

Installation Guide: Secure Credential Management with GPG and Ansible Vault

This guide will walk through the setup of the Ansible Secrets project, secret encryption, the Ansible playbook for local deployment, and finally, the integration with Bash and Python scripts.

Placeholders Used in This Guide (REPLACE WITH YOUR ACTUAL VALUES):

- **Application Passwords:**

1. LDAP DM Password: `Ldap&DmP@sswOrd!2025`
2. LDAP RO Password: `LdapR0n1yP@sswOrd`
3. Oracle DB Password: `S3cure0racle!P@ss`

- **Single GPG Passphrase (to encrypt above passwords):**

`MyV3ryStr0ngGPGPassphr@s3`

- **Ansible Vault Password (to protect the GPG passphrase):**

`MyUltr@S3cureAnsib13VaultP@ss`

- **Project & Secret Directories:**

1. Ansible Project: `/opt/ansible_secrets`
2. Deployed Secrets: `/opt/credential_store`

- **Users & Groups:**

1. Service User: `service_account`
2. Access Group: `appsecretaccess`
3. Admin User (you): `flengye1`

Directory Structure Diagram

```
/
├── opt/
│   ├── ansible_secrets/      (Admin Toolkit: for
deployment only)
│   │   ├── ansible.cfg
│   │   ├── deploy_secrets.yml
│   │   ├── inventory
│   │   ├── .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
```

Section 1: Initial Server Setup

These steps prepare the server environment with the necessary users, groups, and software.

1.1. Create Users and Groups

Run these commands on your RHEL server as a user with `sudo` privileges.

```
# Create the dedicated service user
sudo useradd --system --shell /sbin/nologin --comment
"Service account for Bash and Python apps" service_account

# Create the dedicated access group
sudo groupadd --system appsecretaccess

# Add the service user to the access group
sudo usermod -aG appsecretaccess service_account

# Add yourself and any other required users to the access
```

```
group
sudo usermod -aG appsecretaccess flengyel
# sudo usermod -aG appsecretaccess otheruser1

# IMPORTANT: Any user you add to the group must log out
and log back in
# for their new group membership to take effect.
```

1.2. Install Required Software

```
sudo dnf install ansible-core gnupg2 -y
```

1.3. Set Up Python Virtual Environment & Project Directory

This isolates your Ansible installation.

```
# Create and take ownership of the Ansible project
directory
sudo mkdir -p /opt/ansible_secrets
sudo chown 'flengyel:domain users' /opt/ansible_secrets
cd /opt/ansible_secrets

# Create a Python virtual environment inside the project
directory
python3 -m venv venv

# Activate the virtual environment
source venv/bin/activate
```

```
# Your shell prompt should now start with "(venv)".  
# Install Ansible and the GPG library into the active  
venv.  
pip install ansible-core gnupg
```

Section 2: Credential and Ansible Vault Preparation

This section covers the creation of the Ansible Vault to protect the master GPG passphrase, the installation of the `add-secret.sh` helper script, and finally, the secure creation of your encrypted application secrets.

2.1. Prepare the Ansible Vault

First, we create the Ansible Vault. This encrypted file will hold the single GPG passphrase that is used to encrypt all of your individual application secrets.

```
# Ensure you are in the project root  
(/opt/ansible_secrets) and the venv is active  
cd /opt/ansible_secrets  
source venv/bin/activate  
  
# Create the vault password file that protects the vault  
itself  
echo "MyUltraS3cureAnsibl3VaultP@ss" >  
.ansible_vault_password  
chmod 600 .ansible_vault_password
```

```
# Create the encrypted vault file to hold the GPG
passphrase
mkdir -p group_vars/all
ansible-vault create group_vars/all/vault.yml
```

An editor will open. Enter the following content (this is your single GPG passphrase):

```
app_gpg_passphrase: "MyV3ryStr0ngGPGPassphr@s3"
```

Save and close the file. The GPG passphrase is now securely stored inside the vault.

2.2. Install the add-secret.sh Administrative Script

This helper script automates the creation of new encrypted secrets by securely

retrieving the GPG passphrase from the Ansible Vault you just created.

- Create the file `/opt/ansible_secrets/add-secret.sh` and add the source code below.

```
#!/bin/bash
#
# add-secret.sh - A script to securely encrypt and add a
new secret
# to the Ansible Secrets project.

set -euo pipefail
```

```
# --- Configuration ---
ANSIBLE_PROJECT_DIR="/opt/ansible_secrets"
FILES_DIR="${ANSIBLE_PROJECT_DIR}/files"
VAULT_FILE="${ANSIBLE_PROJECT_DIR}/group_vars/all/vault.yml"
VENV_PATH="${ANSIBLE_PROJECT_DIR}/venv/bin/activate"

# --- Input Validation ---
if [[ $# -ne 1 ]]; then
    echo "Usage: $0 <secret_name>" >&2
    echo "Example: $0 oracle_db" >&2
    exit 1
fi

SECRET_NAME="$1"
OUTPUT_FILE="${FILES_DIR}/${SECRET_NAME}_secret.txt.gpg"

if [[ ! -d "$ANSIBLE_PROJECT_DIR" || ! -f "$VAULT_FILE" ||
! -f "$VENV_PATH" ]]; then
    echo "Error: Required project files or directories not
found in '$ANSIBLE_PROJECT_DIR'." >&2
    exit 1
fi

read -sp "Enter the secret for '${SECRET_NAME}': " SECRET
echo

if [[ -z "$SECRET" ]]; then
    echo "Error: Secret cannot be empty." >&2
    exit 1
fi
```

```
if [[ -f "$OUTPUT_FILE" ]]; then
    read -p "Warning: '${OUTPUT_FILE}' already exists.
Overwrite? (y/N) " -n 1 -r
    echo
    if [[ ! $REPLY =~ ^[Yy]$ ]]; then
        echo "Operation cancelled."
        exit 1
    fi
fi

# --- Main Logic ---
echo "--> Activating virtual environment..."
source "$VENV_PATH"

echo "--> Retrieving GPG passphrase securely from Ansible
Vault..."
GPG_PASSPHRASE=$(ansible-vault view "$VAULT_FILE" | grep
'app_gpg_passphrase:' | awk '{printf "%s", $2}' | tr -d
\'\'\'')

if [[ -z "$GPG_PASSPHRASE" ]]; then
    echo "Error: Failed to retrieve GPG passphrase from
vault. Check vault password or file content." >&2
    exit 1
fi

echo "--> Encrypting new secret for '${SECRET_NAME}'..."
printf '%s' "$SECRET" | gpg --batch --yes --symmetric --
cipher-algo AES256 \
    --passphrase "$GPG_PASSPHRASE" \
    --output "$OUTPUT_FILE"
```



```

if [[ $? -eq 0 ]]; then
    echo "✅ Success! Encrypted secret saved to:
    ${OUTPUT_FILE}"
    sudo chown service_account:appsecretaccess
    "$OUTPUT_FILE"
    echo "--> Ownership set to
    service_account:appsecretaccess"
else
    echo "❌ Error: GPG encryption failed." >&2
    exit 1
fi

deactivate
echo "--> Done."

```

- Make the script executable and set the correct ownership for an administrator.

```

sudo chown 'flengyel:domain users'
/opt/ansible_secrets/add-secret.sh
sudo chmod 750 /opt/ansible_secrets/add-secret.sh

```

2.3. Create Encrypted Secrets Using the Helper Script

Now, with the vault and helper script in place, you can securely create an encrypted file for each of your application secrets.

```
# Ensure you are in the project root and the venv is
active
cd /opt/ansible_secrets
source venv/bin/activate

# Create the subdirectory for the GPG files if it doesn't
exist
mkdir -p files

# Now, use the helper script to create each secret.
# The script will prompt you for the secret value
securely.
# NOTE: the values below are examples only.

echo "Creating LDAP DM secret..."
./add-secret.sh ldap_dm
# --> Enter 'Ldap&DmP@sswOrd!2025' when prompted

echo "Creating LDAP RO secret..."
./add-secret.sh ldap_ro
# --> Enter 'LdapROnlyP@sswOrd' when prompted

echo "Creating Oracle DB secret..."
./add-secret.sh oracle_db
# --> Enter 'S3cureOracle!P@ss' when prompted

# Deactivate the environment when finished creating
secrets
deactivate
```

After running these commands, verify that your encrypted files (e.g., `ldap_dm_secret.txt.gpg`) have been created in the `/opt/ansible_secrets/files/` directory.

Section 3: Ansible Configuration and Playbook

3.1. Configure Ansible (`ansible.cfg` and `inventory`)

- Create `/opt/ansible_secrets/ansible.cfg`:

```
[defaults]
inventory = ./inventory
vault_password_file = ../ansible_vault_password
host_key_checking = False

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = true
```

- Create `/opt/ansible_secrets/inventory`:

```
[local_server]
localhost ansible_connection=local
ansible_python_interpreter={{ ansible_playbook_python }}
```

3.2. Create the Ansible Playbook (`deploy_secrets.yml`)

Create `/opt/ansible_secrets/deploy_secrets.yml` :

```
- name: Deploy Application Secrets Locally
  hosts: local_server
  vars:
    secrets_target_dir: "/opt/credential_store"
    service_user: "service_account"
    secret_access_group: "appsecretaccess"
    encrypted_secret_files:
      - green_dm_secret.txt.gpg
      - yellow_dm_secret.txt.gpg
      - ldap_ro_secret.txt.gpg
      - oracle_db_secret.txt.gpg
  vars_files:
    - group_vars/all/vault.yml

  tasks:
    - name: Ensure base directories and groups are set up
      ansible.builtin.include_tasks: tasks/setup.yml

    - name: Deploy the single GPG passphrase file from
      vaulted variable
      ansible.builtin.copy:
        content: "{{ app_gpg_passphrase }}"
        dest: "{{ secrets_target_dir }}/.gpg_passphrase"
        owner: "{{ service_user }}"
        group: "{{ secret_access_group }}"
        mode: '0440'
      no_log: true
```

```
- name: Deploy all encrypted application password
files
  ansible.builtin.copy:
    src: "./files/{{ item }}" # From the project's
files/ dir
    dest: "{{ secrets_target_dir }}/{{ item }}"
    owner: "{{ service_user }}"
    group: "{{ secret_access_group }}"
    mode: '0440'
  with_items: "{{ encrypted_secret_files }}"
```

Create a supporting task file for clarity, `tasks/setup.yml`:

```
mkdir -p /opt/ansible_secrets/tasks
```

Create `/opt/ansible_secrets/tasks/setup.yml`:

```
- name: Ensure the secret access group exists
  ansible.builtin.group:
    name: "{{ secret_access_group }}"
    state: present
    system: true

- name: Ensure service user is part of the secret access
group
  ansible.builtin.user:
    name: "{{ service_user }}"
    groups: "{{ secret_access_group }}"
    append: true

- name: Ensure secrets target directory exists with
```

```
correct permissions
ansible.builtin.file:
  path: "{{ secrets_target_dir }}"
  state: directory
  owner: "{{ service_user }}"
  group: "{{ secret_access_group }}"
  mode: '0750'
```

Section 4: Deployment

4.1. Run the Ansible Playbook

```
# Ensure you are in the project root and your venv is
active
cd /opt/ansible_secrets
source venv/bin/activate

# Run the playbook
ansible-playbook deploy_secrets.yml
```

4.2. Verify the Deployment

After the playbook runs successfully, check the deployed secrets directory.

```
sudo ls -lA /opt/credential_store/
```

The output should look like this (owner, group, permissions, and files must match):

```
total 12
-r--r----- 1 service_account appsecretaccess 111 Jun 07
08:44 .gpg_passphrase
-r--r----- 1 service_account appsecretaccess 1408 Jun 07
08:44 ldap_dm_secret.txt.gpg
-r--r----- 1 service_account appsecretaccess 1408 Jun 07
08:44 ldap_ro_secret.txt.gpg
-r--r----- 1 service_account appsecretaccess 1408 Jun 07
08:44 oracle_db_secret.txt.gpg
```

Section 5: Script Integration and Runtime Operation

This section details how your application scripts can securely access secrets at runtime. The project provides reusable helper scripts for both Bash and Python.

5.1. Reusable Bash Script (get_secret.sh)

For Bash scripts, the `get_secret.sh` utility is the standard method for retrieving any secret. It takes a single argument—the name of the secret—and prints its value to standard output.

Installation:

- Create the file `/usr/local/bin/get_secret.sh`.
- Add the following source code to the file:

```
#!/usr/bin/env bash
set -euo pipefail

if [[ $# -ne 1 ]]; then
    echo "Usage: $0 <secret_name>" >&2
    echo "Example: $0 oracle_db" >&2
    exit 1
fi

SECRET_NAME="$1"
SECRETS_DIR="/opt/credential_store"
ENC_FILE="${SECRETS_DIR}/${SECRET_NAME}_secret.txt.gpg"
GPG_PASSPHRASE_FILE="${SECRETS_DIR}/.gpg_passphrase"

if [[ ! -r "$ENC_FILE" ]]; then
    echo "Error: Encrypted secret for '${SECRET_NAME}'
not found or not readable." >&2
    exit 1
fi

# Decrypt and print the password to stdout
gpg --batch --quiet --yes \
    --passphrase-file "$GPG_PASSPHRASE_FILE" \
    --decrypt "$ENC_FILE" 2>/dev/null
```

- Set its ownership and permissions:

```
sudo chown service_account:appsecretaccess
/usr/local/bin/get_secret.sh
sudo chmod 0750 /usr/local/bin/get_secret.sh
```


5.2. Reusable Python Modules

For Python applications, a two-layer helper system is provided. Applications can use the low-level `get_secret()` function for direct access to any secret, but the high-level `connection_helpers` module is the recommended approach for database and LDAP connections.

Layer 1: Foundational `get_secret()` Function

The `get_secret()` function, contained in the `secret_retriever.py` module, is the core component for fetching any secret's value as a string.

Installation:

Create the file `/usr/local/lib/ansible_secret_helpers/secret_retriever.py`. Add the following source code:

```
#
/usr/local/lib/ansible_secret_helpers/secret_retriever.py
import os
import subprocess

SECRETS_DIR = "/opt/credential_store"
GPG_PASSPHRASE_FILE = os.path.join(SECRETS_DIR,
".gpg_passphrase")

def get_secret(secret_name: str) -> str:
    """
    Retrieves a decrypted password for a given secret
    name.
    Raises RuntimeError on failure.
    """
```

```
    enc_file = os.path.join(SECRETS_DIR, f"
{secret_name}_secret.txt.gpg")
    if not os.path.exists(enc_file):
        raise FileNotFoundError(f"Encrypted secret for
'{secret_name}' not found.")

    cmd = [
        "gpg", "--batch", "--quiet", "--yes",
        "--passphrase-file", GPG_PASSPHRASE_FILE,
        "--decrypt", enc_file
    ]

    try:
        result = subprocess.run(
            cmd, stdout=subprocess.PIPE,
stderr=subprocess.PIPE,
            universal_newlines=True, check=True
        )
        return result.stdout.strip()
    except subprocess.CalledProcessError as e:
        raise RuntimeError(f"GPG decryption failed for
'{secret_name}': {e.stderr}")
    except FileNotFoundError:
        raise RuntimeError("gpg command not found. Is
GnuPG installed?")
```

Layer 2: Recommended `connection_helpers.py` Module

This high-level module provides pre-built functions like `create_db_connection()` and `create_ldap_connection()`. It uses `secret_retriever.py` internally and is the best practice for connecting to services.

Installation:

Create the file `/usr/local/lib/ansible_secret_helpers/connection_helpers.py`. Add the following source code:

```
#
/usr/local/lib/ansible_secret_helpers/connection_helpers.py
import sys
import ssl
import sqlalchemy
import cx_Oracle as cx
from ldap3 import Server, Connection, ALL, Tls # Requires
ldap3 library
import secret_retriever # Imports the local module

def create_ldap_connection(ldap_server, user_secret,
pswd_secret):
    """
    Retrieves credentials and establishes a secure LDAP
connection.
    Returns a bound ldap3 Connection object.
    """
    oud_user = None
    oud_pswd = None
    try:
```

```
        oud_user =
secret_retriever.get_secret(user_secret)
        oud_pswd =
secret_retriever.get_secret(pswd_secret)

        tls = Tls(validate=ssl.CERT_NONE)
        srv = Server(ldap_server, port=636, get_info=ALL,
use_ssl=True, tls=tls)
        oud = Connection(srv, user=oud_user,
password=oud_pswd, auto_bind=True)

        # Clear credentials from memory immediately after
use
        oud_user = None
        oud_pswd = None

        if not oud.bound:
            # Use the correct server variable in the error
message
            raise ConnectionError(f'Error: cannot bind to
{ldap_server}')
```

```
        # Return the connection object only on success
        return oud

    except Exception as e:
        # Properly handle exceptions and exit
        print(f"Error creating LDAP connection: {e}",
file=sys.stderr)
        sys.exit(1)
```

```

def create_db_connection(dbhost, dbport, dbsid,
user_secret, pswd_secret, engine_only=False):
    """
    Retrieves credentials and creates a DB engine and
    optionally a connection.
    - If engine_only is True, returns only the SQLAlchemy
    engine.
    - If engine_only is False (default), returns a tuple
    of (engine, connection).
    """
    db_user = None
    db_pswd = None
    engine = None
    conn = None
    try:
        db_user = secret_retriever.get_secret(user_secret)
        if not db_user:
            raise RuntimeError(f"Retrieved empty username
secret for '{user_secret}'")

        db_pswd = secret_retriever.get_secret(pswd_secret)
        if not db_pswd:
            raise RuntimeError(f"Retrieved empty password
for '{pswd_secret}'")

        datasource_name = cx.makedsn(dbhost, dbport,
service_name=dbsid)
        connectstring = f'oracle+cx_oracle://{db_user}:
{db_pswd}@{datasource_name}'

        # Clear credentials from memory immediately after
use

```

```

db_user = None
db_pswd = None

engine = sqlalchemy.create_engine(connectstring,
max_identifier_length=128)

# Conditional return based on the new flag
if engine_only:
    return engine
else:
    conn = engine.connect()
    return engine, conn

except Exception as e:
    print(f"Error creating database connection: {e}",
file=sys.stderr)
    # Ensure resources are cleaned up on failure
    if conn:
        conn.close()
    if engine:
        engine.dispose()
    sys.exit(1)

```

Set Permissions for All Python Helpers: Run these commands once to set up the directory and secure both Python helper modules.

```

sudo mkdir -p /usr/local/lib/ansible_secret_helpers
sudo chown service_account:appsecretaccess
/usr/local/lib/ansible_secret_helpers/*.py
sudo chmod 0640 /usr/local/lib/ansible_secret_helpers/*.py

```

Section 6: App script ownership and permissions

This is the recommended ownership and permission model for production Python and bash scripts:

- Owner: `service_account`
- Group: `appsecretaccess`
- Permissions: `0750 (-rwxr-x---)`

Here are the ownership and permission mode commands for scripts used with Ansible Secrets. The script in this example is `getemplid.sh`, however, the commands below apply to Python scripts as well.

```
sudo chown service_account:appsecretaccess getemplid.sh
sudo chmod 0750 getemplid.sh
```

See the `UTILITIES.md` guide for the `secure-app.sh` script to automate this task.