ANSIBLE PLAYBOOKS

DEVSECOPS SECURITY ARCHITECTURE

WWW.DEVSECOPSGUIDES.COM

Ansible Playbooks

· Jun 17, 2024 · 33 min read

Table of contents

Ansible playbooks are essential tools in the DevSecOps toolkit, enabling the automation of complex IT tasks while ensuring security and compliance are integral to every step. These playbooks, written in YAML, define a series of tasks to be executed on remote machines, allowing for consistent and repeatable configurations. In the context of DevSecOps, playbooks can automate security tasks such as patch management, vulnerability assessments, and the application of security policies, integrating security measures directly into the development and deployment pipelines. This automation reduces human error, speeds up the deployment process, and ensures that security practices are consistently applied across all environments.

Moreover, Ansible playbooks support the principle of "security as code," where security practices are codified and version-controlled alongside application code. This approach ensures that security configurations are transparent, auditable, and easily updated in response to emerging threats or compliance requirements. By leveraging Ansible's extensive library of modules, DevSecOps teams can enforce secure configurations, manage firewalls, monitor systems for compliance, and remediate issues automatically. This integration of Ansible playbooks within DevSecOps pipelines not only enhances security posture but also aligns with agile methodologies, promoting rapid and secure delivery of software.

Ansible Inventory Structure

Ansible inventory files define the hosts and groups of hosts on which Ansible commands, modules, and playbooks operate. The inventory can be static (defined in INI or YAML format) or dynamic (using a script or plugin).

INI Format Inventory

Create a file named `inventory.ini`:

```
[webservers]
web1 ansible_host=192.168.1.10 ansible_user=username
ansible_ssh_pass=password
web2 ansible_host=192.168.1.11 ansible_user=username
ansible_ssh_pass=password

[dbservers]
db1 ansible_host=192.168.1.20 ansible_user=username
ansible_ssh_pass=password
db2 ansible_host=192.168.1.21 ansible_user=username
ansible_ssh_pass=password
```

```
[all:vars]
ansible_python_interpreter=/usr/bin/python3
```

- `[webservers]` and `[dbservers]` define groups of hosts.
- `ansible_host`, `ansible_user`, and `ansible_ssh_pass` provide the necessary connection details.
- `[all:vars]` defines variables applied to all hosts.

YAML Format Inventory

Create a file named `inventory.yml`:

COPY

```
all:
  vars:
    ansible_python_interpreter: /usr/bin/python3
  children:
    webservers:
      hosts:
        web1:
          ansible_host: 192.168.1.10
          ansible_user: username
          ansible_ssh_pass: password
        web2:
          ansible_host: 192.168.1.11
          ansible_user: username
          ansible_ssh_pass: password
    dbservers:
      hosts:
        db1:
          ansible_host: 192.168.1.20
          ansible_user: username
          ansible_ssh_pass: password
        db2:
          ansible_host: 192.168.1.21
          ansible_user: username
          ansible_ssh_pass: password
```

- `all` is the top-level group containing all hosts.
- `vars` defines variables applied to all hosts.
- `children` groups hosts into `webservers` and `dbservers`.

Ansible Playbook Structure

An Ansible playbook is a YAML file that defines a series of tasks to be executed on specified hosts. Each playbook consists of one or more plays, and each play targets a group of hosts.

Example Playbook: `setup_webserver.yml`

COPY

```
---
- name: Setup Web Server
  hosts: webservers
  become: yes
  tasks:
    - name: Install Nginx
      ansible.builtin.yum:
        name: nginx
        state: present

    - name: Start and enable Nginx
      ansible.builtin.systemd:
        name: nginx
        state: started
        enabled: yes

    - name: Copy Nginx configuration
      ansible.builtin.copy:
        src: /path/to/local/nginx.conf
        dest: /etc/nginx/nginx.conf
        owner: root
        group: root
        mode: '0644'
        backup: yes

    - name: Restart Nginx
      ansible.builtin.systemd:
        name: nginx
        state: restarted
```

- **Playbook Header**:
  - `name`: Description of the play.
  - `hosts`: Target hosts for the play (e.g., `webservers`).
  - `become`: Indicates that tasks should be executed with elevated privileges (`yes` for `sudo`).
- **Tasks**:
  - **Install Nginx**:
    - Use the `ansible.builtin.yum` module to install Nginx.
  - **Start and enable Nginx**:
    - Use the `ansible.builtin.systemd` module to start and enable the Nginx service.
  - **Copy Nginx configuration**:
    - Use the `ansible.builtin.copy` module to copy the local Nginx configuration file to the remote host.
  - **Restart Nginx**:

- Use the `ansible.builtin.systemd` module to restart Nginx.

Running the Playbook

To run the playbook, navigate to the directory containing `setup_webserver.yml` and execute the following command:

COPY

```
ansible-playbook -i inventory.yml setup_webserver.yml
```

Replace `inventory.yml` with the path to your Ansible inventory file. This command runs the playbook on the hosts specified in the `webservers` group of the inventory file. If you use the INI format, the command would be the same, just ensure the inventory file name is correct:

COPY

```
ansible-playbook -i inventory.ini setup_webserver.yml
```

Ansible Playbook: SSH Audit

This playbook designed to audit SSH configurations on remote hosts. This playbook will check the SSH daemon configuration file (`/etc/ssh/sshd_config`) to ensure it complies with your security policies.

Create a playbook named `ssh_audit.yml` with the following content:

COPY

```
---
- name: Audit SSH Configuration
  hosts: all
  become: yes
  gather_facts: no
  tasks:
    - name: Check if /etc/ssh/sshd_config exists
      ansible.builtin.stat:
        path: /etc/ssh/sshd_config
      register: sshd_config

    - name: Read the SSH configuration file
      ansible.builtin.command: cat /etc/ssh/sshd_config
      when: sshd_config.stat.exists
      register: sshd_config_content

    - name: Ensure PermitRootLogin is set to no
      ansible.builtin.debug:
        msg: "PermitRootLogin is set correctly"
      when: "'PermitRootLogin no' in sshd_config_content.stdout"

    - name: Ensure PasswordAuthentication is set to no
      ansible.builtin.debug:
        msg: "PasswordAuthentication is set correctly"
      when: "'PasswordAuthentication no' in
sshd_config_content.stdout"

    - name: Ensure X11Forwarding is set to no
      ansible.builtin.debug:
        msg: "X11Forwarding is set correctly"
      when: "'X11Forwarding no' in sshd_config_content.stdout"

    - name: Print SSH configuration file if it exists
      ansible.builtin.debug:
        msg: "{{ sshd_config_content.stdout }}"
      when: sshd_config.stat.exists

    - name: Fail if PermitRootLogin is not set to no
      ansible.builtin.fail:
        msg: "PermitRootLogin is not set to 'no'"
      when: sshd_config.stat.exists and "'PermitRootLogin no' not in
```

```
sshd_config_content.stdout"

    - name: Fail if PasswordAuthentication is not set to no
      ansible.builtin.fail:
        msg: "PasswordAuthentication is not set to 'no'"
      when: sshd_config.stat.exists and "'PasswordAuthentication no'
not in sshd_config_content.stdout"

    - name: Fail if X11Forwarding is not set to no
      ansible.builtin.fail:
        msg: "X11Forwarding is not set to 'no'"
      when: sshd_config.stat.exists and "'X11Forwarding no' not in
sshd_config_content.stdout"
```

Explanation:

1. **Playbook Header**:
    - `name`: A description of the playbook.
    - `hosts`: Specifies the target hosts (here, `all` means it will run on all hosts defined in your inventory).
    - `become: yes`: Run tasks with elevated privileges (i.e., `sudo`).
    - `gather_facts: no`: Skip gathering facts for faster execution.
2. **Tasks**:
    - **Check if `/etc/ssh/sshd_config` exists**:
        - Use the `ansible.builtin.stat` module to check if the SSH configuration file exists.
    - **Read the SSH configuration file**:
        - Use the `ansible.builtin.command` module to read the contents of `/etc/ssh/sshd_config` and register the output.

- **Ensure specific settings**:
    - Use the `ansible.builtin.debug` module to print a message if certain settings (`PermitRootLogin no`, `PasswordAuthentication no`, `X11Forwarding no`) are found in the SSH configuration.
- **Fail the playbook if settings are incorrect**:
    - Use the `ansible.builtin.fail` module to fail the playbook if the specific settings are not found, indicating non-compliance.

Ansible Playbook: Linux Kernel Audit

playbook designed to audit the Linux kernel version on remote hosts. This playbook will check the current kernel version and compare it against a specified version to ensure it meets your security or compliance requirements.

Create a playbook named `kernel_audit.yml` with the following content:

COPY

```
---
- name: Audit Linux Kernel Version
  hosts: all
  become: yes
  gather_facts: no

  vars:
    min_kernel_version: "5.4.0"

  tasks:
    - name: Get current kernel version
      ansible.builtin.command: uname -r
      register: kernel_version

    - name: Print current kernel version
      ansible.builtin.debug:
        msg: "Current kernel version is {{ kernel_version.stdout }}"
    - name: Compare kernel version
      ansible.builtin.shell: |
        current_version=$(echo "{{ kernel_version.stdout }}" | sed
's/-.*//')
        min_version="{{ min_kernel_version }}"
        if [ "$(printf '%s\n' "$min_version" "$current_version" | sort
-V | head -n1)" = "$min_version" ]; then
          exit 0
        else
          exit 1
        fi
      register: kernel_comparison
      ignore_errors: yes

    - name: Fail if kernel version is below minimum required
      ansible.builtin.fail:
        msg: "Kernel version {{ kernel_version.stdout }} is below the
minimum required version {{ min_kernel_version }}"
      when: kernel_comparison.rc != 0
```

Explanation:

1. **Playbook Header**:
    - `name`: A description of the playbook.
    - `hosts`: Specifies the target hosts (here, `all` means it will run on all hosts defined in your inventory).
    - `become: yes`: Run tasks with elevated privileges (i.e., `sudo`).
    - `gather_facts: no`: Skip gathering facts for faster execution.
2. **Variables**:
    - `min_kernel_version`: Define the minimum required kernel version for compliance.

3. **Tasks**:
    - **Get current kernel version**:
        - Use the `ansible.builtin.command` module to run the `uname -r` command and register the output in the `kernel_version` variable.
    - **Print current kernel version**:
        - Use the `ansible.builtin.debug` module to print the current kernel version.
    - **Compare kernel version**:
        - Use the `ansible.builtin.shell` module to compare the current kernel version with the minimum required version.
        - This task uses `sed` to strip any extra suffix from the kernel version (e.g., `-generic`), then compares the versions using `sort -V` for proper version sorting.
        - The `ignore_errors: yes` directive ensures that the playbook continues even if the kernel version is below the required version, allowing the next task to handle the failure condition.
    - **Fail if kernel version is below minimum required**:
        - Use the `ansible.builtin.fail` module to fail the playbook if the kernel version is below the minimum required version.

Ansible Playbook: Nginx Audit

Ansible playbook designed to audit Nginx configurations on remote hosts. This playbook will check for common security settings and best practices in the Nginx configuration file (`/etc/nginx/nginx.conf`).

Create a playbook named `nginx_audit.yml` with the following content:

```
---
- name: Audit Nginx Configuration
  hosts: webservers
  become: yes
  gather_facts: no

  tasks:
    - name: Check if /etc/nginx/nginx.conf exists
      ansible.builtin.stat:
        path: /etc/nginx/nginx.conf
      register: nginx_conf

    - name: Read Nginx configuration file
      ansible.builtin.command: cat /etc/nginx/nginx.conf
      when: nginx_conf.stat.exists
      register: nginx_conf_content

    - name: Ensure server_tokens is set to off
      ansible.builtin.debug:
        msg: "server_tokens is set correctly"
      when: "'server_tokens off;' in nginx_conf_content.stdout"

    - name: Ensure SSL protocols are properly configured
      ansible.builtin.debug:
        msg: "SSL protocols are configured correctly"
      when: "'ssl_protocols TLSv1.2 TLSv1.3;' in
nginx_conf_content.stdout"

    - name: Ensure HTTP methods are restricted
      ansible.builtin.debug:
        msg: "HTTP methods are restricted"
      when: "'limit_except GET POST {' in nginx_conf_content.stdout"

    - name: Print Nginx configuration file
      ansible.builtin.debug:
        msg: "{{ nginx_conf_content.stdout }}"
      when: nginx_conf.stat.exists

    - name: Fail if server_tokens is not set to off
      ansible.builtin.fail:
        msg: "server_tokens is not set to 'off'"
      when: nginx_conf.stat.exists and "'server_tokens off;' not in
nginx_conf_content.stdout"

    - name: Fail if SSL protocols are not properly configured
      ansible.builtin.fail:
        msg: "SSL protocols are not properly configured (TLSv1.2 and
TLSv1.3)"
      when: nginx_conf.stat.exists and "'ssl_protocols TLSv1.2
TLSv1.3;' not in nginx_conf_content.stdout"

    - name: Fail if HTTP methods are not restricted
      ansible.builtin.fail:
        msg: "HTTP methods are not restricted (limit_except GET POST)"
      when: nginx_conf.stat.exists and "'limit_except GET POST {' not
in nginx_conf_content.stdout"
```

Explanation:

1. **Playbook Header**:
   - `name`: A description of the playbook.
   - `hosts`: Specifies the target hosts (here, `webservers` is used, assuming this group is defined in your inventory).
   - `become: yes`: Run tasks with elevated privileges (i.e., `sudo`).
   - `gather_facts: no`: Skip gathering facts for faster execution.
2. **Tasks**:
   - **Check if** `/etc/nginx/nginx.conf` exists:

    - Use the `ansible.builtin.stat` module to check if the Nginx configuration file exists.
- **Read Nginx configuration file**:
    - Use the `ansible.builtin.command` module to read the contents of `/etc/nginx/nginx.conf` and register the output.
- **Ensure server_tokens is set to off**:
    - Use the `ansible.builtin.debug` module to print a message if `server_tokens off;` is found in the Nginx configuration.
- **Ensure SSL protocols are properly configured**:
    - Use the `ansible.builtin.debug` module to print a message if `ssl_protocols TLSv1.2 TLSv1.3;` is found in the Nginx configuration.
- **Ensure HTTP methods are restricted**:
    - Use the `ansible.builtin.debug` module to print a message if `limit_except GET POST {` is found in the Nginx configuration.
- **Print Nginx configuration file**:
    - Use the `ansible.builtin.debug` module to print the contents of the Nginx configuration file.
- **Fail if server_tokens is not set to off**:
    - Use the `ansible.builtin.fail` module to fail the playbook if `server_tokens off;` is not found in the Nginx configuration.
- **Fail if SSL protocols are not properly configured**:
    - Use the `ansible.builtin.fail` module to fail the playbook if `ssl_protocols TLSv1.2 TLSv1.3;` is not found in the Nginx configuration.
- **Fail if HTTP methods are not restricted**:

- Use the `ansible.builtin.fail` module to fail the playbook if `limit_except GET POST {` is not found in the Nginx configuration.

Ansible Playbook: Apache Audit

Create a playbook named `apache_audit.yml` with the following content:

Ansible playbook designed to audit the Apache HTTP server configurations on remote hosts. This playbook will check for common security settings and best practices in the Apache configuration file (`/etc/httpd/conf/httpd.conf` or `/etc/apache2/apache2.conf`, depending on the Linux distribution).

```
---
- name: Audit Apache Configuration
  hosts: webservers
  become: yes
  gather_facts: no

  tasks:
    - name: Determine Apache configuration file location
      ansible.builtin.shell: |
        if [ -f /etc/httpd/conf/httpd.conf ]; then
          echo /etc/httpd/conf/httpd.conf
        elif [ -f /etc/apache2/apache2.conf ]; then
          echo /etc/apache2/apache2.conf
        else
          echo "Apache configuration file not found"
        fi
      register: apache_conf_path

    - name: Check if Apache configuration file exists
      ansible.builtin.stat:
        path: "{{ apache_conf_path.stdout }}"
      register: apache_conf
    - name: Read Apache configuration file
      ansible.builtin.command: cat {{ apache_conf_path.stdout }}
      when: apache_conf.stat.exists
      register: apache_conf_content

    - name: Ensure ServerTokens is set to Prod
      ansible.builtin.debug:
        msg: "ServerTokens is set correctly"
      when: "'ServerTokens Prod' in apache_conf_content.stdout"

    - name: Ensure ServerSignature is set to Off
      ansible.builtin.debug:
        msg: "ServerSignature is set correctly"
      when: "'ServerSignature Off' in apache_conf_content.stdout"

    - name: Ensure TraceEnable is set to Off
      ansible.builtin.debug:
        msg: "TraceEnable is set correctly"
      when: "'TraceEnable Off' in apache_conf_content.stdout"

    - name: Print Apache configuration file
      ansible.builtin.debug:
        msg: "{{ apache_conf_content.stdout }}"
      when: apache_conf.stat.exists

    - name: Fail if ServerTokens is not set to Prod
      ansible.builtin.fail:
        msg: "ServerTokens is not set to 'Prod'"
      when: apache_conf.stat.exists and "'ServerTokens Prod' not in
apache_conf_content.stdout"

    - name: Fail if ServerSignature is not set to Off
      ansible.builtin.fail:
        msg: "ServerSignature is not set to 'Off'"
      when: apache_conf.stat.exists and "'ServerSignature Off' not in
apache_conf_content.stdout"
```

```
    - name: Fail if TraceEnable is not set to Off
      ansible.builtin.fail:
        msg: "TraceEnable is not set to 'Off'"
      when: apache_conf.stat.exists and "'TraceEnable Off' not in
apache_conf_content.stdout"
```

Explanation:

1. **Playbook Header**:
   - `name`: A description of the playbook.
   - `hosts`: Specifies the target hosts (e.g., `webservers`).
   - `become: yes`: Run tasks with elevated privileges (i.e., `sudo`).
   - `gather_facts: no`: Skip gathering facts for faster execution.
2. **Tasks**:
   - **Determine Apache configuration file location**:
     - Use the `ansible.builtin.shell` module to determine the location of the Apache configuration file. The file could be in different locations based on the distribution.
   - **Check if Apache configuration file exists**:
     - Use the `ansible.builtin.stat` module to check if the determined Apache configuration file exists.
   - **Read Apache configuration file**:
     - Use the `ansible.builtin.command` module to read the contents of the Apache configuration file and register the output.
   - **Ensure** `ServerTokens` is set to `Prod`:
     - Use the `ansible.builtin.debug` module to print a message if `ServerTokens Prod` is found in the Apache configuration.

- **Ensure** `ServerSignature` is set to `Off`:
  - Use the `ansible.builtin.debug` module to print a message if `ServerSignature Off` is found in the Apache configuration.
- **Ensure** `TraceEnable` is set to `Off`:
  - Use the `ansible.builtin.debug` module to print a message if `TraceEnable Off` is found in the Apache configuration.
- **Print Apache configuration file**:
  - Use the `ansible.builtin.debug` module to print the contents of the Apache configuration file.
- **Fail if** `ServerTokens` is not set to `Prod`:
  - Use the `ansible.builtin.fail` module to fail the playbook if `ServerTokens Prod` is not found in the Apache configuration.
- **Fail if** `ServerSignature` is not set to `Off`:
  - Use the `ansible.builtin.fail` module to fail the playbook if `ServerSignature Off` is not found in the Apache configuration.
- **Fail if** `TraceEnable` is not set to `Off`:
  - Use the `ansible.builtin.fail` module to fail the playbook if `TraceEnable Off` is not found in the Apache configuration.

Ansible Playbook: Environment Secret Audit

Create a playbook named `env_secret_audit.yml` with the following content:

Ansible playbook designed to audit environment variables on remote hosts to ensure that no secrets (e.g., passwords, API keys) are exposed in the environment variables.

```yaml
---
- name: Audit Environment Variables for Secrets
  hosts: all
  become: yes
  gather_facts: no

  vars:
    secret_patterns:
      - "password"
      - "secret"
      - "api_key"
      - "token"

  tasks:
    - name: Check for environment variables containing secrets
      ansible.builtin.shell: printenv
      register: env_vars

    - name: Print all environment variables
      ansible.builtin.debug:
        msg: "{{ env_vars.stdout_lines }}"

    - name: Fail if any environment variables contain secrets
      ansible.builtin.fail:
        msg: "Environment variables contain sensitive information: {{
item }}"
      with_items: "{{ env_vars.stdout_lines }}"
      when: item | regex_search(secret_patterns | join('|'),
ignorecase=True)

    - name: Print safe message if no secrets are found
      ansible.builtin.debug:
        msg: "No sensitive information found in environment
variables."
```

```
      when: env_vars.stdout_lines | selectattr('search', 'none',
secret_patterns | join('|'), ignorecase=True) | list | length == 0
```

Explanation:

1. **Playbook Header**:
   - `name`: A description of the playbook.
   - `hosts`: Specifies the target hosts (e.g., `all`).
   - `become: yes`: Run tasks with elevated privileges (i.e., `sudo`).
   - `gather_facts: no`: Skip gathering facts for faster execution.
2. **Variables**:
   - `secret_patterns`: A list of patterns that represent possible secret keys (e.g., `password`, `secret`, `api_key`, `token`). You can add more patterns as needed.
3. **Tasks**:
   - **Check for environment variables containing secrets**:
     - Use the `ansible.builtin.shell` module to execute the `printenv` command, which prints all environment variables, and register the output.
   - **Print all environment variables**:
     - Use the `ansible.builtin.debug` module to print all environment variables. This is useful for debugging and verifying what is being checked.
   - **Fail if any environment variables contain secrets**:
     - Use the `ansible.builtin.fail` module to fail the playbook if any environment variables match the secret patterns. The `regex_search` function is used to check if any of the patterns are found in the environment variables.

- **Print safe message if no secrets are found**:
  - Use the `ansible.builtin.debug` module to print a message indicating that no sensitive information was found in the environment variables. This task only runs if none of the patterns match.

Ansible Playbook: SCM (GitLab) Audit

Create a playbook named `scm_audit.yml` with the following content:

Ansible playbook designed to audit GitLab or a generic Git server to ensure security and compliance with best practices. This playbook will check for common security settings, such as ensuring that repositories are private, two-factor authentication is enabled, and checking for default branch protection.

```
---
- name: Audit GitLab Configuration
  hosts: scm_servers
  become: yes
  gather_facts: no

  vars:
    gitlab_url: "https://gitlab.example.com"
    gitlab_api_token: "your_personal_access_token"

  tasks:
    - name: Check if GitLab API is accessible
      ansible.builtin.uri:
        url: "{{ gitlab_url }}/api/v4/projects"
        method: GET
        headers:
          PRIVATE-TOKEN: "{{ gitlab_api_token }}"
      register: gitlab_projects

    - name: Print list of projects
```

```
      ansible.builtin.debug:
        msg: "{{ gitlab_projects.json }}"

    - name: Fail if GitLab API is not accessible
      ansible.builtin.fail:
        msg: "Cannot access GitLab API. Check URL and token."
      when: gitlab_projects.status != 200

    - name: Check repository settings
      ansible.builtin.uri:
        url: "{{ gitlab_url }}/api/v4/projects/{{ item.id }}"
        method: GET
        headers:
          PRIVATE-TOKEN: "{{ gitlab_api_token }}"
      with_items: "{{ gitlab_projects.json }}"
      register: repo_settings

    - name: Ensure repositories are private
      ansible.builtin.debug:
        msg: "Repository {{ item.json.name }} is private"
      when: item.json.visibility == "private"

    - name: Fail if repository is not private
      ansible.builtin.fail:
        msg: "Repository {{ item.json.name }} is not private"
      when: item.json.visibility != "private"

    - name: Ensure 2FA is enabled for all users
      ansible.builtin.uri:
        url: "{{ gitlab_url }}/api/v4/users"
        method: GET
        headers:
          PRIVATE-TOKEN: "{{ gitlab_api_token }}"
      register: gitlab_users

    - name: Check if 2FA is enabled for each user
      ansible.builtin.uri:
        url: "{{ gitlab_url }}/api/v4/users/{{ item.id }}"
        method: GET
        headers:
          PRIVATE-TOKEN: "{{ gitlab_api_token }}"
      with_items: "{{ gitlab_users.json }}"
      register: user_settings

    - name: Fail if any user does not have 2FA enabled
      ansible.builtin.fail:
        msg: "User {{ item.json.username }} does not have 2FA enabled"
      when: not item.json.two_factor_enabled

    - name: Ensure default branch protection
      ansible.builtin.uri:
        url: "{{ gitlab_url }}/api/v4/projects/{{ item.id
}}/protected_branches"
        method: GET
        headers:
          PRIVATE-TOKEN: "{{ gitlab_api_token }}"
      with_items: "{{ gitlab_projects.json }}"
      register: protected_branches

    - name: Fail if default branch is not protected
      ansible.builtin.fail:
        msg: "Default branch of repository {{ item.id }} is not
protected"
      when: item.json | length == 0
```

Explanation:

1. **Playbook Header**:
    - `name`: A description of the playbook.
    - `hosts`: Specifies the target hosts (e.g., `scm_servers`).
    - `become: yes`: Run tasks with elevated privileges (i.e., `sudo`).

    - `gather_facts: no`: Skip gathering facts for faster execution.

2. **Variables**:

    - `gitlab_url`: The base URL for your GitLab instance.
    - `gitlab_api_token`: Your GitLab personal access token.

3. **Tasks**:

    - **Check if GitLab API is accessible**:
        - Use the `ansible.builtin.uri` module to send a GET request to the GitLab API to list projects.
        - Register the response in `gitlab_projects`.
    - **Print list of projects**:
        - Use the `ansible.builtin.debug` module to print the list of projects retrieved from GitLab.
    - **Fail if GitLab API is not accessible**:
        - Use the `ansible.builtin.fail` module to fail the playbook if the GitLab API is not accessible.
    - **Check repository settings**:
        - Use the `ansible.builtin.uri` module to get settings for each project.
        - Register the response in `repo_settings`.
    - **Ensure repositories are private**:
        - Use the `ansible.builtin.debug` module to print a message if the repository is private.
    - **Fail if repository is not private**:
        - Use the `ansible.builtin.fail` module to fail the playbook if any repository is not private.

- **Ensure 2FA is enabled for all users**:
  - Use the `ansible.builtin.uri` module to list all users.
  - Register the response in `gitlab_users`.
- **Check if 2FA is enabled for each user**:
  - Use the `ansible.builtin.uri` module to get settings for each user.
  - Register the response in `user_settings`.
- **Fail if any user does not have 2FA enabled**:
  - Use the `ansible.builtin.fail` module to fail the playbook if any user does not have 2FA enabled.
- **Ensure default branch protection**:
  - Use the `ansible.builtin.uri` module to check if the default branch of each project is protected.
  - Register the response in `protected_branches`.
- **Fail if default branch is not protected**:
  - Use the `ansible.builtin.fail` module to fail the playbook if the default branch of any repository is not protected.

Ansible Playbook: Docker Container Audit

Create a playbook named `docker_audit.yml` with the following content:

Ansible playbook designed to audit Docker containers on remote hosts. This playbook will check for common security settings and best practices, such as ensuring that containers are running with non-root users, verifying that containers are using the latest image versions, and checking for exposed ports.

```yaml
---
- name: Audit Docker Containers
  hosts: docker_hosts
  become: yes
  gather_facts: no

  tasks:
    - name: Check if Docker is installed
      ansible.builtin.command: docker --version
      register: docker_version
      changed_when: false
      ignore_errors: yes

    - name: Fail if Docker is not installed
      ansible.builtin.fail:
        msg: "Docker is not installed on this host."
      when: docker_version.rc != 0

    - name: List all running Docker containers
      ansible.builtin.command: docker ps --format "{{ '{{.ID}}' }}"
      register: docker_containers

    - name: Get details of each container
      ansible.builtin.command: docker inspect {{ item }}
      with_items: "{{ docker_containers.stdout_lines }}"
      register: container_details

    - name: Ensure containers are running as non-root users
      ansible.builtin.debug:
        msg: "Container {{ item.item }} is running as non-root user:
{{ item.stdout[0].Config.User }}"
      with_items: "{{ container_details.results }}"
      when: item.stdout[0].Config.User != ""

    - name: Fail if any container is running as root
      ansible.builtin.fail:
        msg: "Container {{ item.item }} is running as root user."
      with_items: "{{ container_details.results }}"
      when: item.stdout[0].Config.User == ""

    - name: Ensure containers are using the latest image versions
      ansible.builtin.command: docker images --format "{{
'{{.Repository}}:{{.Tag}}' }}" | grep {{ item.stdout[0].Config.Image
}}
      with_items: "{{ container_details.results }}"
      register: image_check

    - name: Fail if any container is not using the latest image
version
      ansible.builtin.fail:
        msg: "Container {{ item.item }} is not using the latest image
version."
      with_items: "{{ image_check.results }}"
      when: item.stdout | regex_search('latest') == None

    - name: Check for exposed ports
      ansible.builtin.debug:
        msg: "Container {{ item.item }} has exposed ports: {{
item.stdout[0].NetworkSettings.Ports }}"
      with_items: "{{ container_details.results }}"
      when: item.stdout[0].NetworkSettings.Ports != {}

    - name: Fail if any container has exposed ports
      ansible.builtin.fail:
        msg: "Container {{ item.item }} has exposed ports: {{
item.stdout[0].NetworkSettings.Ports }}"
      with_items: "{{ container_details.results }}"
      when: item.stdout[0].NetworkSettings.Ports != {}
```

Explanation:

1. **Playbook Header:**

- `name`: A description of the playbook.
- `hosts`: Specifies the target hosts (e.g., `docker_hosts`).
- `become: yes`: Run tasks with elevated privileges (i.e., `sudo`).
- `gather_facts: no`: Skip gathering facts for faster execution.

2. **Tasks**:
    - **Check if Docker is installed**:
        - Use the `ansible.builtin.command` module to check the Docker version and register the output.
        - Ignore errors to handle hosts where Docker might not be installed.
    - **Fail if Docker is not installed**:
        - Use the `ansible.builtin.fail` module to fail the playbook if Docker is not installed.
    - **List all running Docker containers**:
        - Use the `ansible.builtin.command` module to list all running Docker containers and register the output.
    - **Get details of each container**:
        - Use the `ansible.builtin.command` module to inspect each running container and register the details.
    - **Ensure containers are running as non-root users**:
        - Use the `ansible.builtin.debug` module to print a message if the container is running as a non-root user.
    - **Fail if any container is running as root**:
        - Use the `ansible.builtin.fail` module to fail the playbook if any container is running as root.

- **Ensure containers are using the latest image versions**:
  - Use the `ansible.builtin.command` module to check if the container is using the latest image version and register the output.
- **Fail if any container is not using the latest image version**:
  - Use the `ansible.builtin.fail` module to fail the playbook if any container is not using the latest image version.
- **Check for exposed ports**:
  - Use the `ansible.builtin.debug` module to print a message if the container has exposed ports.
- **Fail if any container has exposed ports**:
  - Use the `ansible.builtin.fail` module to fail the playbook if any container has exposed ports.

Ansible Playbook: Kubernetes Pod Audit

Create a playbook named `k8s_pod_audit.yml` with the following content:

Ansible playbook designed to audit Kubernetes pods on remote hosts. This playbook will check for common security settings and best practices, such as ensuring that pods are not running as root, using the latest image versions, and not exposing unnecessary ports.

Explanation:

1. **Playbook Header**:
   - `name`: A description of the playbook.
   - `hosts`: Specifies the target hosts (e.g., `k8s_nodes`).
   - `become: yes`: Run tasks with elevated privileges (i.e., `sudo`).
   - `gather_facts: no`: Skip gathering facts for faster execution.

2. **Variables**:
   - `kubeconfig_path`: Path to the Kubernetes configuration file. Update this path based on your environment.
3. **Tasks**:
   - **Ensure kubectl is installed**:
     - Use the `ansible.builtin.command` module to check if `kubectl` is installed and register the output.
     - Ignore errors to handle hosts where `kubectl` might not be installed.
   - **Fail if kubectl is not installed**:
     - Use the `ansible.builtin.fail` module to fail the playbook if `kubectl` is not installed.
   - **List all pods in all namespaces**:
     - Use the `ansible.builtin.command` module to list all pods in all namespaces in JSON format and register the output.
   - **Parse pod details**:
     - Use the `ansible.builtin.set_fact` module to parse the JSON output into a list of pods.
   - **Ensure pods are running as non-root users**:
     - Use the `ansible.builtin.debug` module to print a message if the pod is running as a non-root user.
   - **Fail if any pod is running as root**:
     - Use the `ansible.builtin.fail` module to fail the playbook if any pod is running as root.
   - **Ensure containers in pods are using the latest image versions**:

    - Use the `ansible.builtin.shell` module to check if the container is using the latest image version. This uses `docker pull` to ensure the latest image is available locally.
    - Ignore errors and do not mark the task as changed.
- **Fail if any container is not using the latest image version**:
    - Use the `ansible.builtin.fail` module to fail the playbook if any container is not using the latest image version.
- **Check for exposed ports**:
    - Use the `ansible.builtin.debug` module to print a message if the pod has exposed ports.
- **Fail if any pod has exposed ports**:
    - Use the `ansible.builtin.fail` module to fail the playbook if any pod has exposed ports.

Ansible Playbook: Database Audit (MySQL and PostgreSQL)

Create a playbook named `db_audit.yml` with the following content:

Ansible playbook designed to audit MySQL and PostgreSQL databases. This playbook will check for common security settings and best practices, such as ensuring that remote root access is disabled, verifying password policies, and checking for outdated database versions.

COPY

```
---
- name: Audit MySQL and PostgreSQL Databases
  hosts: db_servers
  become: yes
  gather_facts: no

  vars:
    mysql_root_user: root
    mysql_root_password: your_mysql_root_password
    pgsql_root_user: postgres
    pgsql_root_password: your_pgsql_root_password

  tasks:
    - name: Ensure MySQL is installed
      ansible.builtin.command: mysql --version
      register: mysql_version
      changed_when: false
      ignore_errors: yes

    - name: Fail if MySQL is not installed
      ansible.builtin.fail:
        msg: "MySQL is not installed on this host."
      when: mysql_version.rc != 0

    - name: Ensure PostgreSQL is installed
      ansible.builtin.command: psql --version
      register: pgsql_version
      changed_when: false
      ignore_errors: yes

    - name: Fail if PostgreSQL is not installed
      ansible.builtin.fail:
        msg: "PostgreSQL is not installed on this host."
      when: pgsql_version.rc != 0

    - name: Check MySQL remote root access
      ansible.builtin.shell: >
        mysql -u{{ mysql_root_user }} -p{{ mysql_root_password }} -e
"SELECT user, host FROM mysql.user WHERE user='root' AND
host!='localhost';"
      register: mysql_remote_root
      changed_when: false
      ignore_errors: yes
    - name: Fail if MySQL remote root access is enabled
      ansible.builtin.fail:
        msg: "MySQL remote root access is enabled."
      when: mysql_remote_root.stdout_lines | length > 0

    - name: Check PostgreSQL remote root access
      ansible.builtin.shell: >
        PGPASSWORD={{ pgsql_root_password }} psql -U {{
pgsql_root_user }} -c "\du"
      register: pgsql_users
      changed_when: false
      ignore_errors: yes

    - name: Fail if PostgreSQL remote root access is enabled
      ansible.builtin.fail:
        msg: "PostgreSQL remote root access is enabled."
      when: pgsql_users.stdout | regex_search('postgres') and not
pgsql_users.stdout | regex_search('localhost')

    - name: Check MySQL password policy
      ansible.builtin.shell: >
        mysql -u{{ mysql_root_user }} -p{{ mysql_root_password }} -e
"SHOW VARIABLES LIKE 'validate_password%';"
      register: mysql_password_policy
      changed_when: false

    - name: Ensure MySQL password policy is strong
      ansible.builtin.debug:
        msg: "MySQL password policy: {{ mysql_password_policy.stdout
}}"

    - name: Fail if MySQL password policy is weak
      ansible.builtin.fail:
        msg: "MySQL password policy is weak: {{
mysql_password_policy.stdout }}"
      when: mysql_password_policy.stdout | regex_search('LOW|MEDIUM')
    - name: Check PostgreSQL password policy
      ansible.builtin.shell: >
        PGPASSWORD={{ pgsql_root_password }} psql -U {{
pgsql_root_user }} -c "SHOW password_encryption;"
      register: pgsql_password_policy
      changed_when: false

    - name: Ensure PostgreSQL password policy is strong
      ansible.builtin.debug:
        msg: "PostgreSQL password policy: {{
pgsql_password_policy.stdout }}"

    - name: Fail if PostgreSQL password policy is weak
      ansible.builtin.fail:
        msg: "PostgreSQL password policy is weak: {{
pgsql_password_policy.stdout }}"
      when: not pgsql_password_policy.stdout | regex_search('scram-
sha-256')

    - name: Check MySQL version
      ansible.builtin.debug:
        msg: "MySQL version: {{ mysql_version.stdout }}"

    - name: Check PostgreSQL version
      ansible.builtin.debug:
        msg: "PostgreSQL version: {{ pgsql_version.stdout }}"
```

Explanation:

1. **Playbook Header**:

    - `name`: A description of the playbook.
    - `hosts`: Specifies the target hosts (e.g., `db_servers`).
    - `become: yes`: Run tasks with elevated privileges (i.e., `sudo`).
    - `gather_facts: no`: Skip gathering facts for faster execution.

2. **Variables**:

    - `mysql_root_user`: MySQL root user.
    - `mysql_root_password`: MySQL root user password.
    - `pgsql_root_user`: PostgreSQL root user.
    - `pgsql_root_password`: PostgreSQL root user password.

3. **Tasks**:

    - **Ensure MySQL is installed**:
        - Use the `ansible.builtin.command` module to check the MySQL version and register the output.
        - Ignore errors to handle hosts where MySQL might not be installed.
    - **Fail if MySQL is not installed**:
        - Use the `ansible.builtin.fail` module to fail the playbook if MySQL is not installed.
    - **Ensure PostgreSQL is installed**:
        - Use the `ansible.builtin.command` module to check the PostgreSQL version and register the output.
        - Ignore errors to handle hosts where PostgreSQL might not be installed.
    - **Fail if PostgreSQL is not installed**:
        - Use the `ansible.builtin.fail` module to fail the playbook if PostgreSQL is not installed.
    - **Check MySQL remote root access**:
        - Use the `ansible.builtin.shell` module to check for remote root access in MySQL and register the output.
    - **Fail if MySQL remote root access is enabled**:

    - Use the `ansible.builtin.fail` module to fail the playbook if MySQL remote root access is enabled.
- **Check PostgreSQL remote root access**:
    - Use the `ansible.builtin.shell` module to check for remote root access in PostgreSQL and register the output.
- **Fail if PostgreSQL remote root access is enabled**:
    - Use the `ansible.builtin.fail` module to fail the playbook if PostgreSQL remote root access is enabled.
- **Check MySQL password policy**:
    - Use the `ansible.builtin.shell` module to check the MySQL password policy and register the output.
- **Ensure MySQL password policy is strong**:
    - Use the `ansible.builtin.debug` module to print the MySQL password policy.
- **Fail if MySQL password policy is weak**:
    - Use the `ansible.builtin.fail` module to fail the playbook if the MySQL password policy is weak.
- **Check PostgreSQL password policy**:
    - Use the `ansible.builtin.shell` module to check the PostgreSQL password policy and register the output.
- **Ensure PostgreSQL password policy is strong**:
    - Use the `ansible.builtin.debug` module to print the PostgreSQL password policy.
- **Fail if PostgreSQL password policy is weak**:

    - Use the `ansible.builtin.fail` module to fail the playbook if the PostgreSQL password policy is weak.
- **Check MySQL version**:
    - Use the `ansible.builtin.debug` module to print the MySQL version.
- **Check PostgreSQL version**:
    - Use the `ansible.builtin.debug` module to print the PostgreSQL version.

Ansible Playbook: Manage AWS Security Group Rules

Create a playbook named `manage_security_groups.yml` with the following content:

Below is an Ansible playbook designed to automate the management of AWS security group rules for EC2 instances. This playbook will use the `amazon.aws.ec``2_security_group` module to ensure that specific security group rules are in place.

```
COPY
---
- name: Manage AWS Security Group Rules
  hosts: localhost
  gather_facts: no
  tasks:
    - name: Ensure security group exists
      amazon.aws.ec2_security_group:
        name: my_security_group
        description: Security group for my EC2 instances
        vpc_id: vpc-0abcd1234efgh5678  # Replace with your VPC ID
        region: us-east-1  # Replace with your desired AWS region
        state: present
      register: sg_result

    - name: Ensure security group rules
      amazon.aws.ec2_security_group:
        name: my_security_group
        vpc_id: vpc-0abcd1234efgh5678  # Replace with your VPC ID
        region: us-east-1  # Replace with your desired AWS region
        rules:
          - proto: tcp
            from_port: 22
            to_port: 22
            cidr_ip: 0.0.0.0/0  # SSH from anywhere (modify as needed)
          - proto: tcp
            from_port: 80
            to_port: 80
            cidr_ip: 0.0.0.0/0  # HTTP from anywhere
          - proto: tcp
            from_port: 443
            to_port: 443
            cidr_ip: 0.0.0.0/0  # HTTPS from anywhere
        rules_egress:
          - proto: all
            from_port: -1
            to_port: -1
            cidr_ip: 0.0.0.0/0  # Allow all outbound traffic
      register: sg_rules_result

    - name: Output security group details
      ansible.builtin.debug:
        msg: "Security Group {{ sg_result.group_id }} has been updated
with rules: {{ sg_rules_result }}"
```

Explanation:

1. **Playbook Header**:
   - `name`: A description of the playbook.
   - `hosts`: Specifies the target host(s). In this case, it's `localhost` because the AWS tasks are executed from the control node.

- `gather_facts: no`: Skip gathering facts for faster execution.

2. **Tasks**:

   - **Ensure security group exists**:

     - Use the `amazon.aws.ec2_security_group` module to create or ensure the existence of a security group named `my_security_group`.
     - The `vpc_id` and `region` should be specified according to your AWS setup.
     - Register the result in `sg_result` to capture details about the security group.

   - **Ensure security group rules**:

     - Use the `amazon.aws.ec2_security_group` module again to ensure specific rules are applied to the security group.
     - Define the inbound rules (e.g., SSH, HTTP, HTTPS) and outbound rules (e.g., allow all outbound traffic).
     - Register the result in `sg_rules_result` to capture details about the rules applied.

   - **Output security group details**:

     - Use the `ansible.builtin.debug` module to print out details about the security group and the applied rules for verification purposes.

Ansible Playbook: Monitor Critical Files

Create a playbook named `monitor_files.yml` with the following content:

Ansible playbook designed to monitor critical files for unauthorized changes. This playbook will leverage the `ansible.builtin.stat` module to check the file attributes and the `ansible.builtin.copy` module to save a baseline snapshot of the file for future comparison.

COPY

```yaml
---
- name: Monitor Critical Files for Unauthorized Changes
  hosts: monitored_servers
  become: yes
  gather_facts: no

  vars:
    critical_files:
      - /etc/passwd
      - /etc/shadow
      - /etc/ssh/sshd_config
      - /etc/sudoers

  tasks:
    - name: Ensure baseline directory exists
      ansible.builtin.file:
        path: /var/baseline
        state: directory
        mode: '0755'

    - name: Create baseline snapshots if not exist
      ansible.builtin.copy:
        src: "{{ item }}"
        dest: "/var/baseline/{{ item | basename }}.baseline"
        remote_src: yes
        force: no
      with_items: "{{ critical_files }}"

    - name: Check current file attributes
      ansible.builtin.stat:
        path: "{{ item }}"
      with_items: "{{ critical_files }}"
      register: current_files

    - name: Check baseline file attributes
      ansible.builtin.stat:
        path: "/var/baseline/{{ item | basename }}.baseline"
      with_items: "{{ critical_files }}"
      register: baseline_files

    - name: Compare current and baseline attributes
      ansible.builtin.shell: >
        diff -q "{{ item.0.stat.path }}" "{{ item.1.stat.path }}"
      with_together:
        - "{{ current_files.results }}"
        - "{{ baseline_files.results }}"
      register: diff_results
      ignore_errors: yes

    - name: Report changes if any
      ansible.builtin.debug:
        msg: "File {{ item.item.0.stat.path }} has changed"
      with_items: "{{ diff_results.results }}"
      when: item.rc == 1

    - name: Fail if unauthorized changes detected
      ansible.builtin.fail:
        msg: "Unauthorized changes detected in file {{
item.item.0.stat.path }}"
      with_items: "{{ diff_results.results }}"
          when: item.rc == 1
```

Explanation:

1. **Playbook Header**:
    - `name`: A description of the playbook.
    - `hosts`: Specifies the target hosts (e.g., `monitored_servers`).
    - `become: yes`: Run tasks with elevated privileges (i.e., `sudo`).
    - `gather_facts: no`: Skip gathering facts for faster execution.

2. **Variables**:
    - `critical_files`: A list of critical files to be monitored.
3. **Tasks**:
    - **Ensure baseline directory exists**:
        - Use the `ansible.builtin.file` module to ensure the directory `/var/baseline` exists, where baseline snapshots will be stored.
    - **Create baseline snapshots if not exist**:
        - Use the `ansible.builtin.copy` module to create a baseline snapshot of each critical file. The `force: no` option ensures that the baseline is only created if it does not already exist.
    - **Check current file attributes**:
        - Use the `ansible.builtin.stat` module to gather the current attributes of each critical file and register the results.
    - **Check baseline file attributes**:
        - Use the `ansible.builtin.stat` module to gather the attributes of each baseline file and register the results.
    - **Compare current and baseline attributes**:
        - Use the `ansible.builtin.shell` module with the `diff` command to compare the current file attributes with the baseline attributes. The results are registered and errors are ignored.
    - **Report changes if any**:
        - Use the `ansible.builtin.debug` module to print a message if any file has changed.
    - **Fail if unauthorized changes detected**:

- Use the `ansible.builtin.fail` module to fail the playbook if any unauthorized changes are detected.

Ansible Playbook: Log Collection and Analysis

Centralizing logs from various sources such as syslog and application logs for security analysis is a crucial task in ensuring comprehensive monitoring and analysis capabilities. Below is an Ansible playbook that demonstrates how to collect logs from remote servers and centralize them on a centralized logging server using `rsyslog`.

Create a playbook named `log_collection_analysis.yml` with the following content:

```
COPY
---
- name: Log Collection and Analysis
  hosts: all
  become: yes
  tasks:
    - name: Install rsyslog
      ansible.builtin.package:
        name: rsyslog
        state: present

    - name: Configure rsyslog for centralized logging
      ansible.builtin.lineinfile:
        path: /etc/rsyslog.conf
        regexp: "^#*\\s*\\$ModLoad imtcp"
        line: "$ModLoad imtcp"
        state: present
        backup: yes

    - name: Ensure rsyslog listens on TCP port 514
      ansible.builtin.lineinfile:
        path: /etc/rsyslog.conf
        regexp: "^#*\\s*\\$InputTCPServerRun 514"
        line: "$InputTCPServerRun 514"
```

```yaml
        state: present
        backup: yes

    - name: Restart rsyslog service
      ansible.builtin.service:
        name: rsyslog
        state: restarted

    - name: Ensure rsyslog service is enabled
      ansible.builtin.service:
        name: rsyslog
        enabled: yes
        state: started

- name: Forward logs to centralized server
  hosts: centralized_logging_server
  become: yes
  tasks:
    - name: Install rsyslog
      ansible.builtin.package:
        name: rsyslog
        state: present

    - name: Configure rsyslog to receive logs from clients
      ansible.builtin.lineinfile:
        path: /etc/rsyslog.conf
        regexp: "^#*\\s*\$ModLoad imtcp"
        line: "$ModLoad imtcp"
        state: present
        backup: yes

    - name: Ensure rsyslog listens on TCP port 514
      ansible.builtin.lineinfile:
        path: /etc/rsyslog.conf
        regexp: "^#*\\s*\$InputTCPServerRun 514"
        line: "$InputTCPServerRun 514"
        state: present
        backup: yes

    - name: Restart rsyslog service
      ansible.builtin.service:
        name: rsyslog
        state: restarted

    - name: Ensure rsyslog service is enabled
      ansible.builtin.service:
        name: rsyslog
        enabled: yes
        state: started
```

Explanation:

1. **Playbook Structure**:
    - The playbook consists of two plays:
        - **First Play (Log Collection Configuration on All Hosts)**:
            - Installs `rsyslog` on all hosts (`hosts: all`).
            - Configures `rsyslog` to listen for incoming logs over TCP (`$ModLoad imtcp` and `$InputTCPServerRun 514`).
            - Restarts and enables the `rsyslog` service to apply the configuration changes.
        - **Second Play (Centralized Logging Server Configuration)**:
            - Installs `rsyslog` on the centralized logging server (`hosts: centralized_logging_server`).
            - Configures `rsyslog` on the centralized server to receive logs from clients (`$ModLoad imtcp` and `$InputTCPServerRun 514`).
            - Restarts and enables the `rsyslog` service on the centralized server to apply the configuration changes.

2. **Tasks**:

- **Install rsyslog**: Ensures that `rsyslog` is installed on the hosts.
- **Configure rsyslog**:
  - Uses `ansible.builtin.lineinfile` module to modify `/etc/rsyslog.conf` to enable TCP logging and specify the TCP port 514 for logging.
- **Restart and enable rsyslog service**: Ensures that the `rsyslog` service is restarted and enabled to apply the configuration changes.

Ansible Playbook: Firewall Rules Management with iptables

Automating firewall rule updates based on security policies and requirements analysis is crucial for maintaining a secure environment. Below is an Ansible playbook that demonstrates how to manage firewall rules on Linux servers using `iptables`. Adjustments can be made depending on your specific firewall solution (e.g., `firewalld` for CentOS/RHEL, `ufw` for Ubuntu).

Create a playbook named `firewall_management.yml` with the following content:

COPY

```
---
- name: Firewall Rules Management
  hosts: firewall_servers
  become: yes
  tasks:
    - name: Allow SSH (Port 22) from specific IP addresses
      ansible.builtin.iptables:
        chain: INPUT
        protocol: tcp
        destination_port: 22
        source: "{{ item }}"
        jump: ACCEPT
      with_items:
        - 192.168.1.100   # Replace with your allowed IP addresses
        - 192.168.1.101
        - 10.0.0.1

    - name: Allow HTTP (Port 80) from any IP
      ansible.builtin.iptables:
        chain: INPUT
        protocol: tcp
        destination_port: 80
        jump: ACCEPT

    - name: Allow HTTPS (Port 443) from any IP
      ansible.builtin.iptables:
        chain: INPUT
        protocol: tcp
        destination_port: 443
        jump: ACCEPT

    - name: Deny all other inbound traffic
      ansible.builtin.iptables:
        chain: INPUT
        policy: DROP

    - name: Save iptables rules
      ansible.builtin.shell: iptables-save > /etc/sysconfig/iptables
```

Explanation:

1. **Playbook Structure**:
    - The playbook contains a single play targeting `firewall_servers`.
    - The `become: yes` directive ensures that tasks are executed with root privileges.
2. **Tasks**:
    - **Allow SSH (Port 22) from specific IP addresses**:

    - Uses the `ansible.builtin.iptables` module to allow SSH (TCP port 22) connections from specified IP addresses (`192.168.1.100`, `192.168.1.101`, `10.0.0.1`).
    - Loops through the list of allowed IP addresses using `with_items`.
- **Allow HTTP (Port 80) from any IP**:
    - Allows HTTP (TCP port 80) traffic from any IP address.
- **Allow HTTPS (Port 443) from any IP**:
    - Allows HTTPS (TCP port 443) traffic from any IP address.
- **Deny all other inbound traffic**:
    - Sets the default policy for inbound traffic to `DROP`, effectively denying all other incoming connections.
- **Save iptables rules**:
    - Uses the `ansible.builtin.shell` module to save the current `iptables` rules to `/etc/sysconfig/iptables` (adjust this path for your distribution).

Ansible Playbook: Backup and Restore Procedures

Automating backup tasks and ensuring robust restoration processes are critical for maintaining data integrity and availability. Below is an Ansible playbook that demonstrates how to automate backup and restore procedures for a directory, including encryption of backups using `tar` and `gpg` (GNU Privacy Guard).

Create a playbook named `backup_restore.yml` with the following content:

COPY

```
---
- name: Backup and Restore Procedures
  hosts: backup_server
  become: yes
  vars:
    backup_directory: "/backup"
    source_directory: "/path/to/your/source"
    encrypted_backup_file: "backup.tar.gz.gpg"
    encryption_passphrase: "your_encryption_passphrase"

  tasks:
    - name: Ensure backup directory exists
      ansible.builtin.file:
        path: "{{ backup_directory }}"
        state: directory
        mode: '0755'

    - name: Backup directory with encryption
      ansible.builtin.shell: |
        tar -czf - "{{ source_directory }}" | gpg --symmetric --
passphrase "{{ encryption_passphrase }}" -o "{{ backup_directory }}/{{
encrypted_backup_file }}"
      args:
        warn: no
      environment:
        GNUPGHOME: "{{ backup_directory }}/.gnupg"
      register: backup_result

    - name: Report backup status
      ansible.builtin.debug:
        msg: "Backup task result: {{ backup_result.stdout_lines }}"

- name: Restore Backup
  hosts: restore_server
  become: yes
  vars:
    backup_directory: "/backup"
    encrypted_backup_file: "backup.tar.gz.gpg"
    decryption_passphrase: "your_encryption_passphrase"
    restore_directory: "/path/to/restore"
  tasks:
    - name: Ensure restore directory exists
      ansible.builtin.file:
        path: "{{ restore_directory }}"
        state: directory
        mode: '0755'

    - name: Decrypt and extract backup
      ansible.builtin.shell: |
        gpg --decrypt --passphrase "{{ decryption_passphrase }}" "{{
backup_directory }}/{{ encrypted_backup_file }}" | tar -xzf - -C "{{
restore_directory }}"
      args:
        warn: no
      environment:
        GNUPGHOME: "{{ backup_directory }}/.gnupg"
      register: restore_result

    - name: Report restore status
      ansible.builtin.debug:
        msg: "Restore task result: {{ restore_result.stdout_lines }}"
```

Explanation:

1. **Playbook Structure**:
   - The playbook is divided into two plays:
     - **Backup Procedure**: Executes on `backup_server` to create an encrypted backup of the specified directory.
     - **Restore Procedure**: Executes on `restore_server` to decrypt and restore the backup to a specified directory.
2. **Tasks**:
   - **Ensure backup/restore directory exists**: Uses `ansible.builtin.file` module to ensure that the backup or restore directory exists with the correct

permissions (`mode: '0755'`).

- **Backup directory with encryption**: Uses `ansible.builtin.shell` module to create a backup of `source_directory` using `tar` and encrypt it with `gpg`. The `encryption_passphrase` is used to symmetrically encrypt the backup file. The `GNUPGHOME` environment variable ensures the GPG keyring is stored within the backup directory for security.
- **Report backup status**: Uses `ansible.builtin.debug` module to print the result of the backup task.
- **Decrypt and extract backup**: On the restore server, uses `ansible.builtin.shell` module to decrypt the encrypted backup file using `gpg` and extract it using `tar` to the `restore_directory`. The `decryption_passphrase` is used for decryption.
- **Report restore status**: Uses `ansible.builtin.debug` module to print the result of the restore task.

3. **Variables**:

   - `backup_directory`: Directory where backups are stored.
   - `source_directory`: Directory to be backed up.
   - `encrypted_backup_file`: Name of the encrypted backup file.
   - `encryption_passphrase`: Passphrase used for encrypting the backup.
   - `decryption_passphrase`: Passphrase used for decrypting the backup.

References

- Hands-On Security in DevOps by Tony Hsiang-Chih Hsu

Subscribe to our newsletter

Read articles from **DevSecOpsGuides** directly inside your inbox. Subscribe to the newsletter, and don't miss out.

Enter your email address SUBSCRIBE

ansible Devops DevSecOps audit AWS

Written by

RR Reza Rashidi Follow

Published on

DevSecOpsGuides Follow

MORE ARTICLES

RR Reza Rashidi

eBPF cheatsheet

RR Reza Rashidi

DevSecOps Security Architecture

eBPF (Extended Berkeley Packet Filter) is a powerful technology for monitoring and analyzing system ...

In the rapidly evolving landscape of cybersecurity, DevSecOps Security Architecture emerges as a cri...

RR **Reza Rashidi**

Attacking Secrets

A Secrets and Vault Manager is a critical tool in modern IT infrastructure, designed to securely sto...