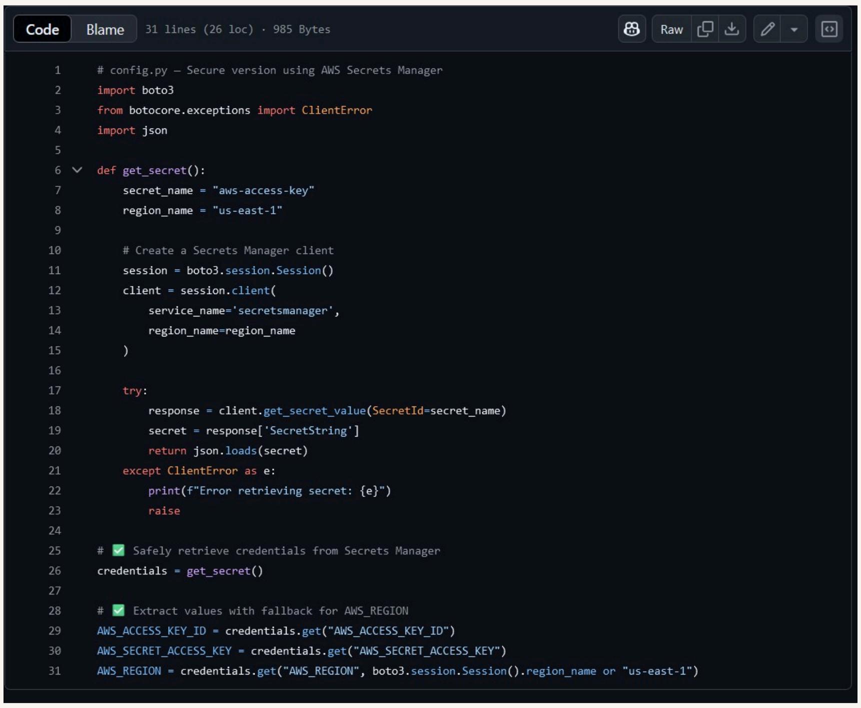
nextwork.org

Secure Secrets with Secrets Manager

Chrispinus Jacob



```python
# config.py — Secure version using AWS Secrets Manager
import boto3
from botocore.exceptions import ClientError
import json

def get_secret():
    secret_name = "aws-access-key"
    region_name = "us-east-1"

    # Create a Secrets Manager client
    session = boto3.session.Session()
    client = session.client(
        service_name='secretsmanager',
        region_name=region_name
    )

    try:
        response = client.get_secret_value(SecretId=secret_name)
        secret = response['SecretString']
        return json.loads(secret)
    except ClientError as e:
        print(f"Error retrieving secret: {e}")
        raise

# ✅ Safely retrieve credentials from Secrets Manager
credentials = get_secret()

# ✅ Extract values with fallback for AWS_REGION
AWS_ACCESS_KEY_ID = credentials.get("AWS_ACCESS_KEY_ID")
AWS_SECRET_ACCESS_KEY = credentials.get("AWS_SECRET_ACCESS_KEY")
AWS_REGION = credentials.get("AWS_REGION", boto3.session.Session().region_name or "us-east-1")
```

Introducing Today's Project!

In this project, I will demonstrate how to use AWS Secrets Manager for secure project management. I'm doing this project to learn how to protect credentials and sensitive configuration data when connecting a live (production) application to an AWS environment and its database. By using AWS Secrets Manager, I want to securely store, manage, and rotate secrets such as database passwords, API keys, and other confidential information instead of hardcoding them in the application code. This approach helps improve security and follows best practices for deploying secure applications in the cloud.

Tools and concepts

Services I used were AWS Secrets Manager, AWS IAM, Git, and GitHub. Key concepts I learnt include secure credential management using Secrets Manager, the importance of avoiding hardcoded secrets in code, how GitHub automatically detects exposed credentials, and how to resolve merge conflicts and rebase branches to maintain a clean commit history.

Project reflection

This project took me approximately 1 hours to complete. The most challenging part was handling the merge conflicts during the rebase and making sure no sensitive credentials were left behind in the commit history. It was most rewarding to see the app working securely with AWS Secrets Manager, knowing that I had improved both the functionality and security of the project.

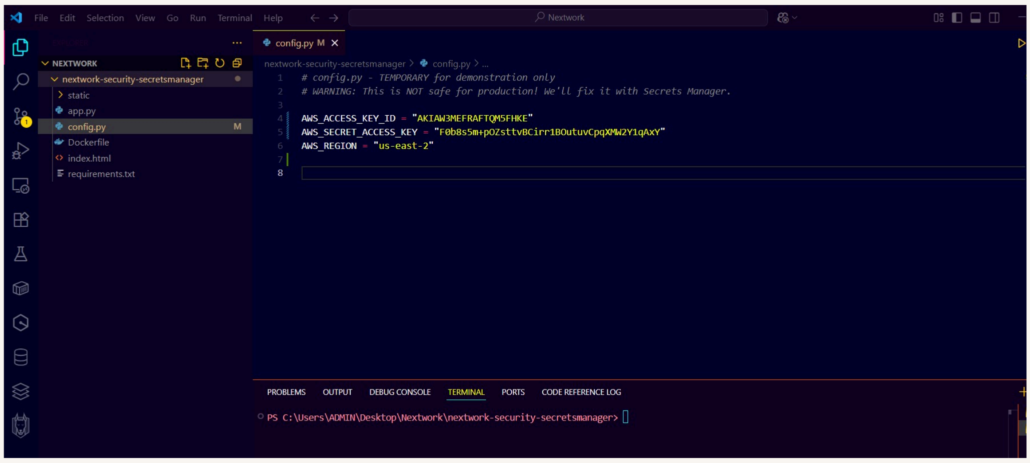
I did this project today to deepen my understanding of secure credential management and to gain hands-on experience with AWS Secrets Manager, Git, and GitHub workflows. Yes, this project met my goals — I successfully replaced hardcoded credentials with a secure solution, practiced using version control effectively, and learned how security tools like GitHub secret scanning work in real time. It was a practical and valuable learning experience.

Hardcoding credentials

In this project, a sample web app is exposing AWS credentials in its source code. It is unsafe to hardcode credentials because anyone with access to the code repository can see and misuse those secrets, which can lead to unauthorized access, data breaches, or unexpected costs. This step highlights why it is important to use a secure solution like AWS Secrets Manager to store and manage sensitive information instead of embedding it directly in the code.

In this project, a sample web app is exposing AWS credentials in its source code. It is unsafe to hardcode credentials because anyone with access to the code repository can see and misuse those secrets, which can lead to unauthorized access, data breaches, or unexpected costs. This step highlights why it is important to use a secure solution like AWS Secrets Manager to store and manage sensitive information instead of embedding it directly in the code.

Chrispinus Jacob
NextWork Student


File Edit Selection View Go Run Terminal Help
Nextwork
NEXTWORK
nextwork-security-secretsmanager
static
app.py
config.py M
Dockerfile
index.html
requirements.txt
config.py M
nextwork-security-secretsmanager > config.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG

```python
# config.py - TEMPORARY for demonstration only
# WARNING: This is NOT safe for production! We'll fix it with Secrets Manager.

AWS_ACCESS_KEY_ID = "AKIAW3MEFRAFTQM5FHKE"
AWS_SECRET_ACCESS_KEY = "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"
AWS_REGION = "us-east-2"

```

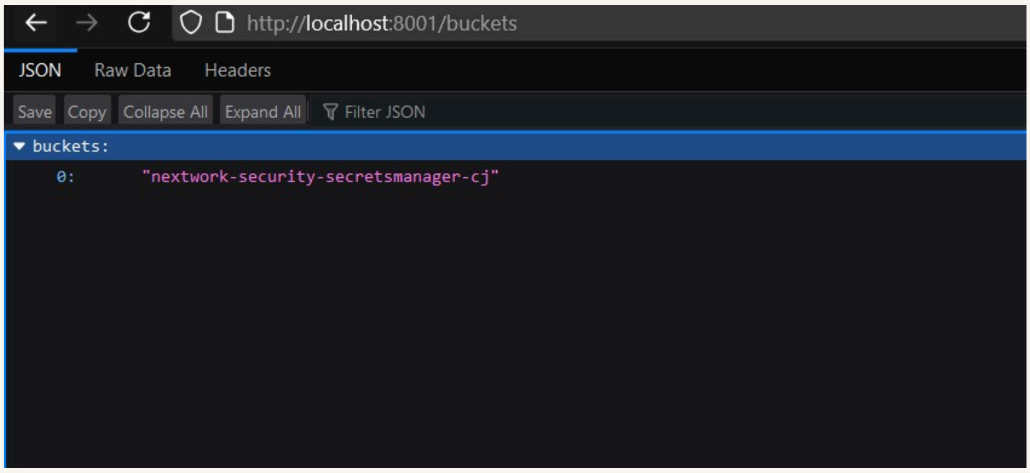
```
PS C:\Users\ADMIN\Desktop\Nextwork\nextwork-security-secretsmanager>
```

Using my own AWS credentials

As an extension for this project, I also decided to run the app locally to test how it connects securely to AWS Secrets Manager. To set up my virtual environment, I installed Python, created a virtual environment using `venv`, and installed the required packages using `pip`. This setup helps me safely test the code and manage dependencies without affecting my global system.

When I first ran the app, I ran into an error because some of the required packages were missing and the hardcoded credentials were either invalid or not configured properly. This showed me how fragile it is to rely on hardcoded secrets and highlighted the importance of setting up the environment correctly and using a secure way to manage credentials like AWS Secrets Manager.

To resolve the 'InvalidAccessKeyId' error, I updated the config file to use our account access ID and secret—no more test credentials.



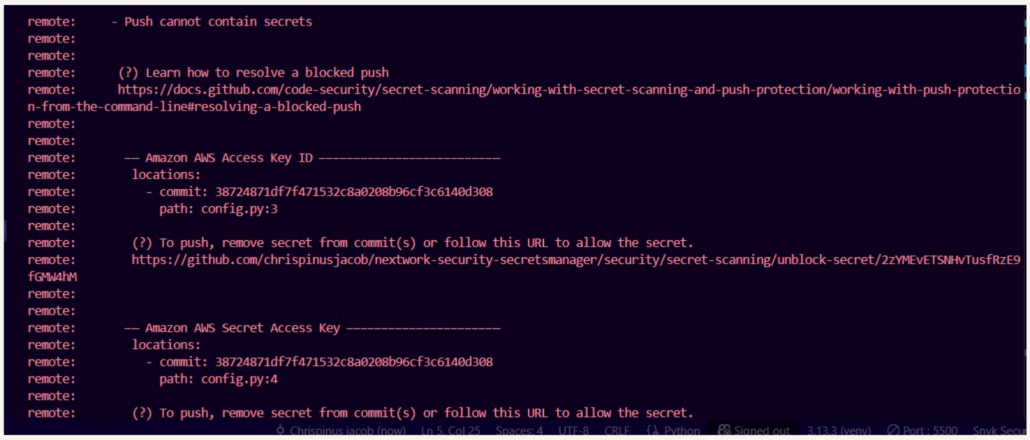
```
buckets:
    0:    "nextwork-security-secretsmanager-cj"
```

Pushing Insecure Code to GitHub

Once I updated the web app code with credentials, I forked the repository because I wanted to create my own copy of the original project on GitHub to experiment safely. A fork is different from a clone — while a clone creates a local copy on your machine, a fork lives on GitHub under your own account, allowing you to make changes without affecting the original project. This setup lets me push code, including the hardcoded credentials, to see how GitHub handles exposed secrets in a public or private repo.

To connect my local repository to the forked repository, I added the forked repo as a remote using `git remote add origin <forked-repo-URL>`. Then I used `git add` and `git commit` to stage and save my changes, including the hardcoded credentials. Finally, `git push` uploads those committed changes to the forked repository on GitHub.

GitHub blocked my push because it detected hardcoded AWS credentials in the code. This is a good security feature because it helps prevent accidental exposure of sensitive information, like API keys and secrets, which could be exploited if accessed by unauthorized users.



```
remote:     - Push cannot contain secrets
remote:
remote:
remote:      (?) Learn how to resolve a blocked push
remote:      https://docs.github.com/code-security/secret-scanning/working-with-secret-scanning-and-push-protection/working-with-push-protectio
n-from-the-command-line#resolving-a-blocked-push
remote:
remote:
remote:       —— Amazon AWS Access Key ID ——————————————————————————
remote:        locations:
remote:          - commit: 38724871df7f471532c8a0208b96cf3c6140d308
remote:            path: config.py:3
remote:
remote:        (?) To push, remove secret from commit(s) or follow this URL to allow the secret.
remote:        https://github.com/chrispinusjacob/nextwork-security-secretsmanager/security/secret-scanning/unblock-secret/2zYMEvETSNHvTusfRzE9
fGMW4hM
remote:
remote:
remote:       —— Amazon AWS Secret Access Key ——————————————————————
remote:        locations:
remote:          - commit: 38724871df7f471532c8a0208b96cf3c6140d308
remote:            path: config.py:4
remote:
remote:        (?) To push, remove secret from commit(s) or follow this URL to allow the secret.
```

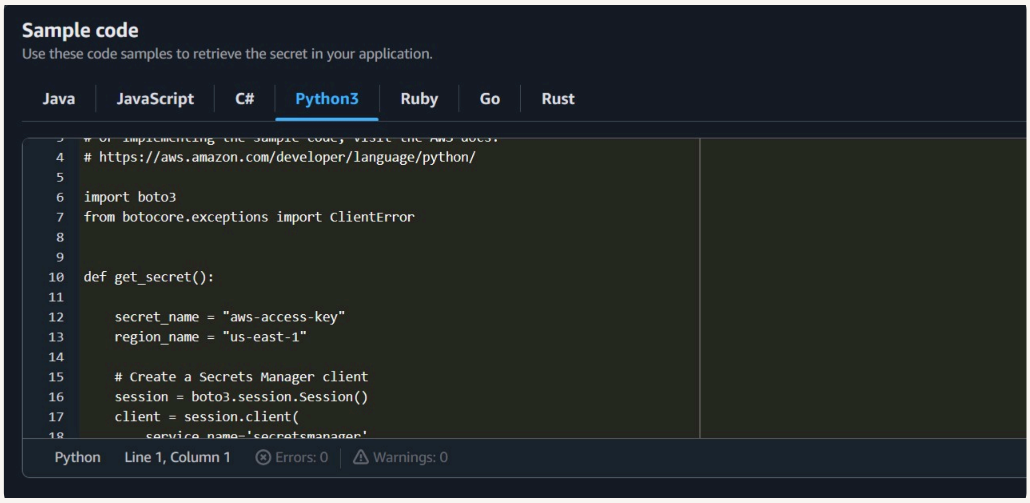
Chrispinus jacob (now)   Ln 5, Col 25   Spaces: 4   UTF-8   CRLF   Python   Signed out   3.13.3 (venv)   Port : 5500   Snyk Secu

Secrets Manager

Secrets Manager is a secure AWS service for storing and managing sensitive information like API keys, passwords, and tokens. I'm using it to store my AWS access key ID and secret access key, so they're no longer hardcoded in my application. Other common use cases include storing database credentials, OAuth tokens, third-party service API keys, and any other configuration secrets that need to be kept safe and accessed programmatically.

Another feature in Secrets Manager is automatic rotation, which means your secrets (like API keys or database credentials) can be updated on a schedule without requiring manual intervention. It's useful in situations where security policies require regular credential changes, or when you want to reduce the risk of long-lived secrets being exposed or misused.

Secrets Manager provides sample code in various languages, like Python, JavaScript, and Java. This is helpful because it allows developers to quickly integrate secret retrieval into their applications without having to write everything from scratch, ensuring secure access to sensitive data with minimal effort.

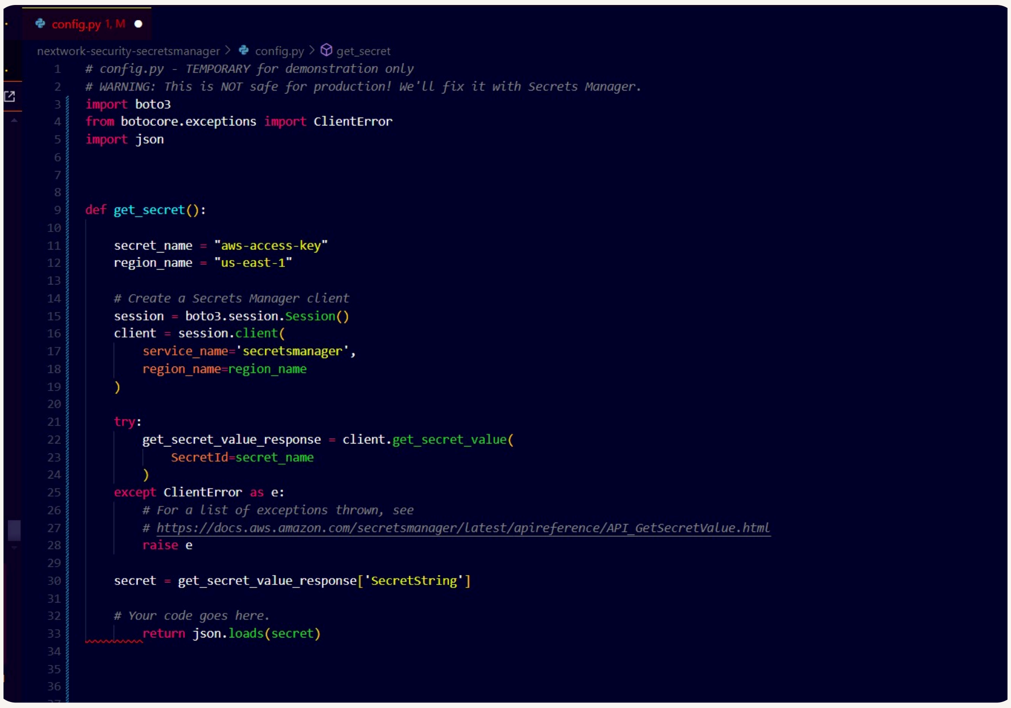

Sample code
Use these code samples to retrieve the secret in your application.
Java
JavaScript
C#
Python3
Ruby
Go
Rust

```python
# https://aws.amazon.com/developer/language/python/

import boto3
from botocore.exceptions import ClientError


def get_secret():

    secret_name = "aws-access-key"
    region_name = "us-east-1"

    # Create a Secrets Manager client
    session = boto3.session.Session()
    client = session.client(
```

Updating the web app code

I updated the `config.py` file to retrieve AWS credentials securely from AWS Secrets Manager. The `get_secret()` function will connect to Secrets Manager, fetch the stored secret (like access key ID and secret access key), and return it for use in the application—eliminating the need to hardcode sensitive information in the codebase.

I also added code to `config.py` to extract specific values like the access key ID and secret access key from the retrieved secret JSON. This is important because it allows the application to use only the necessary parts of the secret in a clean and controlled way, reducing the risk of mishandling sensitive data.



```python
# config.py - TEMPORARY for demonstration only
# WARNING: This is NOT safe for production! We'll fix it with Secrets Manager.
import boto3
from botocore.exceptions import ClientError
import json



def get_secret():

    secret_name = "aws-access-key"
    region_name = "us-east-1"

    # Create a Secrets Manager client
    session = boto3.session.Session()
    client = session.client(
        service_name='secretsmanager',
        region_name=region_name
    )

    try:
        get_secret_value_response = client.get_secret_value(
            SecretId=secret_name
        )
    except ClientError as e:
        # For a list of exceptions thrown, see
        # https://docs.aws.amazon.com/secretsmanager/latest/apireference/API_GetSecretValue.html
        raise e

    secret = get_secret_value_response['SecretString']

    # Your code goes here.
        return json.loads(secret)
```

Rebasing the repository

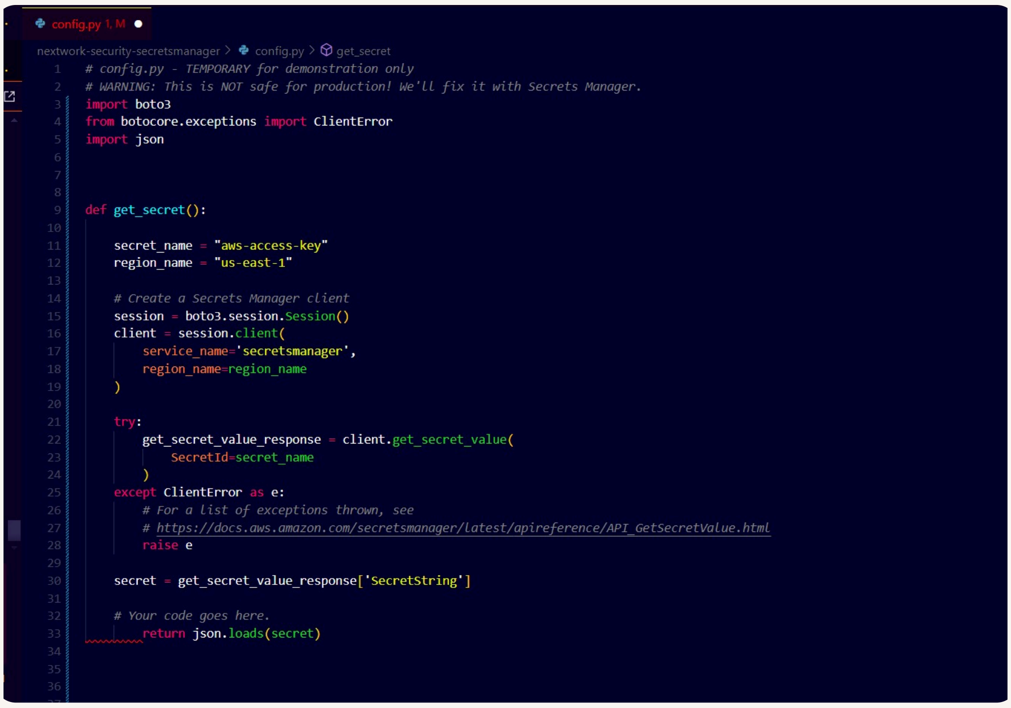
Git rebasing is a way to rewrite commit history by moving or combining a sequence of commits to a new base commit. I used it to clean up my feature branch before merging it with the main branch. This was necessary because it helped create a linear, easy-to-follow commit history and removed any unnecessary merge commits or conflicts.

A merge conflict occurred during rebasing because changes in my branch overlapped with changes in the main branch, specifically in the same lines of code. I resolved the merge conflict by manually editing the conflicting files to keep the correct versions of the code, then staged the changes with `git add` and continued the rebase using `git rebase --continue`.

Once the merge conflict was resolved, I verified that my hardcoded credentials were out of sight in the repository by checking the final committed files with `git diff` and `git log` to ensure no sensitive data remained. I also searched the codebase using keywords like `AWS_SECRET_ACCESS_KEY` and `AKIA` to confirm that no credentials were accidentally pushed. Finally, I reviewed the repository on GitHub to make sure the secrets were not visible in any commit history or code files.

Chrispinus Jacob
NextWork Student

nextwork.org



```python
# config.py - TEMPORARY for demonstration only
# WARNING: This is NOT safe for production! We'll fix it with Secrets Manager.
import boto3
from botocore.exceptions import ClientError
import json



def get_secret():

    secret_name = "aws-access-key"
    region_name = "us-east-1"

    # Create a Secrets Manager client
    session = boto3.session.Session()
    client = session.client(
        service_name='secretsmanager',
        region_name=region_name
    )

    try:
        get_secret_value_response = client.get_secret_value(
            SecretId=secret_name
        )
    except ClientError as e:
        # For a list of exceptions thrown, see
        # https://docs.aws.amazon.com/secretsmanager/latest/apireference/API_GetSecretValue.html
        raise e

    secret = get_secret_value_response['SecretString']

    # Your code goes here.
        return json.loads(secret)
```

nextwork.org

The place to learn & showcase your skills

Check out nextwork.org for more projects