Ansible Secrets

Ansible Secrets provides encrypted secrets (e.g., account usernames, account passwords, API keys, and other sensitive data) to Bash and Python scripts without exposing those secrets in plaintext.

Quick Setup (recommended)

Run the setup script from the **repository root**:

```
./setup.sh
```

The setup script:

- creates/updates the admin toolkit under `/opt/ansible_secrets` (including a Python venv)
- installs runtime helpers to `/usr/local/bin` and `/usr/local/lib`
- creates `.ansible_vault_password` and an encrypted `group_vars/all/vault.yml`
- generates both the **Ansible Vault password** and the **single GPG passphrase** automatically (**48 characters** each)
- prompts only for your **sudo** password (when required)

If the script adds you to the `appsecretaccess` group, you must **log out and log back in** for the new group membership to take effect.

System Components and Directory Structure

Ansible Secrets consists of three security components and relies on a specific directory structure to separate administrative tasks from runtime operations.

Directory Definitions

- Application project directories contain the Bash and Python scripts to protect. Those scripts will consume the secrets at runtime but will never contain them.
- Ansible Deployment Project (e.g., `/opt/ansible_secrets`). This is a self-contained administrative toolkit used **only** for the setup and maintenance of the secrets. It contains the Ansible playbook, the vault, and the source GPG-encrypted files. Your application scripts have no knowledge of, and will never interact with, this directory.
- Runtime Secrets Directory (e.g., `/opt/credential_store`).This is the secure, permissions-restricted location on the server where the Ansible playbook places the necessary files for runtime decryption. This is the **single point of interaction** for the application scripts when they need a secret.

```
/
├── opt/
│   ├── ansible_secrets/      (Admin Toolkit: for deployment only)
│   │   ├── ansible.cfg
│   │   ├── deploy_secrets.yml
│   │   ├── inventory
│   │   ├── venv/               (Python venv for admin tooling)
│   │   ├── .ansible_vault_password
│   │   ├── add-secret.sh         (Admin utility script)
│   │   ├── files/
│   │   │   └── *.txt.gpg         (Source GPG-encrypted secrets)
│   │   ├── group_vars/
│   │   │   └── all/
│   │   │       └── vault.yml     (Ansible Vault with GPG passphrase)
│   │   └── tasks/
│   │       └── setup.yml
│   │
```

```
│   └── credential_store/       (Runtime Secrets: for application use)
│       ├── .gpg_passphrase
│       └── *.txt.gpg           (Deployed GPG-encrypted secrets)
│
└── usr/
    └── local/
        ├── bin/
        │   ├── get_secret.sh     (Runtime helper for Bash)
        │   └── secure-app.sh     (Admin utility script)
        │
        └── lib/
            └── ansible_secret_helpers/ (Runtime helpers for Python)
                ├── secret_retriever.py
                └── connection_helpers.py
```

Security Components

GPG Encryption This component provides confidentiality for your application secrets. Each secret is encrypted with GPG(AES-256 cipher) and the resulting file (e.g., `svc_alpha_secret.txt.gpg`) is stored in the**Runtime Secrets Directory**. A single, strong GPG passphrase is used to encrypt the secrets files.

Ansible Vault This component protects the GPG passphrase. The passphrase–which is the key to unlock all the application secrets–is stored in an encrypted Ansible Vault file that resides within the **Ansible Deployment Project** directory. Access to this vault is controlled by a separate Vault password known only to the administrator.

Linux Permissions This component enforces access control. A dedicated group (`appsecretaccess`) is used to govern who can executethe scripts in your **Application Projects** that require credentials. The permissions on the **Runtime Secrets Directory** are set so that only members of this group can read the encrypted passwords and the GPG passphrase file.

Deployment and Runtime

The Ansible Secrets implementation has two phases: a one-time setup controlled by Ansible, and the normal runtime operation when your scripts execute.

Phase 1: Deployment (Controlled by Ansible)

This is the administrative process for securely placing the secrets on the server, executed from with-inthe **Ansible Deployment Project**. You run an Ansible playbook that performs the following steps:

- Ansible uses its Vault password to decrypt the GPG passphrase in memory.
- It creates the secure **Runtime Secrets Directory** on the server (e.g., `/opt/credential_store`).
- It writes the GPG passphrase from memory into a file within that directory (e.g., `.gpg_passphrase`).
- It copies the GPG-encrypted password files from the **Ansible Deployment Project** into the **Runtime Secrets Directory**.
- It sets strict ownership and permissions on the **Runtime Secrets Directory** and all its files, granting access only to a dedicated service user (`service_account`) and the designated group (`appsecretaccess`).

Runtime Operation (Script Execution)

This is what happens whenever an authorized user runs one of your scripts from an **Application Project**.

- The script starts. It does not interact with the **Ansible Deployment Project** directory.
- The script reads the GPG passphrase from the protected file (`.gpg_passphrase`) inside the **Runtime Secrets Directory** into a variable in memory.
- The script uses this passphrase to decrypt the specific application secret it needs from the corresponding file in the **Runtime Secrets Directory** (e.g., `svc_alpha_secret.txt.gpg`) into a second memory variable.
- It uses the decrypted application password to perform its task (e.g., connect to the Oracle database).
- Once the task is complete, the script immediately clears the variables that held the GPG passphrase and the decrypted application secret, minimizing their lifetime in memory. The plaintext application password is never written to disk.

Note on ownership and permissions

Python and bash application scripts are owned by `service_account` and have group membership `appsecretaccess`. This service account has no password set and logins are not permitted. The Linux permissions of such files are `0750`, which means read-write-execute permission for `service_account`, and `read-execute` permission for `appsecretaccess` group members. No other accounts (except for `root`) have access.

Note: when a user is newly added to `appsecretaccess`, they must start a fresh login session (log out/in) before the new permissions apply.

Examples

The repository includes an `examples/` directory with practical usage patterns:

- `examples/load_github_identity.sh`: sample application script that retrieves a GitHub SSH key passphrase from Ansible Secrets (e.g., secret name `gitphrase`) and loads the identity into `ssh-agent`.
- `examples/EXAMPLES.md` (and `examples/EXAMPLES.pdf`): usage patterns for Bash and Python consumers.