Решение

С решением в более удобном формате можете ознакомиться на github:
https://github.com/Alexander-Komyakov/task_norsi_trans.git

Первое задание до использования Ansible

1. Определимся с задачей. Собрать информацию о том, на каких серверах производились изменения под указанной учетной записью. Под изменениями понимается изменение конфигурации межсетевого экрана.
В линукс все есть файл, конфигурация также является файлом. Существуют обертки для удобного управления межсетевым экраном linux (netfilter). Самая популярная, ныне устаревшая — iptables. Обычно конфиг iptables находится в /etc/sysconfig/iptables. На более современных дистрибутивах используют nftables. Его конфиг хранится в /etc/nftables.conf. Nftables поддерживается начиная с ядра 3.13, а значит в CentOS 7.6 используется iptables, а в Ubuntu 18.04 и Ubuntu 20.04 может быть как iptables, так и nftables.
Следующая задача — определить наличие изменений в файле. Простейший метод команда ls -l. В ней мы увидим последнюю дату изменения файла. Можно достать информацию из столбцов только с датой, для этого пишем | awk '{print $6 « » $7 « » $8}'. С помощью команды stat мы увидим более подробную информацию о файле, но не увидим кто изменял его. В линукс для журналирования важных событий используется audit. С помощью ausearch -f путь_к_файлу мы узнаем более подробную информацию об изменениях. Т.к. мы подключаемся по ssh, на сервере должен быть запущен sshd. Проверяем кто подключался по ssh в требуемое нам время — journalctl --since «2023-01-20» -u sshd. Таким образом мы вычислили кто подключался и вносил изменения в файл, с помощью какой программы они производились, время изменения и т. д.
Теперь автоматизируем эти действия, т. к. у нас 46 серверов.

```
addresses_servers=("192.168.n.m" "192.168.n.m" "2.21.17.194")
user_name="user01"
for i in ${addresses_servers[*]}
do
   ssh $user_name@$i "наши команды"
done
```

С помощью этого кода мы подключаемся к списку наших серверов и выполняем на них требуемые нам команды.

Полный код:

```
#перечисляем адреса серверов
addresses_servers=("192.168.n.m" "192.168.n.m" "2.21.17.194")

#имя пользователя для подключения по ssh
user_name="user01"

#имя пользователя для поиска
find_user_connect="user01"

#команда поиска подключения пользователя в журнале
command_check_connect="journalctl --since "2023-01-20" -u sshd | grep "$find_user_connect

path_to_iptables="/etc/sysconfig/iptables"
path_to_nftables="/etc/nftables.conf"

#команда проверки изменения файла по дате
command_check_change_iptables="stat "$path_to_iptables" | head -7 | tail +7 | grep 2023-01-20 "
command_check_change_nftables="stat "$path_to_nftables" | sed -n '7p' | grep 2023-01-20 "

for i in ${addresses_servers[*]}
do
   checkConnect=$(ssh $user_name@$i $command_check_connect)
   #проверяем подключался ли пользователь
```

```
    if [[ ! -z $checkConnect ]]; then
        echo "USER: " $find_user_connect " connecting to server " $i >> $find_user_connect.log
        iptab_change_bool=$(ssh $user_name@$i $command_check_change_iptables)
        nftab_change_bool=$(ssh $user_name@$i $command_check_change_nftables)
        #изменял ли он iptables
        if [[ ! -z $iptab_change_bool ]]; then
            echo "USER: " $find_user_connect " change IPtables in server: " $i >> find_user_connect.log
        fi
        #изменял ли он nftables
        if [[ ! -z $nftab_change_bool ]]; then
            echo "USER: " $find_user_connect " change NFtables in server: " $i >> find_user_connect.log
        fi
    fi
done
```

В результате получаем файл в котором будет записано, куда подключался пользователь и изменял ли он конфиг межсетевого экрана.

2. На первый взгляд мы видим высокую загрузку оперативной памяти, вследствии чего используется swap. Оперативная память быстрее любых накопителей, а свап как раз на них и размещается, в виде файла или раздела. Обращаясь к свап мы видим заметное снижение производительности, но использование свап может быть следствием другой проблемы. Ошибки в программном или аппаратном обеспечении, выходящие в неконтролируемое увеличение использования оперативной памяти.
Самое быстрое и оптимальное решение — это увеличение ресурсов сервера, т. к. цена оперативной памяти, в данном случае, ниже цены потраченного на оптимизацию времени программистов, но вместо поспешных действий лучше посмотреть на более глобальный мониторинг сервиса и делать вывод.
Так же мы видим load average: 0.05, 0.07, 0.01. Это нагрузка на систему за 1, 5 и 15 минут. Если у нас один процессор, то нагрузка больше 1 является критичной, т. к. процессы стоят в очереди, но load average показывает не нагрузку на процессор, а нагрузку на систему, т. к. процессы могут по разным причинам стоять в очереди. Одна из них — ожидание возвращения результата каким-либо ресурсом.

3. Т.к. требуется решать задачу сейчас, то сделаю это самостоятельно, но чтобы не писать велосипед в следующий раз спрошу у коллег какой метод они приняли для автоматизации выполнения данных действий, оценю их и в зависимости от вывода приму решение.
Первый раз устанавливая пакеты на данные серверы, я сделаю на 3 серверах каждого типа это вручную, дабы понять могу ли столкнуться с неожиданными проблемами. Разрешу их, если возникнут и учитывая этот опыт напишу скрипт для автоматизации данных действий.
Псевдо код:

```
массив адреса_серверов_убунту18_04= [192.168.0.2, 192.168.0.3, 192.168.0.4 …]
массив адреса_серверов_убунту20_04= [192.168.0.6, 192.168.0.7, 192.168.0.8 …]
массив адреса_серверов_центос = [192.168.0.10, 192.168.0.11, 192.168.0.12 ...]
строка имя_пользователя = "user01"
строка пакет_убунту = "package2"
строка пакет_центос = "pckg02"
FOR адрес IN адреса_серверов_убунту:
        установить_на_сервер(адрес, "название_программы")
        ЕСЛИ программа_не_установлена(адрес, "название_программы"):
                записать_в_файл("установка_пакетов.лог", "На сервер" + адрес + " не установлен пакет " + пакет_тип1)
…
```

Из этого кода мы видим, что есть множество задач(подключение к серверу, установка, проверка установки, разрешение зависимостей, оповещение об изменениях и т.д.) , которые при улучшении будут разворачиваться все больше и больше. Есть более простое решение — это использование Ansible, перекладывая реализацию всего перечисленного на плечи разработчиков Ansible, нам требуется только правильно его использовать. Помимо Ansible имеются другие

инструменты, но в вашей компании используется Ansible, а значит именно он выбран и мы будет его придерживаться.

```
---
- name: "Installed package2 in ubuntu 18.04 and 20.04 (local rep)"
  hosts: ubuntu
  become: yes

  tasks:
  - name: "Installed package2"
    apt:  name=package2 state=latest

- name: "Install pckg2 in centos7.6 (local rep)"
  hosts: centos76
  become: yes

  tasks:
  - name: "Installed pckg2"
    yum: name=pckg2 state=latest
...
```

С помощью этого плейбука мы устанавливаем на наши серверы пакеты.
Структура каталога проекта ansible

```
.
├── ansible.cfg
├── check_changed_file.yml
├── group_vars
│   ├── centos76
│   └── ubuntu1804
├── hosts.txt
└── install_package.yml
```

2 directories, 6 files

ansible.cfg

```
[defaults]
host_key_checking   =   false
inventory           =   hosts.txt
```

centos76

```
---
ansible_user: user01
```

ubuntu1804

```
---
ansible_user: user01
```

hosts.txt
Добавляем строки и указываем адреса наших хостов, ubuntu18041, ubuntu18042 и т.д.

```
ubuntu1804]
ubuntu18041 ansible_host=127.0.0.1

[ubuntu2004]
ubuntu20041 ansible_host=127.0.0.1

[ubuntu:children]
ubuntu1804
ubuntu2004

[centos76]
```

```
centos761 ansible_host=127.0.0.1
```

1. Иное решение первого задания:

```
---
- name: Check changed files /etc/sysconfig/iptables and /etc/nftables.conf
  hosts: ubuntu centos76
  become: yes

  vars:
    iptables_mtime: 0
    nftables_mtime: 0

  tasks:
  - name: "Get info file iptables"
    stat:
      path: /etc/sysconfig/iptables
    register: status_file_iptables

  - name: "Check exist file iptables"
    set_fact:
      iptables_mtime: "{{ status_file_iptables.stat.mtime }}"
    when: status_file_iptables.stat.exists

  - name: "Check changed file iptables"
    debug:
      msg: "file /etc/sysconfig/iptables updates in server {{ ansible_ssh_host }}"
    when: (iptables_mtime | int + 18000 > {{ ansible_date_time.epoch | int }}) and (iptables_mtime != 0)

  #DRY - DON'T REPEAT YOUR SELF!
  - name: "Get info file nftables"
    stat:
      path: /etc/nftables.conf
    register: status_file_nftables

  - name: "Check exist file nftables"
    set_fact:
      nftables_mtime: "{{ status_file_nftables.stat.mtime }}"
    when: status_file_nftables.stat.exists

  - name: "Check changed file nftables"
    debug:
      msg: "file /etc/nftables.conf updates in server {{ ansible_ssh_host }}"
    when: (nftables_mtime | int + 18000 > {{ ansible_date_time.epoch | int }}) and (nftables_mtime != 0)
...
```

В этом плейбуке мы вы используем модуль stat, требуемый для наших задач. Проверяем существование файла, сравниваем время изменения с текущим временем. В задании указано 5 часов, а это 18 тысяч секунд и если изменения файла были за этот промежуток времени, то сообщаем в каком файле и на каком сервере они произошли. Код повторяется, оптимальнее было бы создать цикл, nftables и iptables вынести в переменные, но он и так легко читается.