

Ansible

- [Setup Automation Host with Ansible](#)

Setup Automation Host with Ansible

Introduction

In today's dynamic network environments, keeping an accurate and up-to-date state of all the linux hosts is crucial.

This document describes the setup and use of Ansible with the flexibility of Python and the scheduling power of cron jobs.

We setup an automation pipeline which will access your linux hosts via ssh, so that they have the latest packages and configuration.

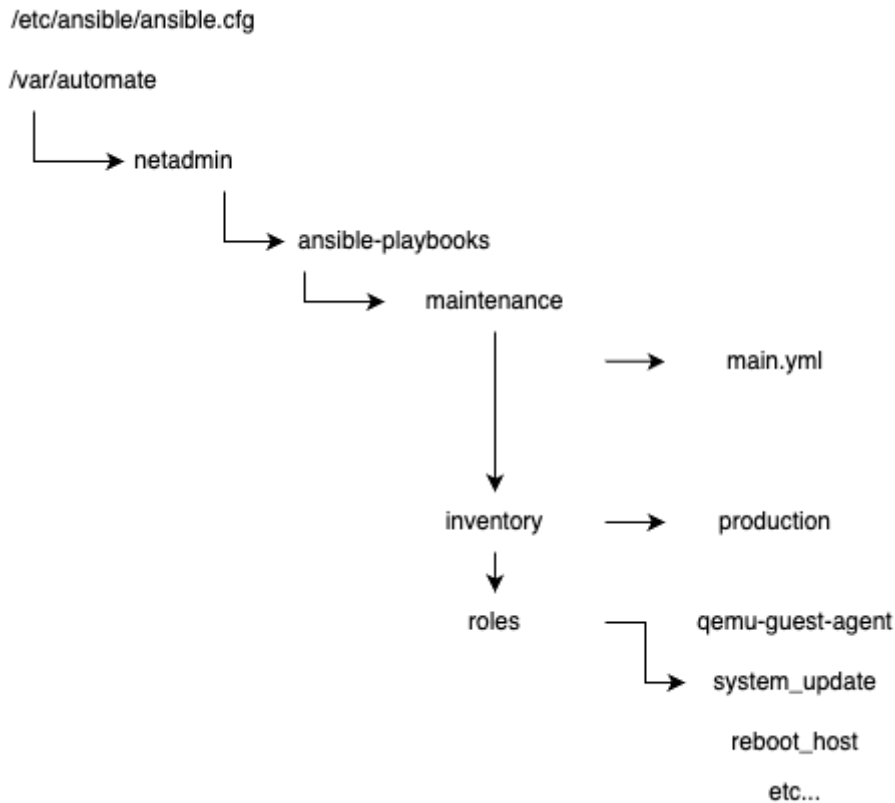
This guide will cover:

- Setup ansible and python: fetching device configurations and updating linux hosts
- Automating with cron jobs: scheduling regular updates to maintain synchronization.
- Practical Example: updating a linux host with the newest packages and kernel updates, rebooting the host after the update.

File Structure

We use the following file structure for the automation host.

In the example we have a host group called adminhosts which includes for the beginning the automation hosts itself, let's call it `automate.lan.domain.com`.



Installation

Install the required tools

- Login as the netadmin user, this user needs to be the default user on the linux host automate.lan.domain.com.
- `sudo -i`
- https://docs.ansible.com/ansible/latest/installation_guide/installation_distros.html#installing-ansible-on-ubuntu

```
sudo apt update
sudo apt install software-properties-common
sudo add-apt-repository --yes --update ppa:ansible/ansible
sudo apt install ansible
```

Create directory for ansible data

```
mkdir -p /var/automate/netadmin/ansible-playbooks/maintenance
cd /var/automate/netadmin/ansible-playbooks/maintenance
mkdir inventory roles
```

Change access rights to /var/automate

```
chgrp -R netadmin /var/automate
chmod -R 774 /var/automate
ls -l /var/automate
drwxrwxr--  4 root netadmin 4096 Mar 13 07:56 automate
```

How to add a User to the System

This will add the <newusername>, creates a new Homedirectory in /home/<newusername> and add the User to the Primary Group netadmin.

```
sudo useradd -m -s $(which bash) -g netadmin <newusername>
```

Configuration

Ansible Roles

Create the ansible role system_update

The following will create the default path structure for a role. Roles can also be used from the ansible-galaxy found here:

<https://galaxy.ansible.com>

```
cd /var/automate/netadmin/ansible-playbooks/maintenance/roles
ansible-galaxy role init system_update
```

Create hosts file in inventory folder

This hosts file lists all the hosts we want to manage with ansible. The hosts are grouped, at the moment we only have the adminservers group with the host automate.lan.domain.com. This will be extended in the future.

```
vi /var/automate/netadmin/ansible-playbooks/maintenance/inventory/production

[adminservers]
automate.lan.domain.com

[webservers]
web1.dmz.domain.com
web2.dmz.domain.com

[dbservers]
```

db1.lan.domain.com

db2.lan.domain.com

Create the main.yml file for our "maintenance" ansible-playbook

This is the master yaml file for running the ansible-playbook.

This will be the yaml config file which does describe which roles should be used. At the moment we only use "system_update" and the "qemu-guest-agent" role.

The "system_update" role will be used to update packages on the selected hosts.

The "qemu-guest-agent" role is used to install/rollout the qemu guest agent on virtual machines.

```
vi /var/automate/netadmin/ansible-playbooks/maintenance/main.yml
```

```
---
```

```
# file: main.yml
```

```
- hosts: all
  gather_facts: yes
```

```
- hosts: adminservers:!automate.lan.gecloud.ch
  roles:
    - system_update
    - qemu-guest-agent
```

```
- hosts: webservers
  roles:
    - system_update
    - qemu-guest-agent
```

```
- hosts: appservers
  roles:
    - system_update
    - qemu-guest-agent
```

```
- hosts: dbservers
  roles:
    - system_update
    - qemu-guest-agent
```

What is the "system_update" role doing ?

The goal is to update packages in the following order:

- Update all packages to their latest version
- Check if a reboot is required
- Reboot if required

Create main.yml file:

```
vi /var/automate/netadmin/ansible-playbooks/maintenance/roles/common/tasks/main.yml

---

- include_tasks: debian.yml
  when: ansible_os_family == "Debian"
```

Create debian.yml:

```
vi /var/automate/netadmin/ansible-playbooks/maintenance/roles/common/tasks/debian.yml

- name: Update all packages to their latest version
  ansible.builtin.apt:
    update_cache: true
    autoremove: true
    cache_valid_time: 3600
    name: "*"
    state: latest

- name: Check if a reboot is required
  stat:
    path: /var/run/reboot-required
  register: reboot_required_file

- name: Reboot if required
  reboot:
    msg: "Reboot initiated by Ansible due to kernel updates"
    connect_timeout: 5
    reboot_timeout: 300
    pre_reboot_delay: 0
    post_reboot_delay: 30
    test_command: uptime
  when: reboot_required_file.stat.exists == true
```

Create the ansible.cfg (See comments for parameter description):

```
sudo vi /etc/ansible/ansible.cfg

[defaults]
# Ansible enables host key checking by default.
# Checking host keys guards against server spoofing and man-in-the-middle attacks, but it does
require some maintenance.
# If this is set to False the host key will be automatically accepted and added.
# Also note that this could be a security problem accepting host keys.
host_key_checking = False

# Python will output warning in the log when a ansible-playbook is running.
# The following will silent these warnings to not overfill the output with these python
warnings.
interpreter_python = auto_silent

# profile how long each task takes
callacks_enabled = profile_tasks

# parallel processing
forks = 10

# Logging
log_path = /var/log/ansible_playbook.log

[privilege_escalation]
# Use sudo on the destination host to run as root the tasks.
become = True
```

When this playbook is now running, it follows the flow:

- Gathering facts about the hosts in the group "adminservers". This will return many information about the host like the host family "debian" or "redhat" etc.
- Update all packages to their latest version
- Check if a reboot is required
- Reboot if required

Cronjobs

Add the cron job

At the moment we only use the user and group netadmin. It will be possible to use more users which are members of the netadmin group and they have their personal ansible playbooks at the path /var/automate/<username>/ansible-playbooks.

Each user can then edit his cron job with the command

```
crontab -e
```

Create a script for running the cron job as a User

```
$ vi /var/automate/<username>/ansible-playbooks/run-cron.sh

PATH="$PATH:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin"
. $HOME/.keychain/`/usr/bin/hostname`-sh

/usr/bin/ansible-playbook -i /var/automate/<username>/ansible-
playbooks/maintenance/inventory/production /var/automate/<username>/ansible-
playbooks/maintenance/main.yml
```

An example for a cron job. This will start every day at 02:00 in the morning.

```
PATH="$PATH:/usr/bin"
0 2 * * * /var/automate/petbau/ansible-playbooks/run-cron.sh
```

Allow to edit crontabs from other Users

The following will allow to edit other Users crontab. It is allowed for the Group netadmin:

```
$ vi /etc/sudoers.d/cronedit

%netadmin ALL = (ALL) NOPASSWD: /usr/bin/crontab
```

Crontab Commands

Command	Action
<i>crontab -l</i>	Lists the crontab entries of the current user
<i>crontab -e</i>	Edit the crontab entry of the current user
<i>crontab -r</i>	Delete the crontab of the current user (Use with care!)

Command	Action
<i>crontab -lu <username></i>	List the crontab of other users. It does only work if you are member of the netadmin group.
<i>crontab -eu <username></i>	Edit the crontab of other users. It does only work if you are member of the netadmin group.

Python Scripts with Virtual Environment (venv)

A Python Script running in a venv does need a Shell Script to activate the venv and run the script. The Script will run in the */var/automate/<username>* path of each user.

Create the Shell Script

The following is an example for an Automation Python Script with venv, we will create ***start.sh***:

```
$ vi /var/automate/<username>/python-automation/start.sh

cd /var/automate/<username>/python-automation
source .venv/bin/activate
python python-automation.py
```

Make the Script Executable:

```
chmod +x start.sh
```

Create the Crontab Entry:

```
$ crontab -e

# m h dom mon dow    command
0 5 * * * /var/automate/<username>/python-automation/start.sh > ~/python-automation.log 1>&1
```

SSH Private Keys and Passwords

Storing SSH credentials securely for Ansible playbooks that are triggered via cron involves a balance between usability and security.

Here's a practical approach...

Best Practice for Storing SSH Keys for Cron-Triggered Ansible Jobs

Use ssh-agent with keychain.

Install it:

```
sudo apt install keychain
```

Add to users .bashrc:

```
$ vi $HOME/.bashrc  
  
eval $(keychain -quiet --eval ~/.ssh/id_rsa)
```

Now also crontabs are using the SSH-Agent provided by the keychain Utility, great!

If needed set a timeout how long the password should be used for the provided SSH Key. The Value is in minutes, in the example below, the password will be cached for 3 Months.

```
keychain --timeout 131487
```

Summary

By following these steps, you'll have a system where ansible periodically fetches the current information from the hosts.

The cron job ensures that this process will be run every day at 02:00 in the morning.

If you add all linux hosts to your ansible hosts file (inventory/production), all devices will be included into this process.