Sessão 5: Gestão de configuração

Algo que já deve ter ficado claro, a este ponto, é que a configuração manual de múltiplas máquinas é um processo tedioso, demorado, e ainda oferece o risco de configuração incorreta por parte do administrador durante o processo. Fica ainda mais evidente a dimensão do problema quando observamos que nosso *datacenter* simulado possui até o momento apenas três servidores — imagine o tamanho do problema se tivéssemos dezenas ou centenas de VMs!

As ferramentas de gestão de configuração oferecem uma alternativa: através delas, podemos ter um processo de implementação sistemática de mudanças em um (ou vários) sistemas de forma a manter sua integridade. A ideia básica por trás dessas ferramentas é a gestão de **estados desejados**: criamos artefatos de configuração que definem um estado-alvo para um grupo de sistemas, e o sistema de gestão de configuração se encarrega de checar se esse estado está atendido — caso positivo, nada precisa ser feito; e, caso negativo, as alterações de configuração previstas nos artefatos de configuração serão aplicadas nos sistemas.

Outro conceito central é o **comportamento idempotente** desse tipo de ferramenta: mesmo com aplicações sucessivas dos artefatos de configuração, o sistema se encarrega de verificar se o estado-alvo está atendido, não reaplicando mudanças desnecessárias. Podemos citar também como vantagens desses sistemas de gestão a existência de *frameworks* de automação que facilitam o trabalho do administrador, uso de *templates* para aproveitar e customizar configuração entre diferentes grupos de máquinas, e grande extensibilidade de capacidades através de módulos e *plugins* de terceiros.

Nesta sessão iremos trabalhar com o Ansible, uma ferramenta de gestão de configuração *open-source* criada por Michael DeHaan em 2012 e atualmente mantida pela Red Hat. O Ansible possibilita também a automatização de provisionamento de software e *deployment* de aplicações, conectando-se via SSH ou PowerShell aos servidores-alvo em um arquitetura que dispensa a instalação de agentes (*agentless*). Usando o Ansible, iremos solucionar a gestão centralizada do arquivo `/etc/sudoers`, problema que encontramos no final da sessão anterior. Finalmente, iremos usar a ferramenta de controle de versão Git para gerenciar as mudanças que faremos nos conjuntos de *scripts* e artefatos do Ansible ao longo das sessões.

1) Topologia desta sessão

A figura abaixo mostra a topologia de rede que será utilizada nesta sessão, com as máquinas relevantes em destaque.

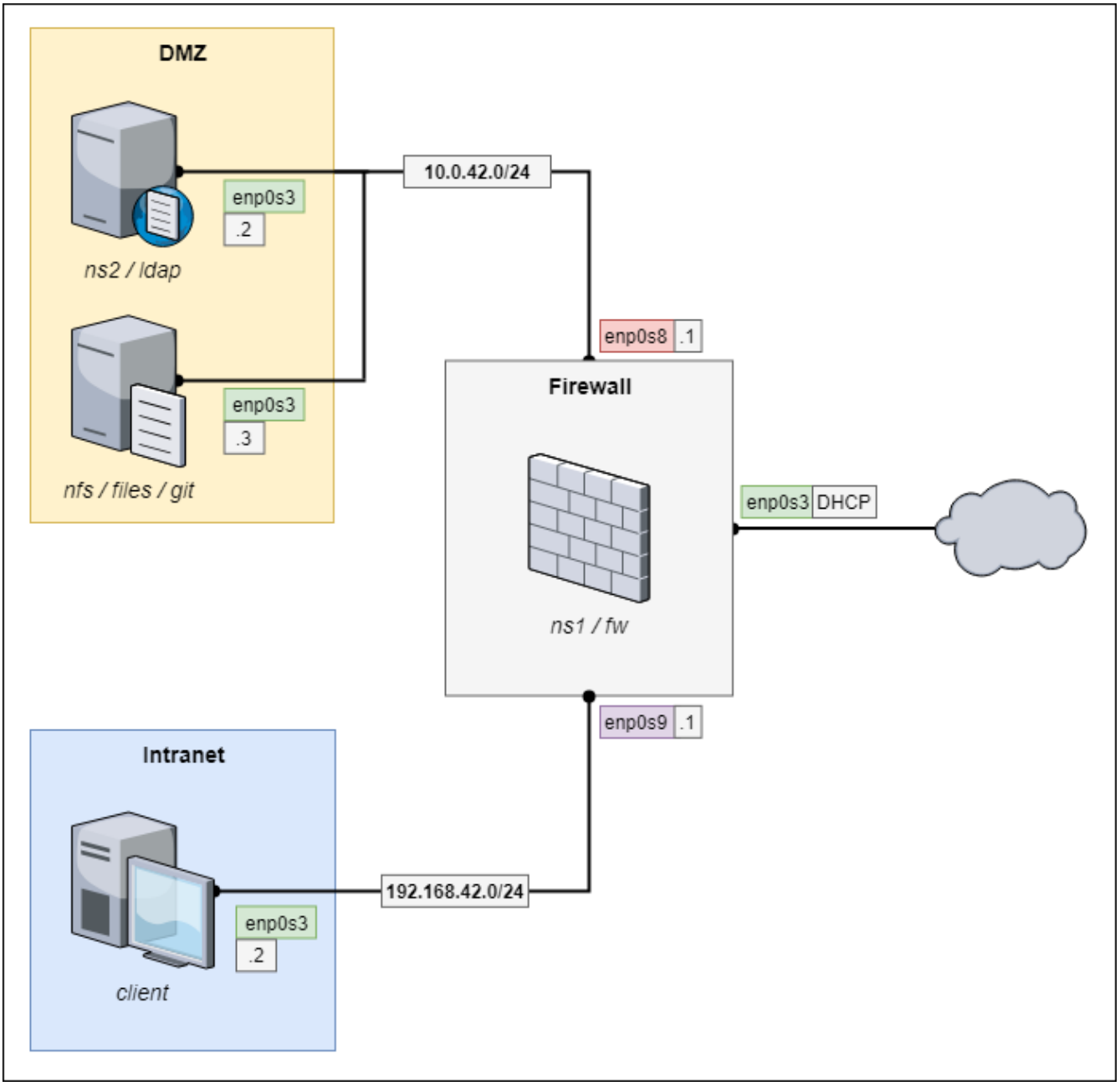

Figura 1. Topologia de rede desta sessão

1. Praticamente não há mudanças em relação à topologia da sessão anterior — a única diferença é que a máquina `nfs` possuirá um novo *alias*, `git`, já que atuará como servidor da solução de controle de versão Git. Vamos ajustar o DNS: acesse a máquina `ns1` como o usuário `root`:

   ```
   # hostname ; whoami
   ns1
   root
   ```

   Edite o arquivo de zonas `/etc/nsd/zones/intnet.zone`, inserindo uma entrada CNAME para a máquina `nfs`, como se segue. **Não se esqueça** de incrementar o valor do serial no topo do arquivo!

```
# nano /etc/nsd/zones/intnet.zone
(...)
```

```
# grep git /etc/nsd/zones/intnet.zone
git     IN    CNAME             nfs
```

Assine o arquivo de zonas usando o *script* criado anteriormente:

```
# bash /root/scripts/signzone-intnet.sh
reconfig start, read /etc/nsd/nsd.conf
ok
ok
ok
ok removed 0 rrsets, 0 messages and 0 key entries
```

Verifique a criação das entradas usando o comando `dig`:

```
# dig git.intnet +short
nfs.intnet.
10.0.42.3
```

2) Instalação e configuração inicial do Ansible

Um dos aspectos mais interessantes do Ansible é sua simplicidade — dependendo apenas do interpretador Python para operar (o qual já vem instalado no sistema-base do Debian), sua instalação é bastante simples. Outro fator relevante: como o sistema dispensa a instalação de agentes, não é necessário criar um servidor dedicado para usar o Ansible, de forma que iremos usar a máquina `client` como um conveniente ponto de partida para os logins remotos.

Vamos, no entanto, criar um usuário específico para o Ansible no servidor LDAP, gerenciando suas permissões de forma granular via `sudo`.

1. Acesse a máquina `ns2` como o usuário `root` e crie um usuário para o Ansible, membro dos grupos `setup` (primário) e `fwadm`, com senha `seg10ansible`:

```
# hostname ; whoami
ns2
root
```

```
# ldapadduser ansible setup
Successfully added user ansible to LDAP
Successfully set password for user ansible
```

```
# ldapaddusertogroup ansible setup
Successfully added user ansible to group cn=setup,ou=Groups,dc=intnet
```

```
# ldapaddusertogroup ansible fwadm
Successfully added user ansible to group cn=fwadm,ou=Groups,dc=intnet
```

```
# ldapsetpasswd ansible
Changing password for user uid=ansible,ou=People,dc=intnet
New Password:
Retype New Password:
Successfully set password for user uid=ansible,ou=People,dc=intnet
```

2. Agora, acesse a máquina `client` como o usuário `root` e instale o Ansible seguindo os passos abaixo. Todas as instruções a seguir referenciam o passo-a-passo da documentação oficial do Ansible, acessível em https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#latest-releases-via-apt-debian .

```
# hostname ; whoami
client
root
```

Primeiro, adicione o repositório de pacotes do Ansible à lista de fontes do `apt-get`:

```
# echo "deb http://ppa.launchpad.net/ansible/ansible/ubuntu trusty main" >
/etc/apt/sources.list.d/ansible.list
```

Em seguida, adicione à lista de chaves confiáveis para instalação de pacotes a *pubkey* dos mantenedores do pacote do Ansible:

```
# apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
93C4A3FD7BB9C367
Executing: /tmp/apt-key-gpghome.UOSyzarqGQ/gpg.1.sh --keyserver
hkp://keyserver.ubuntu.com:80 --recv-keys 93C4A3FD7BB9C367
gpg: key 93C4A3FD7BB9C367: public key "Launchpad PPA for Ansible, Inc." imported
gpg: Total number processed: 1
gpg:               imported: 1
```

Agora, basta atualizar a lista de pacotes disponíveis nos repositórios remotos e instalar o Ansible usando o `apt-get`:

```
# apt-get update
```

```
# apt-get install ansible
```

3) Execução de comandos simples

1. Começaremos utilizando as funções mais básicas do Ansible. Antes de mais nada, no entanto, precisamos configurar o acesso para os diversos servidores do *datacenter* simulado usando o sistema de autenticação LDAP/SSH-CA.

   Acesse a máquina `client` como o usuário `ansible`:

   ```
   $ hostname ; whoami ; pwd
   client
   ansible
   /home/ansible
   ```

   Agora, use o script `~/scripts/sshsign_user.sh` para assine um par de chaves e conseguir logar nos servidores:

   ```
   $ bash ~/scripts/sshsign_user.sh
   Signing ~/.ssh/id_rsa.pub key...
   All done!
   ```

2. Como estabelecido, o Ansible consegue trabalhar com múltiplos sistemas em uma infraestrutura ao mesmo tempo — para isso, ele seleciona conjuntos de máquinas em seu **inventário**. O inventário é um arquivo texto em formato INI ou YAML que lista as máquinas e grupos a serem gerenciados.

   Crie o arquivo novo `/home/ansible/hosts` com o seguinte conteúdo:

   ```
   $ nano ~/hosts
   (...)
   ```

   ```
   $ cat hosts
   [srv]
   ns1
   ns2
   nfs
   ```

   No arquivo acima criamos um único grupo, `srv`, contendo todos os servidores do *datacenter* simulado criados até aqui.

3. Vamos testar? Execute um comando simples em todas as máquinas gerenciadas pelo Ansible:

```
$ ansible -i ~/hosts srv -m shell -a 'hostname --fqdn ; whoami'
ns2 | CHANGED | rc=0 >>
ns2.intnet
ansible

ns1 | CHANGED | rc=0 >>
ns1.intnet
ansible

nfs | CHANGED | rc=0 >>
nfs.intnet
ansible
```

O que aconteceu? Vejamos:

- Com `ansible`, invocamos o Ansible para executar uma única tarefa, com parâmetros passados diretamente via linha de comando.
- Depois, `-i ~/hosts` define o arquivo de inventário a ser usado, `/home/ansible/hosts`.
- O grupo `srv` é indicado a seguir, falando para o Ansible qual dos grupos disponíveis no arquivo de inventário será o alvo dos comandos que se seguem.
- Com `-m shell` carregamos o módulo *shell* do Ansible, que permite execução de comandos diretamente pelo interpretador `/bin/sh` (ou outro à sua escolha) nas máquinas remotas.
- Finalmente, `-a 'hostname --fqdn ; whoami'` indica quais comandos o *shell* sendo executado em cada uma das máquinas-membros do grupo `srv` irá operar.

Analise a saída, agora: note que o Ansible loga em cada uma das máquinas do grupo `srv` (`ns2`, `ns1` e `nfs`) e executa os comandos indicados, mostrando claramente o *hostname* local e usuário logado. Observe, ainda, que a ordem das máquinas escrita no arquivo `/home/ansible/hosts` não foi respeitada — o Ansible inicia conexões nos servidores de forma paralela, e as respostas podem vir fora de ordem.

Um último adendo: mencionamos o uso do módulo *shell* no comando acima, mas quais outros módulos existem? Há muitos outros — mesmo. Confira a lista completa na página https://docs.ansible.com/ansible/latest/modules/modules_by_category.html .

4. A próxima pergunta que pode surgir neste momento é: Ok, conseguimos executar um comando remoto com um usuário não-privilegiado. E se quisermos executar como o `root`? Para isso, podemos executar a escalação de privilégios usando o `become`. Veja:

```
$ ansible -i ~/hosts srv --become --become-user=root --become-method=su --ask
-become-pass -m shell -a 'hostname --fqdn ; head -n1 /etc/shadow'
SU password:
ns1 | CHANGED | rc=0 >>
ns1.intnet
root:$6$s7Gt1cd.$UXQf67CVYxR7HP..h2wvhOx4nOtBT7do28R1uChYdMpZc.uLi43OKdtentrWD2zSTK
v9EyB7Bdqcpwr6nAlNo.:17848:0:99999:7:::

ns2 | CHANGED | rc=0 >>
ns2.intnet
root:$6$s7Gt1cd.$UXQf67CVYxR7HP..h2wvhOx4nOtBT7do28R1uChYdMpZc.uLi43OKdtentrWD2zSTK
v9EyB7Bdqcpwr6nAlNo.:17848:0:99999:7:::

nfs | CHANGED | rc=0 >>
nfs.intnet
root:$6$s7Gt1cd.$UXQf67CVYxR7HP..h2wvhOx4nOtBT7do28R1uChYdMpZc.uLi43OKdtentrWD2zSTK
v9EyB7Bdqcpwr6nAlNo.:17848:0:99999:7:::
```

Quais as diferenças para o comando anterior?

- `--become` indica que desejamos executar comandos como outro usuário — esta opção não implica qual usuário, método ou senha serão usados, mas apenas que haverá a alteração de usuário efetivo.
- Em seguida, `--become-user=root` informa que o usuário a se tornar nas máquinas remotas será, de fato, o `root`.
- A opção `--become-method=su` configura o método de escalada de privilégio usado. Como ainda não configuramos o `sudo` nas máquinas remotas, iremos usar o `su`.
- `--ask-become-pass` solicita ao Ansible que pergunte a senha de escalação de privilégio antes de executar os comandos, evitando que tenhamos que digitá-la diretamente no terminal.
- Por fim, `'hostname --fqdn ; head -n1 /etc/shadow'` visa identificar a máquina remota e depois executar um comando que apenas o `root` conseguiria fazer.

Excelente! Estamos conseguindo controlar remotamente todos os servidores com sucesso. Mas e se quisermos fazer mais que apenas rodar comandos simples?

4) Uso de roles no Ansible

Caso queiramos fazer mais que executar poucos comandos num *shell* remoto, o uso de *roles* (ou papéis) é fundamental para organizar tarefas complexas, cópia remota de arquivos e gerência de dependências entre serviços. Vamos usar *roles* no Ansible para solucionar o problema de gestão centralizada do arquivo `/etc/sudoers`, apresentado no final da sessão anterior.

1. Crie um diretório para armazenar as *roles* que serão criadas, `/home/ansible/roles`. Em seguida, entre nesse diretório.

```
$ mkdir ~/roles
```

```
$ cd ~/roles/
```

2. O comando `ansible-galaxy` nos permite realizar uma série de operações relacionadas a *roles*, desde a criação de simples papéis locais até a busca e importação de *roles* criadas por outros usuários do Ansible e acessíveis na Internet pelo website https://galaxy.ansible.com/ . Recomendamos ao aluno que pesquise por tarefas usuais nesse site — provavelmente já existe uma *role* para solucionar esse problema!

   Para manter a simplicidade das atividades executadas no curso, iremos criar *roles* manualmente. Use o `ansible-galaxy` para criar uma estrutura de diretórios padrão para a *role* `sudoers`:

   ```
   $ ansible-galaxy init --init-path ~/roles/ sudoers
   - sudoers was created successfully
   ```

   O que foi criado? Vejamos:

   ```
   $ ls -R ~/roles/sudoers/
   /home/ansible/roles/sudoers/:
   defaults  files  handlers  meta  README.md  tasks  templates  tests  vars

   /home/ansible/roles/sudoers/defaults:
   main.yml

   /home/ansible/roles/sudoers/files:

   /home/ansible/roles/sudoers/handlers:
   main.yml

   /home/ansible/roles/sudoers/meta:
   main.yml

   /home/ansible/roles/sudoers/tasks:
   main.yml

   /home/ansible/roles/sudoers/templates:

   /home/ansible/roles/sudoers/tests:
   inventory  test.yml

   /home/ansible/roles/sudoers/vars:
   main.yml
   ```

   Cada um dos diretórios criados possui uma função específica, a saber:

- `defaults`: contém variáveis-padrão a serem usadas na *role*.
- `files`: contém arquivos que serão copiados através desta *role*.
- `handlers`: contém funções especiais denominadas *handlers*. Essas funções são bastante parecida com *tasks*, mas são invocadas de forma indireta e servem para automatizar tarefas recorrentes, como o *reload* de *daemons* de serviços.
- `meta`: contém metadados sobre a *role*, como a definição de dependências entre *roles* em um mesmo diretório.
- `tasks`: contém a lista principais de tarefas a serem executados pela *role*.
- `templates`: contém *templates* que podem ser copiados através desta *roles* — *templates* podem basicamente ser entendidos como arquivos que se utilizam de variáveis para customizar seu estado final.
- `tests`: permite a criação de testes para verificar a correta execução de aspectos da *role* corrente. Frequentemente não é necessário desenvolver (muitos) testes no Ansible, já que o sistema de configuração modela um estado-alvo desejado, de forma declarativa.
- `vars`: define outras variáveis para a *role*, que têm precedência sobre as variáveis em `defaults`.

3. Retomemos o problema central, a configuração do arquivo `/etc/sudoers` de forma centralizada:

   - A colaboradora `leia` acaba de se juntar à equipe de `han`, o grupo `fwadm` em nosso sistema LDAP. Imagine que ela ficará responsável por editar regras no firewall de borda, a máquina `ns1`. Mas, por estar começando agora na empresa, `han` quer restringir o conjunto de comandos que `leia` pode executar na máquina, liberando apenas a edição do firewall via `iptables`. Sua senha será `seg10leia`. Nas demais máquinas (`ns2` e `nfs`) `leia` não deve ter qualquer acesso especial, apenas como um usuário regular.
   - O colaborador `chewie` foi contratado para auxiliar na manutenção da base LDAP da empresa. Para desempenhar suas tarefas, iremos colocá-lo em um novo grupo `ldapadm`. Os membros desse grupo devem ter acesso aos principais comandos de edição do LDAP (criação, modificação e deleção de usuários e grupos) na máquina `ns2`. Sua senha será `seg10chewie`. Nas demais máquinas (`ns1` e `nfs`) `chewie` não deve ter qualquer acesso especial, apenas como um usuário regular.
   - Os usuários atuais, `luke` e `han`, terão permissão para executar qualquer comando como o usuário `root`, em qualquer máquina.

   A criação de usuários e grupos já foi tratada no passo (1) da atividade (9) da sessão anterior, como visto. Foquemos, então, apenas arquivo `sudoers`. Como o `sudo` é um programa estático (isto é, não depende de nenhum *daemon* executando para funcionar), o único requerimento para uma *role* que configure o arquivo `/etc/sudoers` é copiar o arquivo corretamente para as máquinas, e ajustar suas permissões.

   Crie o arquivo novo `/home/ansible/roles/sudoers/files/sudoers`, com o seguinte conteúdo:

```
 1 Defaults        env_reset
 2 Defaults        mail_badpass
 3 Defaults        secure_path
="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
 4
 5 User_Alias ADMINS    = aluno, \
 6                        luke,  \
 7                        han
 8
 9 User_Alias FWUSERS   = leia
10
11 User_Alias LDAPUSERS = %ldapadm
12
13 Host_Alias FWHOSTS   = ns1
14
15 Host_Alias LDAPHOSTS = ns2
16
17 Cmnd_Alias FWCMDS    = /sbin/iptables
18
19 Cmnd_Alias LDAPCMDS  = /usr/sbin/ldapaddgroup,             \
20                        /usr/sbin/ldapadduser,              \
21                        /usr/sbin/ldapaddusertogroup,       \
22                        /usr/sbin/ldapdeletegroup,          \
23                        /usr/sbin/ldapdeleteuser,           \
24                        /usr/sbin/ldapdeleteuserfromgroup,  \
25                        /usr/sbin/ldapmodifygroup,          \
26                        /usr/sbin/ldapmodifymachine,        \
27                        /usr/sbin/ldapmodifyuser,           \
28                        /usr/sbin/ldaprenamegroup,          \
29                        /usr/sbin/ldaprenameuser,           \
30                        /usr/sbin/ldapsetpasswd,            \
31                        /usr/sbin/ldapsetprimarygroup
32
33 root      ALL=(ALL:ALL)    ALL
34
35 ansible   ALL=(ALL:ALL)    NOPASSWD: ALL
36
37 ADMINS    ALL=(ALL:ALL)    ALL
38
39 FWUSERS   FWHOSTS=(root)   FWCMDS
40
41 LDAPUSERS LDAPHOSTS=(root) LDAPCMDS
42
43 #includedir /etc/sudoers.d
```

O que esse arquivo faz? Vamos ver:

- Nas linhas [5-11] definimos *aliases* (apelidos) de usuários para agrupar os elementos que serão configurados para usar o `sudo`. Criamos um *alias* `ADMINS` para agrupar os usuários `aluno`, `luke` e `han`, `FWUSERS` para `leia` e `LDAPUSERS` para o **grupo** `ldapadm`. É especialmente

importante manter um *alias* apontando para um usuário local, como o usuário `aluno`, caso haja problemas com o sistema de autenticação LDAP.

- Nas linhas [13-15] definimos *aliases* para máquinas, `ns1` e `ns2`. Também poderíamos usar endereços IP, se desejado.
- Nas linhas [17-31] definimos *aliases* de comandos: para a máquina `ns1`, apenas o comando `/sbin/iptables` é suficiente; já para a máquina `ns2` configuramos uma lista detalhada dos comandos que o *alias* `LDAPUSERS` poderá usar.
- Nas linhas [33-41] fazemos a "amarração" dos *aliases* previamente definidos, atribuindo aos usuários/grupos em quais máquinas eles podem executar os comandos, como quais usuários, e quais são esses comandos. Note que ao usuário `ansible` é permitido executar qualquer comando como `root` em todas as máquinas sem a necessidade de digitação de senha, de forma bastante conveniente para a automação de tarefas via Ansible como faremos em sessões futuras.

Compare as regras sendo definidas pelos *aliases* no arquivo `sudoers` acima e os requisitos de acesso definidos no começo deste passo — todos estão sendo atendidos a contento. É claro, seria possível organizar os usuários/grupos de forma diferente — em particular, poderia ser interessante agrupar os usuário `han` e `luke` em um grupo dedicado de "super-admins" — mas funciona como uma boa prova de conceito.

Para o aluno atento, é claro observar que o conjunto de programas que estão sendo autorizados para o usuário `chewie` (de fato, todo o grupo `ldapadm`) garante a ele uma permissão efetiva **muito** maior que a prevista no escopo inicial do problema. Afinal, bastaria ao usuário `chewie` adicionar-se a si mesmo no grupo `fwadm` (via comando `ldapaddusertogroup`) para acessar a máquina `ns1`, ou alterar a senha dos usuários `han` ou `luke` (via comando `ldapsetpasswd`) e entrar como esses usuários em máquinas que permitam login via senha. Iremos "ignorar" esse problema nesta atividade, em nome da simplicidade.

De fato, fica aqui também um desafio: qual seria o conjunto adequado de programas e permissões garantidas via `sudoers` que permitiria ao usuário `chewie` executar seu trabalho de manutenção de contas no LDAP da empresa, e ao mesmo tempo controlar seu acesso de forma efetiva? Lembre-se que o `sudoers` permite ao administrador definir não apenas quais comandos serão autorizados, mas também parâmetros específicos de linha de comando, se desejado.

4. Criado o arquivo `sudoers` que atende aos requisitos iniciais, temos que criar a tarefa que irá distribuí-lo. Antes disso, observe as permissões do arquivo `sudoers` original:

```
$ ls -ld /etc/sudoers
-r--r----- 1 root root 669 jun  5  2017 /etc/sudoers
```

Com essas permissões em mente, edite o arquivo `/home/ansible/roles/sudoers/tasks/main.yml` com o seguinte conteúdo:

```
---
- name: Propagate sudoers configuration
  become: yes
  become_user: root
  become_method: su
  copy:
    src: sudoers
    dest: /etc
    owner: root
    group: root
    mode: 0440
```

Definimos uma tarefa de propagação do arquivo `sudoers`, com as seguintes características: iremos rodar a tarefa como o usuário `root`, usando o `su` para escalada de privilégio; será copiado o arquivo `sudoers` (do diretório `files/`, implícito aqui) para a pasta `/etc` na máquina remota, com `root` como usuário e grupo donos, e permissões `r—r-----`.

5. Tudo pronto? Vamos executar! Crie o arquivo `/home/ansible/srv.yml` para fazer a amarração entre o grupo de *hosts* e a nova *role*:

```
$ nano ~/srv.yml
(...)
```

```
$ cat ~/srv.yml
---
- hosts: srv
  roles:
    - sudoers
```

Como ainda **não** configuramos o `sudo`, e estamos usando o comando `su` para escalada de privilégio, iremos passar novamente a opção `--ask-become-pass` para que o Ansible solicite a senha do usuário `root` para escalada de privilégio nas máquinas remotas. Execute a role:

```
$ ansible-playbook -i ~/hosts ~/srv.yml --ask-become-pass
SUDO password:

PLAY [srv]
*************************************************************************************
***********************


TASK [Gathering Facts]
*************************************************************************************
************

ok: [ns2]
ok: [ns1]
ok: [nfs]

TASK [sudoers : Propagate sudoers configuration]
*******************************************************************

changed: [ns2]
changed: [ns1]
changed: [nfs]

PLAY RECAP
*************************************************************************************
***********************

nfs                        : ok=2    changed=1    unreachable=0    failed=0
ns1                        : ok=2    changed=1    unreachable=0    failed=0
ns2                        : ok=2    changed=1    unreachable=0    failed=0
```

6. Terá funcionado? Vamos ver: tente executar um comando distribuído com o Ansible usando o `sudo` como método de escalada de privilégio, e sem digitar senha.

```
$ ansible -i ~/hosts srv -b --become-user=root --become-method=sudo -m shell -a
'hostname --fqdn ; grep ansible /etc/sudoers'
nfs | CHANGED | rc=0 >>
nfs.intnet
ansible   ALL=(ALL:ALL)    NOPASSWD: ALL

ns2 | CHANGED | rc=0 >>
ns2.intnet
ansible   ALL=(ALL:ALL)    NOPASSWD: ALL

ns1 | CHANGED | rc=0 >>
ns1.intnet
ansible   ALL=(ALL:ALL)    NOPASSWD: ALL
```

Perfeito! Veja que o Ansible conseguiu acessar todas as máquinas, elevar privilégios usando o `sudo` sem necessidade de senha, e imprimir o conteúdo do arquivo `/etc/sudoers` — o qual pode ser lido apenas pelo `root` e, convenientemente, filtramos a linha em que a autorização ao usuário `ansible` está sendo configurada.

5) Testando os controles do sudo

Conseguimos distribuir o arquivo `sudoers` como objetivado, mas será que os controles estão funcionando?

1. Vamos testar o acesso de `leia` na máquina `ns1` — lembre-se, ela deve conseguir executar apenas o comando `iptables` como o usuário `root`, e nenhum outro. Acesse a máquina `ns1` como `leia`:

```
$ hostname ; whoami
ns1
leia
```

Agora, tente executar o comando `iptables` usando o `sudo`:

```
$ sudo iptables -L POSTROUTING -vn -t nat

Presumimos que você recebeu as instruções de sempre do administrador
de sistema local. Basicamente, resume-se a estas três coisas:

    #1) Respeite a privacidade dos outros.
    #2) Pense antes de digitar.
    #3) Com grandes poderes vêm grandes responsabilidades.

[sudo] senha para leia:
Chain POSTROUTING (policy ACCEPT 322 packets, 25437 bytes)
 pkts bytes target     prot opt in     out     source               destination
    1    60 MASQUERADE  tcp  --  *      enp0s3  10.0.42.0/24         0.0.0.0/0
multiport dports 80,443
    9   540 MASQUERADE  tcp  --  *      enp0s3  192.168.42.0/24      0.0.0.0/0
multiport dports 80,443
```

Excelente! E se tentarmos executar um comando não autorizado?

```
$ sudo rm /etc/shadow
Sinto muito, usuário leia não tem permissão para executar "/bin/rm /etc/shadow"
como root em ns1.intnet.
```

De fato, é possível listar exatamente quais comandos um usuário está apto a executar com o comando `sudo -l`:

```
$ sudo -l
Entradas de Defaults correspondentes a leia em ns1:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

Usuário leia pode executar os seguintes comandos em ns1:
    (root) /sbin/iptables
```

E quanto a `han`? Ele consegue executar qualquer comando como `root`?

```
$ hostname ; whoami
ns1
han
```

```
$ sudo -l

Presumimos que você recebeu as instruções de sempre do administrador
de sistema local. Basicamente, resume-se a estas três coisas:

    #1) Respeite a privacidade dos outros.
    #2) Pense antes de digitar.
    #3) Com grandes poderes vêm grandes responsabilidades.

[sudo] senha para han:
Entradas de Defaults correspondentes a han em ns1:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

Usuário han pode executar os seguintes comandos em ns1:
    (ALL : ALL) ALL
```

Excelente! Os acessos para a máquina `ns1` estão funcionando como previsto.

2. Vamos para o caso do usuário `chewie`. Acesse a máquina `ns2` como `chewie` e verifique quais comandos você está autorizado a executar usando o `sudo`:

```
$ hostname ; whoami
ns2
chewie
```

```
$ sudo -l

Presumimos que você recebeu as instruções de sempre do administrador
de sistema local. Basicamente, resume-se a estas três coisas:

    #1) Respeite a privacidade dos outros.
    #2) Pense antes de digitar.
    #3) Com grandes poderes vêm grandes responsabilidades.

[sudo] senha para chewie:
Entradas de Defaults correspondentes a chewie em ns2:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

Usuário chewie pode executar os seguintes comandos em ns2:
    (root) /usr/sbin/ldapaddgroup, /usr/sbin/ldapadduser,
/usr/sbin/ldapaddusertogroup, /usr/sbin/ldapdeletegroup,
        /usr/sbin/ldapdeleteuser, /usr/sbin/ldapdeleteuserfromgroup,
/usr/sbin/ldapmodifygroup,
        /usr/sbin/ldapmodifymachine, /usr/sbin/ldapmodifyuser,
/usr/sbin/ldaprenamegroup, /usr/sbin/ldaprenameuser,
        /usr/sbin/ldapsetpasswd, /usr/sbin/ldapsetprimarygroup
```

Tente criar um novo grupo no LDAP, `sudotest`, e em seguida delete-o.

```
$ sudo ldapaddgroup sudotest
Successfully added group sudotest to LDAP
```

```
$ sudo ldapdeletegroup sudotest
Successfully deleted group cn=sudotest,ou=Groups,dc=intnet from LDAP
```

Tente executar um comando não-autorizado:

```
$ sudo reboot
Sinto muito, usuário chewie não tem permissão para executar "/sbin/reboot" como
root em ns2.intnet.
```

Tudo de acordo com o esperado, muito bom.

6) Controle da senha do usuário root

1. Pergunte-se agora o seguinte: e se `leia` ou `chewie`, por qualquer motivo, conseguirem obter a senha do usuário `root`? O que não é exatamente difícil, já que estamos usando `rnpesr` como senha — o que ocorre então? Veja o que acontece ao usarmos o comando `su` como `chewie` na máquina `ns2`:

```
$ hostname ; whoami
ns2
chewie
```

```
$ su -
Senha:
```

```
# whoami
root
```

A solução ideal, nesse caso, é desabilitar a senha do `root`. Já que os acessos de superusuário estão corretamente configurados via arquivo `sudoers` distribuído centralmente via Ansible, não teremos prejuízo administrativo — nesse cenário, mesmo que usuários não-autorizados descubram a senha, está já estará desabilitada e não poderá ser usada para efetuar escalada de privilégio.

2. Acesse a máquina `client` como o usuário `ansible`. Para desabilitar a conta do `root`, basta usar o comando `passwd -l` — vamos executar esse comando em todos os servidores do *datacenter*:

```
$ ansible -i ~/hosts srv -b --become-user=root --become-method=sudo -m shell -a
'passwd -l root'
ns1 | CHANGED | rc=0 >>
passwd: informação de expiração de senha alterada.

nfs | CHANGED | rc=0 >>
passwd: informação de expiração de senha alterada.

ns2 | CHANGED | rc=0 >>
passwd: informação de expiração de senha alterada.
```

3. Com a senha desabilitada, apenas aqueles usuários que tenham permissão de `sudo` para executar comandos de escalada de privilégio poderão tornar-se o usuário `root` — todos os demais, restritos a um subconjunto de comandos controlados pelo arquivo `/etc/sudoers`, não conseguirão fazê-lo.

   Por exemplo, acesse a máquina `ns1` como o usuário `han`.

```
$ hostname ; whoami
ns1
han
```

   Note que mesmo o usuário `han`, que possui acesso irrestrito, não consegue executar `su` diretamente, mesmo digitando a senha correta:

```
$ su -
Senha:
su: Falha de autenticação
```

Apenas com comandos como `sudo su -`, `sudo -i` ou `sudo --login` (que equivale a invocar um *shell* de login, como executar `sudo bash`) é possível tornar-se `root`, como podemos ver:

```
$ sudo -i
```

```
# whoami
root
```

4. Mas... observe, nossa configuração não está ortogonal. Imagine que uma nova máquina seja criada em nosso *datacenter*, e queremos configurá-la com a distribuição do arquivo `sudoers` de forma centralizada, assim como fizemos com nossos servidores atuais. Sem problema, certo? Basta adicionar o novo *hostname* no arquivo `~/hosts` e disparar com o `ansible-playbook`!

   A configuração de *lockout* da senha do `root` que fizemos no passo (2), no entanto, não está integrada à *role* `sudoers`. De fato, executamos um comando direto usando o módulo *shell* — e se esquecermos de fazer esse passo com a próxima máquina? Nesse caso, a senha do `root` dessa máquina estará habilitada, e teremos uma configuração divergente em nossos servidores.

   Fica claro, portanto, que o método correto de integração de novas tarefas no Ansible é via *roles*, e não via comandos isolados, sob pena de criar uma grande confusão no parque de servidores num futuro próximo. Vamos reverter o comando que fizemos antes:

```
$ ansible -i ~/hosts srv -b --become-user=root --become-method=sudo -m shell -a
'passwd -u root'
ns1 | CHANGED | rc=0 >>
passwd: informação de expiração de senha alterada.

nfs | CHANGED | rc=0 >>
passwd: informação de expiração de senha alterada.

ns2 | CHANGED | rc=0 >>
passwd: informação de expiração de senha alterada.
```

5. Como desabilitar a conta do `root` do jeito certo? Vamos editar a lista de *tasks* da *role* `sudoers`. Adicione a tarefa a seguir no arquivo `/home/ansible/roles/sudoers/tasks/main.yml` (copie todo o comando, desde `cat` até a palavra `EOF`):

```
$ cat << EOF >> ~/roles/sudoers/tasks/main.yml

- name: Sets root account as expired
  user:
    name: root
    expires: 0
EOF
```

Uma curiosidade: a sintaxe acima é conhecida como *heredoc* (ou *Here Document*), um bloco de código que pode ser usado para diversas tarefas no *shell* Bash. No exemplo acima, estamos usando-o para inserir todas as linhas ao final do arquivo `/home/ansible/roles/sudoers/tasks/main.yml`, até que seja encontrada a *string* `EOF`. Consulte este link para mais informações sobre uso de *heredocs*: https://www.tldp.org/LDP/abs/html/here-docs.html .

De volta à tarefa em mãos, vamos ver como ficou o arquivo após a alteração:

```
$ cat /home/ansible/roles/sudoers/tasks/main.yml
---
- name: Propagate sudoers configuration
  become: yes
  become_user: root
  become_method: su
  copy:
    src: sudoers
    dest: /etc
    owner: root
    group: root
    mode: 0440

- name: Sets root account as expired
  user:
    name: root
    expires: 0
```

A tarefa que acabamos de adicionar irá se valer do módulo *user* do Ansible, e ajustar a data de expiração do usuário `root` para 0 — esse valor é contado em segundos a partir do dia 1 de janeiro de 1970, uma referência conhecida como *Unix Epoch*, ou *POSIX Time*. Com efeito, estamos dizendo que a conta deste usuário está expirada desde 1/1/1970.

6. Falta alguma coisa? Ah sim! Note que o `become_method` do arquivo `/home/ansible/roles/sudoers/tasks/main.yml` ainda está configurado como `su` — como já distribuímos o arquivo `sudoers` anteriormente com sucesso usando essa *role*, podemos alterá-lo para `sudo`.

   Mais além, observe que as diretivas `become*` são específicas da tarefa de propagação do arquivo `sudoers`, e não se aplicam à tarefa de ajuste de expiração de conta do usuário `root`. Ao invés de repetir o bloco `become + become_user + become_method` duas vezes (ou mais, dependendo do

número de tarefas que queremos realizar), o melhor caminho de ação é remover esse bloco do arquivo de *tasks* e inseri-lo no arquivo de configuração da *role*.

Vamos por partes. Primeiro, vamos remover as linhas `become*` do arquivo `/home/ansible/roles/sudoers/tasks/main.yml`:

```
$ t="$( mktemp )" ; \
    cat ~/roles/sudoers/tasks/main.yml | \
    awk 'gsub(/^ *become.*$/,""){printf $0;next;}1' > $t ; \
    mv $t ~/roles/sudoers/tasks/main.yml ; \
    unset t
```

```
$ cat ~/roles/sudoers/tasks/main.yml
---
- name: Propagate sudoers configuration
  copy:
    src: sudoers
    dest: /etc
    owner: root
    group: root
    mode: 0440

- name: Sets root account as expired
  user:
    name: root
    expires: 0
```

Agora, vamos inseri-las no arquivo de amarração da *role*:

```
$ sed -i '/\- hosts\: srv/a\  become: yes\n  become_user: root\n  become_method:
sudo' ~/srv.yml
```

```
$ cat ~/srv.yml
---
- hosts: srv
  become: yes
  become_user: root
  become_method: sudo
  roles:
    - sudoers
```

Observe atentamente a indentação do arquivo acima: caso a cópia do comando tenha inserido ou removido espaços acidentalmente, edite o arquivo diretamente e deixe-o **exatamente** como mostrado acima.

7. Pronto! Vamos disparar a *role* usando o `ansible-playbook` e verificar seu funcionamento:

```
$ ansible-playbook -i ~/hosts ~/srv.yml

PLAY [srv]
*********************************************************************************
***********************


TASK [Gathering Facts]
*********************************************************************************
***********

ok: [ns2]
ok: [ns1]
ok: [nfs]

TASK [sudoers : Propagate sudoers configuration]
*****************************************************************

ok: [ns2]
ok: [ns1]
ok: [nfs]

TASK [sudoers : Sets root account as expired]
*******************************************************************

changed: [ns2]
changed: [nfs]
changed: [ns1]

PLAY RECAP
*********************************************************************************
***********************

nfs                        : ok=3    changed=1    unreachable=0    failed=0
ns1                        : ok=3    changed=1    unreachable=0    failed=0
ns2                        : ok=3    changed=1    unreachable=0    failed=0
```

Terá funcionado? Vamos ver, usando o módulo *shell* do Ansible:

```
$ ansible -i ~/hosts srv -b --become-user=root --become-method=sudo -m shell -a
'chage -l root | grep "Conta expira"'
nfs | CHANGED | rc=0 >>
Conta expira                                          : jan 01, 1970

ns1 | CHANGED | rc=0 >>
Conta expira                                          : jan 01, 1970

ns2 | CHANGED | rc=0 >>
Conta expira                                          : jan 01, 1970
```

Excelente! Vamos fazer um teste *in loco* com o usuário `han` na máquina `ns1`:

```
$ hostname ; whoami
ns1
han
```

```
$ su -
Senha:
Sua conta expirou; entre em contato com o administrador do sistema
su: Falha de autenticação
```

6) Versionamento de configuração com git

À medida que fazemos novas adições de *tasks* e *roles* ao Ansible, pode surgir a necessidade de documentar as modificações sendo feitas, compartilhá-las com outros colegas de trabalho ou, em caso de mudanças incorretas, reverter o estado dos artefatos de configuração para uma versão anterior, conhecida e testada. De certa forma, as configuração que estamos fazendo em servidores do *datacenter* com o Ansible se parecem muito com o código-fonte de uma linguagem de programação — só que, ao invés de produzir programas, nosso código produz configurações e estados-alvo em servidores.

A ferramenta de controle de versão Git é certamente a mais utilizada mundialmente para cumprir as tarefas que delineamos acima. Criado em 2005 por Linus Torvalds (sim, o mesmo criador do kernel Linux, você não leu errado), o Git é um sistema de versionamento distribuído desenhado com o objetivo expresso de aprimorar as limitações dos sistemas de controle de versão mais usados à epoca: CVS e Subversion.

Iremos instalar um servidor Git na máquina `nfs`, e utilizá-lo como repositório para as configurações gerenciadas pelo Ansible. Assim, a cada nova modificação poderemos manter o registro de versão no repositório, comentar quais mudanças foram realizadas, bem como compartilhar o acesso ao código com outros colaboradores e aceitar suas submissões e *patches*, se desejado.

1. O primeiro passo é instalar o Git nas máquinas `nfs` e `client` — queremos usar o Ansible para fazer isso, mas o `sudo` não está configurado na máquina `client`... ainda.

Logue como o usuário `ansible` na máquina `client`.

```
$ hostname ; whoami
client
ansible
```

Como a máquina `client` não está no DNS, iremos usar seu endereço IP no arquivo de *hosts*. Adicione-o:

```
$ echo '192.168.42.2' >> ~/hosts
```

```
$ cat hosts
[srv]
ns1
ns2
nfs
192.168.42.2
```

Lembre-se que o `sudo` ainda não está configurado na máquina local, de modo que teremos que usar o `su` para escalar privilégios. Como o método padrão de escalada de privilégios especificado no arquivo `/home/ansible/srv.yml` é o `sudo`, teremos que configurar um *override* na linha de comando. Outro ponto: queremos executar a *role* apenas na máquina local, de forma que usaremos a opção `--limit`.

Esses fatores considerados, execute a *role*:

```
$ ansible-playbook -i hosts --limit='192.168.42.2' --ask-become-pass -e
ansible_become_method=su srv.yml
SUDO password:

PLAY [srv]
*************************************************************************************
**********************


TASK [Gathering Facts]
*************************************************************************************
***********

ok: [192.168.42.2]

TASK [sudoers : Propagate sudoers configuration]
*********************************************************************

changed: [192.168.42.2]

TASK [sudoers : Sets root account as expired]
*********************************************************************

changed: [192.168.42.2]

PLAY RECAP
*************************************************************************************
**********************

192.168.42.2               : ok=3    changed=2    unreachable=0    failed=0
```

Perfeito. Verifique que o `sudoers` foi configurado com sucesso, e que a conta do `root` foi desabilitada na máquina local:

```
$ sudo grep ansible /etc/sudoers
ansible   ALL=(ALL:ALL)    NOPASSWD: ALL
```

```
$ su -
Senha:
Sua conta expirou; entre em contato com o administrador do sistema
su: Falha de autenticação
```

2. Agora sim, vamos instalar o Git nas máquinas `nfs` e `client` usando o Ansible. Note o uso da opção `--limit` para especificar o escopo do comando:

```
$ ansible -i ~/hosts srv -b --become-user=root --become-method=sudo
--limit='nfs,192.168.42.2' -m shell -a 'apt-get install -y git'
 [WARNING]: Consider using the apt module rather than running apt-get.  If you need
to use command because apt is
insufficient you can add warn=False to this command task or set
command_warnings=False in ansible.cfg to get rid of this message.

nfs | CHANGED | rc=0 >>
(...)
Configurando git (1:2.11.0-3+deb9u4) ...

192.168.42.2 | CHANGED | rc=0 >>
(...)
Configurando git (1:2.11.0-3+deb9u4) ...
```

Terá funcionado? Vamos ver:

```
$ ansible -i ~/hosts srv -m shell -a 'which git'
ns2 | FAILED | rc=1 >>
non-zero return code

nfs | CHANGED | rc=0 >>
/usr/bin/git

ns1 | FAILED | rc=1 >>
non-zero return code

192.168.42.2 | CHANGED | rc=0 >>
/usr/bin/git
```

Perfeito, o Git foi instalado com sucesso, e apenas nas máquinas objetivadas.

3. Iremos usar o Git em sua capacidade mais básica — via SSH, usando os diretórios *home* de cada usuário. É bastante possível fazer uma configuração muito mais sofisticada, configurar uma interface como o GitLab (https://about.gitlab.com/install/), mas para manter a simplicidade das atividades aqui realizadas não iremos fazer isso.

   Primeiro, crie um repositório vazio de nome `ansible` em seu diretório *home* local:

   ```
   $ git init ansible
   Initialized empty Git repository in /home/ansible/ansible/.git/
   ```

   Agora, clone-o para o formato *bare* (mantendo apenas os arquivos administrativos do Git) com o comando:

```
$ git clone --bare ansible ansible.git
Cloning into bare repository 'ansible.git'...
warning: You appear to have cloned an empty repository.
done.
```

```
$ ls -1d ansible*
ansible
ansible.git
```

Devemos copiar o diretório `/home/ansible/ansible.git` para o servidor Git, que será a máquina `nfs`. Use o comando `scp`:

```
$ scp -r ~/ansible.git/ ansible@nfs:~
description
100%   73   179.2KB/s   00:00
HEAD
100%   23    76.8KB/s   00:00
pre-receive.sample
100%  544     1.7MB/s   00:00
commit-msg.sample
100%  896     2.7MB/s   00:00
pre-applypatch.sample
100%  424     1.3MB/s   00:00
pre-push.sample
100% 1348     3.4MB/s   00:00
prepare-commit-msg.sample
100% 1239     3.1MB/s   00:00
update.sample
100% 3610     9.4MB/s   00:00
post-update.sample
100%  189   478.5KB/s   00:00
pre-rebase.sample
100% 4898    13.2MB/s   00:00
applypatch-msg.sample
100%  478     1.3MB/s   00:00
pre-commit.sample
100% 1642     4.4MB/s   00:00
config
100%  113   367.7KB/s   00:00
exclude
100%  240   557.6KB/s   00:00
```

Pronto, o repositório Git remoto está configurado! Se quiser, remova os arquivos locais — iremos usar o repositório remoto a partir de agora:

```
$ rm -rf ~/ansible*
```

4. Para usar o repositório Git remoto, o primeiro passo é cloná-lo — por conveniência (e por já termos configurado o sistema de autenticação centralizado), iremos usar o protocolo SSH no acesso:

```
$ git clone ansible@nfs:/home/ansible/ansible.git
Cloning into 'ansible'...
warning: You appear to have cloned an empty repository.
```

O Git reporta que clonamos um repositório vazio... não por muito tempo! Copie os arquivos de trabalho do Ansible que usamos até aqui para dentro do repositório local:

```
$ mv ~/{hosts,roles,srv.yml} ~/ansible
```

Entre no repositório local e visualize seu estado com `git status`:

```
$ cd ~/ansible/ ; git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        hosts
        roles/
        srv.yml

nothing added to commit but untracked files present (use "git add" to track)
```

O Git reporta que os arquivos adicionados não estão sendo versionados. Para adicioná-los ao sistema de controle de versão, use o comando `git add`:

```
$ git add .
```

O que isso fez?

```
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   hosts
        new file:   roles/sudoers/README.md
        new file:   roles/sudoers/defaults/main.yml
        new file:   roles/sudoers/files/sudoers
        new file:   roles/sudoers/handlers/main.yml
        new file:   roles/sudoers/meta/main.yml
        new file:   roles/sudoers/tasks/main.yml
        new file:   roles/sudoers/tests/inventory
        new file:   roles/sudoers/tests/test.yml
        new file:   roles/sudoers/vars/main.yml
        new file:   srv.yml
```

Agora sim, todos os arquivos foram adicionados ao sistema de versionamento. Para confirmar essas mudanças temos que realizar um *commit*, ou confirmação, das alterações — antes disso, no entanto, é recomendável que configuremos nosso nome de usuário e e-mail no Git, para evitar *warnings* futuros:

```
$ git config user.name Ansible
```

```
$ git config user.email ansible@intnet
```

Pronto, vamos ao *commit*. É sempre recomendável incluir uma mensagem explicativa junto com o *commit* (com a opção `-m`) indicando que alterações foram feitas naquela versão, criando um histórico do repositório.

```
$ git commit -m 'Importacao inicial, role sudoers adicionada'
[master (root-commit) 0c673e4] Importacao inicial, role sudoers adicionada
 11 files changed, 180 insertions(+)
 create mode 100644 hosts
 create mode 100644 roles/sudoers/README.md
 create mode 100644 roles/sudoers/defaults/main.yml
 create mode 100644 roles/sudoers/files/sudoers
 create mode 100644 roles/sudoers/handlers/main.yml
 create mode 100644 roles/sudoers/meta/main.yml
 create mode 100644 roles/sudoers/tasks/main.yml
 create mode 100644 roles/sudoers/tests/inventory
 create mode 100644 roles/sudoers/tests/test.yml
 create mode 100644 roles/sudoers/vars/main.yml
 create mode 100644 srv.yml
```

O passo final consiste em enviar as alterações locais para o repositório remoto através do comando `git push`:

```
$ git push
Counting objects: 22, done.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (22/22), 3.27 KiB | 0 bytes/s, done.
Total 22 (delta 0), reused 0 (delta 0)
To nfs:/home/ansible/ansible.git
 * [new branch]      master -> master
```

Pronto, todos os dados foram enviados para o servidor `nfs` e estão — espera-se — seguros. Será mesmo? Vamos testar: retorne ao seu diretório *home* e remova todos os arquivos do repositório local:

```
$ cd ~ ; rm -rf ~/ansible
```

```
$ ls
scripts
```

Tragédia! Perdemos todo o trabalho realizado com o Ansible até aqui! Ou... perdemos mesmo? Tente clonar o repositório remoto a partir da máquina `nfs`:

```
$ git clone ansible@nfs:/home/ansible/ansible.git
Cloning into 'ansible'...
remote: Counting objects: 22, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 22 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (22/22), done.
```

```
$ ls ~/ansible/
hosts  roles  srv.yml
```

Ufa, está tudo ali ainda. Se quiser verificar o registro dos *commits* anteriores, confirmando que nossas modificações anteriores foram de fato salvas no repositório remoto, basta usar o comando `git log` dentro do repositório local:

```
$ cd ~/ansible/ ; git log
commit 0c673e48dcdc7aaf2bd6738c8033c33815f10cc6
Author: Ansible <ansible@intnet>
Date:   Thu Nov 15 02:04:14 2018 -0200

    Importacao inicial, role sudoers adicionada
```