acme_path": "/home/XXXXX/.acme.sh",
    "acme_domains": "example.com,goodcorp.org",
    "ssh_hostname": "host2.example.com."
}
...
```

```
##  Deploy script, EO2025

def deploy_certs() -> None:
    config.SUDO = True
    reload_svc = set()
    # Loop over each supplied domain
    for domain in host.data.acme_domains.split(','):         # Big Oof #1
        acme_path = path.join(host.data.acme_path, domain)
        # Create combined cert (key + full chain).
        # Big Oof #2
        file_list = [path.join(acme_path, fname) for fname in [domain + '.key', 'fullchain.cer']]
        key_cert = concat_files(*file_list)
        # Update postfix
        if update_cert('postfix', key_cert, '/etc/postfix/certs', domain,
                'root', 'postfix', '400'):
            reload_svc.add('postfix')
        key_cert.seek(0, 0)     # Rewind
        # Update lighttpd
        if update_cert('lighttpd', key_cert, '/etc/lighttpd/certs', domain,
                'root', 'www-data', '440'):
            reload_svc.add('lighttpd')
        key_cert.seek(0, 0)     # Rewind
        # nginx is an exercise left for the reader
    # Reload services as/if required.
    for svc in reload_svc:
        reload_service(name='reloading %s' % svc, service_name=svc)

deploy_certs()
```

```
##  Update remote certificate *only* if required, atomically, race-free, and potentially
##  archived.
##  returns True if cert was updated (indicating a subsystem reload required), False otherwise.

def update_cert(
        subsystem: str,          # subsystem - e.g. postfix, lighttpd
        cert: file,              # new combined cert
        cert_dir: str,           # path to subsystem's cert directory
        domain: str,             # name of cert's domain
        owner: str,              # owner of cert
        group: str,              # group owner of cert
        file_mode: str,          # file permissions
        archive: bool = True     # archive old cert
) -> bool:
    # If cert_dir doesn't exist, abort.
    if host.get_fact(fact.Directory, cert_dir) is None:
        return False
    # If cert doesn't exist, abort (None if doesn't exist, False if not a file)
    cert_path = path.join(cert_dir, domain + '.pem')
    if not host.get_fact(fact.File, cert_path):
        return False

    ...
```

```
# If sha1's match, no update required.
remote_sha1 = host.get_fact(fact.Sha1File, cert_path)
if remote_sha1 is None:     # Hmmm
    logger.warning('%s: %s: unable to obtain hash of %s', host.name, subsystem, cert_path)
    return False
local_sha1 = hashlib.sha1()
local_sha1.update(cert.read())
local_sha1 = local_sha1.hexdigest()
if remote_sha1 == local_sha1:
    return False

...
```

```
##  All checks passed, cert is ripe for upgrading in atomic (nuclear?) fashion.

# Upload cert to temporary (sha1 hash) file name.
files.put(name='%s: upload cert' % subsystem, src=cert,
        dest=path.join(cert_dir, local_sha1 + '.pem'),
        user=owner, group=group, mode=file_mode)
if archive:
    # Create archive directory if required.
    if host.get_fact(fact.Directory, path.join(cert_dir, 'old_certs')) is None:
        files.directory(path.join(cert_dir, 'old_certs'), user='root', group='root',
                mode='700')
    # Create backup of current cert (as hash) if it doesn't already exist.
    backup_pem = path.join(cert_dir, 'old_certs', remote_sha1 + '.pem')
    if host.get_fact(fact.File, backup_pem) is None:
        files.link(path=backup_pem, target=cert_path, symbolic=False)
# Move new temporary named cert into place.
mv(name='%s: move new cert into place' % subsystem,
        source=path.join(cert_dir, local_sha1 + '.pem'), dest=cert_path)

return True
```

```
##  utility functions

##  Simple operation wrapper around "cp" command.
@operation()
def cp(source: str, dest: str):
    yield StringCommand("cp", "-p", QuoteString(source), QuoteString(dest))

# We die like real Unix men!
@operation()
def mv(source: str, dest: str):
    yield StringCommand("mv", QuoteString(source), QuoteString(dest))

...
```

```
##  Reload system service, by hook or by crook.

@operation()
def reload_service(service_name: str) -> None:
    # Check for runit managed service.
    svdir = '/etc/service'      # Thank you, Debian.
    if host.get_fact(facts.files.Directory, svdir):
        runit_svc = host.get_fact(facts.runit.RunitManaged, svdir=svdir)
        if runit_svc and service_name in runit_svc:
            # THIS IS NOT SIGUSR1 - lighttpd needs special love!
            if service_name == 'lighttpd':
                # "name" no workee.
                yield from server.shell._inner(commands='sv 1 lighttpd')
            else:
                yield from runit.service._inner(service_name, running=None, reloaded=True)
            return

    # Check for systemd managed service.  Still needs a systemd guard.
    systemd_svc = host.get_fact(facts.systemd.SystemdStatus)
    if systemd_svc and '%s.service' % service_name in systemd_svc:
        # running=None means leave service state as found
        yield from systemd.service._inner(service_name, running=None, reloaded=True)
        return

    # Bad juju if we got to here ...
    raise OperationError('no matching service handler for %s' % service_name)
```

Insert live demo >here<

Pros and Cons

Pros

- I just gave you a bunch!
- Ground floor hipsterism
- ... Profit???
- Lighter, less filling

Cons

- Doesn’t have the “user community” that Ansible has.
- LLMs may be blissfully unaware.
- No AWX/Tower web like interface.
- Documentation: not terrible, sometimes great.
- Follows same state based model of Ansible

In conclusion ...

Questions?

Le Fin

(You may now safely exit the building)

No LLMs were consulted in the making of this presentation.