Using Ansible Secrets in a Cron Job

This document shows how to modify a Python script to use the Ansible Secrets system when scheduled for exection cron.

Example

A Python script, cunyfirst.py, runs nightly via cron to generate a report from an Oracle database. This script requires a database username and password. The goal is to remove the hardcoded password from the script and retrieve it securely from the credential store.

The execution flow involves three components:

1. **The cron Entry**: Schedules the task to run at a specific time.
2. **The Wrapper Script**: A Bash script that sets up the necessary environment for the Python script.
3. **The Python Script**: The application that performs the main logic.

1. The Cron Job Entry

The role of cron executes the wrapper script at the scheduled time. All logic for logging and notifications should be handled within the wrapper script.

Here is the recommended crontab entry:

```
# Run the daily OIM report job at 1:00 AM
00 1 * * * /ims/cunyfirst/dailyoim_venv.sh
```

This entry, owned by the flengyel user, executes the wrapper script once per day. The user running the cron job must be a member of the appsecureaccess group on the local machine (see INSTALLATION for details).

2. The Wrapper Script (dailyoim_venv.sh)

A wrapper script is essential when running complex applications from cron, which provides a minimal, non-interactive shell environment. The wrapper's job is to prepare the environment before running the main application.

The script below performs several actions:

- Sets the LD_LIBRARY_PATH required by the Oracle client.
- Changes to the correct working directory.
- Sources the Python venv to activate the isolated environment.
- Redirects all output (stdout and stderr) from the entire process to a log file.
- Executes the Python script.
- Upon completion, it emails the log file to a recipient, providing a complete record of the run.

```bash
#!/usr/bin/env bash

# Wrapper script to set up environment and run the cunyfirst.py report

# --- Configuration ---
LOG_FILE="/var/log/cunyfirst_report.log"
EMAIL_RECIPIENT="florian.lengyel@cuny.edu"
EMAIL_SUBJECT="Daily OIM Report"
APP_DIR="/ims/cunyfirst"

# --- Main Logic ---
# Change to the application directory
cd "$APP_DIR" || { echo "Cannot cd to $APP_DIR" >> "$LOG_FILE"; exit 1; }
```

```bash
# Redirect all subsequent output (stdout and stderr) to the log file
exec &> "$LOG_FILE"

echo "---"
echo "Job started at: $(date)"
echo "---"

# Set up the Oracle environment
export LD_LIBRARY_PATH=/usr/lib/oracle/19.19/client64/lib

# Activate the Python virtual environment
source bin/activate

# Execute the main Python script
./cunyfirst.py
PY_EXIT_CODE=$?

if [[ $PY_EXIT_CODE -ne 0 ]]; then
    echo "---"
    echo "ERROR: Python script exited with code $PY_EXIT_CODE."
fi

# Example of conditional logic from original script
if [ "19-Feb-25" == "$(date +%d-%b-%y)" ]; then
    echo "Appending special report..."
    cat ./johnrayvargas_oim_usr_report.txt >> ./daily_oim_usr_report.txt
fi

# SFTP transfer logic
echo "---"
echo "Starting SFTP transfer..."
sftp -i /home/flengyel/.ssh/IMS_SvcAcct -oBatchMode=no -b - IMS_SvcAcct@st-edge.cuny.edu << !
  cd CUNY_IMS
  put daily_oim_usr_report.txt
  ls -l
  bye
!
echo "SFTP transfer finished."
echo "---"
echo "Job finished at: $(date)"
echo "---"

# Send the log file as an email notification
cat "$LOG_FILE" | s-nail -s "$EMAIL_SUBJECT" "$EMAIL_RECIPIENT"

exit 0
```

3. Securing the Python Script (cunyfirst.py)

The final step is to modify the Python script to remove the hardcoded password and use the connection_helpers module.

```python
# ... imports ...
dbhost = 'IAMPRDDB1.cuny.edu'
```

```
dbport = '2483'
dbsid  = 'PDIMOIG_HUD'
datasrc = cx.makedsn(dbhost, dbport,service_name = dbsid)

# This line contains the hardcoded secret password
constr  = 'oracle+cx_oracle://plaintext_db_username:plaintext_password' + datasrc

engine = sqlalchemy.create_engine(constr,  max_identifier_length=128)
# ... rest of script ...
```

Original cunyfirst.py

Revised cunyfirst.py   This version uses the recommended helper module to securely establish the database connection.

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

import csv
import sqlalchemy
import cx_Oracle as cx
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 high-level helper, which is the recommended approach
    from connection_helpers import create_db_connection
except ImportError:
    print(f"CRITICAL ERROR: Could not import helper modules from {HELPER_LIB_PATH}. Check pat
    sys.exit(1)
# --- End of required code block ---


# --- Database Connection Details ---
dbhost = 'IAMPRDDB1.cuny.edu'
dbport = '2483'
dbsid  = 'PDIMOIG_HUD'
# Define the names of the secrets to be used for the connection.
user_secret = 'cunyfirst_user' # The name of the secret for the username
pswd_secret = 'oracle_db'      # The name of the secret for the password


# --- Main Logic ---
engine, conn = None, None
try:
    # Use the helper function to securely create the database connection.
    # It handles secret retrieval and connection string creation internally.
    engine, conn = create_db_connection(
        dbhost, dbport, dbsid, user_secret, pswd_secret
    )

    stmt  = "SELECT DISTINCT "
    stmt += "REST OF SQL NOT SHOWN "
```

```python
        result = conn.execute(sqlalchemy.text(stmt))

        # Use a 'with' block for file handling to ensure it closes automatically
        with open('daily_oim_usr_report.txt', 'w', newline='') as fh:
            outcsv = csv.writer(fh, delimiter='\t')
            outcsv.writerows(result)

except Exception as e:
    print(f"CRITICAL ERROR: An error occurred during script execution. Error: {e}", file=sys.
    sys.exit(1)

finally:
    # Ensure database resources are always released
    if conn:
        conn.close()
    if engine:
        engine.dispose()
```