Ansible

At Firewall Services we use ansible. And we use it a lot !

Basics

Ansible roles

Have a look at our ansible roles, we deploy a lot of stuff with it :-)

First steps

Initial configuration

Clone our ansible-roles repo

```
git clone https://git.fws.fr/fws/ansible-roles.git
cd ansible-roles
```

Create the configuration directories

Those directories will holds configurations of your hosts, groups etc.

```
# This dir will contain your hosts inventories
mkdir inventories
# This one will contain vars for individual hosts
mkdir host_vars
# This one will contain vars for group of hosts
mkdir group_vars
# Will contain SSH related stuff
mkdir ssh
```

Create an SSH key pair

The public key will have to be configured on the hosts you want to manage

```
ssh-keygen -t rsa -b 4096 -f ssh/id_rsa
```

It's advised to protect the private key with a password

Create your inventory file

This inventory will contains all the hosts you manage with ansible. You can have several inventories (eg, one per client). For example **inventories/fws.ini**. Here I create a single group of hosts named **fws**. And a single host **proxyin.fws.fr**

```
[fws]
proxyin.fws.fr
```

Setup the host to be managed

On the machine **proxyin.fws.fr**, we have to configure a few things :

- Create a user named ansible
- Grant ansible full access to the system with sudo
- Configure the public SSH key on this ansible user account

```
useradd -m ansible
mkdir ~ansible/.ssh
cat <<_EOF > ~ansible/.ssh/authorized_keys
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQCj9d6jDy0m7xtqGfR0ywyXnq0lRfqqP0TzBhvCI4rcrJaDSLyA5/mnme0TLfy6YsO
ansible@firewall-services.com
_EOF
chown -R ansible:ansible ~ansible/.ssh/
chmod 700 ~ansible/.ssh/
chmod 600 ~ansible/.ssh/authorized_keys
cat <<_EOF > /etc/sudoers.d/ansible
Defaults:ansible !requiretty
ansible ALL=(ALL) NOPASSWD: ALL
_EOF
chmod 600 /etc/sudoers.d/ansible
```

> Of course, adapt this to your own public SSH key !

Connect a first time

The first time you connect, you have to validate the SSH host key, so, let's do it once, and check everything is OK

```
ansible -m setup -i inventories/fws.ini proxyin.fws.fr
```

You should be prompted to accept the SSH key (which will be recorded in ssh/known_hosts), and ansible will output some info about your host. Your now ready to play !

Roles

Roles and their configuration

Role directories

A role is a set of instruction which describe how to install or update a functionnality. Roles are under the **roles** directory (no joke ;-)). Each role have several sub directories:

- **defaults** : contains default values for the role. These are the available variables you'll be able to set in your host_vars/ or group_vars/ to configure the service
- **tasks** : contains the action to run to configure the role
- **templates**: contains jinja2 templates which will be deployed on the host
- **files**: contains files which will be deployed (as-is, no template processing)
- **vars**: contains variables used by the role. Usualy, variables which you should change are defined in defaults. In vars are defined variables used by the role which you shouldn't have to change
- **handlers** : containers handlers (eg, how to restart services when a configuration file changed)

defaults & variables

defaults is really the most important part of a role. Check the file defaults/main.yml of a role to see which variables you can tune. For example, for the role docker (which can install docker daemon on a host)

```
docker_data_dir: /opt/docker
docker_log_driver: journald

docker_base_conf:
  data-root: /opt/docker
  log-driver: journald
  storage-driver: overlay2
  storage-opts:
```

```
    - 'overlay2.override_kernel_check=true'
docker_extra_conf: {}
# docker_extra_conf:
#   log-opts:
#     max-size: 100m
#     max-file: 5

docker_conf: "{{ docker_base_conf | combine(docker_extra_conf, recursive=True) }}"
```

This is all the variable you can set to modify how Docker will be configured. You do not have to configure everything, just set the variables for which the default value doesn't fit your need.

hosts variables

For example, if you deploy docker on the host docker.fws.fr, just create **host_vars/docker.fws.fr/vars.yml**

```
docker_extra_conf:
  data-root: '/data'
  log-driver: 'json-file'
  log-opts:
    max-size: '100m'
    max-file: '5'
  iptables: False
  group: dockeradmins
  userns-remap: default
  live-restore: True
  dns:
    - 10.118.1.1
```

groups variables

For some settings, you'll want to share them with a group of hosts (eg, the AD domain to join, or the Docker settings above, if you deploy several Docker hosts). In this case, you can create a group of host in your inventory file, for example :

```
[fws]
proxyin.fws.fr
docker1.fws.fr
```

```
docker2.fws.fr

[fws_docker:vars]
ansible_group_priority=2

[fws_docker]
docker1.fws.fr
docker2.fws.fr
```

Please, read ansible documentation if you need more detailed information on this

Now, you can create the files

- **group_vars/fws/vars.yml** : all the variables defined here will be inherited by **proxyin.fws.fr**, **docker1.fws.fr** and **docker2.fws.fr**
- **group_vars/fws_docker/vars.yml** : all the variables defined here will be inherited by **docker1.fws.fr** and **docker2.fws.fr**

With the above **ansible_group_priority**, if a variable is defined in both **fws** and **fws_docker**, the one from **fws_docker** will be used for **docker1.fws.fr** and **docker2.fws.fr**.

encrypted variables

You might need to set secret values in variables, like passwords. In this case, you do not want to store them as cleartext. Then, just use the

https://docs.ansible.com/ansible/latest/user_guide/vault.htmlansible-vault utility.

```
ansible-vault create group_vars/fws/vault.ym
```

You'll be prompted for a password to encrypt the file. The syntaxe is the same as a normal file. If you want to edit an existing vault, use instead :

```
ansible-vault edit group_vars/fws/vault.yml
```

When you run the ansible playbook, if a host requires access to variables in a vault, you'll be prompted to enter the vault password

Playbooks

Playbooks

A playbook is a yaml file which list a set of action to run, and in which order. You can create your own tasks in a playbook, but most of the time, you'll just assign roles to hosts, or group of hosts in a playbook.

You can create your playbooks where you want, for example, in a playbook subdirectory

```
mkdir playbooks
vim playbooks/fws.yml
```

Here's a real world example

```
- name: Deploy outbound proxy server
  hosts: proxyout.fws.fr
  roles:
    - repo_base
    - squid

- name: Deploy AD DC
  hosts: fws_dc
  roles:
    - system_proxy
    - repo_base
    - samba
    - letsencrypt

- name: Deploy common profiles
  hosts: fws
  roles:
    - common
    - backup
    - filebeat
```

```
- name: Deploy Proxmox hosts
  hosts: fws_pve
  roles:
    - pve

- name: Deploy databases servers
  hosts: db.fws.fr
  roles:
    - mysql_server
    - postgresql_server
```

In this example, if the playbook is ran, it'll do the following :

- On the host **proxyout.fws.fr**, it will deploy the roles **repo_base** and **squid**
- On the hosts members of group **fws_dc**, it'll deploy the roles **system_proxy**, **repo_base** , **samba** and **letsencrypt**
- On all the hosts members of the group **fws**, it'll deploy the roles **common**, **backup** and **filebeat**
- etc.

So, you have to define your playbook with what you want to do

The "common" role

This role will setup a lot of different stuff of your system. I use it on all my servers. It's tested on :

- CentOS Linux 7
- CentOS Linux 8
- CentOS Stream 8
- Debian 8
- Debian 9
- Debian 10 (and derivatives like Proxmox VE for example)

Here's the minimum variables you should set

```
# A list of trusted IP.  Will have accesso to the SSH service of all the servers for example

trusted_ip:
  - 10.11.12.13
  - 10.8.0.0/16

# Unix groups whose members will have sudo access.  The group must already exists, it won't be
created (it can be a group from LDAP or AD for example)
system_admin_groups:
  - admins

# Email address which will receive system email, those addressed to root@yourserver
system_admin_email: server-mailbox@example.org

# Some roles which uses a database will try to read mysql_server or pg_server (or fallback to
localhost if not defined)
# If you have a host dedicated to database, you can set it
mysql_server: maria.fws.fr
pg_server: postgres.fws.fr
mysql_admin_pass: "{{ vault_mysql_admin_pass }}"
pg_admin_pass: "{{ vault_pg_admin_pass }}"

# System timezone
system_tz: 'Europe/Paris'
```

> vault_mysql_admin_pass and vault_pg_admin_pass are passwords, so I do not store them in clear. Instead, they are stored in a vault and referenced here for clarity

Of course, there are a lot more variables available. You can look in roles/common/defaults. Some other roles can be pulled in as a dependency if some specific variables are set. For example, if you set the following :

```
ad_auth: True
samba_domain: acme
samba_realm: acme.com
ad_admin_pass: "{{ vault_samba_dc_admin_pass }}"
ad_access_filter:
DOM:FWS.FR:(&(objectCategory=person)(objectClass=user)(primaryGroupId=513)(memberOf:1.2.840.113
```

Then the sssd_ad_auth role will be pulled in, and your server will be joined to the domain during

the playbook (and in this example, users will only be accepted if they are a direct or indirect member of the group CN=Role_Unix,OU=Roles,DC=acme,DC=com)

You can check **roles/common/meta/main.yml** to see which roles will be pulled in as dependency

General

General concepts of FWS' roles

General

Most of our roles follow the same principles :

- Limit the required configuration : most defaults trie to auto configure themselves
- Auto generate password and other secrets when they are not defined
- Handle upgrades : we don't only manage installations, but also upgrades of existing installations
- Backup : during an upgrade, the previous installation will be backed up and archived so you can restore it in case something went wrong
- Integrate with other roles with hooks : dumps are handled by droping hooks in /etc/backup/{pre,post}.d, which will be executed by the pre-backup command if you deploy the backuup role. Let's Encrypt certificates works the same way etc.
- Keep softwares up to date : we try to maintain all the roles we're using up to date, and quickly update them when a new upstream version is released

Roles

BookStack

Status	Production
Supported distro	CentOS 7/8

This roles deploys the BookStack application

Backend server

Sample backend server

```
httpd_ansible_vhosts:
  - name: docs.example.org
    document_root: /opt/bookstack_1/app/public

bookstack_public_url: https://docs.example.org
bookstack_web_alias: False
bookstack_settings:
  AUTH_METHOD: saml2
  SAML2_NAME: EXAMPLE
  SAML2_DISPLAY_NAME_ATTRIBUTES: cn
  SAML2_EXTERNAL_ID_ATTRIBUTE: principal
  SAML2_IDP_ENTITYID: https://sso.example.org/saml/metadata
  SAML2_AUTOLOAD_METADATA: 'false'
  SAML2_IDP_SSO: https://sso.example/saml/singleSignOn
  SAML2_IDP_x509: MIICpjCCAY6gAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  SAML2_USER_TO_GROUPS: 'true'
  SAML2_GROUP_ATTRIBUTE: groups
  SAML2_REMOVE_FROM_GROUPS: 'true'
  DRAWIO: https://draw.example.org/?embed=1&proto=json&spin=1
```

Reverse proxy

Sample reverse proxy config, using the nginx role

```
nginx_vhosts:


  # BookStack
  - name: docs.fws.fr
    allowed_methods: [GET,HEAD,POST,OPTIONS,PUT,DELETE]
    csp: >-
      default-src 'self' 'unsafe-inline' blob:;
      style-src-elem 'self' 'unsafe-inline' data:;
      img-src 'self' data: blob: https://stats.example.org;
      script-src 'self' 'unsafe-inline' 'unsafe-eval' https://stats.example.org blob:;
      font-src 'self' data:;
      frame-src https://sso.example.org https://draw.example.org
    src_ip:
      - "{{ trusted_ip }}"
```

CorwdSec

Status	Production
Supported distro	CentOS 7/8

This role deploys the crowdsec agent.

CrowdSec agent

Here's a sample config for the agent. Running with local and central API enabled, using a MySQL bakend

```
cs_db_engine: mysql
cs_capi_enabled: True
cs_lapi_enabled: True
cs_lapi_src_ip:
  - 10.29.1.14 # Only the rev proxy can reach the API
cs_use_x_forwarded_for: True
cs_acquis:
  - filenames:
      - /run/g2cs/logs/syslog.log
    labels:
      type: syslog
  - filenames:
      - /run/g2cs/logs/nginx/access.log
      - /run/g2cs/logs/nginx/error.log
    labels:
      type: nginx
  - filenames:
      - /run/g2cs/logs/httpd/access.log
      - /run/g2cs/logs/httpd/error.log
    labels:
      type: apache2
```

```
  - filenames:
      - /run/g2cs/logs/zimbra/mailbox.log
    labels:
      type: zimbra


cs_parsers:
  - crowdsecurity/syslog-logs
  - crowdsecurity/geoip-enrich
  - crowdsecurity/dateparse-enrich
  - crowdsecurity/whitelists
  - crowdsecurity/sshd-logs
  - crowdsecurity/iptables-logs
  - crowdsecurity/nginx-logs
  - crowdsecurity/apache2-logs
  - crowdsecurity/http-logs
  - crowdsecurity/postfix-logs
  - crowdsecurity/postscreen-logs
  - firewallservices/zimbra-logs
  - firewallservices/lemonldap-ng
cs_scenarios:
  - crowdsecurity/ban-defcon-drop_range
  - crowdsecurity/ssh-bf
  - crowdsecurity/http-bf-wordpress_bf
  - crowdsecurity/http-generic-bf
  - crowdsecurity/http-sensitive-files
  - crowdsecurity/http-sqli-probing
  - crowdsecurity/http-xss-probing
  - crowdsecurity/http-backdoors-attempts
  - crowdsecurity/http-path-traversal-probing
  - crowdsecurity/iptables-scan-multi_ports
  - crowdsecurity/postfix-spam
  - firewallservices/zimbra-bf
  - firewallservices/lemonldap-ng-bf
```

g2cs

You can use the g2cs companion role. This small daemon can receive a syslog stream from graylog (in CEF format) and write files for crowdsec to consume

```
```

```
cs_user: g2cs
g2cs_port: 514
g2cs_src_ip:
  - 10.29.1.20 # Graylog FWS
```

CrowdSec Firewall Bouncer

You can deploy firewall bouncers with the crowdsec_firewall_bouncer role

```
# Use crowdsec local API
cs_lapi_url: https://cs.example.org/
cs_lapi_server: crowdsec.example.org
```

Reverse proxy

You can use a reverse proxy to expose your local API to your other machines. Here's a sample on a machine using the nginx role to provide the reverse proxy functionnality

```
nginx_vhosts:
  # Crowdsec Local API
  - name: cs.example.org
    proxy:
      backend: http://crowdsec.example.org:8080
    auth: False
    src_ip:
      - "{{ trusted_ip }}"
      - 10.29.0.0/16
      - 10.30.0.0/16
      - 10.99.0.0/16
      - 192.168.7.0/24
```

n8n

Status	Production
Supported distro	CentOS 8

This role deploys the n8n workflow manager.

Here's a sample configuration

```
n8n_src_ip:
  - 10.29.1.14 # Access only for the rev proxy
n8n_public_url: https://workflow.example.org/
```

And a sample reverse proxy configurtion, using the nginx role

```
nginx_vhosts:
  # n8n
  - name: workflow.example.org
    proxy:
      backend: http://n8n.fws.fr:8021
    allowed_methods: [GET,HEAD,POST,OPTIONS,PUT,DELETE,PATCH]
```