Onboarding an Application to Ansible Secrets

This guide provides a checklist for developers to modify an existing Bash or Python script to use the centralized Ansible Secrets credential store. The process involves two main steps:

1. Code Modification:
   Updating the script to remove the hardcoded password and instead call the appropriate helper function to retrieve it at runtime.

2. Set Secure Permissions:
   Using the secure-app.sh utility to apply the final, correct ownership and permissions required for production use.

Step 1: Modify the Application Script

Locate the script you need to secure and modify its source code to fetch credentials from the central store.

For a Bash Script

1. Identify the line where the plaintext password is used.
2. Remove the plaintext password.
3. Add a command to call the reusable helper script `/usr/local/bin/get_secret.sh`, passing the name of the secret you need (e.g., oracle_db).

4. Store the result in a variable and check that the retrieval was successful.
5. Use the variable in your application logic.
6. `unset` the variable as soon as it is no longer needed.

Example Modification

- Before:

```
# Unsafe: password is hardcoded
DB_PASS='S3cureOracle!P@ss'
sqlplus myuser/"$DB_PASS"@ORCL @/path/to/query.sql
```

- After:

```
#!/bin/bash
# Get the Oracle password into a variable using the
helper
ORACLE_PASS=$(/usr/local/bin/get_secret.sh oracle_db)
if [[ -z "$ORACLE_PASS" ]]; then
    echo "Failed to retrieve Oracle password from
credential store." >&2
    exit 1
fi

# Use the retrieved password
sqlplus myuser/"$ORACLE_PASS"@ORCL @/path/to/query.sql

# Clear the password from memory

unset ORACLE_PASS
```

For a Python Script

1. Identify the line where the plaintext password is used.
2. Remove the plaintext password.
3. Add the standard code block at the top of your script to add the helper library path (`/usr/local/lib/ansible_secret_helpers`) to `sys.path` and import the `secret_retriever` module.
4. Call the `secret_retriever.get_password()` function with the name of the secret you need (e.g., `ldap_dm`).
5. Use the returned password variable in your application logic
6. Set the variable to `None` in a finally block to ensure it is cleared.

Example Modification

- Before:

```
# Unsafe: password is hardcoded
ldap_password = "Ldap&DmP@sswOrd!2025"
# ... code to connect to LDAP using ldap_password
```

- After:

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

# --- 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 secret_retriever
except ImportError:
    print(f"CRITICAL ERROR: Could not import
'secret_retriever'.", file=sys.stderr)
    sys.exit(1)
# --- End: Required code block ---

ldap_pass = None
try:
    # Use the helper to get the ldap_dm password
    ldap_pass =
secret_retriever.get_password("ldap_dm")
    print(f"Successfully retrieved LDAP password.")
    # ... now use the ldap_pass variable to connect to
the database

except Exception as e:
  print(f"Error retrieving secret: {e}",
file=sys.stderr)

finally:
    # Clear the variable reference for security
    ldap_pass = None
```

Step 2: Set Secure Production Permissions

Once your script has been modified and tested, use the `secure-app.sh` utility to apply the standard production ownership and permissions. This script ensures the file is owned by `service_account:appsecretaccess`

and has `0750` permissions.

```
# This is an example for a script named 'getemplid.sh'
# Replace with the path to your actual application script.
sudo /usr/local/bin/secure-app.sh
/path/to/your/getemplid.sh
```

This command must be run by an administrator with `sudo` privileges.

Step 3: Final Testing

After setting the final permissions, perform a final test by running the application script.

To run the script interactively for testing, your user account must be a member of the `appsecretaccess` group.

If the script runs successfully, the onboarding process is complete. The script is now ready for its production use (e.g., being called by a `cronjob` running as the `service_account` user).

Step 4: Configuring for Automated Execution (Cron)

For scheduled tasks, the script must be run by the `service_account` user to ensure it has the correct permissions.

Requirements for Cronjobs

- User: The cronjob must be configured to run as the `service_account` user.
- Absolute Paths: The cron environment is minimal. Always use absolute paths for all scripts and executables in your command (e.g., /usr/local/bin/get_secret.sh, /usr/bin/python3).
- Logging: Always redirect standard output (>) and standard error (2>&1) to a log file for debugging.

Example Cronjob Entry

For system services, it is best practice to add a configuration file in the `/etc/cron.d/` directory.

Example for a file named / `etc/cron.d/my-oracle-report` :

```
# Run the Oracle report script daily at 2:00 AM as
service_account
0 2 * * * service_account
/path/to/your/oracle_report_wrapper.sh >>
/var/log/oracle_report.log 2>&1
```

This entry specifies the schedule, the user (`service_account`), the full command to run, and logging redirection.