Adding Secrets to the Ansible Secrets System

This document outlines the procedure for adding new secrets to the credential store using the `add-secret.sh` utility. This script automates the encryption process, ensuring consistency and security.

Procedure for Adding a New Secret

Here is the complete, step-by-step process for adding a new secret—for example, for a secret named `mssql_db`.

Step 1: Run the `add-secret.sh` Utility

The `add-secret.sh` script handles the creation and GPG encryption of the new secret file. It must be run from within the Ansible project directory.

```
# Ensure you are in the project root directory
cd /opt/ansible_secrets

# Run the script, providing the name of the secret you want to add.
# The script will prompt you to enter the secret value securely.
./add-secret.sh mssql_db
```

After the script runs successfully, a new encrypted file named `mssql_db_secret.txt.gpg` will be created in the `/opt/ansible_secrets/files/` directory with the correct ownership.

Step 2: Update the Ansible Playbook

Edit your playbook, `/opt/ansible_secrets/deploy_secrets.yml`, to add the new filename to the `encrypted_secret_files` list.

- Find this section in `deploy_secrets.yml`:

```
vars:
  secrets_target_dir: "/opt/credential_store"
  service_user: "service_account"
  secret_access_group: "appsecretaccess"
  encrypted_secret_files:
    - svc_alpha_secret.txt.gpg
    - svc_beta_secret.txt.gpg
    - db_gamma_secret.txt.gpg
```

- Add the new filename to the list**

```
  encrypted_secret_files:
    - svc_alpha_secret.txt.gpg
    - svc_beta_secret.txt.gpg
```

```
    - db_gamma_secret.txt.gpg
    - mssql_db_secret.txt.gpg  # <-- Add the new file here
```

Save the playbook file.

Step 3: Re-run the Ansible Playbook

Now, deploy the new secret to the credential store.

```
# 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
```

Because Ansible is idempotent, running the operation more than once with the same configuration files will not change the result. Ansible will check the existing files, see that they are already in the correct state (they will appear as "ok"), and will only copy the new `mssql_db_secret.txt.gpg` file to `/opt/credential_store` (this will appear as "changed").

Step 4: Use the New Secret in Your Scripts

Your reusable helper scripts are designed to use the new secret without any changes to them.

- **In a Bash script**:

```
#!/bin/bash
# Get the MSSQL secret into a variable
MSSQL_PASS=$(/usr/local/bin/get_secret.sh mssql_db)
if [[ -z "$MSSQL_PASS" ]]; then
    echo "Failed to retrieve MSSQL secret." >&2
    exit 1
fi
echo "Successfully retrieved secret. Now connecting to database..."
# ... use "$MSSQL_PASS"
unset MSSQL_PASS # Clean up
```

- In a Python script (Recommended Method)**: If the new secret is for a database or LDAP connection, you should use the high-level `connection_helpers` module.

```
#!/usr/bin/env python3
import sys

# --- Start: Required code block for secret retrieval ---
```

```
HELPER_LIB_PATH = "/usr/local/lib/ansible_secret_helpers"
if HELPER_LIB_PATH not in sys.path:
    sys.path.append(HELPER_LIB_PATH)
try:
    # Import the specific, high-level helper you need
    from connection_helpers import create_db_connection
except ImportError:
    print(f"CRITICAL: Could not import helper modules from {HELPER_LIB_PATH}.", file=sys.std
    sys.exit(1)
# --- End: Required code block ---

engine, conn = None, None
try:
    # Use the name of the new secret ('mssql_db') in the helper function call
    engine, conn = create_db_connection(
        dbhost='mssql.example.com',
        dbport='1433',
        dbsid='PRODDB',
        user_secret='mssql_user',
        pswd_secret='mssql_db' # <-- Use the new secret here
    )
    print("Successfully connected to the MSSQL database.")
    # ... your database logic ...

finally:
    if conn:
        conn.close()
    if engine:
        engine.dispose()
```