# Ansible, best practices.

Bas Meijer 2016 Amsterdam

# Topics in this workshop

- Ansible.
- Get organized.
- Format with style.
- Inventories and environments.
- Variable precendence.
- A vault with secrets.
- Build quality in.
- Roles in detail.
- Managing efforts.
- Benchmarking where you are.

# What is Ansible?

- A radically simple IT automation engine for:
  - cloud provisioning
  - configuration management
  - application construction and deployment
  - intra-service orchestration
  - many other IT needs

# Why Ansible?

- Simple to learn and share. Easy to audit.
- Global open source community.
- Minimal requirements.
- Secure, only SSH.
- Integrates really well with cloud, Docker, etc.
- Role-based access & delegation in Tower.

# Application Construction

- vagrant, local mode
- packer, image creation
- docker build
- cloud & bare metal provisioning
- deployment & remote execution

# Get organized

- Try to schedule provisioning and deployment as early and often as possible.
- Roll-out changes, in stages, to all environments using a CI server and Tower.
- Use the same playbooks in each environment.
- Create playground environments to test things.

# Be Idempotent

- Same operation? Same result, again & again.
- Ensure no changes unless things change.
- No uncertainty: describe desired state.
- System life cycle as transaction log, accounts for all changes done on purpose.

# Use version control

- Collaboration and preservation.
- Local validation, merge-requests, tests.
- git-flow: audit trail / 4 eyes.
- Role sharing / joy of modular playbooks.
- Galaxy & GitHub have a wealth of stuff.

# Git version control

- git init
- git add playbook.yml inventory
- git commit -m "why I made changes" -a
- git push, after you tested those changes
- echo ansible.cfg >> .gitignore

# Keep it simple

- If you can make it simple, make it simple.
- If something feels complicated, it probably is.
- Do one thing at a time, commit related changes.
- Make it readable for your succesor.
- Always define state.

# Organize your content

- Format playbooks: with native YAML style.
- Magic happens when you put your files in the right locations.
- Editors use file extensions for syntax coloring.
- Version control and Ansible go hand in hand.

# Whitespace and comments

- # comments start with a hash-mark.
- Yml indents with 2 spaces.
- White space helps readability.
- Add a blank line before vars, roles, tasks, handlers.

# Directory layout

- group_vars
- host_vars
- roles

```
ansible.cfg                 # parameters that affect running ansible
inventory/                  # an inventory defines an environment
    hosts                   # defines the hosts in an inventory
    group_vars/             # here we assign variables to particular groups
        all                 # global variables for all groups
        dbservers/          # directory for dbservers group
            secrets         # -- encrypted variables for dbservers group
            vars            # -- plaintext variables for dbservers group
        group2              # plaintext variables for group2
    host_vars/              # here we assign variables to particular hosts
        hostname1           # if systems need specific variables, put them here
        hostname2           # “”
site.yml                    # master playbook
webservers.yml              # playbook for webserver tier
dbservers.yml               # playbooke for database tier
galaxy_roles/               # roles imported from galaxy
roles/                      # in-house roles
    common/                 # this hierarchy represents a “role"
        tasks/              # 'tasks' contains the actions that implement role
            main.yml        # -- main.yml could include other files if warranted
        handlers/           # 'handlers' can be notified by tasks on change
            main.yml        # -- handlers file often defines service actions
        templates/          # files for use with the template module
            hosts.j2        # -- Jinja templates, should end in .j2
        files/              # 'files' is the start for relative paths
```

# Format your playbooks

- shebang #!
- use a name
- tags in header
- blank lines
- linebreak=80
- align args

```
#!/usr/bin/env ansible-playbook

- name: 'install.yml'           # quote names for syntax highlighting
  hosts: localhost              # scope the play appropriately
  connection: local             #
  gather_facts: False           # booleans: /^(y|yes|n|no|true|false|on|off)$/i

  tags:                         # use tags for plays, and actions
    - preparation

  vars:                         # use group_vars for environment specifics
    - url: "https://galaxy.ansible.com"  # quote when value has ':'

  tasks:                        # list tasks, but consider using a role
    - name: 'check network'     # format parameters for small terminal size
      uri:                      # the best way is to use 'Native YML' format
        url: "{{ url }}"
        method: HEAD
        return_content: no
        status_code: 200
        timeout: 60
        follow_redirects: all

    - name: 're-import roles from Galaxy'
      command: ansible-galaxy install --force -r roles/requirements.yml
```

# Tag all the things

- Tags help organizing playbooks.
- Tags can help in testing.
- You can run or skip parts of playbooks:

  --tags=only,run,these,tags

  --skip-tags=tags,to,skip

# Deployment vs configuration

- Deploy all?
- --tags
- --limit
- Ad-Hoc
- playbooks/

```
ansible-playbook -i production site.yml --limit webservers
```

```
ansible-playbook -i production site.yml --tags ntp
```

```
ansible-playbook -i acceptance dbservers.yml --tags restore
```

# Start your model with the inventory

- Hosts used in infrastructure/ cloud
- Arbitrary grouping
- Can be used for staging (vagrant, int, acc, prod).

```
[local]
127.0.0.1 ansible_connection=local

[vagrant:children]
control
dbservers
webservers

[control]
control.example.com

[dbservers]
sql.example.com

[webservers]
web.example.com

[vagrant:vars]
ansible_ssh_user=vagrant
ansible_ssh_private_key_file=~/.vagrant.d/insecure_private_key
```

# Inventories

- Ansible Tower keeps inventories and credentials securely.
- If you have another system maintaining a list of systems in your infrastructure, use it for your dynamic inventory.
- Executable inventory file must emit JSON.
- Use .ini files for static inventories.

# Stage your environments

- Use separate inventories for stages like 'test' and 'production'.
- Target an environment with the -i flag.
- Test things first in a stage environment before running them in 'production'.
- Use group_vars to manage differences.

# Import Ansible inventories

To import an existing static inventory and the accompanying host and group vars into Tower, your inventory should be in a structure that looks similar to the following (don't use subdirs):

```
inventory/
    group_vars
        mygroup
    host_vars
        myhost
    hosts
```

# Import Ansible inventories

**tower-manage inventory_import —source=inventory --inventory-name="inventory"**

```
1.076 INFO     Group "webservers" added
1.108 INFO     Host "127.0.0.1" added
1.120 INFO     Host "control" added
1.127 INFO     Host "sql" added
1.134 INFO     Host "web" added
1.158 INFO     Group "webservers" added as child of "vagrant"
1.203 INFO     Host "control" added to group "control"
1.223 INFO     Host "sql" added to group "dbservers"
1.238 INFO     Host "127.0.0.1" added to group "local"
1.257 INFO     Host "web" added to group "webservers"
1.362 INFO     Inventory import completed for "imported" (id=3) in 0.5s
```

# Variables

- All systems are not exactly alike, use variables to deal with differences.
- Separate environment/staging data from configuration.
- Use the template language Jinja2.

```
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

{% for item in groups['vagrant'] %}
{{ hostvars[item]['ansible_ssh_host'] }} {{ item }}
{% endfor %}
```

# Group variables

- Hosts are listed in the inventory under [groups]
- The group_vars directory has the configuration data in files and folders.
- Use a file for each group with its variable values.
- Check and compare group_vars files often.

# Use vars to define your values

- Roles have defaults.
- group_vars is for overrides.
- 'all' is for globals

```
---
# file: group_vars/all
# here we assign variables to all groups

ansible_user: ansible

google_nameserver: 8.8.4.4
dns_nameserver: "{{ google_nameserver }}"

pg_host: sql
pg_ip: 192.168.20.22
pg_subnet: 192.168.20.0/24

log_host: sql
log_host_ip: 192.168.20.22
```

# Use ansible-vault for secrets

- Use ansible-vault to store secrets safely.
- Use subdirs in group_vars
- vars + secrets

```
$ touch group_vars/dbservers/secrets

$ ansible-vault encrypt group_vars/dbservers/secrets
Vault password:
Confirm Vault password:
Encryption successful

$ ansible-vault edit group_vars/dbservers/secrets
Vault password:

$ cat group_vars/dbservers/secrets
$ANSIBLE_VAULT;1.1;AES256
30623164636337303064313565393361656437343739396235643861336265373138653965303866
39333063336361643533303931376330616532303666643 10a3137343233633062613533393064
31623732373933333666656665646135656637356366664623133616132383831366163623261336
64316361323730363 00a6666333361353763613261636339616262313964333935336630643063
65306365323836633838306639336230383039353035343239306432313535326633
```

# Variable precendence

| Ansible | Tower |
| --- | --- |
| role defaults | |
| dynamic inventory variables | |
| inventory variables | Tower inventory variables |
| inventory group_vars | Tower group variables |
| inventory host_vars | Tower host variables |
| playbook group_vars | |
| playbook host_vars | |
| host facts | |
| registered variables | |
| set_facts | |
| play variables | |
| play vars_prompt | (not supported in Tower) |
| play vars_files | |
| role variables and include variables | |
| block variables | |
| task variables | |
| extra variables | Job Template extra variables Job Template Survey (defaults) Job Launch extra_vars |

# ansible_ssh_user

- Don't login as root
- Don't use service account interactively
  `echo logout > ~/.bash_profile`
- Settle on become_method: sudo/su/doas
- Consider to use signed ssh keys.

```
TrustedUserCAKeys /etc/ssh/ca_key.pub
AuthorizedKeysFile /dev/null
```

# Ansible modules

- Ansible comes with hundreds of modules.
- Avoid using command as much as possible.
- Modules return JSON data that you can use.
- Modules report about state, about change.
- Modules deal with the corner cases.
- Write a module for a resource if there is none.

# Use this file: ansible.cfg

- current directory
- home directory
- /etc/ansible/ansible.cfg

Has/hides ansible-playbook options so your commands can be short.

```
[defaults]
hostfile = vagrant.ini
host_key_checking = False
ask_vault_pass = True
retry_files_enabled = False
executable = sh
remote_tmp=
log_path = /tmp/ansible_run.log

gather_facts = smart
roles_path = galaxy_roles:roles:/etc/ansible/roles

[privilege_escalation]
become_method = sudo

[ssh_connection]
scp_if_ssh = True
pipelining = True
```

# Blocks

- Error handling

```
---

- name: 'blocks.yml'
  hosts: localhost
  connection: local

  tasks:
    - block:
        - debug:
            msg: 'i execute normally'
        - command: /usr/bin/false
        - debug:
            msg: 'i never execute, cause ERROR!'
      rescue:
        - debug:
            msg: 'I caught an error'
        - command: /usr/bin/false
        - debug:
            msg: 'I also never execute :-('
      always:
        - debug:
            msg: "this always executes"
```

# Add tests to your playbooks

- --tags test
- fail early.
- keep adding test cases.

```
- name: 'verify listening on port 9200'
  wait_for:
    port: 9200
    host: "{{ inventory_hostname }}”
    timeout: 5
  tags:
       - test

- name: 'query cluster health'
  uri:
    url: "http://{{inventory_hostname}}:9200/_cluster/health?pretty"
  register: cluster
  tags:
    - test

- name: 'verify that nodes have joined the cluster'
  assert:
  that:
    - cluster.json.number_of_data_nodes != 0
  tags:
    - test
```

# Manage your dependencies

requirements.txt: extra Python packages for modules.

requirements.yml: list of extra roles to import.

```
apache-libcloud
azure
boto
ConfigParser
cs
django
docker-py
dopy
httplib2
linode-python
netaddr
psycopg2
pycrypto
pycurl
pysphere
python-keyczar
python-keystoneclient
python-novaclient
python-quantumclient
pretty
pyrax
requests
shade
wsgiref
zmq
```

```
@ roles/requirements.yml

# galaxy role
- src: hostclick.tomcat

# role in git
- src: https://github.com/bbaassssiiee/RHEL6-STIG.git
  version: devel
```

# Separate top level playbooks by role

- --limit

```
#!/usr/bin/env ansible-playbook
# master playbook
- name: 'site.yml'
  hosts: all:!local
  remote_user: "{{ ansible_user }}"

- include: dbservers.yml
- include: webservers.yml
```

```
#!/usr/bin/env ansible-playbook
# playbook for dbserver tier

- name: 'dbservers.yml'
  hosts: dbservers
  become: yes
  gather_facts: True

  roles:
    - common
    - postgres
```

# Decouple roles

- Roles are ways of automatically loading certain vars_files, tasks, and handlers based on a known file structure.
- Grouping content by roles also allows easy sharing of roles with other users.
- http://galaxy.ansible.com

```
ssh
|-- files
|   `-- bash_profile
|-- handlers
|   `-- main.yml
|-- tasks
|   `-- main.yml
`-- templates
    |-- ssh_config.j2
    `-- sshd_config.j2
```

# Roles precendence

- Separate your own roles and Galaxy roles.
- Leave room for “corporate roles”.
- Configure roles_path to search for roles.
- galaxy_roles:roles:/etc/ansible/roles

# Organization of a role

- defaults
- files
- **handlers**
- library
- meta
- **tasks/main.yml**
- **templates**
- tests
- vars

```
---

- name: 'ensure package ntp is installed'
  package:
    name: ntp
    state: latest
  tags:
    - ntp

- name: 'build and write /etc/ntp.conf file'
  template:
    src: ntp.conf.j2
    dest: /etc/ntp.conf
    owner: root
    group: root
    mode: 0644
  notify:
    - restart ntpd
  tags:
    - ntp
```

# Handlers for a role

- defaults
- files
- **handlers**
- library
- meta
- tasks
- templates
- tests
- vars

```
- name: restart ntpd
  service:
    name: ntpd
    state: restarted
```

# Templates for a role

- defaults
- files
- handlers
- library
- meta
- tasks
- **templates**
- tests
- vars

```
tinker panic 0
restrict 127.0.0.1
restrict default kod nomodify notrap

# NTP server pool, use 3 minimally
{% for server in ntp_servers %}
server {{ server }}
{% endfor %}

driftfile /var/lib/ntp/drift

# Avoid using the local clock
server 127.127.1.0 # local clock
fudge 127.127.1.0 stratum 10
```

# Default vars for a role

- **defaults/main.yml**
- files
- handlers
- library
- meta
- tasks
- templates
- tests
- vars

```
---

# ntp
ntp_servers:
  # use nearby ntp servers if available
  - 0.pool.ntp.org
  - 1.pool.ntp.org
  - 2.pool.ntp.org
  - 3.pool.ntp.org
```

# Bundling modules with a role

- defaults
- files
- handlers
- **library/pam.py**
- meta
- **tasks/main.yml**
- templates
- tests
- vars

```
---

- name: 'Disable accounts after three login failures in a 15-minute interval.'
  pam:
    service: "{{ item }}"
    type: auth
    control: required
    pam_module: pam_faillock.so
    before_line: auth sufficient pam_unix.so try_first_pass
    arguments: preauth silent deny=3 unlock_time=604800 fail_interval=900
    state: present
  with_items:
    - password-auth
    - system-auth
  tags:
    - pam
```

# Management

- Is the team applying best practice and common principles when building / configuring applications and environments?
- Is the team automating the deployment and promotion process for applications and environments?
- Is the team validating functional and non-functional requirements (automatically) before promoting applications to Production environments?
- Is the team providing and receiving the feedback needed to be in control of the solutions they are responsible for?
- Is the team pro-actively collaborating with other teams to integrate applications and environments and share technical knowledge?

# Benchmarks

**A good Config Management Strategy should allow to answer all of the following questions with „yes“:**

- Can I exactly reproduce any of my environments, including the version of the operating system, its patch level, the network configuration, the SW stack, the application deployed into it, and their configuration?
- Can I easily make an incremental change to any of these individual items and deploy the change to any, and all, of my environments?
- Can I easily see each change that occurred to a particular environment and trace it back to see exactly what the change was, who made it, and when?
- Can I satisfy all of the compliance regulations that I am subject to?
- Is it easy for every member of the team to get the information that they need, and to make the changes that they need to make? Or does the strategy get in the way of efficient delivery, increasing cycle time and decreasing feedback?

# Download final project

```
brew install python ansible

git clone https://github.com/bbaassssiiee/lunchbox
cd lunchbox
vagrant up --no-provision
ansible-playbook -v install.yml
ansible-playbook -v site.yml
```

Requires: VirtualBox, Vagrant

# Ansible, best practices.

Bas Meijer 2016 Amsterdam