

Store your environment variables with **AWS Secret Manager**

AWS Secret Manager, Advantages, Disadvantages, Examples

AWS Secret Manager

The WHAT:

- A fully managed service to store and retrieve secrets securely
- Supports rotation, auditing, and fine-grained access control
- Integrates easily with AWS services and custom applications

The WHY:

- Centralized management of secrets
- Very easy integration with AWS services
- Automated secret rotation
- Secure and encrypted storage

Advantages and disadvantages of using AWS Secret Manager vs local storage of secrets

AWS Secret Manager	Local storage .env
<ul style="list-style-type: none">- Secrets encrypted at rest and in transit. ✓- Granular access control with IAM. ✓	<ul style="list-style-type: none">- .env or config files can be leaked through: ✗<ul style="list-style-type: none">• Git commits• Server misconfiguration- Easy to use and create ✓
<ul style="list-style-type: none">- Seamless integration with AWS services ✓- Secure and encrypted storage (KMS-based) ✓<ul style="list-style-type: none">- Extra costs ✗	<ul style="list-style-type: none">- Need to copy every time the secrets to new services (Servers, CI pipelines, Environments) ✗<ul style="list-style-type: none">- No extra costs ✓
<ul style="list-style-type: none">- All access and updates logged in CloudTrail (must for compliance (SOC2, ISO, HIPAA) ✓- Supports rotation for RDS, Redshift, and custom apps ✓<ul style="list-style-type: none">- Deeply tied to AWS services ✗	<ul style="list-style-type: none">- No log by default of accessing the secrets ✗- No automatic rotation of leaked/expired secrets ✗- Fast read time. Secrets are loaded directly from disk or environment variables ✓

DEMO

Let's create a new secret and consume it



Select the secret type

Configure secret

Step 3 - optional
Configure rotation

Step 4
Review

☐ Credentials for Amazon RDS database

☐ Credentials for Amazon DocumentDB database

☐ Credentials for Amazon Redshift cluster

☐ Credentials for other database

☒ Other type of secret
API key, OAuth token, other.

Key/value pairs [Info](#)

Key/value

Plaintext

MySecretKey

MySecretPasswordValue

+ Add row

Encryption key [Info](#)


You can encrypt using the KMS key that Secrets Manager creates or a customer managed KMS key that you create.

aws/secretsmanager

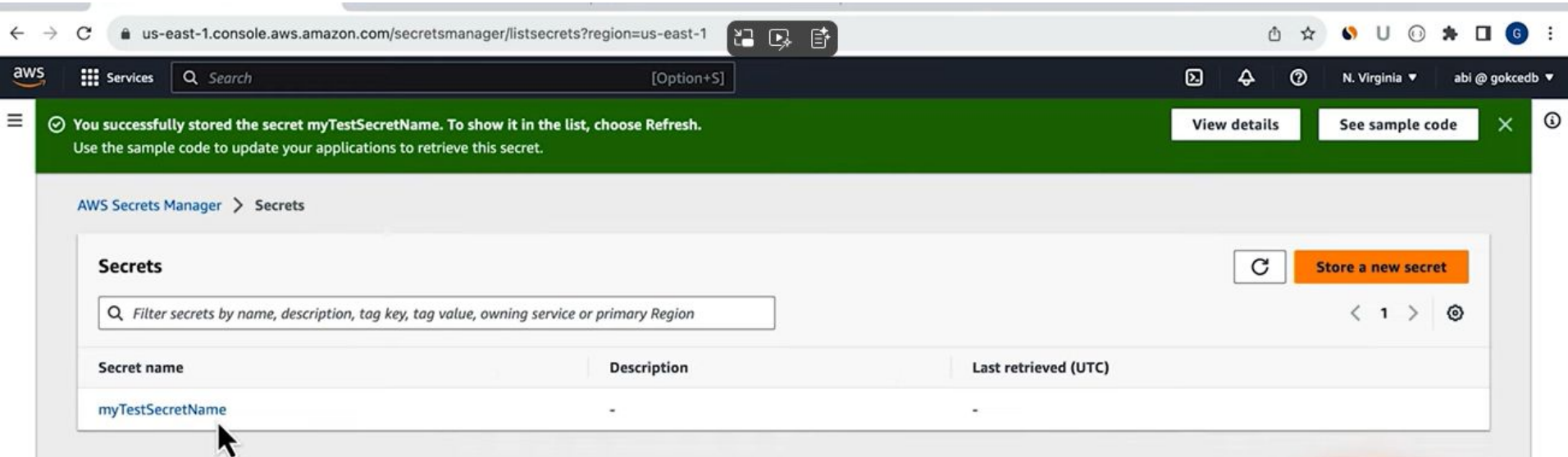
▼

↺

[Add new key](#) [↗](#)



Check if secret was created successfully



The screenshot shows the AWS Secrets Manager console in the us-east-1 region. A green notification banner at the top states: "You successfully stored the secret myTestSecretName. To show it in the list, choose Refresh. Use the sample code to update your applications to retrieve this secret." Below the notification, the "Secrets" section is visible, featuring a search bar and a table of secrets. The table has columns for "Secret name", "Description", and "Last retrieved (UTC)". A single secret, "myTestSecretName", is listed with a hyphen in the description and last retrieved fields. A mouse cursor is pointing at the secret name.

us-east-1.console.aws.amazon.com/secretsmanager/listsecrets?region=us-east-1

aws Services Search [Option+S]

✓ You successfully stored the secret myTestSecretName. To show it in the list, choose Refresh. Use the sample code to update your applications to retrieve this secret. View details See sample code

AWS Secrets Manager > Secrets

Secrets [Refresh](#) [Store a new secret](#)

Filter secrets by name, description, tag key, tag value, owning service or primary Region

Secret name	Description	Last retrieved (UTC)
myTestSecretName	-	-

Create an AWS Lambda function with Python

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/create/function

Create function [Info](#)

AWS Serverless Application Repository applications have moved to [Create application](#).

☒ **Author from scratch**
Start with a simple Hello World example.

☐ **Use a blueprint**
Build a Lambda application from sample code and configuration presets for common use cases.

☐ **Container image**
Select a container image from Amazon ECR or a public repository.

Basic information

Function name
Enter a name that describes the purpose of your function.

mySecretsMgrLambdaFunc

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Python 3.10

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.

☒ x86_64

☐ arm64

Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

Add permissions to use the Secrets Manager

The screenshot shows the AWS IAM console interface. The browser address bar indicates the URL: `us-east-1.console.aws.amazon.com/iamv2/home#/roles/details/mySecretsMgrLambdaFunc-role-m83lrfrg/attach-policies`. The AWS logo and navigation menu are visible at the top. The breadcrumb trail shows the path: `IAM > Roles > mySecretsMgrLambdaFunc-role-m83lrfrg > Add permissions`. The main heading is `Attach policy to mySecretsMgrLambdaFunc-role-m83lrfrg`. Below this, there is a section for `Current permissions policies (1)`. Further down, the `Other permissions policies (1081)` section is shown. A search bar contains the text `secrets`, and a dropdown menu is set to `Filter by Type: All types`, showing `1 match`. The search results table has two columns: `Policy name` and `Type`. The first result is `SecretsManagerReadWrite`, which is an `AWS managed` policy.

us-east-1.console.aws.amazon.com/iamv2/home#/roles/details/mySecretsMgrLambdaFunc-role-m83lrfrg/attach-policies

aws Services Search [Option+S]

IAM > Roles > mySecretsMgrLambdaFunc-role-m83lrfrg > Add permissions

Attach policy to mySecretsMgrLambdaFunc-role-m83lrfrg

► Current permissions policies (1)

Other permissions policies (1081)

Filter by Type

secrets X All types 1 match

<input type="checkbox"/>	Policy name	Type
<input type="checkbox"/>	SecretsManagerReadWrite	AWS managed

Access the secret

```
1 import boto3
2 from botocore.exceptions import ClientError
3
4 def lambda_handler(event, context):
5     secret_name: str = "myTestSecretName"
6     secret_json = get_secret(secret_name)
7     print(secret_json)
8
9     return {
10         'statusCode': 200,
11         'body': json.dumps("Hello from lambda")
12     }
13
14 def get_secret(secret_name):
15     region_name: str = "us-east-1"
16
17     session = boto3.session.Session()
18     client = session.client(
19         service_name='secretsmanager'
20         region_name=region_name
21     )
22
23     try:
24         secret_resp = client.get_secret_value(
25             SecretId=secret_name
26         )
27     except ClientError as e:
28         raise e
29
30     secret = get_secret_value_response('SecretString')
31
32     return secret
```

Run and see the result

Successfully updated the function **mySecretsMgrLambdaFunc**.

mySecretsMgrLambdaFunc

► **Function overview** [Info](#)

[Code](#) | [Test](#) | [Monitor](#) | [Configuration](#) | [Aliases](#) | [Versions](#)

Code source [Info](#)

File Edit Find View Go Tools Window **Test** Deploy

Go to Anything (% P)

Environment

- mySecretsMgrLamb
 - lambda_function.py

▼ Execution results

Test Event Name
(unsaved) test event

Response

```
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}
```

Function Logs

```
START RequestId: 075924bf-eee2-4923-95a7-15f349dc9012 Version: $LATEST
secret_json: {"MySecretKey":"MySecretPasswordValue"}
END RequestId: 075924bf-eee2-4923-95a7-15f349dc9012
REPORT RequestId: 075924bf-eee2-4923-95a7-15f349dc9012 Duration: 1519.13 ms Billed Duration: 1520 ms
```

Request ID
075924bf-eee2-4923-95a7-15f349dc9012