

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) - Extra costs | <ul style="list-style-type: none">- Need to copy every time the secrets to new services (Servers, CI pipelines, Environments) - 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 - 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

The screenshot shows the AWS Secrets Manager landing page in a web browser. The URL in the address bar is `us-east-1.console.aws.amazon.com/secretsmanager/landing?region=us-east-1`. The browser interface includes standard navigation buttons, a search bar, and user information like "N. Virginia" and "abi @ gokcedb".

The main content area features the title "AWS Secrets Manager" in large white font, followed by the subtitle "Easily rotate, manage, and retrieve secrets throughout their lifecycle". Below this, a subtext states "AWS Secrets Manager helps you protect access to your applications, services, and IT resources." A cursor icon is positioned over the word "resources". To the right, a "Get started" section with an orange "Store a new secret" button is visible.

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 a web browser. The URL in the address bar is `us-east-1.console.aws.amazon.com/secretsmanager/listsecrets?region=us-east-1`. The browser's title bar includes the AWS logo and the region `N. Virginia`.

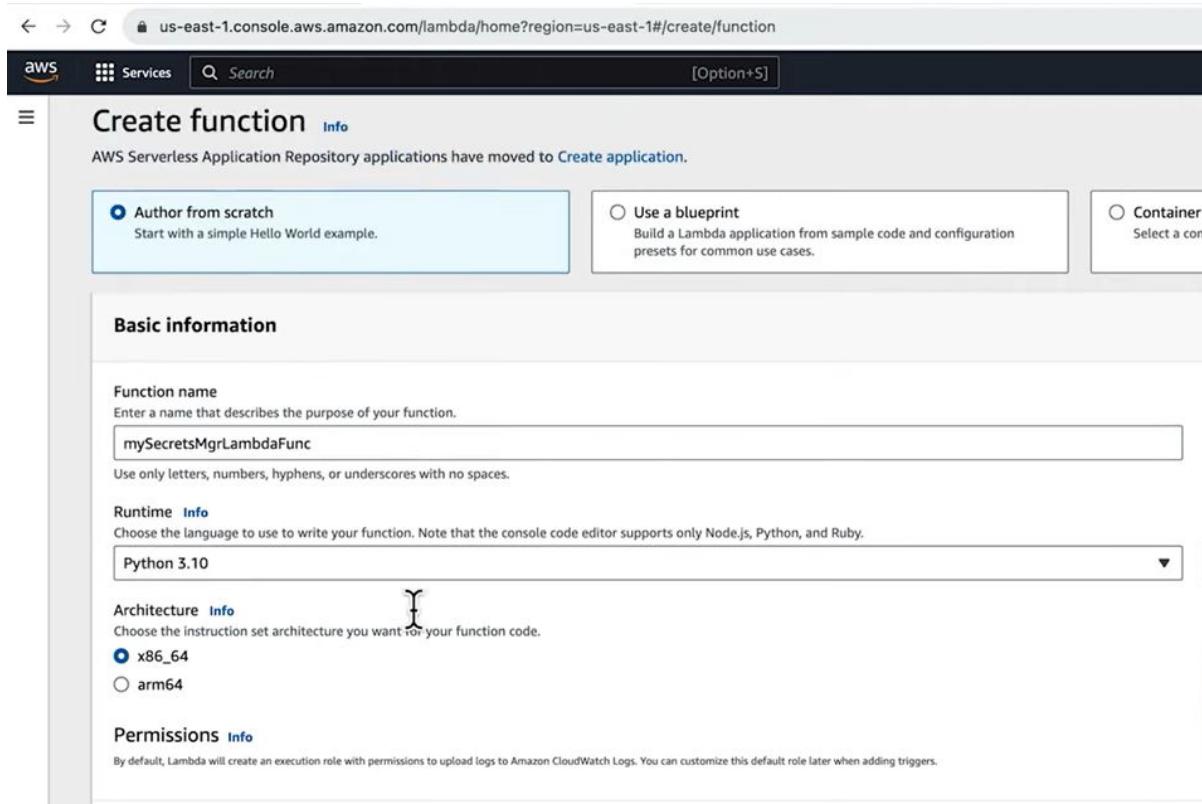
A green success message at the top states: **You successfully stored the secret myTestSecretName. To show it in the list, choose Refresh.** Use the sample code to update your applications to retrieve this secret.

The main interface shows a table of secrets. The first row in the table is highlighted with a yellow background, corresponding to the message above. The table has columns: **Secret name**, **Description**, and **Last retrieved (UTC)**. The single entry in the table is **myTestSecretName**.

At the top right of the table area, there is a **Store a new secret** button. Below the table, there are navigation controls for pages and a refresh icon.

| Secret name | Description | Last retrieved (UTC) |
|------------------|-------------|----------------------|
| myTestSecretName | - | - |

Create an AWS Lambda function with Python



The screenshot shows the 'Create function' wizard on the AWS Lambda console. The top navigation bar includes the AWS logo, a search bar, and a 'Services' dropdown. The main title is 'Create function' with an 'Info' link. A note states: 'AWS Serverless Application Repository applications have moved to [Create application](#)'. Three options are available:

- 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.
- Containerize: Select a container image.

Basic information

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

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.

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 URL in the address bar is `us-east-1.console.aws.amazon.com/iamv2/home#/roles/details/mySecretsMgrLambdaFunc-role-m83lrfrg/attach-policies`. The navigation path is `IAM > Roles > mySecretsMgrLambdaFunc-role-m83lrfrg > Add permissions`. The main title is `Attach policy to mySecretsMgrLambdaFunc-role-m83lrfrg`. Below it, there are two sections: `Current permissions policies (1)` and `Other permissions policies (1081)`. A search bar at the top of the list allows filtering by policy name, with the term `secrets` entered. The results table shows one policy: `SecretsManagerReadWrite`, which is described as `AWS managed`.

| Policy name | Type |
|---|-------------|
| 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

The screenshot shows the AWS Lambda console for the function "mySecretsMgrLambdaFunc". A green banner at the top indicates a successful update. The "Code" tab is selected, showing the code source and execution results.

Code source

File Edit Find View Go Tools Window Test Deploy

Environment

lambda_function.x Environment Var Execution result

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