Ansible Playbook Documentation

Play | Update

```
- name: Update
  hosts: staging
  become: yes
```

This play would run the tasks on the staging server. The `become: yes` is used to run the tasks as root.

Task 01 - Update and upgrade apt packages

```
tasks:
  - name: Update apt packages
    ansible.builtin.apt:
      upgrade: yes
      update_cache: yes
      cache_valid_time: 86400
```

Equivalent with the following bash command:

```
sudo apt update && sudo apt upgrade -y
```

This task would update and upgrade all apt packages on the staging server. The `become: yes` is used to run the task as root. The `ansible.builtin.apt` module is used to update and upgrade the packages. The `upgrade: yes` is used to upgrade the packages. The `update_cache: yes` is used to update the cache. The `cache_valid_time: 86400` is used to set the cache time to 24 hours.

TOP

Play | Clone Repository

```
- name: Clone repository
  hosts: staging
  become: true
```

Task 02 - Clear destination directory

```
- name: Clear destination directory
  ansible.builtin.file:
    path: /home/runner/gear5
    state: absent
```

Equivalent with the following bash command:

```
sudo rm -rf /home/runner/gear5
```

This task would clear the destination directory on the staging server. `state` is used to set the state of the directory. `absent` is used to remove the directory. It uses the `ansible.builtin.file` module found in Ansible core and is used to manage files and file properties.

Task 03 - Clone private repository

```
- name: Clone private repository
  ansible.builtin.git:
    repo: https://x-access-token:{{ lookup('env', 'GITHUB_TOKEN')
}}@github.com/SamanPetfat/gear5.git
    dest: /home/runner/gear5
    version: staging
```

This task clones the staging branch of the SamanPetfat/gear5 repository from GitHub onto the target system at the /home/runner/gear5 directory. The lookup function is used to retrieve the value of the GITHUB_TOKEN environment variable, which is used to authenticate with the private repository.

Play | Install Pre-requisites

```
- name: Install pre-requisites
  hosts: staging
  become: true
    - vars/default.yml
```

`vars/default.yml` is used to store the variables used in the tasks. For example, the `mysql_root_password` variable is used to set the root password for the MySQL database.

Task 04 - Install pre-requisites

```
- name: Install pre-requisites
  ansible.builtin.apt:
    name: aptitude
    state: present
  tags: [system]
```

Equivalent with the following bash command:

```
sudo apt install aptitude -y
```

This task ensures that the aptitude package is installed on the target system. If the package is already installed, the task will have no effect. If the package is not installed, it will be installed. The tags parameter is used to apply the system tag to the task, which can be used to selectively run or skip tasks based on their tags.

Task 05 - Install LAMP stack

```
ansible.builtin.apt:
        name: "{{ item }}"
        state: present
      loop:
        - apache2
        - mysql-server
        - python3-pymysql
        - php
        - php-mysql
        - libapache2-mod-php
        - jq
        - swaks
      tags: [system]
```

Equivalent with the following bash command:

```
sudo apt install apache2 mysql-server python3-pymysql php php-mysql
libapache2-mod-php jq swaks -y
```

This task installs the following LAMP packages on the target system:

- apache2: The Apache web server.

- mysql-server: The MySQL database server.
- python3-pymysql: A Python library for connecting to MySQL databases.
- php: The PHP scripting language.
- php-mysql: A PHP extension for connecting to MySQL databases.
- libapache2-mod-php: An Apache module for running PHP scripts.
- jq: A command-line JSON processor.
- swaks: A command-line SMTP testing tool.

TOP

Task 06 - Install PHP Extensions

```yaml
- name: Install PHP extensions
  ansible.builtin.apt:
    name: "{{ item }}"
    state: present
  loop:
    - php-curl
    - php-gd
    - php-mbstring
    - php-xml
    - php-xmlrpc
    - php-zip
    - php-soap
    - php-intl
  tags: [system]
```

Equivalent with the following bash command:

```
sudo apt install php-curl php-gd php-mbstring php-xml php-xmlrpc php-zip
php-soap php-intl -y
```

This task installs the following PHP extensions on the target system:

- php-curl: A PHP extension for interacting with remote resources via the libcurl library.
- php-gd: A PHP extension for manipulating images.
- php-mbstring: A PHP extension for handling multibyte character encodings.
- php-xml: A PHP extension for parsing XML documents.
- php-xmlrpc: A PHP extension for communicating with XML-RPC services.
- php-zip: A PHP extension for manipulating ZIP archives.
- php-soap: A PHP extension for communicating with SOAP services.
- php-intl: A PHP extension for internationalization support.

TOP

Play | Configure Staging Server

```
- name: Configure Staging
  hosts: staging
  become: true
  vars_files:
    - vars/default.yml
  vars:
    wordpress_dir: "/var/www/{{ http_host }}/wordpress"
    temp_dir: "/tmp"
```

This play sets up the staging environment for a WordPress site by defining two variables: wordpress_dir and temp_dir. The wordpress_dir variable specifies the path to the WordPress installation directory, which is /var/www/{{ http_host }}/wordpress. The http_host variable is defined in the vars/default.yml file, which is loaded before the play runs. The temp_dir variable specifies the path to a temporary directory that can be used during the play.

TOP

Task 07 - Apache | Create document root directory

```
- name: Create document root
      ansible.builtin.file:
        path: "/var/www/{{ http_host }}"
        state: directory
        owner: "www-data"
        group: "www-data"
        mode: "0755"
      tags: [apache]
```

Equivalent with the following bash command:

```
sudo mkdir -p /var/www/your_domain
chown -R www-data:www-data /var/www/your_domain
chmod -R 755 /var/www/your_domain
```

This task generates a configuration file for the Apache web server at /etc/apache2/sites-available/{{ http_host }}.conf on the target system. The src/path parameter specifies the path to the Jinja2 template file to use, which is templates/apache2.conf.j2. The dest parameter specifies the path to the file to generate on the target system. The owner, group, and mode parameters specify the ownership and permissions of the generated file. mode: "0755" is used to set the permissions of the directory to 755.

TOP

Task 08 - Apache | Set up virtual host

```
- name: Set up Apache VirtualHost
      ansible.builtin.template:
        src: "files/apache.conf.j2"
        dest: "/etc/apache2/sites-available/{{ http_conf }}"
      notify: Reload Apache
      tags: [apache]
```

This task generates an Apache VirtualHost configuration file at /etc/apache2/sites-available/{{ http_conf }} on the target system. The notify parameter is used to trigger the Reload Apache handler when the task completes successfully. `ansible.builtin.template` is used to generate the configuration file from a Jinja2 template file.

```
http_host: "your_domain"
http_conf: "your_domain.conf"
http_port: "80"
```

Task 09 - Apache | Enable rewrite module

```
 - name: Enable rewrite module
      ansible.builtin.shell: /usr/sbin/a2enmod rewrite
      notify: Restart Apache
      tags: [apache]
```

Equivalent with the following bash command:

```
sudo a2enmod rewrite
```

It enables the Apache rewrite module on the target system. The notify parameter is used to trigger the Restart Apache handler when the task completes successfully.

Task 10 - MySQL | Ensure .my.cnf exists

```
- name: Ensure .my.cnf for root is present
      ansible.builtin.template:
        src: files/my.cnf.j2
```

```
        dest: /root/.my.cnf
        owner: root
        mode: "0600"
      tags: [mysql, mysql-root]
```

It generates a .my.cnf file for the root user on the target system. This file contains the credentials for the root user to access the MySQL database. The generated file is placed at /root/.my.cnf on the target system with the ownership set to root and the permissions set to 0600.

TOP

Task 11 - MySQL | Check if the root user has a password

```
- name: Check if root password is already set
      community.mysql.mysql_user:
        name: root
        password: "{{ mysql_root_password }}"
        login_unix_socket: /var/run/mysqld/mysqld.sock
      check_mode: true
      register: root_password_check
      failed_when: false
      tags: [mysql, mysql-root]
```

Equivalent with the following bash command:

```
sudo mysql -u root -p
```

Checks if the root user has a password set on the target system. The community.mysql.mysql_user module is used to manage MySQL users. The check_mode parameter is used to run the task in check mode, which means that the task will not make any changes to the target system. The register parameter is used to register the result of the task in the root_password_check variable. The failed_when parameter is used to prevent the task from failing if the root user does not have a password set.

TOP

Task 12 - MySQL | Set root password

```
- name: Set the root password
  community.mysql.mysql_user:
    name: root
    password: "{{ mysql_root_password }}"
    login_unix_socket: /var/run/mysqld/mysqld.sock
  when: root_password_check.changed
  tags: [mysql, mysql-root]
```

Equivalent with the following sql command:

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'MYpassword123!';
```

This task sets the root password for the MySQL database. The when parameter is used to run the task only if the root_password_check variable has changed, which means that the root user did not have a password set.

TOP

Task 13 - MySQL | Remove anonymous users

```
- name: Remove all anonymous user accounts
    community.mysql.mysql_user:
      name: ""
      host_all: true
      state: absent
      login_user: root
      login_password: "{{ mysql_root_password }}"
    tags: [mysql]
```

Equivalent with the following sql command:

```
DELETE FROM mysql.user WHERE User='';
```

It removes all anonymous user accounts from the MySQL database on the target system using the community.mysql.mysql_user module.

TOP

Task 14 - MySQL | Remove test database

```
- name: Remove the MySQL test database
    community.mysql.mysql_db:
      name: test
      state: absent
      login_user: root
      login_password: "{{ mysql_root_password }}"
    tags: [mysql]
```

Equivalent with the following sql command:

```
DROP DATABASE test;
```

It removes the test database. This database is created by default when the MySQL database server is installed. It is mostly used for testing purposes and is not needed for a production environment.

Task 15 - MySQL | Create database for WordPress

```
- name: Creates database for WordPress
    community.mysql.mysql_db:
      name: "{{ mysql_db }}"
      state: present
      login_user: root
      login_password: "{{ mysql_root_password }}"
    tags: [mysql]
```

Equivalent with the following sql command:

```
CREATE DATABASE wordpress;
```

It creates a database for the WordPress site. The name of the database is specified by the mysql_db variable, which is defined in the vars/default.yml file. The state parameter is used to ensure that the database exists.

Task 16 - MySQL | Create user for WordPress

```
- name: Create MySQL user for WordPress
      community.mysql.mysql_user:
        name: "{{ mysql_user }}"
        password: "{{ mysql_password }}"
        priv: "{{ mysql_db }}.*:ALL"
        state: present
        login_user: root
        login_password: "{{ mysql_root_password }}"
      tags: [mysql]
```

Equivalent with the following sql command:

```
CREATE USER 'root'@'localhost' IDENTIFIED BY 'MYpassword123!';
```

It creates a MySQL user for the WordPress site.

TOP

Task 17 - WordPress | Download WordPress archive

```
- name: Download latest WordPress archive
    ansible.builtin.get_url:
      url: https://wordpress.org/latest.tar.gz
      dest: "/tmp/latest.tar.gz"
      mode: "0755"
    register: download_result
    until: download_result is success
    retries: 5
    delay: 10
    tags: [wordpress]
```

Equivalent with the following bash command:

```
wget https://wordpress.org/latest.tar.gz
```

This task downloads the latest WordPress archive from the official website using the ansible.builtin.get_url module. The until parameter is used to retry the task until the download is successful. The retries parameter is used to set the number of retries. The delay parameter is used to set the delay between retries.

TOP

Task 18 - WordPress | Extract WordPress archive

```
- name: Unpack WordPress archive if download succeeded
    ansible.builtin.unarchive:
      src: "/tmp/latest.tar.gz"
      dest: "/var/www/{{ http_host }}"
      remote_src: true
      creates: "/var/www/{{ http_host }}/wordpress"
    when: download_result is success
    tags: [wordpress]
```

Equivalent with the following bash command:

```
tar -xzf latest.tar.gz
```

This task extracts the WordPress archive to the /var/www/{{ http_host }} directory on the target system. The creates parameter is used to check if the WordPress directory already exists. If it does, the task will have no effect. If it does not, the WordPress archive will be extracted.

TOP

Task 19 - WordPress | Set ownership

```
- name: Set ownership
      ansible.builtin.file:
        path: "/var/www/{{ http_host }}"
        state: directory
        recurse: true
        owner: www-data
        group: www-data
      when: download_result is changed
      tags: [wordpress]
```

Equivalent with the following bash command:

```
chown -R www-data:www-data /var/www/your_domain
```

This task sets the ownership of the WordPress installation directory to the www-data user and group. The when parameter is used to run the task only if the download_result variable has changed, which means that the WordPress archive was downloaded and extracted.

TOP

Task 20 - WordPress | Set permissions for folders

```
- name: Set permissions for directories
  ansible.builtin.shell: "/usr/bin/find /var/www/{{ http_host
}}/wordpress/ -type d -exec chmod 777 {} \\;"
  when: download_result.changed
```

Equivalent with the following bash command:

```
find /var/www/your_domain/wordpress/ -type d -exec chmod 777 {} \;
```

This sets the permissions of all directories in the WordPress installation directory to 777 using the ansible.builtin.shell module, but only if the WordPress archive was downloaded in a previous task. `type: d` is used to set the permissions to directories only.

Task 21 - WordPress | Set permissions for files

```
- name: Set permissions for files
      ansible.builtin.shell: "/usr/bin/find /var/www/{{ http_host
}}/wordpress/ -type f -exec chmod 777 {} \\;"
      when: download_result.changed
```

Equivalent with the following bash command:

```
find /var/www/your_domain/wordpress/ -type f -exec chmod 777 {} \;
```

This sets the permissions of all files in the WordPress installation directory to 777 using the ansible.builtin.shell module, but only if the WordPress archive was downloaded in a previous task. `type: f` is used to set the permissions to files only.

Task 22 - WordPress | Set up wp-config.php

```
- name: Set up wp-config
      ansible.builtin.template:
        src: "files/wp-config.php.j2"
        dest: "/var/www/{{ http_host }}/wordpress/wp-config.php"
      when: download_result.changed
      tags: [wordpress]
```

This task generates a wp-config.php file for the WordPress. This file contains the database credentials for the WordPress site. The generated file is placed at /var/www/{{ http_host }}/wordpress/wp-config.php on the target system.

Task 23 - WordPress | Remove folder `html``

```
- name: Remove folder (html)
      ansible.builtin.file:
```

```
        path: "/var/www/html"
        state: absent
      when: download_result.changed
      tags: [wordpress]
```

Equivalent with the following bash command:

```
sudo rm -rf /var/www/html
```

It removes the html folder from the WordPress installation directory. This folder is created by default when the WordPress archive is extracted. It is not needed for a production environment.

TOP

Task 24 - WordPress | Remove folder `plugins`

```
- name: Remove folder (Plugins)
    ansible.builtin.file:
      path: "/var/www/{{ http_host }}/wordpress/wp-content/plugins"
      state: absent
    register: plugins_result
    tags: [wordpress]
```

Equivalent with the following bash command:

```
sudo rm -rf /var/www/your_domain/wordpress/wp-content/plugins
```

It removes the plugins folder from the WordPress installation directory.

TOP

Task 25 - WordPress | Create symlink to plugins

```
- name: Create a symbolic link (Plugins)
    ansible.builtin.file:
      src: "/home/runner/gear5/www/html/wp-content/plugins"
      dest: "/var/www/{{ http_host }}/wordpress/wp-content/plugins"
      owner: www-data
      group: www-data
      state: link
    tags: [wordpress]
```

Equivalent with the following bash command:

```
sudo ln -s /home/runner/gear5/www/html/wp-content/plugins
/var/www/your_domain/wordpress/wp-content/plugins
```

It creates a symbolic link to the plugins folder in the WordPress installation directory. This is done to make it easier to install plugins on the WordPress site.

TOP

Task 26 - WordPress | Remove folder `themes`

```yaml
- name: Remove folder (Themes)
      ansible.builtin.file:
        path: "/var/www/{{ http_host }}/wordpress/wp-content/themes"
        state: absent
      register: themes_result
      tags: [wordpress]
```

Equivalent with the following bash command:

```
sudo rm -rf /var/www/your_domain/wordpress/wp-content/themes
```

It removes the themes folder from the WordPress installation directory.

TOP

Task 27 - WordPress | Create symlink to themes

```yaml
- name: Create a symbolic link (Themes)
      ansible.builtin.file:
        src: "/home/runner/gear5/www/html/wp-content/themes"
        dest: "/var/www/{{ http_host }}/wordpress/wp-content/themes"
        owner: www-data
        group: www-data
        state: link
      tags: [wordpress]
```

Equivalent with the following bash command:

```
sudo ln -s /home/runner/gear5/www/html/wp-content/themes
/var/www/your_domain/wordpress/wp-content/themes
```

It creates a symbolic link to the themes folder in the WordPress installation directory.

TOP

Task 28 - WordPress | Remove folder `uploads`

```
- name: Remove folder (Uploads)
      ansible.builtin.file:
        path: "/var/www/{{ http_host }}/wordpress/wp-content/uploads"
        state: absent
      tags: [wordpress]
```

Equivalent with the following bash command:

```
sudo rm -rf /var/www/your_domain/wordpress/wp-content/uploads
```

It removes the uploads folder from the WordPress installation directory.

TOP

Task 29 - WordPress | Create symlink to uploads

```
- name: Create a symbolic link (Uploads)
      ansible.builtin.file:
        src: "/home/runner/gear5/www/html/wp-content/uploads"
        dest: "/var/www/{{ http_host }}/wordpress/wp-content/uploads"
        owner: www-data
        group: www-data
        state: link
      tags: [wordpress]
```

Equivalent with the following bash command:

```
sudo ln -s /home/runner/gear5/www/html/wp-content/uploads
/var/www/your_domain/wordpress/wp-content/uploads
```

It creates a symbolic link to the uploads folder in the WordPress installation directory.

TOP

Task 30 - WordPress | Create symlink | Wordpress

```
- name: Create a symbolic link (Wordpress)
      ansible.builtin.file:
        src: "/var/www/{{ http_host }}/wordpress"
        dest: "/var/www/html"
        owner: www-data
        group: www-data
        state: link
      when: download_result.changed
      tags: [wordpress]
```

Equivalent with the following bash command:

```
sudo ln -s /var/www/your_domain/wordpress /var/www/html
```

It creates a symbolic link to the WordPress installation directory.

TOP

Task 31 - Extract Backup

```
- name: Extract Backup
      ansible.builtin.unarchive:
        src:
https://storage.googleapis.com/gear5_backup_bucket/dbbackup.tar.gz
        dest: /tmp/
        remote_src: true
      when: download_result.changed
      tags: [mysql]
```

Equivalent with the following bash command:

It extracts a compressed backup file from a remote URL to the /tmp directory on the target system using the ansible.builtin.unarchive module, but only if a previous task has indicated that the file needs to be downloaded. The backup file is downloaded from a Google Cloud Storage bucket using a signed URL. The URL is generated using the gsutil command-line tool. The backup file contains a MySQL database dump that can be used to restore the database on the target system.

```
gsutil signurl -d 10m /home/runner/gear5/gear5-credentials.json
gs://gear5_backup_bucket/dbbackup.tar.gz
```

TOP

Task 32 - Update URLs in the database dump (http)

```
- name: Update WordPress URLs in the database dump (HTTP)
      ansible.builtin.replace:
        path: /tmp/wordpress.bak.sql
        regexp: "http://gear5.online"
        replace: "http://{{ lookup('env', 'INSTANCE_IP') }}"
      when: download_result.changed
      tags: [wordpress]
```

Equivalent with the following bash command:

```
sed -i 's/http:\/\/gear5.online/http:\/\/{{ lookup('env', 'INSTANCE_IP')
}}/g' /tmp/wordpress.bak.sql
```

It replaces all instances of the http://gear5.online URL with the IP address of the target system in a WordPress database dump file located at /tmp/wordpress.bak.sql using the ansible.builtin.replace module, but only if a previous task has indicated that the file needs to be downloaded.

Task 33 - Update URLs in the database dump (https)

```
- name: Update WordPress URLs in the database dump (HTTPS)
      ansible.builtin.replace:
        path: /tmp/wordpress.bak.sql
        regexp: "https://gear5.online"
        replace: "https://{{ lookup('env', 'INSTANCE_IP') }}"
      when: download_result.changed
      tags: [wordpress]
```

Equivalent with the following bash command:

```
sed -i 's/https:\/\/gear5.online/https:\/\/{{ lookup('env', 'INSTANCE_IP')
}}/g' /tmp/wordpress.bak.sql
```

It replaces all instances of the https://gear5.online URL with the IP address of the target system. This is done to ensure that the WordPress site will work correctly after the database dump is restored.

Task 34 - Restore database dump

```
- name: Import .sql database
      community.mysql.mysql_db:
        name: "{{ mysql_db }}"
        state: import
        target: /tmp/wordpress.bak.sql
      when: download_result.changed
```

Equivalent with the following bash command:

```
mysql -u root -p wordpress < /tmp/wordpress.bak.sql
```

It restores a MySQL database dump from a file located at /tmp/wordpress.bak.sql on the target system using the community.mysql.mysql_db module, but only if a previous task has indicated that the file needs to be downloaded.

TOP

Task 35 - Cleanup temporary files

```
- name: Cleanup temporary files
      ansible.builtin.file:
        path: /tmp/latest.tar.gz
        state: absent
      when: download_result.changed
      tags: [wordpress]
```

Equivalent with the following bash command:

```
sudo rm -rf /tmp/latest.tar.gz
```

It removes the archve file from the target system. This is done to ensure that the file is not left on the target system after the WordPress site has been set up.

TOP

Task 36 - Cleanup database dump

```
- name: Cleanup database dump
      ansible.builtin.file:
```

```
        path: "{{ temp_dir }}/wordpress.bak.sql"
        state: absent
      tags: [cleanup]
```

Equivalent with the following bash command:

```
sudo rm -rf /tmp/wordpress.bak.sql
```

It removes the database dump file from the target system.

TOP

Handlers

Handlers are tasks that are only executed when notified by other tasks. Handlers are typically used to restart services or perform other actions that need to be triggered after a change has been made to the system. Handlers can be notified by other tasks using the notify parameter, which specifies the name of the handler to notify. When a handler is notified, Ansible will queue the handler to be executed at the end of the current play. If multiple tasks notify the same handler, the handler will only be executed once. Handlers are a powerful feature of Ansible that can help ensure that changes to a system are properly applied and that services are restarted when necessary.

TOP

Reload Apache

```
- name: Reload Apache
    ansible.builtin.service:
      name: apache2
      state: reloaded
```

Equivalent with the following bash command:

```
sudo systemctl reload apache2
```

It reloads the Apache web server on the target system using the ansible.builtin.service module.

TOP

Restart Apache

```
- name: Restart Apache
      ansible.builtin.service:
        name: apache2
        state: restarted
```

Equivalent with the following bash command:

```
sudo systemctl restart apache2
```

It restarts the Apache web server on the target system using the ansible.builtin.service module.

TOP

Play | Run tests

```
- name: Run tests
  hosts: staging
  become: true
```

This play runs the tests on the staging server. The `become: yes` is used to run the tasks as root.

TOP

Task 37 - Test | Run the tests script

```
- name: Run tests
      ansible.builtin.shell:
        cmd: "cd /home/runner/gear5 && ./tests.sh"
      tags: [tests]
```

Equivalent with the following bash command:

```
cd /home/runner/gear5 && ./tests.sh
```

It runs a shell script located at /home/runner/gear5/tests.sh on the target system using the ansible.builtin.shell module. The script performs a series of tests and actions.

The script sends an email notification with an attached log file if any of the tests fail, and creates a pull request on GitHub if all tests pass. The email notification is sent using the swaks command-line tool, and the pull request is created using the GitHub API.

The script uses various environment variables to configure the email notification and pull request creation, including the email addresses and SMTP server for the email notification, the GitHub repository and access token for the pull request creation, and the base and head branches for the pull request. The script also writes the output of the tests to a log file and attaches the log file to the email notification.