
AWS Secrets Manager

User Guide



AWS Secrets Manager: User Guide

Copyright © Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Secrets Manager?	1
Basic scenario	1
Features	2
Programmatically retrieve encrypted secret values at runtime	2
Store different types of secrets	2
Encrypt your secret data	3
Automatically rotate your secrets	3
Control access to secrets	4
Compliance with standards	4
Pricing	6
Support and feedback	6
Access Secrets Manager	8
Secrets Manager console	8
Command line tools	8
AWS SDKs	8
HTTPS Query API	9
Get started	10
Secrets Manager concepts	10
Secret	10
Rotation	11
Version	11
Tutorials	12
Replace hardcoded secrets	12
Step 1: Create the secret	12
Step 2: Update your code	14
Step 3: Update the secret	14
Next steps	14
Replace hardcoded DB credentials	15
Step 1: Create the secret	15
Step 2: Update your code	16
Step 3: Rotate the secret	16
Next steps	17
Single user rotation	17
Permissions	18
Prerequisites	18
Step 1: Connect with original password	21
Step 2: Create a Secrets Manager endpoint	22
Step 3: Rotate the secret	22
Step 4: Test the rotated password	23
Step 5: Clean up resources	23
Next steps	24
Alternating users rotation	24
Permissions	24
Prerequisites	24
Step 1: Create an Amazon RDS database user	25
Step 2: Create a secret for the user credentials	25
Step 3: Test the rotated secret	26
Step 4: Clean up resources	26
Next steps	26
Authentication and access control	27
Secrets Manager administrator permissions	27
Permissions to access secrets	27
Permissions for Lambda rotation functions	27
Permissions for encryption keys	27

Attach a permissions policy to an identity	28
Attach a permissions policy to a secret	28
AWS CLI	29
AWS SDK	29
AWS managed policy	30
Determine who has permissions to your secrets	30
Cross-account access	31
Permissions policy examples	32
Example: Permission to retrieve secret values	33
Example: Wildcards	34
Example: Permission to create secrets	35
Example: Permissions and VPCs	35
Example: Control access to secrets using tags	36
Example: Limit access to identities with tags that match secrets' tags	37
Example: Service principal	37
Permissions reference	38
Secrets Manager actions	38
Secrets Manager resources	45
Condition keys	46
BlockPublicPolicy condition	47
IP address conditions	47
VPC endpoint conditions	48
Create and manage secrets	49
Create a database secret	49
AWS CLI	50
AWS SDK	51
JSON structure of a database secret	51
Create a secret	53
AWS CLI	54
AWS SDK	55
Modify a secret	55
AWS CLI	55
AWS SDK	56
Find secrets	57
AWS CLI	57
AWS SDK	58
Delete a secret	58
AWS CLI	59
AWS SDK	60
Restore a secret	60
AWS CLI	60
AWS SDK	61
Replicate a secret to other Regions	61
AWS CLI	62
AWS SDK	62
Promote a replica secret to a standalone secret	62
AWS CLI	63
AWS SDK	63
Tag secrets	63
AWS CLI	64
AWS SDK	64
AWS CloudFormation	64
Create a simple secret	65
Create a secret with Amazon RDS credentials	66
Create a secret with Amazon RDS credentials with automatic rotation	69
Create a secret with Amazon Redshift credentials with automatic rotation	75
Create a secret with Amazon DocumentDB credentials with automatic rotation	80

Retrieve secrets	86
Within other AWS services	86
Connect to a SQL database	86
Establish a connection to a database	87
Establish a connection to a replica database	88
Use c3p0 connection pooling to establish a connection	90
Use c3p0 connection pooling to establish a connection to a replica database	90
Java applications	91
SecretCache	92
SecretCacheConfiguration	93
SecretCacheHook	95
Python applications	96
SecretCache	96
SecretCacheConfig	97
SecretCacheHook	98
@InjectSecretString	99
@InjectKeywordedSecretString	99
.NET applications	100
SecretsManagerCache	101
SecretCacheConfiguration	102
ISecretCacheHook	103
Go applications	104
type Cache	104
type CacheConfig	106
type CacheHook	106
AWS Batch	106
AWS CloudFormation	106
Example: Use a secret to set a database password	107
Amazon ECS	108
Amazon EKS	108
Install the ASCP	109
Step 1: Set up access control	109
Step 2: Mount secrets in Amazon EKS	109
SecretProviderClass	110
Tutorial	111
AWS IoT Greengrass	113
Parameter Store	114
Rotate secrets	115
Rotation strategies	115
Single user	115
Alternating users	116
Rotate DB credentials	117
AWS CLI	118
AWS SDK	119
Rotate a secret	119
AWS SDK and AWS CLI	120
AWS SDK	120
Schedule expressions	120
Rate expressions	120
Cron expressions	121
Rotate a secret immediately	122
AWS SDK and AWS CLI	122
AWS SDK	122
How rotation works	122
Network access for rotation	123
Permissions for rotation	124
Lambda function resource policy	124

Lambda function execution role inline policy	125
Customize a rotation function	127
Rotation function templates	128
Amazon RDS databases	128
Amazon DocumentDB databases	130
Amazon Redshift	131
Other types of secrets	131
Troubleshoot rotation	132
I want to find the diagnostic logs for my Lambda rotation function	132
I get "access denied" when trying to configure rotation for my secret	132
My first rotation fails after I enable rotation	133
Rotation fails because the secret value is not formatted as expected by the rotation function. ...	133
Secrets Manager says I successfully configured rotation, but the password isn't rotating	133
Rotation fails with an "Internal failure" error message	134
CloudTrail shows access-denied errors during rotation	134
My database requires an SSL/TLS connection but the Lambda rotation function isn't using SSL/TLS	135
VPC endpoint	137
Considerations for Secrets Manager VPC endpoints	137
Creating an interface VPC endpoint for Secrets Manager	137
Creating a VPC endpoint policy for Secrets Manager	138
Monitor secrets	139
Logging with AWS CloudTrail	139
AWS CLI or SDK	140
Examples of Secrets Manager log entries	140
Monitoring with CloudWatch	142
Secrets Manager metrics and dimensions	142
Create alarms to monitor Secrets Manager metrics	143
Secrets Manager events	143
Amazon CloudWatch Synthetics canaries	143
Monitor secrets scheduled for deletion	143
Compliance validation	146
Audit secrets for compliance by using AWS Config	146
.....	147
Aggregate secrets from your AWS accounts and AWS Regions	147
Services that use Secrets Manager secrets	148
Alexa for Business	149
AWS App2Container	149
Amazon AppFlow	149
AWS AppSync	149
Amazon Athena	149
AWS CodeBuild	150
AWS Directory Service	150
Amazon DocumentDB	150
AWS Elemental Live	150
AWS Elemental MediaConnect	150
AWS Elemental MediaConvert	151
CodeGuru Reviewer	151
AWS Elemental MediaPackage	151
AWS Elemental MediaTailor	151
Amazon EMR	151
Amazon EventBridge	152
Amazon FSx	152
AWS Glue DataBrew	152
AWS Glue Studio	152
AWS IoT SiteWise	153
Amazon Kendra	153

AWS Launch Wizard	153
Amazon Lookout for Metrics	153
Amazon Managed Streaming for Apache Kafka	153
Amazon Managed Workflows for Apache Airflow	154
AWS Migration Hub	154
AWS OpsWorks for Chef Automate	154
Amazon RDS	154
Amazon Redshift	155
Amazon SageMaker	155
AWS Toolkit for JetBrains	155
AWS Transfer Family	156
Security in Secrets Manager	157
Best practices	157
Mitigate the risks of using the AWS CLI to store your secrets	158
Data protection in Secrets Manager	159
Encryption at rest	160
Encryption in transit	160
Encryption key management	160
Inter-network traffic privacy	161
Secret encryption and decryption	161
Encryption and decryption processes	162
How Secrets Manager uses your KMS key	162
Permissions for the KMS key	163
Secrets Manager encryption context	164
Monitor Secrets Manager interaction with AWS KMS	165
Infrastructure security	167
Resilience	167
Troubleshooting	169
"Access denied" messages when sending requests to Secrets Manager	169
"Access denied" for temporary security credentials	169
Changes I make aren't always immediately visible.	170
"Cannot generate a data key with an asymmetric KMS key" when creating a secret	170
An AWS CLI or AWS SDK operation can't find my secret from a partial ARN	170
Quotas	172
Secret name constraints	172
Maximum quotas	172
Rate quotas	172
Add retries to your application	173
Cross-account requests	174

What is AWS Secrets Manager?

In the past, when you created a custom application to retrieve information from a database, you typically embedded the credentials, the secret, for accessing the database directly in the application. When the time came to rotate the credentials, you had to do more than just create new credentials. You had to invest time to update the application to use the new credentials. Then you distributed the updated application. If you had multiple applications with shared credentials and you missed updating one of them, the application failed. Because of this risk, many customers choose not to regularly rotate credentials, which effectively substitutes one risk for another.

Secrets Manager enables you to replace hardcoded credentials in your code, including passwords, with an API call to Secrets Manager to retrieve the secret programmatically. This helps ensure the secret can't be compromised by someone examining your code, because the secret no longer exists in the code. Also, you can configure Secrets Manager to automatically rotate the secret for you according to a specified schedule. This enables you to replace long-term secrets with short-term ones, significantly reducing the risk of compromise.

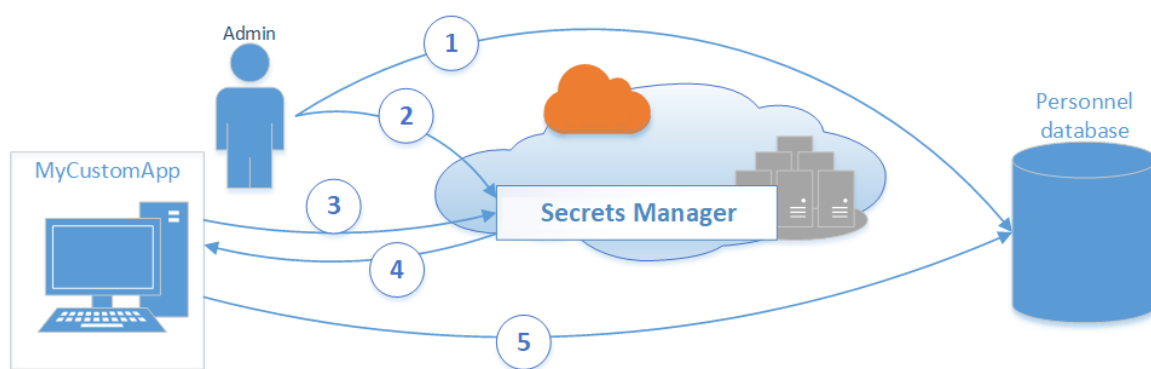
For a list of terms and concepts you need to understand to make full use of Secrets Manager, see [Get started](#) (p. 10).

Topics

- [Basic AWS Secrets Manager scenario](#) (p. 1)
- [Features of AWS Secrets Manager](#) (p. 2)
- [Compliance with standards for AWS Secrets Manager](#) (p. 4)
- [Pricing for AWS Secrets Manager](#) (p. 6)
- [Support and feedback for AWS Secrets Manager](#) (p. 6)

Basic AWS Secrets Manager scenario

The following diagram illustrates the most basic scenario. The diagram displays you can store credentials for a database in Secrets Manager, and then use those credentials in an application to access the database.



1. The database administrator creates a set of credentials on the Personnel database for use by an application called MyCustomApp. The administrator also configures those credentials with the permissions required for the application to access the Personnel database.
2. The database administrator stores the credentials as a secret in Secrets Manager named *MyCustomAppCreds*. Then, Secrets Manager encrypts and stores the credentials within the secret as the *protected secret text*.

3. When MyCustomApp accesses the database, the application queries Secrets Manager for the secret named *MyCustomAppCreds*.
4. Secrets Manager retrieves the secret, decrypts the protected secret text, and returns the secret to the client app over a secured (HTTPS with TLS) channel.
5. The client application parses the credentials, connection string, and any other required information from the response and then uses the information to access the database server.

Note

Secrets Manager supports many types of secrets. However, Secrets Manager can *natively* rotate credentials for [supported AWS databases \(p. 3\)](#) without any additional programming. However, rotating the secrets for other databases or services requires creating a custom Lambda function to define how Secrets Manager interacts with the database or service. You need some programming skill to create the function. For more information, see [Rotate AWS Secrets Manager secrets \(p. 115\)](#).

Features of AWS Secrets Manager

Programmatically retrieve encrypted secret values at runtime

Secrets Manager helps you improve your security posture by removing hard-coded credentials from your application source code, and by not storing credentials within the application, in any way. Storing the credentials in or with the application subjects them to possible compromise by anyone who can inspect your application or the components. Since you have to update your application and deploy the changes to every client before you can deprecate the old credentials, this process makes rotating your credentials difficult.

Secrets Manager enables you to replace stored credentials with a runtime call to the Secrets Manager Web service, so you can retrieve the credentials dynamically when you need them.

Most of the time, your client requires access to the most recent version of the encrypted secret value. When you query for the encrypted secret value, you can choose to provide only the secret name or Amazon Resource Name (ARN), without specifying any version information at all. If you do this, Secrets Manager automatically returns the most recent version of the secret value.

However, other versions can exist at the same time. Most systems support secrets more complicated than a simple password, such as full sets of credentials including the connection details, the user ID, and the password. Secrets Manager allows you to store multiple sets of these credentials at the same time. Secrets Manager stores each set in a different version of the secret. During the secret rotation process, Secrets Manager tracks the older credentials, as well as the new credentials you want to start using, until the rotation completes.

Store different types of secrets

Secrets Manager enables you to store text in the encrypted secret data portion of a secret. This typically includes the connection details of the database or service. These details can include the server name, IP address, and port number, as well as the user name and password used to sign in to the service. For details on secrets, see the [maximum and minimum values](#). The protected text doesn't include:

- Secret name and description
- Rotation or expiration settings
- ARN of the KMS key associated with the secret
- Any attached AWS tags

Encrypt your secret data

Secrets Manager encrypts the protected text of a secret by using [AWS Key Management Service \(AWS KMS\)](#). Many AWS services use AWS KMS for key storage and encryption. AWS KMS ensures secure encryption of your secret when at rest. Secrets Manager associates every secret with a KMS key. It can be either AWS managed key for Secrets Manager for the account (`aws/secretsmanager`), or a customer managed key you create in AWS KMS.

Whenever Secrets Manager encrypt a new version of the protected secret data, Secrets Manager requests AWS KMS to generate a new data key from the KMS key. Secrets Manager uses this data key for [envelope encryption](#). Secrets Manager stores the encrypted data key with the protected secret data. Whenever the secret needs decryption, Secrets Manager requests AWS KMS to decrypt the data key, which Secrets Manager then uses to decrypt the protected secret data. Secrets Manager never stores the data key in unencrypted form, and always disposes the data key immediately after use.

In addition, Secrets Manager, by default, only accepts requests from hosts using open standard [Transport Layer Security \(TLS\)](#) and [Perfect Forward Secrecy](#). Secrets Manager ensures encryption of your secret while in transit between AWS and the computers you use to retrieve the secret.

Automatically rotate your secrets

You can configure Secrets Manager to automatically rotate your secrets without user intervention and on a specified schedule.

You define and implement rotation with an AWS Lambda function. This function defines how Secrets Manager performs the following tasks:

- Creates a new version of the secret.
- Stores the secret in Secrets Manager.
- Configures the protected service to use the new version.
- Verifies the new version.
- Marks the new version as production ready.

Staging labels help you to keep track of the different versions of your secrets. Each version can have multiple staging labels attached, but each staging label can only be attached to one version. For example, Secrets Manager labels the currently active and in-use version of the secret with `AWSCURRENT`. You should configure your applications to always query for the current version of the secret. When the rotation process creates a new version of a secret, Secrets Manager automatically adds the staging label `AWSPENDING` to the new version until testing and validation completes. Only then does Secrets Manager add the `AWSCURRENT` staging label to this new version. Your applications immediately start using the new secret the next time they query for the `AWSCURRENT` version.

Databases with fully configured and ready-to-use rotation support

When you choose to enable rotation, Secrets Manager supports the following Amazon Relational Database Service (Amazon RDS) databases with AWS written and tested Lambda rotation function templates, and full configuration of the rotation process:

- Amazon Aurora on Amazon RDS
- MySQL on Amazon RDS
- PostgreSQL on Amazon RDS
- Oracle on Amazon RDS
- MariaDB on Amazon RDS

- Microsoft SQL Server on Amazon RDS

Other services with fully configured and ready-to-use rotation support

You can also choose to enable rotation on the following services, fully supported with AWS written and tested Lambda rotation function templates, and full configuration of the rotation process:

- Amazon DocumentDB
- Amazon Redshift

You can also store secrets for almost any other kind of database or service. However, to automatically rotate the secrets, you need to create and configure a custom Lambda rotation function. For more information about writing a custom Lambda function for a database or service, see [the section called “How rotation works”](#) (p. 122).



Control access to secrets

You can attach AWS Identity and Access Management (IAM) permission policies to your users, groups, and roles that grant or deny access to specific secrets, and restrict management of those secrets. For example, you might attach one policy to a group with members that require the ability to fully manage and configure your secrets. Another policy attached to a role used by an application might grant only read permission on the one secret the application needs to run.

Alternatively, you can attach a resource-based policy directly to the secret to grant permissions specifying users who can read or modify the secret and the versions. Unlike an identity-based policy which automatically applies to the user, group, or role, a resource-based policy attached to a secret uses the `Principal` element to identify the target of the policy. The `Principal` element can include users and roles from the same account as the secret or principals from other accounts.

Compliance with standards for AWS Secrets Manager

AWS Secrets Manager has undergone auditing for the following standards and can be part of your solution when you need to obtain compliance certification.

	AWS has expanded its Health Insurance Portability and Accountability Act (HIPAA) compliance program to include AWS Secrets Manager as a HIPAA-eligible service . If you have an executed Business Associate Agreement (BAA) with AWS, you can use Secrets Manager to help build your HIPAA-compliant applications. AWS offers a HIPAA-focused whitepaper for customers who are interested in learning more about how they can leverage AWS for the processing and storage of health information. For more information, see HIPAA Compliance .
	AWS Secrets Manager has an Attestation of Compliance for Payment Card Industry (PCI) Data Security Standard (DSS) version 3.2 at Service Provider Level 1. Customers who use AWS products and services to store, process, or transmit cardholder data can use AWS Secrets Manager as they manage their own PCI DSS compliance certification. For more information about PCI DSS, including how to request a copy of the AWS PCI Compliance Package, see PCI DSS Level 1 .



AWS Secrets Manager has successfully completed compliance certification for ISO/IEC 27001, ISO/IEC 27017, ISO/IEC 27018, and ISO 9001. For more information, see [ISO 27001](#), [ISO 27017](#), [ISO 27018](#), [ISO 9001](#).

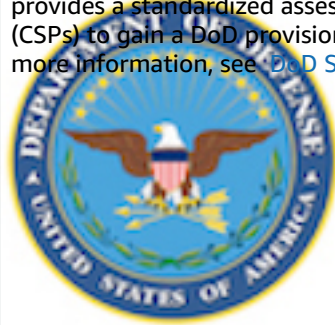


System and Organization Control (SOC) reports are independent third-party examination reports that demonstrate how Secrets Manager achieves key compliance controls and objectives. The purpose of these reports is to help you and your auditors understand the AWS controls that are established to support operations and compliance. For more information, see [SOC Compliance](#).



FedRAMP

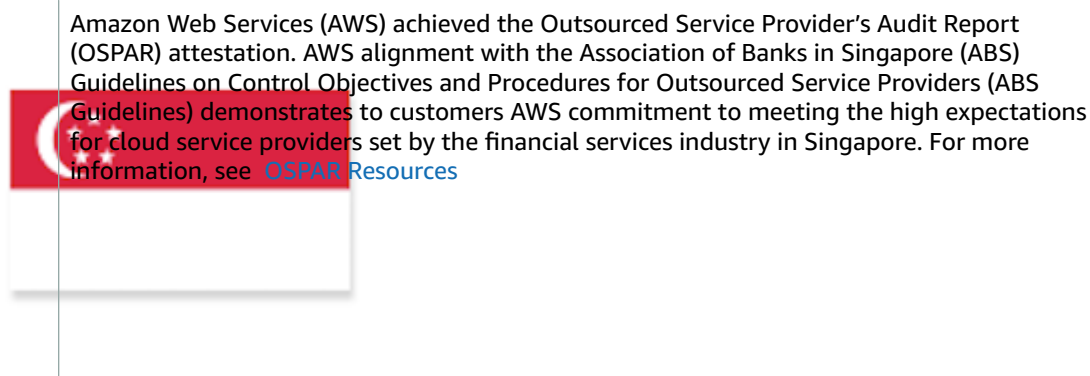
The Federal Risk and Authorization Management Program (FedRAMP) is a government-wide program that provides a standardized approach to security assessment, authorization, and continuous monitoring for cloud products and services. The FedRAMP Program also provides provisional authorizations for services and regions for East/West and GovCloud to consume government or regulated data. For more information, see [FedRAMP Compliance](#).



The Department of Defense (DoD) Cloud Computing Security Requirements Guide (SRG) provides a standardized assessment and authorization process for cloud service providers (CSPs) to gain a DoD provisional authorization, so that they can serve DoD customers. For more information, see [DoD SRG Resources](#)



The Information Security Registered Assessors Program (IRAP) enables Australian government customers to validate that appropriate controls are in place and determine the appropriate responsibility model for addressing the requirements of the Australian government Information Security Manual (ISM) produced by the Australian Cyber Security Centre (ACSC). For more information, see [IRAP Resources](#)



Pricing for AWS Secrets Manager

When you use Secrets Manager, you pay only for what you use, and no minimum or setup fees. There is no charge for secrets that you have marked for deletion. For the current complete pricing list, see [AWS Secrets Manager Pricing](#).

You can use the AWS managed key (`aws/secretsmanager`) that Secrets Manager creates to encrypt your secrets for free. If you create your own KMS keys to encrypt your secrets, AWS charges you at the current AWS KMS rate. For more information, see [AWS Key Management Service pricing](#).

If you enable AWS CloudTrail on your account, you can obtain logs of the API calls that Secrets Manager sends out. Secrets Manager logs all events as management events. AWS CloudTrail stores the first copy of all management events for free. However, you can incur charges for Amazon S3 for log storage and for Amazon SNS if you enable notification. Also, if you set up additional trails, the additional copies of management events can incur costs. For more information, see [AWS CloudTrail pricing](#).

Support and feedback for AWS Secrets Manager

We welcome your feedback. You can send comments to awssecretsmanager-feedback@amazon.com. You also can post your feedback and questions in our [AWS Secrets Manager support forum](#). For more information about the AWS Support forums, see [Forums Help](#).

To request new features for the AWS Secrets Manager console or command line tools, we recommend you submit them in email to awssecretsmanager-feedback@amazon.com.

To provide feedback for our documentation, you can use the feedback link at the bottom of each web page. Be specific about the issue you face and how the documentation failed to help you. Let us know what you saw and how that differed from what you expected. That helps us to understand what we need to do to improve the documentation.

Here are some additional resources available to you:

- [AWS Training Catalog](#) – Role-based and specialty courses, as well as self-paced labs, to help you sharpen your AWS skills and gain practical experience.
- [AWS Developer Tools](#) – Tools and resources that provide documentation, code examples, release notes, and other information to help you build innovative applications with AWS.
- [AWS Support Center](#) – The hub for creating and managing your AWS Support cases. It includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.

- [AWS Support](#) – A one-on-one, fast-response support channel for helping you build and run applications in the cloud.
- [Contact Us](#) – A central contact point for inquiries about AWS billing, accounts, events, and other issues.
- [AWS Site Terms](#) – Detailed information about our copyright and trademark, your account, your license, site access, and other topics.

Access Secrets Manager

You can work with Secrets Manager in any of the following ways:

- [Secrets Manager console](#) (p. 8)
- [Command line tools](#) (p. 8)
- [AWS SDKs](#) (p. 8)
- [HTTPS Query API](#) (p. 9)

Secrets Manager console

You can manage your secrets using the browser-based [Secrets Manager console](#) and perform almost any task related to your secrets by using the console.

Command line tools

The AWS command line tools allows you to issue commands at your system command line to perform Secrets Manager and other AWS tasks. This can be faster and more convenient than using the console. The command line tools can be useful if you want to build scripts to perform AWS tasks.

AWS provides two sets of command line tools:

- [AWS Command Line Interface \(AWS CLI\)](#)
- [AWS Tools for Windows PowerShell](#)

AWS SDKs

The AWS SDKs consist of libraries and sample code for various programming languages and platforms, for example, Java, Python, Ruby, .NET, and others. The SDKs include tasks such as cryptographically signing requests, managing errors, and retrying requests automatically. For more information, see [the section called "AWS SDKs" \(p. 8\)](#).

To download and install any of the SDKs, see [Tools for Amazon Web Services](#).

For SDK documentation, see:

- [C++](#)
- [Java](#)
- [PHP](#)
- [Python](#)
- [Ruby](#)
- [.NET](#)
- [Node.js](#)
- [Go](#)

HTTPS Query API

The HTTPS Query API gives you [programmatic access](#) to Secrets Manager and AWS. The HTTPS Query API allows you to issue HTTPS requests directly to the service.

Although you can make direct calls to the Secrets Manager HTTPS Query API, we recommend that you use one of the SDKs instead. The SDK performs many useful tasks you otherwise must perform manually. For example, the SDKs automatically sign your requests and convert responses into a structure syntactically appropriate to your language.

Get started with AWS Secrets Manager

There are many different types of secrets you might have in your organization. Here are some of them, and where you can store them in AWS:

- AWS credentials – [AWS Identity and Access Management](#)
- Encryption keys – [AWS Key Management Service](#)
- SSH keys – [Amazon EC2 Instance Connect](#)
- Private keys and certificates – [AWS Certificate Manager](#)
- **Database credentials – Secrets Manager**
- **Application credentials – Secrets Manager**
- **OAuth tokens – Secrets Manager**
- **Application Programming Interface (API) keys – Secrets Manager**

Secrets Manager concepts

The following concepts are important for understanding how Secrets Manager works.

Secret

In Secrets Manager, a *secret* consists of secret information, the *secret value*, plus metadata about the secret. A secret value can be a string or binary. To store multiple string values in one secret, we recommend that you use a JSON text string with key/value pairs, for example:

```
{
  "host"      : "ProdServer-01.databases.example.com",
  "port"      : "8888",
  "username"  : "administrator",
  "password"  : "EXAMPLE-PASSWORD",
  "dbname"    : "MyDatabase",
  "engine"    : "mysql"
}
```

A secret's metadata includes:

- An Amazon Resource Name (ARN) with the following format:

```
arn:aws:secretsmanager:<Region>:<AccountId>:secret:SecretName-6RandomCharacters
```

- The name of the secret, a description, a resource policy, and tags.
- The ARN for an *encryption key*, an AWS KMS key that Secrets Manager uses to encrypt and decrypt the secret value. Secrets Manager stores secret text in an encrypted form and encrypts the secret in transit. See [the section called "Secret encryption and decryption" \(p. 161\)](#).
- Information about how to rotate the secret, if you set up rotation. See [the section called "Rotation" \(p. 11\)](#).

Secrets Manager uses IAM permission policies to make sure that only authorized users can access or modify a secret. See [Authentication and access control for AWS Secrets Manager](#) (p. 27).

A secret has *versions* which hold copies of the encrypted secret value. When you change the secret value, or the secret is rotated, Secrets Manager creates a new version. See [the section called "Version"](#) (p. 11).

You can use a secret across multiple AWS Regions by *replicating* it. When you replicate a secret, you create a copy of the original or *primary secret* called a *replica secret*. The replica secret remains linked to the primary secret. See [the section called "Replicate a secret to other Regions"](#) (p. 61).

See [Create and manage secrets](#) (p. 49).

Rotation

Rotation is the process of periodically updating a secret to make it more difficult for an attacker to access the credentials. In Secrets Manager, you can set up automatic rotation for your secrets. When Secrets Manager rotates a secret, it updates the credentials in both the secret and the database or service. See [Rotate secrets](#) (p. 115).

Version

A secret has *versions* which hold copies of the encrypted secret value. When you change the secret value, or the secret is rotated, Secrets Manager creates a new version. A secret always has a version with the staging label `AWSCURRENT`, which is the current secret value.

During rotation, Secrets Manager uses staging labels to indicate the different versions of a secret:

- `AWSCURRENT` indicates the version that is actively used by clients. A secret always has an `AWSCURRENT` version.
- `AWSPENDING` indicates the version that will become `AWSCURRENT` when rotation completes.
- `AWSPREVIOUS` indicates the *last known good* version, in other words, the previous `AWSCURRENT` version.

Secrets Manager deprecates versions with no staging labels and removes them when there are more than 100. Secrets Manager doesn't remove versions created less than 24 hours ago.

When you use the AWS CLI or AWS SDK to get the secret value, you can specify the version of the secret. If you don't specify a version, either by version ID or staging label, Secrets Manager gets the version with the staging label `AWSCURRENT` attached.

You can also attach your own staging label to a version, for example to indicate development or production versions. You can attach up to 20 staging labels to a secret. Two versions of a secret can't have the same staging label.

AWS Secrets Manager tutorials

Topics

- [Move hardcoded secrets to AWS Secrets Manager \(p. 12\)](#)
- [Move hardcoded database credentials to AWS Secrets Manager \(p. 15\)](#)
- [Set up single user rotation for AWS Secrets Manager \(p. 17\)](#)
- [Set up alternating users rotation for AWS Secrets Manager \(p. 24\)](#)

Move hardcoded secrets to AWS Secrets Manager

If you have plaintext secrets in your code, we recommend that you rotate them and store them in Secrets Manager. Moving the secret to Secrets Manager solves the problem of the secret being visible to anyone who sees the code, because going forward, your code retrieves the secret directly from Secrets Manager. Rotating the secret revokes the current hardcoded secret so that it is no longer valid.

For database credential secrets, see [Move hardcoded database credentials to AWS Secrets Manager \(p. 15\)](#).

Before you begin, you need to determine who needs access to the secret. We recommend using two IAM roles to manage permission to your secret:

- A role that manages the secrets in your organization. For more information, see [the section called “Secrets Manager administrator permissions” \(p. 27\)](#). You'll create and rotate the secret using this role.
- A role that can use the secret at runtime, for example in this tutorial you use `RoleToRetrieveSecretAtRuntime`. Your code assumes this role to retrieve the secret. In this tutorial, you grant the role only the permission to retrieve one secret value, and you grant permission by using the secret's resource policy. For other alternatives, see [the section called “Next steps” \(p. 14\)](#).

Steps:

- [Step 1: Create the secret \(p. 12\)](#)
- [Step 2: Update your code \(p. 14\)](#)
- [Step 3: Update the secret \(p. 14\)](#)
- [Next steps \(p. 14\)](#)

Step 1: Create the secret

The first step is to copy the existing hardcoded secret into Secrets Manager. If the secret is related to an AWS resource, store it in the same Region as the resource. Otherwise, store it in the Region that has lowest latency for your use case.

To create a secret (console)

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose **Store a new secret**.
3. On the **Store a new secret** page, do the following:

- a. For **Secret type**, choose **Other type of secret**.
- b. Enter your secret as **Key/value pairs** or in **Plaintext**. Some examples:

API key key/value pairs: ClientID: <i>my_client_id</i> ClientSecret: <i>wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY</i>
Credentials key/value pairs: Username: <i>saanvis</i> Password: <i>EXAMPLE-PASSWORD</i>
OAuth token plaintext: <i>AKIAI44QH8DHBEXAMPLE</i>
Digital certificate plaintext: <div><pre>-----BEGIN CERTIFICATE----- EXAMPLE -----END CERTIFICATE-----</pre></div>
Private key plaintext: <div><pre>-----BEGIN PRIVATE KEY --- EXAMPLE ----- END PRIVATE KEY ----</pre></div>

- c. For **Encryption key**, choose **aws/secretsmanager** to use the AWS managed key for Secrets Manager. There is no cost for using this key. You can also use your own customer managed key, for example to [access the secret from another AWS account \(p. 31\)](#).
 - d. Choose **Next**.
4. On the **Secret name and description** page, do the following:
 - a. Enter a descriptive **Secret name** and **Description**.
 - b. In **Resource permissions**, choose **Edit permissions**. Paste the following policy, which allows *RoleToRetrieveSecretAtRuntime* to retrieve the secret, and then choose **Save**.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": "arn:aws:iam::AccountId:role/RoleToRetrieveSecretAtRuntime"  
      },  
      "Action": "secretsmanager:GetSecretValue",  
      "Resource": "*"   
    }  
  ]  
}
```

- c. At the bottom of the page, choose **Next**.
5. On the **Secret rotation** page, keep rotation off. Choose **Next**.

6. On the **Review** page, review your secret details, and then choose **Store**.

Step 2: Update your code

Your code must assume the IAM role `RoleToRetrieveSecretAtRuntime` to be able to retrieve the secret. For more information, see [Switching to an IAM role \(AWS API\)](#).

Next, you update your code to retrieve the secret from Secrets Manager using the sample code provided by Secrets Manager.

To find the sample code

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. On the **Secrets** page, choose your secret.
3. Scroll down to **Sample code**. Choose your programming language, and then copy the code snippet.

In your application, remove the hardcoded secret and paste the code snippet. Depending on your code language, you might need to add a call to the function or method in the snippet.

Test that your application works as expected with the secret in place of the hardcoded secret.

Step 3: Update the secret

The last step is to revoke and update the hardcoded secret. Refer to the source of the secret to find instructions to revoke and update the secret. For example, you might need to deactivate the current secret and generate a new secret.

To update the secret with the new value

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose **Secrets**, and then choose the secret.
3. On the **Secret details** page, scroll down and choose **Retrieve secret value**, and then choose **Edit**.
4. Update the secret and then choose **Save**.

Next, test that your application works as expected with the new secret.

Next steps

After you remove a hardcoded secret from your code, some ideas to consider next:

- To find hardcoded secrets in your Java and Python applications, we recommend [Amazon CodeGuru Reviewer](#).
- You can improve performance and reduce costs by caching secrets. For more information, see [Retrieve secrets](#) (p. 86).
- For secrets that you access from multiple Regions, consider replicating your secret to improve latency. For more information, see [the section called "Replicate a secret to other Regions"](#) (p. 61).
- In this tutorial, you granted `RoleToRetrieveSecretAtRuntime` only the permission to retrieve the secret value. To grant the role more permissions, for example to get metadata about the secret or to view a list of secrets, see [the section called "Permissions policy examples"](#) (p. 32).
- In this tutorial, you granted permission to `RoleToRetrieveSecretAtRuntime` by using the secret's resource policy. For other ways to grant permission, see [the section called "Attach a permissions policy to an identity"](#) (p. 28).

Move hardcoded database credentials to AWS Secrets Manager

If you have plaintext database credentials in your code, we recommend that you move the credentials to Secrets Manager and then rotate them immediately. Moving the credentials to Secrets Manager solves the problem of the credentials being visible to anyone who sees the code, because going forward, your code retrieves the credentials directly from Secrets Manager. Rotating the secret updates the password and then revokes the current hardcoded password so that it is no longer valid.

For Amazon RDS, Amazon Redshift, and Amazon DocumentDB databases, use the steps in this page to move hardcoded credentials to Secrets Manager. For other types of credentials and other secrets, see [the section called "Replace hardcoded secrets "](#) (p. 12).

Before you begin, you need to determine who needs access to the secret. We recommend using two IAM roles to manage permission to your secret:

- A role that manages the secrets in your organization. For more information, see [the section called "Secrets Manager administrator permissions"](#) (p. 27). You'll create and rotate the secret using this role.
- A role that can use the credentials at runtime, `RoleToRetrieveSecretAtRuntime` in this tutorial. Your code assumes this role to retrieve the secret.

Steps:

- [Step 1: Create the secret](#) (p. 15)
- [Step 2: Update your code](#) (p. 16)
- [Step 3: Rotate the secret](#) (p. 16)
- [Next steps](#) (p. 17)

Step 1: Create the secret

The first step is to copy the existing hardcoded credentials into a secret in Secrets Manager. For the lowest latency, store the secret in the same Region as the database.

To create a secret

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose **Store a new secret**.
3. On the **Store a new secret** page, do the following:
 - a. For **Secret type**, choose the type of database credentials to store:
 - **Amazon RDS database**
 - **Amazon DocumentDB database**
 - **Amazon Redshift cluster**.
 - For other types of secrets, see [Replace hardcoded secrets](#) .
 - b. For **Credentials**, enter the existing hardcoded credentials for the database.
 - c. For **Encryption key**, choose **aws/secretsmanager** to use the AWS managed key for Secrets Manager. There is no cost for using this key. You can also use your own customer managed key, for example to [access the secret from another AWS account](#) (p. 31).
 - d. For **Database**, choose your database.
 - e. Choose **Next**.

4. On the **Secret name and description** page, do the following:
 - a. Enter a descriptive **Secret name** and **Description**.
 - b. In **Resource permissions**, choose **Edit permissions**. Paste the following policy, which allows *RoleToRetrieveSecretAtRuntime* to retrieve the secret, and then choose **Save**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountId:role/RoleToRetrieveSecretAtRuntime"
      },
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "*"
    }
  ]
}
```

- c. At the bottom of the page, choose **Next**.
5. On the **Secret rotation** page, keep rotation off for now. You'll turn it on later. Choose **Next**.
6. On the **Review** page, review your secret details, and then choose **Store**.

Step 2: Update your code

Your code must assume the IAM role *RoleToRetrieveSecretAtRuntime* to be able to retrieve the secret. For more information, see [Switching to an IAM role \(AWS API\)](#).

Next, you update your code to retrieve the secret from Secrets Manager using the sample code provided by Secrets Manager.

To find the sample code

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. On the **Secrets** page, choose your secret.
3. Scroll down to **Sample code**. Choose your language, and then copy the code snippet.

In your application, remove the hardcoded credentials and paste the code snippet. Depending on your code language, you might need to add a call to the function or method in the snippet.

Test that your application works as expected with the secret in place of the hardcoded credentials.

Step 3: Rotate the secret

The last step is to revoke the hardcoded credentials by rotating the secret. *Rotation* is the process of periodically updating a secret. When you rotate a secret, you update the credentials in both the secret and the database. Secrets Manager can automatically rotate a secret for you on a schedule you set.

Part of setting up rotation is ensuring that the Lambda rotation function can access both Secrets Manager and your database. When you turn on automatic rotation, Secrets Manager creates the Lambda rotation function in the same VPC as your database so that it has network access to the database. The Lambda rotation function must also be able to make calls to Secrets Manager to update the secret. We recommend that you create a Secrets Manager endpoint in the VPC so that calls from Lambda to Secrets Manager don't leave AWS infrastructure. For more information, see [the section called "Network access for rotation" \(p. 123\)](#). For instructions, see [VPC endpoint \(p. 137\)](#).

To turn on rotation

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. On the **Secrets** page, choose your secret.
3. On the **Secret details** page, in the **Rotation configuration** section, choose **Edit rotation**.
4. In the **Edit rotation configuration** dialog box, do the following:
 - a. Turn on **Automatic rotation**.
 - b. Under **Rotation schedule**, enter your schedule in UTC time zone.
 - c. Choose **Rotate immediately when the secret is stored** to rotate your secret when you save your changes.
 - d. Under **Rotation function**, choose **Create a new Lambda function** and enter a name for your new function. Secrets Manager adds "SecretsManager" to the beginning of your function name.
 - e. For **Use separate credentials to rotate this secret**, choose **No**.
 - f. Choose **Save**.

To check that the secret rotated

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose **Secrets**, and then choose the secret.
3. On the **Secret details** page, scroll down and choose **Retrieve secret value**.

If the secret value changed, then rotation succeeded. If the secret value didn't change, you need to [Troubleshoot rotation \(p. 132\)](#) by looking at the CloudWatch Logs for the rotation function.

Test that your application works as expected with the rotated secret.

Next steps

After you remove a hardcoded secret from your code, some ideas to consider next:

- You can improve performance and reduce costs by caching secrets. For more information, see [Retrieve secrets \(p. 86\)](#).
- You can choose a different rotation schedule. For more information, see [the section called "Schedule expressions" \(p. 120\)](#).
- You can choose a different rotation strategy. For more information, see [the section called "Rotation strategies" \(p. 115\)](#).
- To find hardcoded secrets in your Java and Python applications, we recommend [Amazon CodeGuru Reviewer](#).

Set up single user rotation for AWS Secrets Manager

In this tutorial, you learn how to set up single user rotation for a secret that contains database admin credentials. *Single user rotation* is a rotation strategy where Secrets Manager updates a single user's credentials in both the secret and the database. For more information, see [the section called "Rotation strategies" \(p. 115\)](#).

A large part of this tutorial is setting up a realistic environment. To show you how rotation works, this tutorial uses an example Amazon RDS MySQL database. For security, the database is in a VPC that

doesn't allow internet access. To connect to the database from your local computer through the internet, you use a *bastion host*, a server in the VPC that can connect to the database, but that also allows SSH connections from the internet. The bastion host in this tutorial is an Amazon EC2 instance, and the security groups for the instance prevent other types of connections.

As part of the prerequisites for the tutorial, you also create a secret that contains admin credentials for the database. The secret doesn't initially have rotation turned on. In this tutorial, you'll learn how to turn on automatic rotation.

Contents

- [Permissions \(p. 18\)](#)
- [Prerequisites \(p. 18\)](#)
 - [Prereq A: Amazon RDS database, Amazon VPC, and a Secrets Manager secret \(p. 18\)](#)
 - [Prereq B: Internet gateway \(p. 19\)](#)
 - [Prereq C: Security group \(p. 20\)](#)
 - [Prereq D: Amazon EC2 instance \(p. 20\)](#)
 - [Prereq E: MySQL Workbench \(p. 21\)](#)
- [Step 1: Connect with original password \(p. 21\)](#)
- [Step 2: Create a Secrets Manager endpoint \(p. 22\)](#)
- [Step 3: Rotate the secret \(p. 22\)](#)
- [Step 4: Test the rotated password \(p. 23\)](#)
- [Step 5: Clean up resources \(p. 23\)](#)
- [Next steps \(p. 24\)](#)

Permissions

For the tutorial prerequisites, you need administrative permissions to your AWS account. In a production setting, it is a best practice to use different roles for each of the steps. For example, a role with database admin permissions would create the Amazon RDS database, and a role with network admin permissions would set up the VPC and security groups. For the tutorial steps, we recommend you continue using the same identity.

For information about how to set up permissions in a production environment, see [Authentication and access control \(p. 27\)](#).

Prerequisites

For this tutorial, you need the following:

- [Prereq A: Amazon RDS database, Amazon VPC, and a Secrets Manager secret \(p. 18\)](#)
- [Prereq B: Internet gateway \(p. 19\)](#)
- [Prereq C: Security group \(p. 20\)](#)
- [Prereq D: Amazon EC2 instance \(p. 20\)](#)
- [Prereq E: MySQL Workbench \(p. 21\)](#)

Prereq A: Amazon RDS database, Amazon VPC, and a Secrets Manager secret

Rather than creating these resources through the console, for this tutorial you use AWS CloudFormation with a provided template to create a CloudFormation stack. For more information about CloudFormation and templates, see [AWS CloudFormation concepts](#).

To get the CloudFormation template

- Go to [the section called "Create a secret with Amazon RDS credentials" \(p. 66\)](#), and save the code to a new file. You can use either JSON or YAML.

To create the stack from the template

- Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
- Under **Stacks**, choose **Create stack** and then choose **With new resources**.
- On the **Create stack** page, for **Prepare template**, choose **Template is ready**.
- For **Template source**, choose **Upload a template file**.
- Choose **Choose file**, and then choose the file you saved.
- Choose **Next**.
- On the **Specify stack details** page, name the stack **SecretsManagerRotationTutorial**, and then choose **Next**.
- On the **Configure stack options** page, choose **Next**.
- On the Review page, choose **Create stack**.

CloudFormation opens the new stack in the console and begins creating the resources in the template. This process can take up to 30 minutes. You can see how far along it is in the process under **Events**. Choose the refresh button to update the events.

When the stack is complete, under **Logical ID** you see **SecretsManagerRotationTutorial** with **Status CREATE_COMPLETE**.

Prereq B: Internet gateway

For this tutorial, you need to create an internet gateway and attach it to the VPC to allow traffic to leave the VPC. You create a route in the route table so that traffic destined for outside the VPC is sent to the internet gateway. For more information, see [Connect subnets to the internet using an internet gateway](#).

To create an internet gateway

- Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
- Choose **Internet Gateways** and then choose **Create internet gateway**.
- On the **Create internet gateway** page, for **Name tag**, enter **SecretsManagerTutorialGateway**, and then choose **Create internet gateway**.
- On the **igw-**** / SecretsManagerTutorialGateway** page, in the green banner, choose **Attach to a VPC**.
- On the **Attach to VPC** page, for **Available VPCs**, choose **vpc-**** - SecretsManagerTutorial**, and then choose **Attach internet gateway**.

To add a route for the internet gateway

- Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
- Choose **Route tables**, and then select the **Route table ID** associated with the VPC **vpc-****** | **SecretsManagerTutorial**. You might need to scroll to see the column **VPC** in the table.
- Choose **Actions** and then choose **Edit routes**.
- On the **Edit routes** page, choose **Add Route**, and then do the following:
 - For **Destination**, enter **0.0.0.0/0**.

- b. For **Target**, choose **Internet Gateway** and then choose **igw-**** (SecretsManagerTutorialGateway)**.
- c. Choose **Save changes**.

Prereq C: Security group

Create a security group to allow inbound SSH traffic to access a Amazon EC2 bastion host you'll create in a later step.

To allow SSH access to the bastion host

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. Choose **Security Groups** and then choose **Create Security Group**.
3. For **Security group name**, enter **SecretsManagerTutorialAccess**.
4. For **Description**, enter **Allows SSH access to bastion host**.
5. For **VPC**, choose **vpc-**** (SecretsManagerTutorial)**. You might need to delete the prefilled text to see other choices.
6. Under **Inbound Rules**, choose **Add Rule**.
7. For Inbound rule 1, do the following:
 - a. For **Type**, choose **SSH**.
 - b. For **Source**, choose **My IP**.
8. Choose **Create security group**.

Prereq D: Amazon EC2 instance

To access the database in the VPN, you use a bastion host. The bastion host is also in the VPN, but it allows your local computer to connect to it with SSH. From the bastion host, you can access the database.

To create an EC2 instance for your bastion host

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Choose **Instances** and then choose **Launch Instances**.
3. On the **Step 1** page, choose the default **Amazon Linux 2 AMI (HVM) Kernel 5.10** and then choose **Select**.
4. On the **Step 2** page, choose the default **t2.micro** and then choose **Next: Configure Instance Details**.
5. On the Step 3 page, do the following:
 - a. For **Network**, choose **vpc-**** SecretsManagerTutorial**.
 - b. For **Auto-assign Public IP**, choose **Enable**.
 - c. Choose **Next: Add Storage**.
6. On the **Step 4** page, choose **Next: Add Tags**.
7. On the **Step 5** page, choose **Add Tag**.

For **Key** enter **Name**, and for **Value** enter **SecretsManagerTutorialInstance**, and then choose **Next: Configure Security Group**.

8. On the **Step 6** page, for **Assign a security group**, choose **Select an existing security group**.

9. For Security group ID, choose the security groups with the names **default** and **SecretsManagerTutorialAccess**, and then choose **Review and Launch**.
10. On the **Step 7** page, choose **Launch**.
11. In the **Select an existing key pair** dialog box, do the following:
 - a. Select **Create a new key pair**.
 - b. For **Key pair name**, enter **SecretsManagerTutorialKeyPair**.
 - c. Choose **Download Key Pair**. You will use this private key file to connect to the instance in a later step.
 - d. Choose **Launch Instances**.

Prereq E: MySQL Workbench

To connect to the database, you use a MySQL client tool. In this tutorial, you use MySQL Workbench, a GUI-based application.

To install MySQL Workbench, see [Download MySQL Workbench](#).

To connect to the database, you first create a connection configuration in MySQL Workbench. For the configuration, you need some information from both Amazon EC2 and Amazon RDS.

To create a database connection in MySQL Workbench

1. In MySQL Workbench, next to **MySQL Connections**, choose the (+) button.
2. In the **Setup New Connection** dialog box, do the following:
 - a. For **Connection Name**, enter **SecretsManagerTutorial**.
 - b. For **Connection Method**, choose **Standard TCP/IP over SSH**.
 - c. On the **Parameters** tab, do the following:
 - i. For **SSH Hostname**, enter the public IP address of the Amazon EC2 instance.

You can find the IP address on the Amazon EC2 console by choosing the instance **SecretsManagerTutorialInstance**. Copy the IP address under **Public IPv4 DNS**.
 - ii. For **SSH Username**, enter **ec2-user**.
 - iii. For **SSH Keyfile**, choose the key pair file **SecretsManagerTutorialKeyPair.pem** you downloaded in the previous prerequisite.
 - iv. For **MySQL Hostname**, enter the Amazon RDS endpoint address.

You can find the endpoint address on the Amazon RDS console by choosing the database instance **secretsmanagertutorialdb**. Copy the address under **Endpoint**.
 - v. For **Username**, enter **admin**.
 - d. Choose **OK**.

Step 1: Connect with original password

The first step is to check that the information in the secret contains valid credentials for the database.

To retrieve the password from the secret

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose **Secrets**, and then choose the secret **SecretsManagerTutorialAdmin-******.

3. On the **Secret details** page, scroll down and choose **Retrieve secret value**.
4. In the **Key/value** table, copy the **Secret value** for **password**.

To test the credentials

1. In MySQL Workbench, choose the connection **SecretsManagerTutorial**.
2. In the **Open SSH Connection** dialog box, for **Password**, paste the password you retrieved from the secret, and then choose **OK**.

The first time you connect, you might see a warning dialog box about the server fingerprint. Choose **OK** to continue.

If the credentials are valid, then MySQL Workbench opens to the design page for the database.

Step 2: Create a Secrets Manager endpoint

The next step is to create a Secrets Manager endpoint within the VPC. When you set up automatic rotation, Secrets Manager creates the Lambda rotation function within the VPC so that it has access to the database. The Lambda rotation function also calls Secrets Manager. By creating a Secrets Manager endpoint within the VPC, you ensure that calls from Lambda to Secrets Manager don't leave AWS infrastructure. Instead, they are routed to the Secrets Manager endpoint within the VPC. For more information, see [the section called "Network access for rotation" \(p. 123\)](#).

To create a Secrets Manager endpoint within the VPC

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. Under **Endpoints**, choose **Create Endpoint**.
3. Scroll down to **Services**, enter **secretsmanager** to filter the list, and then select the Secrets Manager endpoint in your AWS Region. For example, in the US East (N. Virginia), choose `com.amazonaws.us-east-1.secretsmanager`.
4. For **VPC**, choose **vpc**** (SecretsManagerTutorial)**.
5. For **Subnets**, select all **Availability Zones**, and then for each one, choose a **Subnet ID** to include.
6. For **Security groups**, choose the default security group.
7. For **Policy**, choose **Full access**.
8. Choose **Create endpoint**.

Step 3: Rotate the secret

Now you are ready to turn on rotation. You start an immediate rotation, so Secrets Manager rotates the secret immediately when you save the secret. You also turn on automatic rotation, so the secret is rotated every 10 days between midnight and 2:00 AM UTC.

To turn on automatic rotation

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Open the secret **SecretsManagerTutorialAdmin-a1b2c3d4e5f6**, scroll down to **Rotation configuration**, and choose **Edit rotation**.
3. In the **Edit rotation configuration** dialog, turn on **Automatic rotation**.
4. For **Rotation schedule**, set a schedule of **Days: 10 Days** with **Duration: 2h**. Keep **Rotate immediately** selected.

5. For **Rotation function**, do the following:
 - a. Choose **Create a rotation function**, and then for **Lambda rotation function**, enter **tutorial-single-user-rotation**.
 - b. Secrets Manager will create a Lambda rotation function named **SecretsManagertutorial-single-user-rotation**.
 - c. For **Use separate credentials**, choose **No**.
6. Choose **Save**.

Secrets Manager returns to the the secret details page. Secrets Manager uses CloudFormation to create resources such as the Lambda rotation function and an execution role that runs the Lambda function. At the top of the secret details page, you can see the status of the CloudFormation resources. When CloudFormation finishes deploying the resources, the banner changes to **Secret scheduled for rotation**. Now Secrets Manager begins the first rotation.

Step 4: Test the rotated password

After the first secret rotation, which might take a few seconds, you can check that the secret still contains valid credentials. The password in the secret has changed from the original credentials.

To retrieve the new password from the secret

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose **Secrets**, and then choose the secret **SecretsManagerTutorialAdmin-******.
3. On the **Secret details** page, scroll down and choose **Retrieve secret value**.
4. In the **Key/value** table, copy the **Secret value** for **password**.

To test the credentials

1. In MySQL Workbench, choose the connection **SecretsManagerTutorial**.
2. In the **Open SSH Connection** dialog box, for **Password**, paste the password you retrieved from the secret, and then choose **OK**.

If the credentials are valid, then MySQL Workbench opens to the design page for the database.

This shows that the secret rotation is successful. The updated password in the secret is a valid password to connect to the database. You have finished setting up automatic rotation, and the next rotation will happen in 10 days.

Step 5: Clean up resources

If you want to try another rotation strategy, *alternating users rotation*, skip cleaning up resources and go to [Set up alternating users rotation for AWS Secrets Manager \(p. 24\)](#).

Otherwise, to avoid potential charges, and to remove the EC2 instance that has access to the internet, delete the following resources you created in this tutorial:

- Amazon EC2 instance. For instructions, see [Terminate an instance](#).
- Secrets Manager endpoint. For instructions, see [Delete a VPC endpoint](#).
- Internet gateway. First [Detach an internet gateway from your VPC](#), then [Delete an internet gateway](#).
- AWS CloudFormation stack. For instructions, see [Delete a stack](#).

Next steps

- Learn how to retrieve secrets in your applications. See [Retrieve secrets](#) (p. 86).
- Learn how to create a secret with automatic rotation using AWS CloudFormation, see [AWS::SecretsManager::RotationSchedule](#) in the AWS CloudFormation User Guide.
- Learn about other rotation schedules. See [the section called “Schedule expressions”](#) (p. 120).

Set up alternating users rotation for AWS Secrets Manager

In this tutorial, you learn how to set up alternating users rotation for a secret that contains database credentials. *Alternating users rotation* is a rotation strategy where Secrets Manager clones the user and then alternates which user's credentials are updated. This strategy is a good choice if you need high availability for your secret, because one of the alternating users has current credentials to RDS while the other one is being updated. For more information, see [the section called “Rotation strategies”](#) (p. 115).

To set up alternating users rotation, you need two secrets:

- One secret with the credentials that you want to rotate.
- A second secret that has credentials for an administrator or superuser who has permissions to both change the first users's password and clone the first user.

Contents

- [Permissions](#) (p. 24)
- [Prerequisites](#) (p. 24)
- [Step 1: Create an Amazon RDS database user](#) (p. 25)
- [Step 2: Create a secret for the user credentials](#) (p. 25)
- [Step 3: Test the rotated secret](#) (p. 26)
- [Step 4: Clean up resources](#) (p. 26)
- [Next steps](#) (p. 26)

Permissions

For the tutorial prerequisites, you need administrative permissions to your AWS account. In a production setting, it is a best practice to use different roles for each of the steps. For example, a role with database admin permissions would create the Amazon RDS database, and a role with network admin permissions would set up the VPC and security groups. For the tutorial steps, we recommend you continue using the same identity.

For information about how to set up permissions in a production environment, see [Authentication and access control](#) (p. 27).

Prerequisites

The prerequisite for this tutorial is [the section called “Single user rotation”](#) (p. 17). Don't clean up the resources at the end of the first tutorial. After that tutorial, you have a realistic environment with an Amazon RDS database and a Secrets Manager secret. The secret contains admin credentials for the database, and it is set up to rotate every 10 days.

You also have a connection configured in MySQL Workbench to connect to the database with the admin credentials.

Step 1: Create an Amazon RDS database user

First, you need a user whose credentials will be stored in the secret.

To create a database user

1. In MySQL Workbench, choose the connection **SecretsManagerTutorial**.
2. In the **Query** window, enter the following commands (including a strong password) and then choose **Execute**.

```
CREATE DATABASE myDB;  
CREATE USER 'appuser'@'%' IDENTIFIED BY 'EXAMPLE-PASSWORD';  
GRANT ALL PRIVILEGES ON myDB . * TO 'appuser'@'%';
```

In the **Output** window, you see the commands are successful.

Step 2: Create a secret for the user credentials

Next, you create a secret to store the credentials of the user you just created. This is the secret you'll be rotating. You turn on automatic rotation, and to indicate the alternating users strategy, you choose a separate superuser secret that has permission to change the first user's password.

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose **Store a new secret**.
3. On the **Store a new secret** page, do the following:
 - a. For **Secret type**, choose **Credentials for Amazon RDS database**.
 - b. For **Credentials**, enter the username **newuser** and the password you entered for the database user you created using MySQL Workbench.
 - c. For **Database**, choose **secretsmanagertutorialdb**.
4. On the **Secret name and description** page, for **Secret name**, enter **SecretsManagerTutorialAppuser** and then choose **Next**.
5. On the **Secret rotation** page, do the following:
 - a. Turn on **Automatic rotation**.
 - b. For **Rotation schedule**, set a schedule of **Days: 2 Days** with **Duration: 2h**. Keep **Rotate immediately** selected.
 - c. For **Rotation function**, choose **Create a rotation function**, and then for the function name, enter **tutorial-alternating-users-rotation**.
 - d. For **Use separate credentials**, choose **Yes**, and then under **Secrets**, choose **SecretsManagerTutorialAdmin-a1b2c3d4e5f6**.
 - e. Choose **Next**.
6. On the **Review** page, choose **Store**.

Secrets Manager returns to the the secret details page. At the top of the page, you can see the rotation configuration status.

Secrets Manager uses CloudFormation to create resources such as the Lambda rotation function and an execution role that runs the Lambda function. When CloudFormation finishes, the banner changes to **Secret scheduled for rotation**. The first rotation is complete.

Step 3: Test the rotated secret

Now that the secret is rotated, you can check that the secret still contains valid credentials. The password in the secret has changed from the original credentials.

To retrieve the new password from the secret

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose **Secrets**, and then choose the secret **SecretsManagerTutorialAppuser**.
3. On the **Secret details** page, scroll down and choose **Retrieve secret value**.
4. In the **Key/value** table, copy the **Secret value** for **password**.

To test the credentials

1. In MySQL Workbench, right-click the connection **SecretsManagerTutorial** and then choose **Edit Connection**.
2. In the **Manage Server Connections** dialog box, for **Username**, enter **appuser**, and then choose **Close**.
3. Back in MySQL Workbench, choose the connection **SecretsManagerTutorial**.
4. In the **Open SSH Connection** dialog box, for **Password**, paste the password you retrieved from the secret, and then choose **OK**.

If the credentials are valid, then MySQL Workbench opens to the design page for the database.

This shows that the secret rotation is successful. The credentials in the secret have been updated and it is a valid password to connect to the database.

Step 4: Clean up resources

To avoid potential charges, and to remove the EC2 instance that has access to the internet, delete the following resources you created in this tutorial and its prerequisites:

- Amazon EC2 instance. For instructions, see [Terminate an instance](#).
- Secrets Manager secret **SecretsManagerTutorialAppuser**. See [the section called "Delete a secret" \(p. 58\)](#).
- Secrets Manager endpoint. For instructions, see [Delete a VPC endpoint](#).
- Internet gateway. First [Detach an internet gateway from your VPC](#), then [Delete an internet gateway](#).
- AWS CloudFormation stack. For instructions, see [Delete a stack](#).

Next steps

- Learn how to retrieve secrets in your applications. See [Retrieve secrets \(p. 86\)](#).
- Learn how to create a secret with automatic rotation using AWS CloudFormation, see [AWS::SecretsManager::RotationSchedule](#) in the AWS CloudFormation User Guide.
- Learn about other rotation schedules. See [the section called "Schedule expressions" \(p. 120\)](#).

Authentication and access control for AWS Secrets Manager

Secrets Manager uses [AWS Identity and Access Management \(IAM\)](#) to secure access to secrets. IAM provides authentication and access control. *Authentication* verifies the identity of individuals' requests. Secrets Manager uses a sign-in process with passwords, access keys, and multi-factor authentication (MFA) tokens to verify the identity of the users. See [Signing in to AWS](#). *Access control* ensures that only approved individuals can perform operations on AWS resources such as secrets. Secrets Manager uses policies to define who has access to which resources, and which actions the identity can take on those resources. See [Policies and permissions in IAM](#).

Secrets Manager administrator permissions

To grant Secrets Manager administrator permissions, follow the instructions at [Adding and removing IAM identity permissions](#), and attach the following policies:

- [SecretsManagerReadWrite](#)
- [IAMFullAccess](#)

We recommend you do not grant administrator permissions to end users. While this allows your users to create and manage their secrets, the permission required to enable rotation ([IAMFullAccess](#)) grants significant permissions that are not appropriate for end users.

Permissions to access secrets

By using IAM permission policies, you control which users or services have access to your secrets. A *permissions policy* describes who can perform which actions on which resources. You can:

- [the section called "Attach a permissions policy to an identity"](#) (p. 28)
- [the section called "Attach a permissions policy to a secret"](#) (p. 28)

Permissions for Lambda rotation functions

Secrets Manager uses AWS Lambda functions to [rotate secrets](#). The Lambda function must have access to the secret as well as the database or service that the secret contains credentials for. See [the section called "Permissions for rotation"](#) (p. 124).

Permissions for encryption keys

Secrets Manager uses AWS Key Management Service (AWS KMS) keys to [encrypt secrets](#). The AWS managed key `aws/secretsmanager` automatically has the correct permissions. If you use a different

KMS key, Secrets Manager needs permissions to that key. See [the section called “Permissions for the KMS key” \(p. 163\)](#).

Attach a permissions policy to an identity

You can attach permissions policies to [IAM identities: users, user groups, and roles](#). In an identity-based policy, you specify which secrets the identity can access and the actions the identity can perform on the secrets. For more information, see [Adding and removing IAM identity permissions](#).

You can grant permissions to a role that represents an application or user in another service. For example, an application running on an Amazon EC2 instance might need access to a database. You can create an IAM role attached to the EC2 instance profile and then use a permissions policy to grant the role access to the secret that contains credentials for the database. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#). Other services that you can attach roles to include [Amazon Redshift](#), [AWS Lambda](#), and [Amazon ECS](#).

You can also grant permissions to users authenticated by an identity system other than IAM. For example, you can associate IAM roles to mobile app users who sign in with Amazon Cognito. The role grants the app temporary credentials with the permissions in the role permission policy. Then you can use a permissions policy to grant the role access to the secret. For more information, see [Identity providers and federation](#).

You can use identity-based policies to:

- Grant an identity access to multiple secrets.
- Control who can create new secrets, and who can access secrets that haven't been created yet.
- Grant an IAM group access to secrets.

For more information, see [the section called “Permissions policy examples” \(p. 32\)](#).

Attach a permissions policy to a secret

In a resource-based policy, you specify who can access the secret and the actions they can perform on the secret. You can use resource-based policies to:

- Grant access to a single secret to multiple users and roles.
- Grant access to users or roles in other AWS accounts.

See [the section called “Permissions policy examples” \(p. 32\)](#).

When you attach a resource-based policy to a secret in the console, Secrets Manager uses the automated reasoning engine [Zelkova](#) and the API `ValidateResourcePolicy` to prevent you from granting a wide range of IAM principals access to your secrets. Alternatively, you can call the `PutResourcePolicy` API with the `BlockPublicPolicy` parameter from the CLI or SDK.

To view, change, or delete the resource policy for a secret (console)

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. In the secret details page for your secret, in the **Resource permissions** section, choose **Edit permissions**.
3. In the code field, do one of the following, and then choose **Save**:
 - To attach or modify a resource policy, enter the policy.
 - To delete the policy, clear the code field.

AWS CLI

To retrieve the policy attached to the secret, use [get-resource-policy](#).

Example

The following CLI command retrieves the policy attached to the secret.

```
$ aws secretsmanager get-resource-policy --secret-id production/MyAwesomeAppSecret
{
  "ARN": "arn:aws:secretsmanager:us-east-2:123456789012:secret:production/MyAwesomeAppSecret-alb2c3",
  "Name": "MyAwesomeAppSecret",
  "ResourcePolicy": "{\n  \"Version\": \"2012-10-17\", \"Statement\": [\n    {\n      \"Effect\": \"Allow\", \"Principal\": {\n        \"AWS\": \"arn:aws:iam::111122223333:root\", \"arn:aws:iam::444455556666:root\"\n      }, \"Action\": [\n        \"secretsmanager:GetSecret\", \"secretsmanager:GetSecretValue\" \n      ], \"Resource\": \"*\n    }\n  ]\n}"
```

To delete the policy attached to the secret, use [delete-resource-policy](#).

Example

The following CLI command deletes the policy attached to the secret.

```
$ aws secretsmanager delete-resource-policy --secret-id production/MyAwesomeAppSecret
{
  "ARN": "arn:aws:secretsmanager:us-east-2:123456789012:secret:production/MyAwesomeAppSecret-alb2c3",
  "Name": "production/MyAwesomeAppSecret"
}
```

To attach a policy for the secret, use [put-resource-policy](#). If there is already a policy attached, the command first removes it, and then attaches the new policy. The policy must be formatted as JSON structured text. See [JSON policy document structure](#).

Example

The following CLI command attaches the resource-based policy attached to the secret. The policy is defined in the file `secretpolicy.json`. Use the [the section called “Permissions policy examples” \(p. 32\)](#) to get started writing your policy.

```
$ aws secretsmanager put-resource-policy --secret-id production/MyAwesomeAppSecret --resource-policy file://secretpolicy.json
{
  "ARN": "arn:aws:secretsmanager:us-east-2:123456789012:secret:production/MyAwesomeAppSecret-alb2c3",
  "Name": "MyAwesomeAppSecret"
}
```

AWS SDK

To retrieve the policy attached to a secret, use [GetResourcePolicy](#).

To delete a policy attached to a secret, use [DeleteResourcePolicy](#).

To attach a policy to a secret, use [PutResourcePolicy](#). If there is already a policy attached, the command first removes it, and then attaches the new policy. The policy must be formatted as JSON

structured text. See [JSON policy document structure](#). Use the [the section called "Permissions policy examples"](#) (p. 32) to get started writing your policy.

For more information, see [the section called "AWS SDKs"](#) (p. 8).

AWS managed policies available for use with AWS Secrets Manager

AWS addresses many common use cases by providing *managed policies*, standalone IAM policies created and administered by AWS. Managed policies grant permissions for common use cases so you can avoid investigating the necessary permissions. You can attach or remove an AWS managed policy to users in your account, but you can't modify or delete the policy. For more information, see [AWS managed policies](#) in the *IAM User Guide*.

The following table describes the AWS managed policy you can use to help manage access to Secrets Manager secrets.

Policy Name	Description	ARN
SecretsManagerReadOnlyAccess	Provides access to Secrets Manager operations. The policy doesn't allow the identity to configure rotation because rotation requires IAM permissions to create roles. If you need to enable rotation and configure Lambda rotation functions, you need to also assign the IAMFullAccess managed policy. See the section called "Permissions for rotation" (p. 124).	arn:aws:iam::aws:policy/SecretsManagerReadWrite

Determine who has permissions to your secrets

By default, IAM identities don't have permission to access secrets. When authorizing access to a secret, Secrets Manager evaluates the resource-based policy attached to the secret and all identity-based policies attached to the IAM user or role sending the request. To do this, Secrets Manager uses a process similar to the one described in [Determining whether a request is allowed or denied](#) in the *IAM User Guide*.

When multiple policies apply to a request, Secrets Manager uses a hierarchy to control permissions:

1. If a statement in any policy with an explicit deny matches the request action and resource:

The explicit deny overrides everything else and blocks the action.
2. If there is no explicit deny, but a statement with an explicit allow matches the request action and resource:

The explicit allow grants the action in the request access to the resources in the statement.

If the identity and the secret are in two different accounts, there must be an allow in both the resource policy for the secret and the policy attached to the identity, otherwise AWS denies the request. For more information, see [Cross-account access](#) (p. 31).
3. If there is no statement with an explicit allow that matches the request action and resource:

AWS denies the request by default, which is called an *implicit deny*.

To view the resource-based policy for a secret

- Do one of the following:
 - Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>. In the secret details page for your secret, in the **Resource permissions** section, choose **Edit permissions**.
 - Use the AWS CLI or AWS SDK to call `GetResourcePolicy`.

To determine who has access through identity-based policies

- Use the IAM policy simulator. See [Testing IAM policies with the IAM policy simulator](#)

Permissions for users in a different account

To allow users in one account to access secrets in another account (*cross-account access*), you must allow access both in a resource policy and in an identity policy. This is different than granting access to identities in the same account as the secret.

You must also allow the identity to use the KMS key that the secret is encrypted with. This is because you can't use the AWS managed key (`aws/secretsmanager`) for cross-account access. Instead, you must encrypt your secret with a KMS key that you create, and then attach a key policy to it. There is a charge for creating KMS keys. To change the encryption key for a secret, see [the section called "Modify a secret" \(p. 55\)](#).

The following example policies assume you have a secret and encryption key in *Account1*, and an identity in *Account2* that you want to allow to access the secret value.

Step 1: Attach a resource policy to the secret in *Account1*

- The following policy allows *ApplicationRole* in *Account2* to access the secret in *Account1*. To use this policy, see [the section called "Attach a permissions policy to a secret" \(p. 28\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::Account2:role/ApplicationRole"
      },
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "*"
    }
  ]
}
```

Step 2: Add a statement to the key policy for the KMS key in *Account1*

- The following key policy statement allows *ApplicationRole* in *Account2* to use the KMS key in *Account1* to decrypt the secret in *Account1*. To use this statement, add it to the key policy for your KMS key. For more information, see [Changing a key policy](#).

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::Account2:role/ApplicationRole"
  }
}
```

```
    },
    "Action": [
      "kms:Decrypt",
      "kms:DescribeKey"
    ],
    "Resource": "*"
  }
}
```

Step 3: Attach an identity policy to the identity in *Account2*

- The following policy allows *ApplicationRole* in *Account2* to access the secret in *Account1* and decrypt the secret value by using the encryption key which is also in *Account1*. To use this policy, see [the section called “Attach a permissions policy to an identity” \(p. 28\)](#). You can find the ARN for your secret in the Secrets Manager console on the secret details page under **Secret ARN**. Alternatively, you can call [DescribeSecret](#).

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect": "Allow",
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "SecretARN"
    },
    {
      "Effect": "Allow",
      "Action": "kms:Decrypt",
      "Resource": "arn:aws:kms:Region:Account1:key/EncryptionKey"
    }
  ]
}
```

Permissions policy examples

A permissions policy is JSON structured text. See [JSON policy document structure](#).

Permissions policies that you attach to resources and identities are very similar. Some elements you include in a policy for access to secrets include:

- Principal:** who to grant access to. See [Specifying a principal](#) in the *IAM User Guide*. When you attach a policy to an identity, you don't include a **Principal** element in the policy.
- Action:** what they can do. See [the section called “Secrets Manager actions” \(p. 38\)](#).
- Resource:** which secrets they can access. See [the section called “Secrets Manager resources” \(p. 45\)](#).

The wildcard character (*) has different meaning depending on what you attach the policy to:

- In a policy attached to a secret, * means the policy applies to this secret.
- In a policy attached to an identity, * means the policy applies to all resources, including secrets, in the account.

To attach a policy to a secret, see [the section called “Attach a permissions policy to a secret” \(p. 28\)](#).

To attach a policy to an identity, see [the section called “Attach a permissions policy to an identity” \(p. 28\)](#).

Topics

- [Example: Permission to retrieve secret values \(p. 33\)](#)
- [Example: Wildcards \(p. 34\)](#)
- [Example: Permission to create secrets \(p. 35\)](#)
- [Example: Permissions and VPCs \(p. 35\)](#)
- [Example: Control access to secrets using tags \(p. 36\)](#)
- [Example: Limit access to identities with tags that match secrets' tags \(p. 37\)](#)
- [Example: Service principal \(p. 37\)](#)

Example: Permission to retrieve secret values

To grant permission to retrieve secret values, you can attach policies to secrets or identities. For help determining which type of policy to use, see [Identity-based policies and resource-based policies](#). For information about how to attach a policy, see [the section called "Attach a permissions policy to a secret" \(p. 28\)](#) and [the section called "Attach a permissions policy to an identity" \(p. 28\)](#).

The following examples show two different ways to grant access to a secret. The first example is a resource-based policy that you can attach to a secret. This example is useful when you want to grant access to a single secret to multiple users or roles. The second example is an identity-based policy that you can attach to a user or role in IAM. This example is useful when you want to grant access to an IAM group.

Example Read one secret (attach to a secret)

You can grant access to a secret by attaching the following policy to the secret. To use this policy, see [the section called "Attach a permissions policy to a secret" \(p. 28\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountId:role/EC2RoleToAccessSecrets"
      },
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "*"
    }
  ]
}
```

Example Read one secret (attach to an identity)

You can grant access to a secret by attaching the following policy to an identity. To use this policy, see [the section called "Attach a permissions policy to an identity" \(p. 28\)](#). If you attach this policy to the role *EC2RoleToAccessSecrets*, it grants the same permissions as the previous policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "SecretARN"
    }
  ]
}
```


Example: Wildcards

You can use wildcards to include a set of values in a policy element.

Example Access all secrets in a path (attach to identity)

The following policy grants access to retrieve all secrets with a name beginning with `TestEnv/`. To use this policy, see [the section called "Attach a permissions policy to an identity" \(p. 28\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "secretsmanager:GetSecretValue",
    "Resource": "arn:aws:secretsmanager:Region:AccountId:secret:TestEnv/*"
  }
}
```

Example Access metadata on all secrets (attach to identity)

The following policy grants `DescribeSecret` and permissions beginning with `List:ListSecrets` and `ListSecretVersionIds`. To use this policy, see [the section called "Attach a permissions policy to an identity" \(p. 28\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "secretsmanager:DescribeSecret",
      "secretsmanager:List*"
    ],
    "Resource": "*"
  }
}
```

Example Match secret name (attach to identity)

The following policy grants all Secrets Manager permissions for a secret by name. To use this policy, see [the section called "Attach a permissions policy to an identity" \(p. 28\)](#).

To match a secret name, you create the ARN for the secret by putting together the Region, Account ID, secret name, and the wildcard (?) to match individual random characters. Secrets Manager appends six random characters to secret names as part of their ARN, so you can use this wildcard to match those characters. If you use the syntax `another_secret_name-*`, Secrets Manager matches not only the intended secret with the 6 random characters, but also matches `another_secret_name-<anything-here>alb2c3`.

Because you can predict all of the parts of the ARN of a secret except the 6 random characters, using the wildcard character `'?????'` syntax enables you to securely grant permissions to a secret that doesn't yet exist. Be aware, however, if you delete the secret and recreate it with the same name, the user automatically receives permission to the new secret, even though the 6 characters changed.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "secretsmanager:*",
      "Resource": [
```

```
    "arn:aws:secretsmanager:Region:AccountId:secret:a_specific_secret_name-a1b2c3",  
    "arn:aws:secretsmanager:Region:AccountId:secret:another_secret_name-?????"  
  ]  
}  
]
```

Example: Permission to create secrets

To grant a user permissions to create a secret, we recommend you attach a permissions policy to an IAM group the user belongs to. See [IAM user groups](#).

Example Create secrets (attach to identity)

The following policy grants permission to create secrets and view a list of secrets. To use this policy, see [the section called "Attach a permissions policy to an identity"](#) (p. 28).

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "secretsmanager:CreateSecret",  
        "secretsmanager:ListSecrets"  
      ],  
      "Resource": "*"   
    }  
  ]  
}
```

Example: Permissions and VPCs

If you need to access Secrets Manager from within a VPC, you can make sure that requests to Secrets Manager come from the VPC by including a condition in your permissions policies. For more information, see [VPC endpoint conditions](#) (p. 48) and [VPC endpoint](#) (p. 137).

Make sure that requests to access the secret from other AWS services also come from the VPC, otherwise this policy will deny them access.

Example Require requests to come through a VPC endpoint (attach to secret)

The following policy allows a user to perform Secrets Manager operations only when the request comes through the VPC endpoint `vpce-1234a5678b9012c`. To use this policy, see [the section called "Attach a permissions policy to a secret"](#) (p. 28).

```
{  
  "Id": "example-policy-1",  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "RestrictGetSecretValueoperation",  
      "Effect": "Deny",  
      "Principal": "*",  
      "Action": "secretsmanager:GetSecretValue",  
      "Resource": "*",  
      "Condition": {  
        "StringNotEquals": {  
          "aws:sourceVpce": "vpce-1234a5678b9012c"  
        }  
      }  
    }  
  ]  
}
```

```
}  
}  
]  
}
```

Example Require requests to come from a VPC (attach to secret)

The following policy allows commands to create and manage secrets only when they come from `vpc-12345678`. In addition, the policy allows operations that use access the secret encrypted value only when the requests come from `vpc-2b2b2b2b`. You might use a policy like this one if you run an application in one VPC, but you use a second, isolated VPC for management functions. To use this policy, see [the section called "Attach a permissions policy to a secret"](#) (p. 28).

```
{  
  "Id": "example-policy-2",  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "AllowAdministrativeActionsfromONLYvpc-12345678",  
      "Effect": "Deny",  
      "Principal": "*",  
      "Action": [  
        "secretsmanager:Create*",  
        "secretsmanager:Put*",  
        "secretsmanager:Update*",  
        "secretsmanager:Delete*",  
        "secretsmanager:Restore*",  
        "secretsmanager:RotateSecret",  
        "secretsmanager:CancelRotate*",  
        "secretsmanager:TagResource",  
        "secretsmanager:UntagResource"  
      ],  
      "Resource": "*",  
      "Condition": {  
        "StringNotEquals": {  
          "aws:sourceVpc": "vpc-12345678"  
        }  
      }  
    },  
    {  
      "Sid": "AllowSecretValueAccessfromONLYvpc-2b2b2b2b",  
      "Effect": "Deny",  
      "Principal": "*",  
      "Action": [  
        "secretsmanager:GetSecretValue"  
      ],  
      "Resource": "*",  
      "Condition": {  
        "StringNotEquals": {  
          "aws:sourceVpc": "vpc-2b2b2b2b"  
        }  
      }  
    }  
  ]  
}
```

Example: Control access to secrets using tags

You can use tags to control access to your secrets. Using tags to control permissions is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome. One strategy is to attach tags to secrets and then grant permissions to an identity when a secret has a specific tag.

Example Allow access to secrets with a specific tag (attach to an identity)

The following policy allows `DescribeSecret` on secrets with a tag with the key `"ServerName"` and the value `"ServerABC"`. To use this policy, see [the section called "Attach a permissions policy to an identity" \(p. 28\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "secretsmanager:DescribeSecret",
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "secretsmanager:ResourceTag/ServerName": "ServerABC"
      }
    }
  }
}
```

Example: Limit access to identities with tags that match secrets' tags

One strategy is to attach tags to both secrets and IAM identities. Then you create permissions policies to allow operations when the identity's tag matches the secret's tag. For a complete tutorial, see [Define permissions to access secrets based on tags](#).

Using tags to control permissions is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome. For more information, see [What is ABAC for AWS?](#)

Example Allow access to roles that have the same tags as secrets (attach to a secret)

The following policy grants `GetSecretValue` to account `123456789012` only if the tag `AccessProject` has the same value for the secret and the role. To use this policy, see [the section called "Attach a permissions policy to a secret" \(p. 28\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {
      "AWS": "123456789012"
    },
    "Condition": {
      "StringEquals": {
        "aws:ResourceTag/AccessProject": "${ aws:PrincipalTag/AccessProject }"
      }
    },
    "Action": "secretsmanager:GetSecretValue",
    "Resource": "*"
  }
}
```

Example: Service principal

If the resource policy attached to your secret includes an [AWS service principal](#), we recommend that you use the `aws:SourceArn` and `aws:SourceAccount` global condition keys. The ARN and account values are

included in the authorization context only when a request comes to Secrets Manager from another AWS service. This combination of conditions avoids a potential [confused deputy scenario](#).

If a resource ARN includes characters that are not permitted in a resource policy, you cannot use that resource ARN in the value of the `aws:SourceArn` condition key. Instead, use the `aws:SourceAccount` condition key. For more information, see [IAM requirements](#).

Service principals are not typically used as principals in a policy attached to a secret, but some AWS services require it. For information about resource policies that a service requires you to attach to a secret, see the service's documentation.

Example Allow a service to access a secret using a service principal (attach to a secret)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "service-name.amazonaws.com"
        ]
      },
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "*",
      "Condition": {
        "ArnLike": {
          "aws:sourceArn": "arn:aws:service-name::123456789012:*"
        },
        "StringEquals": {
          "aws:sourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Permissions reference for Secrets Manager

To see the elements that make up a permissions policy, see [JSON policy document structure](#) and [IAM JSON policy elements reference](#).

To get started writing your own permissions policy, see [the section called "Permissions policy examples" \(p. 32\)](#).

Secrets Manager actions

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
CancelRotateSecret	Grants permission to cancel an in-progress secret rotation	Write	Secret* (p. 45)		
				<code>secretsmanager:SecretId</code> (p. 47)	
				<code>secretsmanager:resource/AllowRotationLambdaArn</code> (p. 47)	

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
				secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	
CreateSecret	Grants permission to create a secret that stores encrypted data that can be queried and rotated	Write	Secret* (p. 45)	secretsmanager:Name (p. 46) secretsmanager:Description (p. 46) secretsmanager:KmsKeyId (p. 46) aws:RequestTag/\${TagKey} (p. 46) aws:ResourceTag/\${TagKey} (p. 46) aws:TagKeys (p. 46) secretsmanager:ResourceTag/tag-key (p. 46) secretsmanager:AddReplicaRegions (p. 46) secretsmanager:ForceOverwriteReplica	
DeleteResourcePolicy	Grants permission to delete the resource policy attached to a secret	Permissions management	Secret* (p. 45)	secretsmanager:SecretId (p. 47) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	
DeleteSecret	Grants permission to delete a secret	Write	Secret* (p. 45)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
				secretsmanager:SecretId (p. 47) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:RecoveryWindowInDays secretsmanager:ForceDeleteWithoutRecovery secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	
DescribeSecret	Grants permission to retrieve the metadata about a secret, but not the encrypted data	Read	Secret* (p. 45)		
				secretsmanager:SecretId (p. 47) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	
GetRandomPassword	Grants permission to generate a random string for use in password creation	Read			
GetResourcePolicy	Grants permission to get the resource policy attached to a secret	Read	Secret* (p. 45)		
				secretsmanager:SecretId (p. 47) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	
GetSecretValue	Grants permission to retrieve and decrypt the encrypted data	Read	Secret* (p. 45)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
				secretsmanager:SecretId (p. 47) secretsmanager:VersionId (p. 47) secretsmanager:VersionStage (p. 47) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	
ListSecretVersions	Grants permission to list the available versions of a secret	Read	Secret* (p. 45)	secretsmanager:SecretId (p. 47) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	
ListSecrets	Grants permission to list the available secrets	List			
PutResourcePolicy	Grants permission to attach a resource policy to a secret	Permissions management	Secret* (p. 45)	secretsmanager:SecretId (p. 47) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:BlockPublicPolicy (p. 4) secretsmanager:SecretPrimaryRegion	
PutSecretValue		Write	Secret* (p. 45)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to create a new version of the secret with new encrypted data			secretsmanager:SecretId (p. 47) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	
RemoveRegionsFromReplication	Grants permission to remove regions from replication	Write	Secret* (p. 45)	secretsmanager:SecretId (p. 47) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	
ReplicateSecretToExistingRegions	Grants permission to convert an existing secret to a multi-Region secret and begin replicating the secret to a list of new regions	Write	Secret* (p. 45)	secretsmanager:SecretId (p. 47) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion secretsmanager:AddReplicaRegions (p. 47) secretsmanager:ForceOverwriteReplica	
RestoreSecret	Grants permission to cancel deletion of a secret	Write	Secret* (p. 45)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
				secretsmanager:SecretId (p. 47) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	
RotateSecret	Grants permission to start rotation of a secret	Write	Secret* (p. 45)		
				secretsmanager:SecretId (p. 47) secretsmanager:RotationLambdaARN secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion secretsmanager:ModifyRotationRules secretsmanager:RotateImmediately (p. 47)	
StopReplicationToReplica	Grants permission to remove the secret from replication and promote the secret to a regional secret in the replica Region	Write	Secret* (p. 45)		
				secretsmanager:SecretId (p. 47) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	
TagResource	Grants permission to add tags to a secret	Tagging	Secret* (p. 45)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
				secretsmanager:SecretId (p. 47) aws:RequestTag/\${TagKey} (p. 46) aws:TagKeys (p. 46) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	
UntagResource	Grants permission to remove tags from a secret	Tagging	Secret* (p. 45)		
				secretsmanager:SecretId (p. 47) aws:TagKeys (p. 46) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	
UpdateSecret	Grants permission to update a secret with new metadata or with a new version of the encrypted data	Write	Secret* (p. 45)		
				secretsmanager:SecretId (p. 47) secretsmanager:Description (p. 46) secretsmanager:KmsKeyId (p. 46) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
UpdateSecretVersionStage	Grants permission to move a stage from one secret to another	Write	Secret* (p. 45)	secretsmanager:SecretId (p. 47) secretsmanager:VersionStage (p. 47) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	
ValidateResourcePolicy	Grants permission to validate a resource policy before attaching policy	Permissions management	Secret* (p. 45)	secretsmanager:SecretId (p. 47) secretsmanager:resource/AllowRotationLambdaArn (p. 47) secretsmanager:ResourceTag/tag-key (p. 46) aws:ResourceTag/\${TagKey} (p. 46) secretsmanager:SecretPrimaryRegion	

Secrets Manager resources

Resource types	ARN	Condition keys
Secret	arn:\${Partition}:secretsmanager:\${Region}:\${Account}:secret:\${SecretId}	aws:RequestTag/\${TagKey} (p. 46) aws:ResourceTag/\${TagKey} (p. 46) aws:TagKeys (p. 46) secretsmanager:ResourceTag/tag-key (p. 46) secretsmanager:resource/AllowRotationLambdaArn (p. 47)

Secrets Manager constructs the last part of the secret ARN by appending a dash and six random alphanumeric characters at the end of the secret name. If you delete a secret and then recreate another with the same name, this formatting helps ensure that individuals with permissions to the original secret don't automatically get access to the new secret because Secrets Manager generates six new random characters.

You can find the ARN for a secret in the Secrets Manager console on the secret details page or by calling [DescribeSecret](#).

Condition keys

If you include string conditions from the following table in your permissions policy, callers to Secrets Manager must pass the matching parameter or they are denied access. To avoid denying callers for a missing parameter, add `IfExists` to the end of the condition operator name, for example `StringLikeIfExists`. For more information, see [IAM JSON policy elements: Condition operators](#).

Condition keys	Description	Type
aws:RequestTag/\${TagKey}	Filters access by a key that is present in the request the user makes to the Secrets Manager service	String
aws:ResourceTag/\${TagKey}	Filters access by the tags associated with the resource	String
aws:TagKeys	Filters access by the list of all the tag key names present in the request the user makes to the Secrets Manager service	ArrayOfString
secretsmanager:AddSecretRegions	Filters access by the list of Regions in which to replicate the secret	ArrayOfString
secretsmanager:BlockAWSServiceAccess	Filters access by whether the resource policy blocks broad AWS service access	Bool
secretsmanager:Description	Filters access by the description text in the request	String
secretsmanager:ForceDeleteWithoutRecovery	Filters access by whether the secret is to be deleted immediately without any recovery window	Bool
secretsmanager:ForceOverwriteInDestinationRegion	Filters access by whether to overwrite a secret with the same name in the destination Region	Bool
secretsmanager:KmsKeyId	Filters access by the ARN of the KMS key in the request	String
secretsmanager:ModifyRotationRules	Filters access by whether the rotation rules of the secret are to be modified	Bool
secretsmanager:Name	Filters access by the friendly name of the secret in the request	String
secretsmanager:RecoveryWindowInDays	Filters access by the number of days that Secrets Manager waits before it can delete the secret	Numeric
secretsmanager:ResourceTag/tag-key	Filters access by a tag key and value pair	String

Condition keys	Description	Type
secretsmanager:RotateImmediately	Filters access by whether the secret is to be rotated immediately	Bool
secretsmanager:RotationLambdaArn	Filters access by the ARN of the rotation Lambda function in the request	ARN
secretsmanager:SecretId	Filters access by the SecretId value in the request	ARN
secretsmanager:SecretPrimaryRegion	Filters access by primary region in which the secret is created	String
secretsmanager:SecretVersion	Filters access by the unique identifier of the version of the secret in the request	String
secretsmanager:VersionStage	Filters access by the list of version stages in the request	String
secretsmanager:resource/AllowRotationLambdaArn	Filters access by the ARN of the rotation Lambda function associated with the secret	ARN

Block broad access to secrets with BlockPublicPolicy condition

In identity policies that allow the action `PutResourcePolicy`, we recommend you use `BlockPublicPolicy: true`. This condition means that users can only attach a resource policy to a secret if the policy doesn't allow broad access.

Secrets Manager uses Zelkova automated reasoning to analyze resource policies for broad access. For more information about Zelkova, see [How AWS uses automated reasoning to help you achieve security at scale](#) on the AWS Security Blog.

The following example shows how to use `BlockPublicPolicy`.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "secretsmanager:PutResourcePolicy",
    "Resource": "SecretId",
    "Condition": {
      "Bool": {
        "secretsmanager:BlockPublicPolicy": "true"
      }
    }
  }
}
```

IP address conditions

Use caution when you specify the [IP address condition operators](#) or the `aws:SourceIp` condition key in a policy statement that allows or denies access to Secrets Manager. For example, if you attach a policy that restricts AWS actions to requests from your corporate network IP address range to a secret,

then your requests as an IAM user invoking the request from the corporate network work as expected. However, if you enable other services to access the secret on your behalf, such as when you enable rotation with a Lambda function, that function calls the Secrets Manager operations from an AWS-internal address space. Requests impacted by the policy with the IP address filter fail.

Also, the `aws:sourceIP` condition key is less effective when the request comes from an Amazon VPC endpoint. To restrict requests to a specific VPC endpoint, use [the section called “VPC endpoint conditions” \(p. 48\)](#).

VPC endpoint conditions

To allow or deny access to requests from a particular VPC or VPC endpoint, use `aws:SourceVpc` to limit access to requests from the specified VPC or `aws:SourceVpce` to limit access to requests from the specified VPC endpoint. See [the section called “Example: Permissions and VPCs” \(p. 35\)](#).

- `aws:SourceVpc` limits access to requests from the specified VPC.
- `aws:SourceVpce` limits access to requests from the specified VPC endpoint.

If you use these condition keys in a resource policy statement that allows or denies access to Secrets Manager secrets, you can inadvertently deny access to services that use Secrets Manager to access secrets on your behalf. Only some AWS services can run with an endpoint within your VPC. If you restrict requests for a secret to a VPC or VPC endpoint, then calls to Secrets Manager from a service not configured for the service can fail.

See [VPC endpoint \(p. 137\)](#).

Create and manage secrets with AWS Secrets Manager

A *secret* can be a password, a set of credentials such as a user name and password, an OAuth token, or other secret information that you store in an encrypted form in Secrets Manager.

Topics

- [Create a database secret \(p. 49\)](#)
- [Create a secret \(p. 53\)](#)
- [Modify a secret \(p. 55\)](#)
- [Find secrets in AWS Secrets Manager \(p. 57\)](#)
- [Delete a secret \(p. 58\)](#)
- [Restore a secret \(p. 60\)](#)
- [Replicate an AWS Secrets Manager secret to other AWS Regions \(p. 61\)](#)
- [Promote a replica secret to a standalone secret \(p. 62\)](#)
- [Tag secrets \(p. 63\)](#)
- [Create secrets in AWS CloudFormation \(p. 64\)](#)

Create a database secret

To store credentials for Amazon Relational Database Service (Amazon RDS), Amazon Aurora, Amazon Redshift, or Amazon DocumentDB, follow these steps. When you use the AWS CLI or one of the SDKs to store the secret, you must provide the secret in the [JSON structure of a database secret \(p. 51\)](#). When you use the console to store a database secret, Secrets Manager automatically creates it in the correct JSON structure.

When you store database credentials for a source database that is replicated to other Regions, the secret contains connection information for the source database. If you then replicate the secret, the replicas are copies of the source secret and contain the same connection information. You can add additional key/value pairs to the secret for regional connection information.

To create a secret, you need the permissions granted by the **SecretsManagerReadWrite** [AWS managed policy \(p. 30\)](#).

To create a secret (console)

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose **Store a new secret**.
3. On the **Store a new secret** page, do the following:
 - a. For **Secret type**, choose the type of database credentials to store:
 - **Amazon RDS database** (includes Aurora)
 - **Amazon DocumentDB database**
 - **Amazon Redshift cluster**

- b. For **Credentials**, enter the credentials for the database.
- c. For **Encryption key**, choose the AWS KMS key that Secrets Manager uses to encrypt the secret value:
 - For most cases, choose **aws/secretsmanager** to use the AWS managed key for Secrets Manager. There is no cost for using this key.
 - If you need to access the secret from another AWS account, choose a customer managed key from the list or choose **Add new key** to create one. You will be charged for KMS keys that you create.

You must have the following permissions to the key: `kms:Encrypt`, `kms:Decrypt`, and `kms:GenerateDataKey`. For more information about cross-account access, see [the section called "Cross-account access" \(p. 31\)](#).
- d. For **Database**, choose your database.
- e. Choose **Next**.
4. On the **Secret name and description** page, do the following:
 - a. Enter a descriptive **Secret name** and **Description**.
 - b. (Optional) In the **Tags** section, add tags to your secret. For tagging strategies, see [the section called "Tag secrets" \(p. 63\)](#). Don't store sensitive information in tags because they aren't encrypted.
 - c. (Optional) In **Resource permissions**, to add a resource policy to your secret, choose **Edit permissions**. For more information, see [the section called "Attach a permissions policy to a secret" \(p. 28\)](#).
 - d. (Optional) In **Replicate secret**, to replicate your secret to another AWS Region, choose **Replicate secret**. You can replicate your secret now or come back and replicate it later. For more information, see [Replicate a secret to other Regions \(p. 61\)](#).
 - e. Choose **Next**.
5. (Optional) On the **Secret rotation** page, you can turn on automatic rotation. You can also keep rotation off for now and then turn it on later. For more information, see [Rotate secrets \(p. 115\)](#). Choose **Next**.
6. On the **Review** page, review your secret details, and then choose **Store**.

AWS CLI

To create a secret by using the AWS CLI, first create a JSON file that contains your secret. For Secrets Manager to be able to rotate the secret, you must make sure the JSON matches the [JSON structure of a database secret \(p. 51\)](#). For more information, see [the section called "Rotate DB credentials" \(p. 117\)](#).

Then use the `create-secret` operation to store the secret in Secrets Manager.

To create a secret

1. Create your secret in a file, for example a JSON file named `mycreds.json`.

```
{
  "engine": "mysql",
  "host": "<instance host name/resolvable DNS name>",
  "username": "<username>",
  "password": "<password>",
  "dbname": "<database name. If not specified, defaults to None>",
  "port": "<TCP port number. If not specified, defaults to 3306>"
}
```

2. In the AWS CLI, use the following command.

```
$ aws secretsmanager create-secret --name MySecret --secret-string file://mycreds.json
```

The following shows the output.

```
{
  "SecretARN": "arn:aws:secretsmanager:Region:AccountId:secret:MySecret-a1b2c3",
  "SecretName": "MySecret",
  "SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE"
}
```

AWS SDK

To create a secret by using one of the AWS SDKs, use the [CreateSecret](#) action. For more information, see [the section called “AWS SDKs”](#) (p. 8).

JSON structure of AWS Secrets Manager database credential secrets

If you want to turn on automatic rotation in Secrets Manager for a database credential secret, the secret must be in the correct JSON structure. During rotation, Secrets Manager uses the information in the secret to connect to the database and update the credentials there. When you use the AWS CLI or one of the SDKs to store a secret, you must provide the secret in one of the following structures. When you use the console to store a database secret, Secrets Manager automatically creates it in the correct JSON structure.

You can add more key/value pairs to a database secret, for example to contain connection information for replica databases in other Regions.

Topics

- [Amazon RDS MariaDB secret structure](#) (p. 51)
- [Amazon RDS MySQL secret structure](#) (p. 52)
- [Amazon RDS Oracle secret structure](#) (p. 52)
- [Amazon RDS PostgreSQL secret structure](#) (p. 52)
- [Amazon RDS Microsoft SQLServer secret structure](#) (p. 52)
- [Amazon DocumentDB secret structure](#) (p. 53)
- [Amazon Redshift secret structure](#) (p. 53)

Amazon RDS MariaDB secret structure

```
{
  "engine": "mariadb",
  "host": "<instance host name/resolvable DNS name>",
  "username": "<username>",
  "password": "<password>",
  "dbname": "<database name. If not specified, defaults to None>",
  "port": "<TCP port number. If not specified, defaults to 3306>"
}
```

To use the [the section called “Alternating users”](#) (p. 116), also include the name-value pair:

```
"masterarn": "<the ARN of the elevated secret>"
```

Amazon RDS MySQL secret structure

```
{
  "engine": "mysql",
  "host": "<instance host name/resolvable DNS name>",
  "username": "<username>",
  "password": "<password>",
  "dbname": "<database name. If not specified, defaults to None>",
  "port": "<TCP port number. If not specified, defaults to 3306>"
}
```

To use the [the section called “Alternating users”](#) (p. 116), also include the name-value pair:

```
"masterarn": "<the ARN of the elevated secret>"
```

Amazon RDS Oracle secret structure

```
{
  "engine": "oracle",
  "host": "<required: instance host name/resolvable DNS name>",
  "username": "<required: username>",
  "password": "<required: password>",
  "dbname": "<required: database name>",
  "port": "<optional: TCP port number. If not specified, defaults to 1521>"
}
```

To use the [the section called “Alternating users”](#) (p. 116), also include the name-value pair:

```
"masterarn": "<the ARN of the elevated secret>"
```

Amazon RDS PostgreSQL secret structure

```
{
  "engine": "postgres",
  "host": "<instance host name/resolvable DNS name>",
  "username": "<username>",
  "password": "<password>",
  "dbname": "<database name. If not specified, defaults to 'postgres'>",
  "port": "<TCP port number. If not specified, defaults to 5432>"
}
```

To use the [the section called “Alternating users”](#) (p. 116), also include the name-value pair:

```
"masterarn": "<the ARN of the elevated secret>"
```

Amazon RDS Microsoft SQLServer secret structure

```
{
  "engine": "sqlserver",
  "host": "<instance host name/resolvable DNS name>",

```

```
"username": "<username>",  
"password": "<password>",  
"dbname": "<database name. If not specified, defaults to 'master'>",  
"port": "<TCP port number. If not specified, defaults to 1433>"  
}
```

To use the [the section called “Alternating users” \(p. 116\)](#), also include the name-value pair:

```
"masterarn": "<the ARN of the elevated secret>"
```

Amazon DocumentDB secret structure

```
{  
  "engine": "mongo",  
  "host": "<instance host name/resolvable DNS name>",  
  "username": "<username>",  
  "password": "<password>",  
  "dbname": "<database name. If not specified, defaults to None>",  
  "port": "<TCP port number. If not specified, defaults to 27017>"  
}
```

To use the [the section called “Alternating users” \(p. 116\)](#), also include the name-value pair:

```
"masterarn": "<the ARN of the elevated secret>"
```

Amazon Redshift secret structure

```
{  
  "engine": "redshift",  
  "host": "<instance host name/resolvable DNS name>",  
  "username": "<username>",  
  "password": "<password>",  
  "dbname": "<database name. If not specified, defaults to None>",  
  "port": "<TCP port number. If not specified, defaults to 5439>"  
}
```

To use the [the section called “Alternating users” \(p. 116\)](#), also include the name-value pair:

```
"masterarn": "<the ARN of the elevated secret>"
```

Create a secret

To store API keys, access tokens, credentials that aren't for databases, and other secrets in Secrets Manager, follow these steps.

To create a secret, you need the permissions granted by the **SecretsManagerReadWrite** [AWS managed policy \(p. 30\)](#).

To create a secret (console)

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose **Store a new secret**.

3. On the **Store a new secret** page, do the following:
 - a. For **Secret type**, choose **Other type of secret**.
 - b. In **Key/value pairs**, either enter your secret in **Key/value** pairs, or choose the **Plaintext** tab and enter the secret in any format. We recommend JSON. You can store up to 65536 bytes in the secret.
 - c. For **Encryption key**, choose the AWS KMS key that Secrets Manager uses to encrypt the secret value:
 - For most cases, choose **aws/secretsmanager** to use the AWS managed key for Secrets Manager. There is no cost for using this key.
 - If you need to access the secret from another AWS account, choose a customer managed key from the list or choose **Add new key** to create one. You will be charged for KMS keys that you create.

You must have the following permissions to the key: `kms:Encrypt`, `kms:Decrypt`, and `kms:GenerateDataKey`. For more information about cross-account access, see [the section called "Cross-account access" \(p. 31\)](#).
 - d. Choose **Next**.
4. On the **Secret name and description** page, do the following:
 - a. Enter a descriptive **Secret name** and **Description**.
 - b. (Optional) In the **Tags** section, add tags to your secret. For tagging strategies, see [the section called "Tag secrets" \(p. 63\)](#). Don't store sensitive information in tags because they aren't encrypted.
 - c. (Optional) In **Resource permissions**, to add a resource policy to your secret, choose **Edit permissions**. For more information, see [the section called "Attach a permissions policy to a secret" \(p. 28\)](#).
 - d. (Optional) In **Replicate secret**, to replicate your secret to another AWS Region, choose **Replicate secret**. You can replicate your secret now or come back and replicate it later. For more information, see [Replicate a secret to other Regions \(p. 61\)](#).
 - e. Choose **Next**.
5. (Optional) On the **Secret rotation** page, you can turn on automatic rotation. You can also keep rotation off for now and then turn it on later. For more information, see [Rotate secrets \(p. 115\)](#). Choose **Next**.
6. On the **Review** page, review your secret details, and then choose **Store**.

AWS CLI

To create a secret by using the AWS CLI, you first create a JSON file or binary file that contains your secret. Then you use the `create-secret` operation.

To create a secret

1. Create your secret in a file, for example a JSON file named `mycreds.json`.

```
{
  "username": "saanvi",
  "password": "EXAMPLE-PASSWORD"
}
```

2. In the AWS CLI, use the following command.

```
$ aws secretsmanager create-secret --name MySecret --secret-string file://mycreds.json
```

The following shows the output.

```
{
  "SecretARN": "arn:aws:secretsmanager:Region:AccountId:secret:MySecret-a1b2c3",
  "SecretName": "MySecret",
  "SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE"
}
```

AWS SDK

To create a secret by using one of the AWS SDKs, use the [CreateSecret](#) action. For more information, see [the section called "AWS SDKs" \(p. 8\)](#).

Modify a secret

You can modify some parts of a secret after you create it: the description, resource-based policy, the encryption key, and tags. You can also change the encrypted secret value; however, we recommend you use rotation to update secret values that contain credentials. Rotation updates both the secret in Secrets Manager and the credentials on the database or service. This keeps the secret automatically synchronized so when clients request a secret value, they always get a working set of credentials. For more information, see [Rotate secrets \(p. 115\)](#).

To update a secret (console)

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. From the list of secrets, choose your secret.
3. On the secret details page, do any of the following:
 - To update the description, in the **Secrets details** section, choose **Actions**, and then choose **Edit description**.
 - To update the encryption key, in the **Secrets details** section, choose **Actions**, and then choose **Edit encryption key**. See [the section called "Secret encryption and decryption" \(p. 161\)](#).
 - To update tags, in the **Tags** section, choose **Edit**. See [the section called "Tag secrets" \(p. 63\)](#).
 - To update the secret value, in the **Secret value** section, choose **Retrieve secret value** and then choose **Edit**.

Secrets Manager creates a new version of the secret with the staging label `AWSCURRENT`. You can still access the old version. From the CLI, use the [get-secret-value](#) action with `version-id` `AWSPREVIOUS`.

- To update rotation for your secret, choose **Edit rotation**. See [Rotate secrets \(p. 115\)](#).
- To update permissions for your secret, choose **Edit permissions**. See [the section called "Attach a permissions policy to a secret" \(p. 28\)](#).
- To replicate your secret to other Regions, see [Replicate a secret to other Regions \(p. 61\)](#).
- If your secret has replicas, you can change the encryption key for a replica. In the **Replicate secret** section, select the radio button for the replica, and then on the **Actions** menu, choose **Edit encryption key**. See [the section called "Secret encryption and decryption" \(p. 161\)](#).

AWS CLI

To update a secret by using the AWS CLI, use the [update-secret](#) or [put-secret-value](#) operation. To tag a secret, see [the section called "Tag secrets" \(p. 63\)](#).

Example Example: Update secret description

The following example adds or replaces the description with the one in the `--description` parameter.

```
$ aws secretsmanager update-secret --secret-id production/MyAwesomeAppSecret --description
'This is the description I want to attach to the secret.'
{
  "ARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:production/
MyAwesomeAppSecret-AbCdEf",
  "Name": "production/MyAwesomeAppSecret",
  "VersionId": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE"
}
```

Example Example: Update encryption key

The following example adds or replaces the encryption key for this secret.

When you change the encryption key, Secrets Manager re-encrypts versions of the secret that have the staging labels `AWSCURRENT`, `AWSPENDING`, and `AWSPREVIOUS` under the new encryption key. When the secret value changes, Secrets Manager also encrypts it under the new key. You can use the old key or the new one to decrypt the secret when you retrieve it.

```
$ aws secretsmanager update-secret --secret-id production/MyAwesomeAppSecret --kms-key-id
arn:aws:kms:Region:AccountId:key/EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE
```

Example Example: Update secret value

When you update the secret value for a secret, Secrets Manager creates a new version with the `AWSCURRENT` staging label and moves the `AWSPREVIOUS` staging label to the version that previously had the label `AWSCURRENT`.

We recommend you avoid calling `PutSecretValue` or `UpdateSecret` at a sustained rate of more than once every 10 minutes. When you call `PutSecretValue` or `UpdateSecret` to update the secret value, Secrets Manager creates a new version of the secret. Secrets Manager removes outdated versions when there are more than 100, but it does not remove versions created less than 24 hours ago. If you update the secret value more than once every 10 minutes, you create more versions than Secrets Manager removes, and you will reach the quota for secret versions.

The following example AWS CLI command updates the secret value for a secret.

```
$ aws secretsmanager put-secret-value --secret-id production/MyAwesomeAppSecret --secret-
string '{"username":"anika","password":"EXAMPLE-PASSWORD"}'
{
  "SecretARN": "arn:aws:secretsmanager:us-east-2:123456789012:secret:production/
MyAwesomeAppSecret-AbCdEf",
  "SecretName": "production/MyAwesomeAppSecret",
  "VersionId": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE"
}
```

AWS SDK

We recommend you avoid calling `PutSecretValue` or `UpdateSecret` at a sustained rate of more than once every 10 minutes. When you call `PutSecretValue` or `UpdateSecret` to update the secret value, Secrets Manager creates a new version of the secret. Secrets Manager removes outdated versions when there are more than 100, but it does not remove versions created less than 24 hours ago. If you update the secret value more than once every 10 minutes, you create more versions than Secrets Manager removes, and you will reach the quota for secret versions.

To update a secret, use the following actions: [UpdateSecret](#), [ReplicateSecretToRegions](#), or [PutSecretValue](#). For more information, see [the section called "AWS SDKs"](#) (p. 8).

Find secrets in AWS Secrets Manager

When you search for secrets without a filter, Secrets Manager matches keywords in the secret name, description, tag key, and tag value. Searching without filters is not case-sensitive and ignores special characters, such as space, /, _, =, #, and only uses numbers and letters. When you search without a filter, Secrets Manager analyzes the search string to convert it to separate words. The words are separated by any change from uppercase to lowercase, from letter to number, or from number/letter to punctuation. For example, entering the search term `credsDatabase#892` searches for `creds`, `Database`, and `892` in name, description, and tag key and value.

You can apply the following filters to your search:

Name

Matches the beginning of secret names; case-sensitive. For example, **Name: Data** returns a secret named `DatabaseSecret`, but not `databaseSecret` or `MyData`.

Description

Matches the words in secret descriptions, not case-sensitive. For example, **Description: My Description** matches secrets with the following descriptions:

- My Description
- my description
- My basic description
- Description of my secret

Replicated secrets

You can filter for primary secrets, replica secrets, or secrets that aren't replicated.

Tag keys

Matches the beginning of tag keys; case-sensitive. For example, **Tag key: Prod** returns secrets with the tag `Production` and `Prod1`, but not secrets with the tag `prod` or `1 Prod`.

Tag values

Matches the beginning of tag values; case-sensitive. For example, **Tag value: Prod** returns secrets with the tag `Production` and `Prod1`, but not secrets with the tag value `prod` or `1 Prod`.

Secrets Manager is a regional service and only secrets within the selected region are returned.

AWS CLI

To find secrets stored in Secrets Manager, use `list-secrets`, as shown in the following example.

The following example searches for secrets with the keyword **conducts** in the description.

```
$ aws secretsmanager list-secrets --filters Key=description,Values=conducts
{
  [
    {
      "Description": "Conducts an AWS SecretsManager rotation for RDS MySQL using single
user rotation scheme",
```



```
        "SecretName": "SecretsManager-rotation-lambda"
    },
    {
        "Description": "Conducts an AWS SecretsManager rotation for RDS MySQL using single
user rotation scheme",
        "SecretName": "SecretsManager-rotation-Developers"
    }
]
}
```

AWS SDK

To find secrets by using one of the AWS SDKs, use `ListSecrets`. For more information, see [the section called “AWS SDKs” \(p. 8\)](#).

Delete a secret

Because of the critical nature of secrets, AWS Secrets Manager intentionally makes deleting a secret difficult. Secrets Manager does not immediately delete secrets. Instead, Secrets Manager immediately makes the secrets inaccessible and scheduled for deletion after a recovery window of a minimum of seven days. Until the recovery window ends, you can recover a secret you previously deleted. There is no charge for secrets that you have marked for deletion.

You can't delete a primary secret if it is replicated to other Regions. First delete the replicas, then delete the primary secret. When you delete a replica, it is deleted immediately.

You can't directly delete a version of a secret. Instead, you remove all staging labels from the version using the AWS CLI or AWS SDK. This marks the version as deprecated, and then Secrets Manager can automatically delete the version in the background.

If you don't know whether an application still uses a secret, you can create an Amazon CloudWatch alarm to alert you to any attempts to access a secret during the recovery window. For more information, see [Monitor secrets scheduled for deletion \(p. 143\)](#).

To delete a secret, you must have `secretsmanager:ListSecrets` and `secretsmanager:DeleteSecret` permissions.

To delete a secret (console)

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. In the list of secrets, choose the secret you want to delete.
3. In the **Secret details** section, choose **Actions**, and then choose **Delete secret**.
4. In the **Disable secret and schedule deletion** dialog box, in **Waiting period**, enter the number of days to wait before the deletion becomes permanent. Secrets Manager attaches a field called `DeletionDate` and sets the field to the current date and time, plus the number of days specified for the recovery window.
5. Choose **Schedule deletion**.

To view deleted secrets

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. On the **Secrets** page, choose **Preferences** (⚙️).

3. In the Preferences dialog box, select **Show disabled secrets**, and then choose **Save**

To delete a replica secret

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose the primary secret.
3. In the **Replicate Secret** section, choose the replica secret.
4. From the **Actions** menu, choose **Delete Replica**.

AWS CLI

To delete a secret, use the `delete-secret` action. To delete a version of a secret, use the `update-secret-version-stage` action to remove all of the staging labels. Secrets Manager can then delete the version in the background. To find the version ID of the version you want to delete, use `ListSecretVersionIds`.

To delete a replica, use the `remove-regions-from-replication` action.

Example

The following example marks for deletion the secret named "MyTestDatabase" and schedules deletion after a recovery window of 14 days. At any time after the date and time specified in the `DeletionDate` field, Secrets Manager permanently deletes the secret.

```
$ aws secretsmanager delete-secret --secret-id development/MyTestDatabase --recovery-  
window-in-days 14  
{  
  "ARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:development/  
MyTestDatabase-AbCdEf",  
  "Name": "development/MyTestDatabase",  
  "DeletionDate": 1510089380.309  
}
```

Example

The following example immediately deletes the secret without a recovery window. The `DeletionDate` response field shows the current date and time instead of a future time. **This secret cannot be recovered.**

```
$ aws secretsmanager delete-secret --secret-id development/MyTestDatabase --force-delete-  
without-recovery  
{  
  "ARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:development/  
MyTestDatabase-AbCdEf",  
  "Name": "development/MyTestDatabase",  
  "DeletionDate": 1508750180.309  
}
```

Example

The following example deletes a replica secret.

```
$ aws secretsmanager remove-regions-from-replication --secret-id development/MyTestDatabase  
--remove-replica-regions us-east-1
```

Example

The following example removes the `AWSPREVIOUS` staging label from a version of the secret named "MyTestDatabase".

```
$ aws secretsmanager update-secret-version-stage \
    --secret-id development/MyTestDatabase \
    --remove-from-version-id EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE \
    --version-stage AWSPREVIOUS
{
  "ARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:development/MyTestDatabase-AbCdEf",
  "Name": "development/MyTestDatabase"
}
```

AWS SDK

To delete a secret, use the `DeleteSecret` command. To delete a version of a secret, use the `UpdateSecretVersionStage` command. To delete a replica, use the `StopReplicationToReplica` command. For more information, see [the section called "AWS SDKs" \(p. 8\)](#).

Restore a secret

Secrets Manager considers a secret scheduled for deletion *deprecated* and you can no longer directly access it. After the recovery window has passed, Secrets Manager deletes the secret permanently. Once Secrets Manager deletes the secret, you can't recover it. Before the end of the recovery window, you can recover the secret and make it accessible again. This removes the `DeletionDate` field, which cancels the scheduled permanent deletion.

To restore a secret and the metadata in the console, you must have `secretsmanager:ListSecrets` and `secretsmanager:RestoreSecret` permissions.

To restore a secret (console)

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. In the list of secrets, choose the secret you want to restore.

If deleted secrets don't appear in your list of secrets, choose **Preferences** (⚙️). In the Preferences dialog box, select **Show disabled secrets**, and then choose **Save**.

3. On the **Secret details** page, choose **Cancel deletion**.
4. In the **Cancel secret deletion** dialog box, choose **Cancel deletion**.

AWS CLI

You can use the `restore-secret` command to retrieve a secret stored in Secrets Manager.

Example

The following example restores a previously deleted secret named "MyTestDatabase". This cancels the scheduled deletion and restores access to the secret.

```
$ aws secretsmanager restore-secret --secret-id development/MyTestDatabase
```

```
{
  "ARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:development/
MyTestDatabase-AbCdEf",
  "Name": "development/MyTestDatabase"
}
```

AWS SDK

To restore a secret marked for deletion, use the [RestoreSecret](#) command. For more information, see [the section called "AWS SDKs" \(p. 8\)](#).

Replicate an AWS Secrets Manager secret to other AWS Regions

You can replicate your secrets in multiple AWS Regions to support applications spread across those Regions to meet Regional access and low latency requirements. If you later need to, you can promote a replica secret to a standalone and then set it up for replication independently. Secrets Manager replicates the encrypted secret data and metadata such as tags and resource policies across the specified Regions.

The ARN for replicated secrets shows the Region the replica is in, for example:

- Primary secret: `arn:aws::secretsmanager:Region1:123456789012:secret:MySecret-a1b2c3`
- Replica secret: `arn:aws::secretsmanager:Region2:123456789012:secret:MySecret-a1b2c3`.

When you store database credentials for a source database that is replicated to other Regions, the secret contains connection information for the source database. If you then replicate the secret, the replicas are copies of the source secret and contain the same connection information. You can add additional key/value pairs to the secret for regional connection information.

If you turn on rotation for your primary secret, Secrets Manager rotates the secret in the primary Region, and the new secret value propagates to all of the associated replica secrets. You don't have to manage rotation individually for all of the replica secrets.

You can replicate secrets across all of your enabled AWS Regions. However, if you use Secrets Manager in special AWS Regions such as AWS GovCloud (US) or China Regions, you can only configure secrets and the replicas within these specialized AWS Regions. You can't replicate a secret in your enabled AWS Regions to a specialized Region or replicate secrets from a specialized region to a commercial region.

Before you can replicate a secret to another Region, you must enable that Region. For more information, see [Managing AWS Regions](#).

It is possible to use a secret across multiple Regions without replicating it by calling the Secrets Manager endpoint in the Region where the secret is stored. For a list of endpoints, see [AWS Secrets Manager endpoints](#).

To replicate a secret to other Regions (console)

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. On the **Secrets** page, choose your secret.
3. On the Secret details page, do one of the following:
 - If your secret is not replicated, choose **Replicate secret**.

- If your secret is replicated, in the **Replicate secret** section, choose **Add Region**.
4. In the **Add replica regions** dialog box, do the following:
 - a. For **AWS Region**, choose the Region you want to replicate the secret to.
 - b. (Optional) For **Encryption key**, choose a KMS key to encrypt the secret with. The key must be in the replica Region, and you can choose the same key as the primary secret.
 - c. (Optional) To add another Region, choose **Add more regions**.
 - d. Choose **Replicate**.

You return to the secret details page. In the **Replicate secret** section, the **Replication status** shows for each Region. The following are some reasons that replication can fail and how to resolve them:

- **Failed** - Secret with the same name exists in the selected Region. One option to resolve is to overwrite the duplicate name secret in the replica Region. Choose the **Actions** menu and then choose **Retry replication**. In the **Retry replication** dialog box, choose **Overwrite** and then choose **Retry replication**.
- **Failed** - No permissions available on the KMS key to complete the replication. One option to resolve is to update permissions policies for the KMS key so that you have `kms:Decrypt` permission.
- **Failed** - Secret replication failed due to a network error. When the network is available, choose the **Actions** menu and then choose **Retry replication**.
- **Failed** - You have not enabled the Region where the replication occurs. For more information about how to enable a Region, see [Managing AWS Regions](#).

AWS CLI

To replicate a secret, use the [replicate-secret-to-regions](#) action. The following example replicates a secret to US East (N. Virginia).

```
$ aws secretsmanager replicate-secret-to-regions --secret-id production/DBWest --add-replica-regions region us-east-1
```

AWS SDK

To replicate a secret, use the [ReplicateSecretToRegions](#) command. For more information, see [the section called "AWS SDKs"](#) (p. 8).

Promote a replica secret to a standalone secret

A replica secret is a secret that is replicated from a primary in another AWS Region. It has the same secret value and metadata as the primary, but it can be encrypted with a different KMS key. A replica secret can't be updated independently from its primary secret, except for its encryption key. Promoting a replica secret disconnects the replica secret from the primary secret and makes the replica secret a standalone secret. Changes to the primary secret won't replicate to the standalone secret.

You might want to promote a replica secret to a standalone secret as a disaster recovery solution if the primary secret becomes unavailable. Or you might want to promote a replica to a standalone secret if you want to turn on rotation for the replica.

If you promote a replica, be sure to update the corresponding applications to use the standalone secret.

To promote a replica secret (console)

1. Log in to the Secrets Manager at <https://console.aws.amazon.com/secretsmanager/>.
2. On the **Secrets** page, choose the primary secret.
3. On the Secret details page, in the **Replicate secret** section, choose the ARN of the replica you want to promote.
4. On the replica secret details page, choose **Promote to standalone secret**. choose **Promote to standalone secret**.

AWS CLI

To promote a replica to a standalone secret, use the [stop-replication-to-replica](#) action. You must call this action from the replica secret Region.

Example

The following example promotes a replica secret to a standalone.

```
$ aws secretsmanager stop-replication-to-replica \
    --secret-id development/MyTestDatabase
```

AWS SDK

To promote a replica to a standalone secret, use the [StopReplicationToReplica](#) command. You must call this command from the replica secret Region. For more information, see [the section called "AWS SDKs"](#) (p. 8).

Tag secrets

Secrets Manager defines a *tag* as a label consisting of a key that you define and an optional value. You can use tags to make it easy to manage, search, and filter secrets and other resources in your AWS account. When you tag your secrets, use a standard naming scheme across all of your resources. Tags are case sensitive. Never store sensitive information for a secret in a tag.

To find secrets with a specific tag, see [the section called "Find secrets"](#) (p. 57).

Create tags for:

- **Security/access control** – You can grant or deny access to a secret by checking the tags attached to the secret. See [the section called "Example: Control access to secrets using tags"](#) (p. 36).
- **Automation** – You can use tags to filter resources for automation. For example, some customers run automated start/stop scripts to turn off development environments during non-business hours to reduce costs. You can create and then check for a tag indicating if a specific Amazon EC2 instance should be included in the shutdown.
- **Filtering** – You can find secrets by tags in the console, AWS CLI, and SDKs. AWS also provides the Resource Groups tool to create a custom console that consolidates and organizes your resources based on their tags. For more information, see [Working with Resource Groups](#) in the *AWS Management Console Getting Started Guide*.

For more information, see [AWS Tagging Strategies](#) on the *AWS Answers* website.

You can tag your secrets when you create them or when you edit them.

To change tags for your secret (console)

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. From the list of secrets, choose your secret.
3. In the secret details page, in the **Tags** section, choose **Edit**. Tag key names and values are case sensitive, and tag keys must be unique.

AWS CLI

To change tags for your secret, use the [tag-resource](#) or [untag-resource](#) operation.

Example

The following example adds or replaces the tags with those provided by the `--tags` parameter. Tag key names and values are case sensitive, and tag keys must be unique. The parameter is expected to be a JSON array of `Key` and `Value` elements:

```
$ aws secretsmanager tag-resource --secret-id MySecret2 --tags Key=costcenter,Value=12345
```

Example

The following example AWS CLI command removes the tags with the key "environment" from the specified secret:

```
$ aws secretsmanager untag-resource --secret-id MySecret2 --tag-keys 'environment'
```

The `tag-resource` command doesn't return any output.

AWS SDK

To change tags for your secret, use [TagResource](#) or [UntagResource](#). For more information, see [the section called "AWS SDKs" \(p. 8\)](#).

Create secrets in AWS CloudFormation

You can create secrets in a CloudFormation stack by using the `AWS::SecretsManager::Secret` resource in a CloudFormation template.

A common scenario is to first create a secret with a password generated by Secrets Manager, and then use a [dynamic reference \(p. 106\)](#) to retrieve the username and password from the secret to use as credentials for a new database. See the examples below.

To attach a resource policy to your secret, use the `AWS::SecretsManager::ResourcePolicy` resource.

If the secret contains Amazon RDS, Amazon Redshift, or Amazon DocumentDB credentials, to turn on automatic rotation for a secret, use the `AWS::SecretsManager::SecretTargetAttachment` resource to add details about the database to the secret that Secrets Manager needs to rotate the secret. Then use the `AWS::SecretsManager::RotationSchedule` resource to turn on automatic rotation. You specify both the Lambda rotation function and the rotation schedule in this resource. For a secret that contains Amazon RDS, Amazon Redshift, or Amazon DocumentDB credentials, use one of the provided [Rotation function templates \(p. 128\)](#).

For other types of secrets, you create your own rotation function and then use the [AWS::SecretsManager::RotationSchedule](#) resource to turn on automatic rotation. Secrets Manager provides a [the section called "Generic rotation function template" \(p. 131\)](#) that you can use as a starting point.

For information about creating resources with AWS CloudFormation, see [Learn template basics](#) in the AWS CloudFormation User Guide. You can also use the AWS Cloud Development Kit (CDK). For more information, see [AWS Secrets Manager Construct Library](#).

Examples

- [Create a Secrets Manager secret with AWS CloudFormation \(p. 65\)](#)
- [Create a Secrets Manager secret for an Amazon RDS MySQL DB instance with AWS CloudFormation \(p. 66\)](#)
- [Create a Secrets Manager secret with automatic rotation and an Amazon RDS MySQL DB instance with AWS CloudFormation \(p. 69\)](#)
- [Create a Secrets Manager secret with automatic rotation and an Amazon Redshift cluster with AWS CloudFormation \(p. 75\)](#)
- [Create a Secrets Manager secret with automatic rotation and an Amazon DocumentDB instance with AWS CloudFormation \(p. 80\)](#)

Create a Secrets Manager secret with AWS CloudFormation

This example creates a secret named **CloudFormationCreatedSecret-*a1b2c3d4e5f6***. The secret value is the following JSON, with a 32-character password that is generated when the secret is created.

```
{
  "password": "EXAMPLE-PASSWORD",
  "username": "saanvi"
}
```

This example uses the following CloudFormation resource:

- [AWS::SecretsManager::Secret](#)

For information about creating resources with AWS CloudFormation, see [Learn template basics](#) in the AWS CloudFormation User Guide.

JSON

```
{
  "Resources": {
    "CloudFormationCreatedSecret": {
      "Type": "AWS::SecretsManager::Secret",
      "Properties": {
        "Description": "Simple secret created by AWS CloudFormation.",
        "GenerateSecretString": {
          "SecretStringTemplate": "{\"username\": \"saanvi\"}",
          "GenerateStringKey": "password",
          "PasswordLength": 32
        }
      }
    }
  }
}
```



```
}
```

YAML

```
Resources:
  CloudFormationCreatedSecret:
    Type: 'AWS::SecretsManager::Secret'
    Properties:
      Description: Simple secret created by AWS CloudFormation.
      GenerateSecretString:
        SecretStringTemplate: '{"username": "saanvi"}'
        GenerateStringKey: password
        PasswordLength: 32
```

Create a Secrets Manager secret for an Amazon RDS MySQL DB instance with AWS CloudFormation

This example creates a secret and an Amazon RDS MySQL DB instance using the credentials in the secret as the user and password. Secrets Manager generates a password with 32 characters. As a security best practice, the database is in an Amazon VPC.

For a tutorial to turn on rotation for the secret created in this template, see [the section called “Single user rotation” \(p. 17\)](#).

For an example with automatic rotation already turned on, see [Create a secret with Amazon RDS credentials with automatic rotation \(p. 69\)](#).

This example uses the following CloudFormation resources for Secrets Manager:

- [AWS::SecretsManager::Secret](#)
- [AWS::SecretsManager::SecretTargetAttachment](#)

For information about creating resources with AWS CloudFormation, see [Learn template basics](#) in the AWS CloudFormation User Guide.

JSON

```
{
  "Description": "This is an example template to demonstrate CloudFormation resources for Secrets Manager",
  "Resources": {
    "TestVPC": {
      "Type": "AWS::EC2::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/16",
        "EnableDnsHostnames": true,
        "EnableDnsSupport": true,
        "Tags": [
          {
            "Key": "Name",
            "Value": "SecretsManagerTutorial"
          }
        ]
      }
    },
    "TestSubnet01": {
      "Type": "AWS::EC2::Subnet",
```

```
    "Properties": {
      "CidrBlock": "10.0.96.0/19",
      "AvailabilityZone": {
        "Fn::Select": [
          "0",
          {
            "Fn::GetAZs": {
              "Ref": "AWS::Region"
            }
          }
        ]
      },
      "VpcId": {
        "Ref": "TestVPC"
      }
    }
  },
  "TestSubnet02": {
    "Type": "AWS::EC2::Subnet",
    "Properties": {
      "CidrBlock": "10.0.128.0/19",
      "AvailabilityZone": {
        "Fn::Select": [
          "1",
          {
            "Fn::GetAZs": {
              "Ref": "AWS::Region"
            }
          }
        ]
      },
      "VpcId": {
        "Ref": "TestVPC"
      }
    }
  },
  "SecretsManagerTutorialAdmin": {
    "Type": "AWS::SecretsManager::Secret",
    "Properties": {
      "Description": "AWS RDS admin credentials",
      "GenerateSecretString": {
        "SecretStringTemplate": "{\"username\": \"admin\"}",
        "GenerateStringKey": "password",
        "PasswordLength": 32,
        "ExcludeCharacters": "/@\"'\"\\\"
      }
    }
  },
  "MyDBInstance": {
    "Type": "AWS::RDS::DBInstance",
    "Properties": {
      "AllocatedStorage": 20,
      "DBInstanceClass": "db.t2.micro",
      "DBInstanceIdentifier": "SecretsManagerTutorialDB",
      "Engine": "mysql",
      "DBSubnetGroupName": {
        "Ref": "MyDBSubnetGroup"
      },
      "MasterUsername": {
        "Fn::Sub": "${resolve:secretsmanager:
${SecretsManagerTutorialAdmin}:username}"
      },
      "MasterUserPassword": {
        "Fn::Sub": "${resolve:secretsmanager:
${SecretsManagerTutorialAdmin}:password}"
      }
    }
  },
```

```
        "BackupRetentionPeriod": 0,
        "VPCSecurityGroups": [
            {
                "Fn::GetAtt": [
                    "TestVPC",
                    "DefaultSecurityGroup"
                ]
            }
        ]
    },
    "MyDBSubnetGroup": {
        "Type": "AWS::RDS::DBSubnetGroup",
        "Properties": {
            "DBSubnetGroupDescription": "Test Group",
            "SubnetIds": [
                {
                    "Ref": "TestSubnet01"
                },
                {
                    "Ref": "TestSubnet02"
                }
            ]
        }
    },
    "SecretRDSInstanceAttachment": {
        "Type": "AWS::SecretsManager::SecretTargetAttachment",
        "Properties": {
            "SecretId": {
                "Ref": "SecretsManagerTutorialAdmin"
            },
            "TargetId": {
                "Ref": "MyDBInstance"
            },
            "TargetType": "AWS::RDS::DBInstance"
        }
    }
}
```

YAML

```
Description: >-
  This is an example template to demonstrate CloudFormation resources for
  Secrets Manager
Resources:
  TestVPC:
    Type: 'AWS::EC2::VPC'
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsHostnames: true
      EnableDnsSupport: true
      Tags:
        - Key: Name
          Value: SecretsManagerTutorial
  TestSubnet01:
    Type: 'AWS::EC2::Subnet'
    Properties:
      CidrBlock: 10.0.96.0/19
      AvailabilityZone: !Select
        - '0'
        - !GetAZs
      Ref: 'AWS::Region'
    VpcId: !Ref TestVPC
```

```
TestSubnet02:
  Type: 'AWS::EC2::Subnet'
  Properties:
    CidrBlock: 10.0.128.0/19
    AvailabilityZone: !Select
      - '1'
      - !GetAZs
    Ref: 'AWS::Region'
    VpcId: !Ref TestVPC
SecretsManagerTutorialAdmin:
  Type: 'AWS::SecretsManager::Secret'
  Properties:
    Description: AWS RDS admin credentials
    GenerateSecretString:
      SecretStringTemplate: '{"username": "admin"}'
      GenerateStringKey: password
      PasswordLength: 32
      ExcludeCharacters: '@/\'
MyDBInstance:
  Type: 'AWS::RDS::DBInstance'
  Properties:
    AllocatedStorage: 20
    DBInstanceClass: db.t2.micro
    DBInstanceIdentifier: SecretsManagerTutorialDB
    Engine: mysql
    DBSubnetGroupName: !Ref MyDBSubnetGroup
    MasterUsername: !Sub '{{resolve:secretsmanager:
${SecretsManagerTutorialAdmin}:username}}'
    MasterUserPassword: !Sub '{{resolve:secretsmanager:
${SecretsManagerTutorialAdmin}:password}}'
    BackupRetentionPeriod: 0
    VPCSecurityGroups:
      - !GetAtt
        - TestVPC
      - DefaultSecurityGroup
MyDBSubnetGroup:
  Type: 'AWS::RDS::DBSubnetGroup'
  Properties:
    DBSubnetGroupDescription: Test Group
    SubnetIds:
      - !Ref TestSubnet01
      - !Ref TestSubnet02
SecretRDSInstanceAttachment:
  Type: 'AWS::SecretsManager::SecretTargetAttachment'
  Properties:
    SecretId: !Ref SecretsManagerTutorialAdmin
    TargetId: !Ref MyDBInstance
    TargetType: 'AWS::RDS::DBInstance'
```

Create a Secrets Manager secret with automatic rotation and an Amazon RDS MySQL DB instance with AWS CloudFormation

This example creates a secret and an Amazon RDS MySQL DB instance using the credentials in the secret as the user and password. Secrets Manager generates a password with 32 characters. The template also creates a Lambda rotation function from the [Rotation function templates \(p. 128\)](#) and configures the secret to automatically rotate between 8:00 AM and 10:00 AM UTC on the first day of every month. As a security best practice, the DB instance is in an Amazon VPC.

To see an example without automatic rotation, see [Create a secret with Amazon RDS credentials \(p. 66\)](#).

This example uses the following CloudFormation resources for Secrets Manager:

- [AWS::SecretsManager::Secret](#)
- [AWS::SecretsManager::SecretTargetAttachment](#)
- [AWS::SecretsManager::RotationSchedule](#)

For information about creating resources with AWS CloudFormation, see [Learn template basics](#) in the AWS CloudFormation User Guide.

JSON

```
{
  "Transform": "AWS::SecretsManager-2020-07-23",
  "Description": "This is an example template to demonstrate CloudFormation resources for Secrets Manager",
  "Resources": {
    "TestVPC": {
      "Type": "AWS::EC2::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/16",
        "EnableDnsHostnames": true,
        "EnableDnsSupport": true
      }
    },
    "TestSubnet01": {
      "Type": "AWS::EC2::Subnet",
      "Properties": {
        "CidrBlock": "10.0.96.0/19",
        "AvailabilityZone": {
          "Fn::Select": [
            "0",
            {
              "Fn::GetAZs": {
                "Ref": "AWS::Region"
              }
            }
          ]
        },
        "VpcId": {
          "Ref": "TestVPC"
        }
      }
    },
    "TestSubnet02": {
      "Type": "AWS::EC2::Subnet",
      "Properties": {
        "CidrBlock": "10.0.128.0/19",
        "AvailabilityZone": {
          "Fn::Select": [
            "1",
            {
              "Fn::GetAZs": {
                "Ref": "AWS::Region"
              }
            }
          ]
        },
        "VpcId": {
          "Ref": "TestVPC"
        }
      }
    }
  }
},
```



```
        {
            "Fn::GetAtt": [
                "TestVPC",
                "DefaultSecurityGroup"
            ]
        }
    ]
},
"SecretRDSInstanceAttachment": {
    "Type": "AWS::SecretsManager::SecretTargetAttachment",
    "Properties": {
        "SecretId": {
            "Ref": "MyRDSInstanceRotationSecret"
        },
        "TargetId": {
            "Ref": "MyDBInstance"
        },
        "TargetType": "AWS::RDS::DBInstance"
    }
},
"MySecretRotationSchedule": {
    "Type": "AWS::SecretsManager::RotationSchedule",
    "DependsOn": "SecretRDSInstanceAttachment",
    "Properties": {
        "SecretId": {
            "Ref": "MyRDSInstanceRotationSecret"
        },
        "HostedRotationLambda": {
            "RotationType": "MySQLSingleUser",
            "RotationLambdaName": "SecretsManagerRotation",
            "VpcSecurityGroupIds": {
                "Fn::GetAtt": [
                    "TestVPC",
                    "DefaultSecurityGroup"
                ]
            },
            "VpcSubnetIds": {
                "Fn::Join": [
                    ",",
                    [
                        {
                            "Ref": "TestSubnet01"
                        },
                        {
                            "Ref": "TestSubnet02"
                        }
                    ]
                ]
            }
        }
    }
},
"RotationRules": {
```

```
        "Duration": 2h,  
        "ScheduleExpression": "cron(0 8 1 * ? *)"  
      }  
    }  
  }  
}
```

YAML

```
---  
Transform: AWS::SecretsManager-2020-07-23  
Description: This is an example template to demonstrate CloudFormation resources for  
Secrets Manager  
Resources:  
  
  #This is the VPC that the rotation Lambda function and the RDS instance will be placed in  
  TestVPC:  
    Type: AWS::EC2::VPC  
    Properties:  
      CidrBlock: 10.0.0.0/16  
      EnableDnsHostnames: true  
      EnableDnsSupport: true  
  
  # Subnet that the rotation Lambda function and the RDS instance will be placed in  
  TestSubnet01:  
    Type: AWS::EC2::Subnet  
    Properties:  
      CidrBlock: 10.0.96.0/19  
      AvailabilityZone:  
        Fn::Select:  
          - '0'  
          - Fn::GetAZs:  
              Ref: AWS::Region  
      VpcId:  
        Ref: TestVPC  
  TestSubnet02:  
    Type: AWS::EC2::Subnet  
    Properties:  
      CidrBlock: 10.0.128.0/19  
      AvailabilityZone:  
        Fn::Select:  
          - '1'  
          - Fn::GetAZs:  
              Ref: AWS::Region  
      VpcId:  
        Ref: TestVPC  
  
  #VPC endpoint that will enable the rotation Lambda function to make api calls to Secrets  
  Manager  
  SecretsManagerVPCEndpoint:  
    Type: AWS::EC2::VPCEndpoint  
    Properties:  
      SubnetIds:  
        - Ref: TestSubnet01  
        - Ref: TestSubnet02  
      SecurityGroupIds:  
        - Fn::GetAtt:  
            - TestVPC  
            - DefaultSecurityGroup  
      VpcEndpointType: Interface  
      ServiceName:  
        Fn::Sub: com.amazonaws.${AWS::Region}.secretsmanager  
      PrivateDnsEnabled: true
```



```
VpcId:
  Ref: TestVPC

#This is a Secret resource with a randomly generated password in its SecretString JSON.
MyRDSInstanceRotationSecret:
  Type: AWS::SecretsManager::Secret
  Properties:
    Description: This is my rds instance secret
    GenerateSecretString:
      SecretStringTemplate: '{"username": "admin"}'
      GenerateStringKey: password
      PasswordLength: 32
      ExcludeCharacters: "/@\"'\\\"
    Tags:
      - Key: AppName
        Value: MyApp

#This is an RDS instance resource. Its master username and password use dynamic
references to resolve values from
#SecretsManager. The dynamic reference guarantees that CloudFormation will not log or
persist the resolved value
#We sub the Secret resource's logical id in order to construct the dynamic reference,
since the Secret's name is being #generated by CloudFormation
MyDBInstance:
  Type: AWS::RDS::DBInstance
  Properties:
    AllocatedStorage: 20
    DBInstanceClass: db.t2.micro
    Engine: mysql
    DBSubnetGroupName:
      Ref: MyDBSubnetGroup
    MasterUsername:
      Fn::Sub: "${resolve:secretsmanager:${MyRDSInstanceRotationSecret}::username}"
    MasterUserPassword:
      Fn::Sub: "${resolve:secretsmanager:${MyRDSInstanceRotationSecret}::password}"
    BackupRetentionPeriod: 0
    VPCSecurityGroups:
      - Fn::GetAtt:
          - TestVPC
          - DefaultSecurityGroup

#Database subnet group for the RDS instance
MyDBSubnetGroup:
  Type: AWS::RDS::DBSubnetGroup
  Properties:
    DBSubnetGroupDescription: Test Group
    SubnetIds:
      - Ref: TestSubnet01
      - Ref: TestSubnet02

#This is a SecretTargetAttachment resource which updates the referenced Secret resource
with properties about
#the referenced RDS instance
SecretRDSInstanceAttachment:
  Type: AWS::SecretsManager::SecretTargetAttachment
  Properties:
    SecretId:
      Ref: MyRDSInstanceRotationSecret
    TargetId:
      Ref: MyDBInstance
    TargetType: AWS::RDS::DBInstance

#This is a RotationSchedule resource. It configures rotation of password for the
referenced secret using a rotation lambda function
#The first rotation happens at resource creation time, with subsequent rotations
scheduled according to the rotation rules
```

```
#We explicitly depend on the SecretTargetAttachment resource being created to ensure that
the secret contains all the
#information necessary for rotation to succeed
MySecretRotationSchedule:
  Type: AWS::SecretsManager::RotationSchedule
  DependsOn: SecretRDSInstanceAttachment
  Properties:
    SecretId:
      Ref: MyRDSInstanceRotationSecret
    HostedRotationLambda:
      RotationType: MySQLSingleUser
      RotationLambdaName: SecretsManagerRotation
      VpcSecurityGroupIds:
        Fn::GetAtt:
          - TestVPC
          - DefaultSecurityGroup
      VpcSubnetIds:
        Fn::Join:
          - ","
          - - Ref: TestSubnet01
            - Ref: TestSubnet02
    RotationRules:
      Duration: 2h
      ScheduleExpression: 'cron(0 8 1 * ? *)'
```

Create a Secrets Manager secret with automatic rotation and an Amazon Redshift cluster with AWS CloudFormation

This example creates a secret and an Amazon Redshift cluster using the credentials in the secret as the user and password. The template also creates a Lambda rotation function from the [Rotation function templates \(p. 128\)](#) and configures the secret to automatically rotate between 8:00 AM and 10:00 AM UTC on the first day of every month. As a security best practice, the cluster is in an Amazon VPC.

This example uses the following CloudFormation resources for Secrets Manager:

- [AWS::SecretsManager::Secret](#)
- [AWS::SecretsManager::SecretTargetAttachment](#)
- [AWS::SecretsManager::RotationSchedule](#)

For information about creating resources with AWS CloudFormation, see [Learn template basics](#) in the AWS CloudFormation User Guide.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Transform": "AWS::SecretsManager-2020-07-23",
  "Resources": {
    "TestVPC": {
      "Type": "AWS::EC2::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/16",
        "EnableDnsHostnames": true,
        "EnableDnsSupport": true
      }
    },
  },
}
```

```
"TestSubnet01":{
  "Type":"AWS::EC2::Subnet",
  "Properties":{
    "CidrBlock":"10.0.96.0/19",
    "AvailabilityZone":{
      "Fn::Select":[
        "0",
        {
          "Fn::GetAZs":{
            "Ref":"AWS::Region"
          }
        }
      ]
    },
    "VpcId":{
      "Ref":"TestVPC"
    }
  }
},
"TestSubnet02":{
  "Type":"AWS::EC2::Subnet",
  "Properties":{
    "CidrBlock":"10.0.128.0/19",
    "AvailabilityZone":{
      "Fn::Select":[
        "1",
        {
          "Fn::GetAZs":{
            "Ref":"AWS::Region"
          }
        }
      ]
    },
    "VpcId":{
      "Ref":"TestVPC"
    }
  }
},
"SecretsManagerVPCEndpoint":{
  "Type":"AWS::EC2::VPCEndpoint",
  "Properties":{
    "SubnetIds":[
      {
        "Ref":"TestSubnet01"
      },
      {
        "Ref":"TestSubnet02"
      }
    ],
    "SecurityGroupIds":[
      {
        "Fn::GetAtt":[
          "TestVPC",
          "DefaultSecurityGroup"
        ]
      }
    ],
    "VpcEndpointType":"Interface",
    "ServiceName":{
      "Fn::Sub":"com.amazonaws.${AWS::Region}.secretsmanager"
    },
    "PrivateDnsEnabled":true,
    "VpcId":{
      "Ref":"TestVPC"
    }
  }
}
```

```
},
"MyRedshiftSecret":{
  "Type":"AWS::SecretsManager::Secret",
  "Properties":{
    "Description":"This is my rds instance secret",
    "GenerateSecretString":{
      "SecretStringTemplate":{"\"username\": \"admin\""},
      "GenerateStringKey":"password",
      "PasswordLength":16,
      "ExcludeCharacters":"\"@/\\\" "
    },
    "Tags":[
      {
        "Key":"AppName",
        "Value":"MyApp"
      }
    ]
  }
},
"MyRedshiftCluster":{
  "Type":"AWS::Redshift::Cluster",
  "Properties":{
    "DBName":"myyamldb",
    "NodeType":"ds2.xlarge",
    "ClusterType":"single-node",
    "ClusterSubnetGroupName":{
      "Ref":"ResdshiftSubnetGroup"
    },
    "MasterUsername":{
      "Fn::Sub":"{{resolve:secretsmanager:${MyRedshiftSecret}::username}}"
    },
    "MasterUserPassword":{
      "Fn::Sub":"{{resolve:secretsmanager:${MyRedshiftSecret}::password}}"
    },
    "PubliclyAccessible":false,
    "VpcSecurityGroupIds":[
      {
        "Fn::GetAtt":[
          "TestVPC",
          "DefaultSecurityGroup"
        ]
      }
    ]
  }
},
"ResdshiftSubnetGroup":{
  "Type":"AWS::Redshift::ClusterSubnetGroup",
  "Properties":{
    "Description":"Test Group",
    "SubnetIds":[
      {
        "Ref":"TestSubnet01"
      },
      {
        "Ref":"TestSubnet02"
      }
    ]
  }
},
"SecretRedshiftAttachment":{
  "Type":"AWS::SecretsManager::SecretTargetAttachment",
  "Properties":{
    "SecretId":{
      "Ref":"MyRedshiftSecret"
    },
    "TargetId":{
```

```
        "Ref": "MyRedshiftCluster"
      },
      "TargetType": "AWS::Redshift::Cluster"
    }
  },
  "MySecretRotationSchedule": {
    "Type": "AWS::SecretsManager::RotationSchedule",
    "DependsOn": "SecretRedshiftAttachment",
    "Properties": {
      "SecretId": {
        "Ref": "MyRedshiftSecret"
      },
      "HostedRotationLambda": {
        "RotationType": "RedshiftSingleUser",
        "RotationLambdaName": "SecretsManagerRotationRedshift",
        "VpcSecurityGroupIds": {
          "Fn::GetAtt": [
            "TestVPC",
            "DefaultSecurityGroup"
          ]
        },
        "VpcSubnetIds": {
          "Fn::Join": [
            ",",
            [
              {
                "Ref": "TestSubnet01"
              },
              {
                "Ref": "TestSubnet02"
              }
            ]
          ]
        }
      },
      "RotationRules": {
        "Duration": "2h",
        "ScheduleExpression": "cron(0 8 1 * ? *)"
      }
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::SecretsManager-2020-07-23
Resources:
  TestVPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsHostnames: true
      EnableDnsSupport: true
  TestSubnet01:
    Type: AWS::EC2::Subnet
    Properties:
      CidrBlock: 10.0.96.0/19
      AvailabilityZone:
        Fn::Select:
          - '0'
        - Fn::GetAZs:
            Ref: AWS::Region
```

```
VpcId:
  Ref: TestVPC
TestSubnet02:
  Type: AWS::EC2::Subnet
  Properties:
    CidrBlock: 10.0.128.0/19
    AvailabilityZone:
      Fn::Select:
        - '1'
      - Fn::GetAZs:
          Ref: AWS::Region
    VpcId:
      Ref: TestVPC
SecretsManagerVPCEndpoint:
  Type: AWS::EC2::VPCEndpoint
  Properties:
    SubnetIds:
      - Ref: TestSubnet01
      - Ref: TestSubnet02
    SecurityGroupIds:
      - Fn::GetAtt:
          - TestVPC
          - DefaultSecurityGroup
    VpcEndpointType: Interface
    ServiceName:
      Fn::Sub: com.amazonaws.${AWS::Region}.secretsmanager
    PrivateDnsEnabled: true
    VpcId:
      Ref: TestVPC
MyRedshiftSecret:
  Type: AWS::SecretsManager::Secret
  Properties:
    Description: This is my rds instance secret
    GenerateSecretString:
      SecretStringTemplate: '{"username": "admin"}'
      GenerateStringKey: password
      PasswordLength: 16
      ExcludeCharacters: "\"@/\\\"
    Tags:
      - Key: AppName
        Value: MyApp
MyRedshiftCluster:
  Type: AWS::Redshift::Cluster
  Properties:
    DBName: myyamldb
    NodeType: ds2.xlarge
    ClusterType: single-node
    ClusterSubnetGroupName:
      Ref: ResdshiftSubnetGroup
    MasterUsername:
      Fn::Sub: "${resolve:secretsmanager:${MyRedshiftSecret}:username}"
    MasterUserPassword:
      Fn::Sub: "${resolve:secretsmanager:${MyRedshiftSecret}:password}"
    PubliclyAccessible: false
    VpcSecurityGroupIds:
      - Fn::GetAtt:
          - TestVPC
          - DefaultSecurityGroup
ResdshiftSubnetGroup:
  Type: AWS::Redshift::ClusterSubnetGroup
  Properties:
    Description: Test Group
    SubnetIds:
      - Ref: TestSubnet01
      - Ref: TestSubnet02
SecretRedshiftAttachment:
```

```
Type: AWS::SecretsManager::SecretTargetAttachment
Properties:
  SecretId:
    Ref: MyRedshiftSecret
  TargetId:
    Ref: MyRedshiftCluster
  TargetType: AWS::Redshift::Cluster
MySecretRotationSchedule:
  Type: AWS::SecretsManager::RotationSchedule
  DependsOn: SecretRedshiftAttachment
  Properties:
    SecretId:
      Ref: MyRedshiftSecret
    HostedRotationLambda:
      RotationType: RedshiftSingleUser
      RotationLambdaName: SecretsManagerRotationRedshift
      VpcSecurityGroupIds:
        Fn::GetAtt:
          - TestVPC
          - DefaultSecurityGroup
      VpcSubnetIds:
        Fn::Join:
          - ","
          - - Ref: TestSubnet01
            - Ref: TestSubnet02
    RotationRules:
      Duration: 2h
      ScheduleExpression: 'cron(0 8 1 * ? *)'
```

Create a Secrets Manager secret with automatic rotation and an Amazon DocumentDB instance with AWS CloudFormation

This example creates a secret and an Amazon DocumentDB instance using the credentials in the secret as the user and password. The secret has a resource-based policy attached that defines who can access the secret. The template also creates a Lambda rotation function from the [Rotation function templates \(p. 128\)](#) and configures the secret to automatically rotate between 8:00 AM and 10:00 AM UTC on the first day of every month. As a security best practice, the instance is in an Amazon VPC.

This example uses the following CloudFormation resources for Secrets Manager:

- [AWS::SecretsManager::Secret](#)
- [AWS::SecretsManager::SecretTargetAttachment](#)
- [AWS::SecretsManager::RotationSchedule](#)

For information about creating resources with AWS CloudFormation, see [Learn template basics](#) in the AWS CloudFormation User Guide.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Transform": "AWS::SecretsManager-2020-07-23",
  "Resources": {
    "TestVPC": {
      "Type": "AWS::EC2::VPC",
```

```
    "Properties":{
      "CidrBlock":"10.0.0.0/16",
      "EnableDnsHostnames":true,
      "EnableDnsSupport":true
    }
  },
  "TestSubnet01":{
    "Type":"AWS::EC2::Subnet",
    "Properties":{
      "CidrBlock":"10.0.96.0/19",
      "AvailabilityZone":{
        "Fn::Select":[
          "0",
          {
            "Fn::GetAZs":{
              "Ref":"AWS::Region"
            }
          }
        ]
      },
      "VpcId":{
        "Ref":"TestVPC"
      }
    }
  },
  "TestSubnet02":{
    "Type":"AWS::EC2::Subnet",
    "Properties":{
      "CidrBlock":"10.0.128.0/19",
      "AvailabilityZone":{
        "Fn::Select":[
          "1",
          {
            "Fn::GetAZs":{
              "Ref":"AWS::Region"
            }
          }
        ]
      },
      "VpcId":{
        "Ref":"TestVPC"
      }
    }
  },
  "SecretsManagerVPCEndpoint":{
    "Type":"AWS::EC2::VPCEndpoint",
    "Properties":{
      "SubnetIds":[
        {
          "Ref":"TestSubnet01"
        },
        {
          "Ref":"TestSubnet02"
        }
      ],
      "SecurityGroupIds":[
        {
          "Fn::GetAtt":[
            "TestVPC",
            "DefaultSecurityGroup"
          ]
        }
      ],
      "VpcEndpointType":"Interface",
      "ServiceName":{
        "Fn::Sub":"com.amazonaws.${AWS::Region}.secretsmanager"
```



```
    },
    "PrivateDnsEnabled":true,
    "VpcId":{
      "Ref":"TestVPC"
    }
  }
},
"MyDocDBClusterRotationSecret":{
  "Type":"AWS::SecretsManager::Secret",
  "Properties":{
    "GenerateSecretString":{
      "SecretStringTemplate":"{\"username\": \"someadmin\\\", \"ssl\": true}",
      "GenerateStringKey":"password",
      "PasswordLength":16,
      "ExcludeCharacters":"\"@/\\\" "
    },
    "Tags":[
      {
        "Key":"AppName",
        "Value":"MyApp"
      }
    ]
  }
},
"MyDocDBCluster":{
  "Type":"AWS::DocDB::DBCluster",
  "Properties":{
    "DBSubnetGroupName":{
      "Ref":"MyDBSubnetGroup"
    },
    "MasterUsername":{
      "Fn::Sub":"${resolve:secretsmanager:
${MyDocDBClusterRotationSecret}::username}"
    },
    "MasterUserPassword":{
      "Fn::Sub":"${resolve:secretsmanager:
${MyDocDBClusterRotationSecret}::password}"
    },
    "VpcSecurityGroupIds":[
      {
        "Fn::GetAtt":[
          "TestVPC",
          "DefaultSecurityGroup"
        ]
      }
    ]
  }
},
"DocDBInstance":{
  "Type":"AWS::DocDB::DBInstance",
  "Properties":{
    "DBClusterIdentifier":{
      "Ref":"MyDocDBCluster"
    },
    "DBInstanceClass":"db.r5.large"
  }
},
"MyDBSubnetGroup":{
  "Type":"AWS::DocDB::DBSubnetGroup",
  "Properties":{
    "DBSubnetGroupDescription":"",
    "SubnetIds":[
      {
        "Ref":"TestSubnet01"
      },
      {

```

```
        "Ref": "TestSubnet02"
      }
    ]
  },
  "SecretDocDBClusterAttachment": {
    "Type": "AWS::SecretsManager::SecretTargetAttachment",
    "Properties": {
      "SecretId": {
        "Ref": "MyDocDBClusterRotationSecret"
      },
      "TargetId": {
        "Ref": "MyDocDBCluster"
      },
      "TargetType": "AWS::DocDB::DBCluster"
    }
  },
  "MySecretRotationSchedule": {
    "Type": "AWS::SecretsManager::RotationSchedule",
    "DependsOn": "SecretDocDBClusterAttachment",
    "Properties": {
      "SecretId": {
        "Ref": "MyDocDBClusterRotationSecret"
      },
      "HostedRotationLambda": {
        "RotationType": "MongoDBSingleUser",
        "RotationLambdaName": "MongoDBSingleUser",
        "VpcSecurityGroupIds": {
          "Fn::GetAtt": [
            "TestVPC",
            "DefaultSecurityGroup"
          ]
        },
        "VpcSubnetIds": {
          "Fn::Join": [
            ",",
            [
              {
                "Ref": "TestSubnet01"
              },
              {
                "Ref": "TestSubnet02"
              }
            ]
          ]
        }
      },
      "RotationRules": {
        "Duration": "2h",
        "ScheduleExpression": "cron(0 8 1 * ? *)"
      }
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::SecretsManager-2020-07-23
Resources:
  TestVPC:
    Type: AWS::EC2::VPC
    Properties:
```

```
CidrBlock: 10.0.0.0/16
EnableDnsHostnames: true
EnableDnsSupport: true
TestSubnet01:
  Type: AWS::EC2::Subnet
  Properties:
    CidrBlock: 10.0.96.0/19
    AvailabilityZone:
      Fn::Select:
        - '0'
      - Fn::GetAZs:
          Ref: AWS::Region
    VpcId:
      Ref: TestVPC
TestSubnet02:
  Type: AWS::EC2::Subnet
  Properties:
    CidrBlock: 10.0.128.0/19
    AvailabilityZone:
      Fn::Select:
        - '1'
      - Fn::GetAZs:
          Ref: AWS::Region
    VpcId:
      Ref: TestVPC
SecretsManagerVPCEndpoint:
  Type: AWS::EC2::VPCEndpoint
  Properties:
    SubnetIds:
      - Ref: TestSubnet01
      - Ref: TestSubnet02
    SecurityGroupIds:
      - Fn::GetAtt:
          - TestVPC
          - DefaultSecurityGroup
    VpcEndpointType: Interface
    ServiceName:
      Fn::Sub: com.amazonaws.${AWS::Region}.secretsmanager
    PrivateDnsEnabled: true
    VpcId:
      Ref: TestVPC
MyDocDBClusterRotationSecret:
  Type: AWS::SecretsManager::Secret
  Properties:
    GenerateSecretString:
      SecretStringTemplate: '{"username": "someadmin", "ssl": true}'
      GenerateStringKey: password
      PasswordLength: 16
      ExcludeCharacters: "\"@/\\\"
    Tags:
      - Key: AppName
        Value: MyApp
MyDocDBCluster:
  Type: AWS::DocDB::DBCluster
  Properties:
    DBSubnetGroupName:
      Ref: MyDBSubnetGroup
    MasterUsername:
      Fn::Sub: "${resolve:secretsmanager:${MyDocDBClusterRotationSecret}::username}"
    MasterUserPassword:
      Fn::Sub: "${resolve:secretsmanager:${MyDocDBClusterRotationSecret}::password}"
    VpcSecurityGroupIds:
      - Fn::GetAtt:
          - TestVPC
          - DefaultSecurityGroup
DocDBInstance:
```

```
Type: AWS::DocDB::DBInstance
Properties:
  DBClusterIdentifier:
    Ref: MyDocDBCluster
  DBInstanceClass: db.r5.large
MyDBSubnetGroup:
  Type: AWS::DocDB::DBSubnetGroup
  Properties:
    DBSubnetGroupDescription: ''
    SubnetIds:
      - Ref: TestSubnet01
      - Ref: TestSubnet02
SecretDocDBClusterAttachment:
  Type: AWS::SecretsManager::SecretTargetAttachment
  Properties:
    SecretId:
      Ref: MyDocDBClusterRotationSecret
    TargetId:
      Ref: MyDocDBCluster
    TargetType: AWS::DocDB::DBCluster
MySecretRotationSchedule:
  Type: AWS::SecretsManager::RotationSchedule
  DependsOn: SecretDocDBClusterAttachment
  Properties:
    SecretId:
      Ref: MyDocDBClusterRotationSecret
    HostedRotationLambda:
      RotationType: MongoDBSingleUser
      RotationLambdaName: MongoDBSingleUser
      VpcSecurityGroupIds:
        Fn::GetAtt:
          - TestVPC
          - DefaultSecurityGroup
      VpcSubnetIds:
        Fn::Join:
          - ','
          - - Ref: TestSubnet01
            - Ref: TestSubnet02
    RotationRules:
      Duration: 2h
      ScheduleExpression: 'cron(0 8 1 * ? *)'
```

Retrieve secrets from AWS Secrets Manager

You can retrieve your secrets by using the console (<https://console.aws.amazon.com/secretsmanager/>) or the AWS CLI ([get-secret-value](#)).

In applications, you can retrieve your secrets by calling `GetSecretValue` in any of the AWS SDKs. However, we recommend that you cache your secret values by using client-side caching. Caching secrets improves speed and reduces your costs.

- For Java applications:
 - If you store database credentials in the secret, use the [Secrets Manager SQL connection drivers](#) (p. 86) to connect to a database using the credentials in the secret.
 - For other types of secrets, use the [Secrets Manager Java-based caching component](#) (p. 91).
- For Python applications, use the [Secrets Manager Python-based caching component](#) (p. 96).
- For .NET applications, use the [Secrets Manager .NET-based caching component](#) (p. 100).
- For Go applications, use the [Secrets Manager Go-based caching component](#) (p. 104).
- For JavaScript applications, call the SDK directly with `getSecretValue`.
- For PHP applications, call the SDK directly with `GetSecretValue`.
- For Ruby applications, call the SDK directly with `get_secret_value`.

Within other AWS services

You can also retrieve secrets within other AWS services:

- For Amazon EKS, you can use [AWS Secrets and Configuration Provider \(ASCP\)](#) (p. 108) to mount secrets as files in Amazon EKS.
- For AWS Batch, you can [reference secrets](#) (p. 106) in a job definition.
- For AWS CloudFormation, you can [create secrets](#) (p. 64) and [reference secrets](#) in a CloudFormation stack.
- For Amazon ECS, you can [reference secrets](#) (p. 108) in a container definition.
- For AWS IoT Greengrass, you can [reference secrets](#) (p. 113) in a Greengrass group.
- For Parameter Store, you can [reference secrets](#) (p. 114) in a parameter.

Connect to a SQL database with credentials in an AWS Secrets Manager secret

In Java applications, you can use the Secrets Manager SQL Connection drivers to connect to MySQL, PostgreSQL, Oracle, and MSSQLServer databases using credentials stored in Secrets Manager. Each driver wraps the base JDBC driver, so you can use JDBC calls to access your database. However, instead

of passing a username and password for the connection, you provide the ID of a secret. The driver calls Secrets Manager to retrieve the secret value, and then uses the credentials in the secret to connect to the database. The driver also caches the credentials using the [Java client-side caching library](#) (p. 91), so future connections don't require a call to Secrets Manager. By default, the cache refreshes every hour and also when the secret is rotated. To configure the cache, see [the section called "SecretCacheConfiguration"](#) (p. 93).

You can download the source code from [GitHub](#).

To use the Secrets Manager SQL Connection drivers:

- Your application must be in Java 8 or higher.
- Your secret must be in the [JSON structure of a database secret](#) (p. 51). To check the format, in the Secrets Manager console, view your secret and choose **Retrieve secret value**. Alternatively, in the AWS CLI, call [get-secret-value](#).
- If your database is replicated to other Regions, to connect to a replica database in another Region, you specify the regional endpoint and port when you create the connection. You can store regional connection information in the secret as extra key/value pairs, in SSM Parameter Store parameters, or in your code configuration.

To add the driver to your project, in your Maven build file `pom.xml`, add the following dependency for the driver. For more information, see [Secrets Manager SQL Connection Library](#) on the Maven Repository website.

```
<dependency>
  <groupId>com.amazonaws.secretsmanager</groupId>
  <artifactId>aws-secretsmanager-jdbc</artifactId>
  <version>1.0.5</version>
</dependency>
```

Examples:

- [Establish a connection to a database](#) (p. 87)
- [Establish a connection to a replica database](#) (p. 88)
- [Use c3p0 connection pooling to establish a connection](#) (p. 90)
- [Use c3p0 connection pooling to establish a connection to a replica database](#) (p. 90)

Establish a connection to a database

The following example shows how to establish a connection to a database using the credentials and connection information in a secret. Once you have the connection, you can use JDBC calls to access the database. For more information, see [JDBC Basics](#) on the Java documentation website.

MySQL

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerMySQLDriver" ).newInstance();

// Retrieve the connection info from the secret
String URL = "secretId";

// Populate the user property with the secret ARN to retrieve user and password from
the secret
Properties info = new Properties( );
info.put( "user", "secretId" );
```

```
// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

PostgreSQL

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerPostgreSQLDriver" ).newInstance();

// Retrieve the connection info from the secret
String URL = "secretId";

// Populate the user property with the secret ARN to retrieve user and password from
the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

Oracle

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerOracleDriver" ).newInstance();

// Retrieve the connection info from the secret
String URL = "secretId";

// Populate the user property with the secret ARN to retrieve user and password from
the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

MSSQLServer

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerMSSQLServerDriver" ).newInstance();

// Retrieve the connection info from the secret
String URL = "secretId";

// Populate the user property with the secret ARN to retrieve user and password from
the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

Establish a connection to a replica database

The following example shows how to establish a connection to a database using the credentials in a secret. To connect to a replica database in another Region, you specify the endpoint and port for the connection.

Secrets that are replicated to other Regions can improve latency for the connection to the regional database, but they do not contain different connection information from the source secret. Each replica

is a copy of the source secret. To store regional connection information in the secret, add more key/value pairs for the endpoint and port information for the Regions.

Once you have the connection, you can use JDBC calls to access the database. For more information, see [JDBC Basics](#) on the Java documentation website.

MySQL

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerMySQLDriver" ).newInstance();

// Set the endpoint and port. You can also retrieve it from a key/value pair in the
// secret.
String URL = "jdbc-secretsmanager:mysql://example.com:3306";

// Populate the user property with the secret ARN to retrieve user and password from
// the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

PostgreSQL

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerPostgreSQLDriver" ).newInstance();

// Set the endpoint and port. You can also retrieve it from a key/value pair in the
// secret.
String URL = "jdbc-secretsmanager:postgresql://example.com:5432/database";

// Populate the user property with the secret ARN to retrieve user and password from
// the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

Oracle

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerOracleDriver" ).newInstance();

// Set the endpoint and port. You can also retrieve it from a key/value pair in the
// secret.
String URL = "jdbc-secretsmanager:oracle:thin:@example.com:1521/ORCL";

// Populate the user property with the secret ARN to retrieve user and password from
// the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

MSSQLServer

```
// Load the JDBC driver
Class.forName( "com.amazonaws.secretsmanager.sql.AWSSecretsManagerMSSQLServerDriver" ).newInstance();
```



```
// Set the endpoint and port. You can also retrieve it from a key/value pair in the
secret.
String URL = "jdbc-secretsmanager:sqlserver://example.com:1433";

// Populate the user property with the secret ARN to retrieve user and password from
the secret
Properties info = new Properties( );
info.put( "user", "secretId" );

// Establish the connection
conn = DriverManager.getConnection(URL, info);
```

Use c3p0 connection pooling to establish a connection

The following example shows a `c3p0.properties` file that uses the driver to retrieve credentials and connection information from the secret. For `user` and `jdbcUrl`, enter the secret ID to configure the connection pool. Then you can retrieve connections from the pool and use them as any other database connections. For more information, see [JDBC Basics](#) on the Java documentation website.

For more information about c3p0, see [c3p0](#) on the Machinery For Change website.

MySQL

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerMySQLDriver
c3p0.jdbcUrl=secretId
```

PostgreSQL

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerPostgreSQLDriver
c3p0.jdbcUrl=secretId
```

Oracle

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerOracleDriver
c3p0.jdbcUrl=secretId
```

MSSQLServer

```
c3p0.user=secretId
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerMSSQLServerDriver
c3p0.jdbcUrl=secretId
```

Use c3p0 connection pooling to establish a connection to a replica database

The following example shows a `c3p0.properties` file that uses the driver to retrieve credentials and from the secret. To connect to a replica database in another Region, you specify the endpoint and port directly or by retrieving them from key/value pairs in the secret. Then you can retrieve connections from

the pool and use them as any other database connections. For more information, see [JDBC Basics](#) on the Java documentation website.

MySQL

```
c3p0.user=secretId  
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerMySQLDriver  
c3p0.jdbcUrl=jdbc-secretsmanager:mysql://example.com:3306
```

PostgreSQL

```
c3p0.user=secretId  
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerPostgreSQLDriver  
c3p0.jdbcUrl=jdbc-secretsmanager:postgresql://example.com:5432/database
```

Oracle

```
c3p0.user=secretId  
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerOracleDriver  
c3p0.jdbcUrl=jdbc-secretsmanager:oracle:thin:@example.com:1521/ORCL
```

MSSQLServer

```
c3p0.user=secretId  
c3p0.driverClass=com.amazonaws.secretsmanager.sql.AWSSecretsManagerMSSQLServerDriver  
c3p0.jdbcUrl=jdbc-secretsmanager:sqlserver://example.com:1433
```

Retrieve AWS Secrets Manager secrets in Java applications

When you retrieve a secret, you can use the Secrets Manager Java-based caching component to cache it for future use. Retrieving a cached secret is faster than retrieving it from Secrets Manager. Because there is a cost for calling Secrets Manager APIs, using a cache can reduce your costs.

The cache policy is Least Recently Used (LRU), so when the cache must discard a secret, it discards the least recently used secret. By default, the cache refreshes secrets every hour. You can configure how often the secret is refreshed in the cache, and you can hook into the secret retrieval to add more functionality.

To use the component, you must have the following:

- A Java 8 or higher development environment. See [Java SE Downloads](#) on the Oracle website.
- The AWS SDK for Java. See [the section called “AWS SDKs”](#) (p. 8).

To download the source code, see [Secrets Manager Java-based caching client component](#) on GitHub.

To add the component to your project, in your Maven pom.xml file, include the following dependency. For more information about Maven, see the [Getting Started Guide](#) on the Apache Maven Project website.

```
<dependency>  
  <groupId>com.amazonaws.secretsmanager</groupId>  
  <artifactId>aws-secretsmanager-caching-java</artifactId>  
  <version>1.0.1</version>
```

</dependency>

Reference

- [SecretCache](#) (p. 92)
- [SecretCacheConfiguration](#) (p. 93)
- [SecretCacheHook](#) (p. 95)

Example Example: Retrieve a secret

The following code example shows a Lambda function that retrieves a secret string. It follows the [best practice](#) of instantiating the cache outside of the function handler, so it doesn't keep calling the API if you call the Lambda function again.

```
package com.amazonaws.secretsmanager.caching.examples;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.LambdaLogger;

import com.amazonaws.secretsmanager.caching.SecretCache;

public class SampleClass implements RequestHandler<String, String> {

    private final SecretCache cache = new SecretCache();

    @Override public String handleRequest(String secretId, Context context) {
        final String secret = cache.getSecretString(secretId);

        // Use the secret, return success;

    }
}
```

SecretCache

An in-memory cache for secrets requested from Secrets Manager. You use [the section called "getSecretString" \(p. 93\)](#) or [the section called "getSecretBinary" \(p. 93\)](#) to retrieve a secret from the cache. You can configure the cache settings by passing in a [the section called "SecretCacheConfiguration" \(p. 93\)](#) object in the constructor.

For more information, including examples, see [the section called "Java applications" \(p. 91\)](#).

Constructors

```
public SecretCache()
```

Default constructor for a `SecretCache` object.

```
public SecretCache(AWSSecretsManagerClientBuilder builder)
```

Constructs a new cache using a Secrets Manager client created using the provided [AWSSecretsManagerClientBuilder](#). Use this constructor to customize the Secrets Manager client, for example to use a specific region or endpoint.

```
public SecretCache(AWSSecretsManager client)
```

Constructs a new secret cache using the provided [AWSSecretsManagerClient](#). Use this constructor to customize the Secrets Manager client, for example to use a specific region or endpoint.

```
public SecretCache(SecretCacheConfiguration config)
```

Constructs a new secret cache using the provided [the section called "SecretCacheConfiguration" \(p. 93\)](#).

Methods

getSecretString

```
public String getSecretString(final String secretId)
```

Retrieves a string secret from Secrets Manager. Returns a [String](#).

getSecretBinary

```
public ByteBuffer getSecretBinary(final String secretId)
```

Retrieves a binary secret from Secrets Manager. Returns a [ByteBuffer](#).

refreshNow

```
public boolean refreshNow(final String secretId) throws InterruptedException
```

Forces the cache to refresh. Returns `true` if the refresh completed without error, otherwise `false`.

close

```
public void close()
```

Closes the cache.

SecretCacheConfiguration

Cache configuration options for a [the section called "SecretCache" \(p. 92\)](#), such as max cache size and Time to Live (TTL) for cached secrets.

Constructor

```
public SecretCacheConfiguration
```

Default constructor for a `SecretCacheConfiguration` object.

Methods

getClient

```
public AWSSecretsManager getClient()
```

Returns the [AWSSecretsManagerClient](#) that the cache retrieves secrets from.

setClient

```
public void setClient(AWSSecretsManager client)
```

Sets the [AWSSecretsManagerClient](#) client that the cache retrieves secrets from.

getCacheHook

```
public SecretCacheHook getCacheHook()
```

Returns the [the section called "SecretCacheHook" \(p. 95\)](#) interface used to hook cache updates.

setCacheHook

```
public void setCacheHook(SecretCacheHook cacheHook)
```

Sets the [the section called "SecretCacheHook" \(p. 95\)](#) interface used to hook cache updates.

getMaxCacheSize

```
public int getMaxCacheSize()
```

Returns the maximum cache size. The default is 1024 secrets.

setMaxCacheSize

```
public void setMaxCacheSize(int maxCacheSize)
```

Sets the maximum cache size. The default is 1024 secrets.

getCacheItemTTL

```
public long getCacheItemTTL()
```

Returns the TTL in milliseconds for the cached items. When a cached secret exceeds this TTL, the cache retrieves a new copy of the secret from the [AWSecretsManagerClient](#). The default is 1 hour in milliseconds.

The cache refreshes the secret synchronously when the secret is requested after the TTL. If the synchronous refresh fails, the cache returns the stale secret.

setCacheItemTTL

```
public void setCacheItemTTL(long cacheItemTTL)
```

Sets the TTL in milliseconds for the cached items. When a cached secret exceeds this TTL, the cache retrieves a new copy of the secret from the [AWSecretsManagerClient](#). The default is 1 hour in milliseconds.

getVersionStage

```
public String getVersionStage()
```

Returns the version of secrets that you want to cache. For more information, see [Secret versions \(p. 11\)](#). The default is "AWSCURRENT".

setVersionStage

```
public void setVersionStage(String versionStage)
```

Sets the version of secrets that you want to cache. For more information, see [Secret versions \(p. 11\)](#). The default is "AWSCURRENT".

SecretCacheConfiguration withClient

```
public SecretCacheConfiguration withClient(AWSSecretsManager client)
```

Sets the [AWSSecretsManagerClient](#) to retrieve secrets from. Returns the updated `SecretCacheConfiguration` object with the new setting.

SecretCacheConfiguration withCacheHook

```
public SecretCacheConfiguration withCacheHook(SecretCacheHook cacheHook)
```

Sets the interface used to hook the in-memory cache. Returns the updated `SecretCacheConfiguration` object with the new setting.

SecretCacheConfiguration withMaxCacheSize

```
public SecretCacheConfiguration withMaxCacheSize(int maxCacheSize)
```

Sets the maximum cache size. Returns the updated `SecretCacheConfiguration` object with the new setting.

SecretCacheConfiguration withCacheItemTTL

```
public SecretCacheConfiguration withCacheItemTTL(long cacheItemTTL)
```

Sets the TTL in milliseconds for the cached items. When a cached secret exceeds this TTL, the cache retrieves a new copy of the secret from the [AWSSecretsManagerClient](#). The default is 1 hour in milliseconds. Returns the updated `SecretCacheConfiguration` object with the new setting.

SecretCacheConfiguration withVersionStage

```
public SecretCacheConfiguration withVersionStage(String versionStage)
```

Sets the version of secrets that you want to cache. For more information, see [Secret versions \(p. 11\)](#). Returns the updated `SecretCacheConfiguration` object with the new setting.

SecretCacheHook

An interface to hook into a [the section called “SecretCache” \(p. 92\)](#) to perform actions on the secrets being stored in the cache.

put

```
Object put(final Object o)
```

Prepare the object for storing in the cache.

Returns the object to store in the cache.

get

```
Object get(final Object cachedObject)
```

Derive the object from the cached object.

Returns the object to return from the cache

Retrieve AWS Secrets Manager secrets in Python applications

When you retrieve a secret, you can use the Secrets Manager Python-based caching component to cache it for future use. Retrieving a cached secret is faster than retrieving it from Secrets Manager. Because there is a cost for calling Secrets Manager APIs, using a cache can reduce your costs.

The cache policy is Least Recently Used (LRU), so when the cache must discard a secret, it discards the least recently used secret. By default, the cache refreshes secrets every hour. You can configure how often the secret is refreshed in the cache, and you can hook into the secret retrieval to add more functionality.

To use the component, you must have the following:

- Python 3.6 or later.
- boto3 1.12 or higher. See [AWS SDK for Python](#) and [Boto3](#).
- setuptools_scm 3.2 or higher. See <https://pypi.org/project/setuptools-scm/>.

To download the source code, see [Secrets Manager Python-based caching client component](#) on GitHub.

To install the component, use the following command.

```
$ pip install aws-secretsmanager-caching
```

Reference

- [SecretCache](#) (p. 96)
- [SecretCacheConfig](#) (p. 97)
- [SecretCacheHook](#) (p. 98)
- [@InjectSecretString](#) (p. 99)
- [@InjectKeywordedSecretString](#) (p. 99)

Example Example: Retrieve a secret

The following example shows how to get the secret value for a secret named *mysecret*.

```
import boto3
import boto3.session
from aws_secretsmanager_caching import SecretCache, SecretCacheConfig

client = boto3.session.get_session().create_client('secretsmanager')
cache_config = SecretCacheConfig()
cache = SecretCache( config = cache_config, client = client)

secret = cache.get_secret_string('mysecret')
```

SecretCache

An in-memory cache for secrets retrieved from Secrets Manager. You use [the section called "get_secret_string" \(p. 97\)](#) or [the section called "get_secret_binary" \(p. 97\)](#) to retrieve a secret from the cache. You can configure the cache settings by passing in a [the section called "SecretCacheConfig" \(p. 97\)](#) object in the constructor.

For more information, including examples, see [the section called “Python applications” \(p. 96\)](#).

```
cache = SecretCache(  
    config = the section called “SecretCacheConfig”,  
    client = client  
)
```

These are the available methods:

- [get_secret_string](#) (p. 97)
- [get_secret_binary](#) (p. 97)

get_secret_string

Retrieves the secret string value.

Request syntax

```
response = cache.get_secret_string(  
    secret_id='string',  
    version_stage='string' )
```

Parameters

- `secret_id` (*string*) -- [Required] The name or ARN of the secret.
- `version_stage` (*string*) -- The version of secrets that you want to retrieve. For more information, see [Secret versions](#). The default is 'AWSCURRENT'.

Return type

string

get_secret_binary

Retrieves the secret binary value.

Request syntax

```
response = cache.get_secret_binary(  
    secret_id='string',  
    version_stage='string'  
)
```

Parameters

- `secret_id` (*string*) -- [Required] The name or ARN of the secret.
- `version_stage` (*string*) -- The version of secrets that you want to retrieve. For more information, see [Secret versions](#). The default is 'AWSCURRENT'.

Return type

[base64-encoded](#) string

SecretCacheConfig

Cache configuration options for a [the section called “SecretCache” \(p. 96\)](#) such as max cache size and Time to Live (TTL) for cached secrets.

Parameters

`max_cache_size` (*int*)

The maximum cache size. The default is 1024 secrets.

`exception_retry_delay_base` (*int*)

The number of seconds to wait after an exception is encountered before retrying the request. The default is 1.

`exception_retry_growth_factor` (*int*)

The growth factor to use for calculating the wait time between retries of failed requests. The default is 2.

`exception_retry_delay_max` (*int*)

The maximum amount of time in seconds to wait between failed requests. The default is 3600.

`default_version_stage` (*str*)

The version of secrets that you want to cache. For more information, see [Secret versions \(p. 11\)](#). The default is 'AWSCURRENT'.

`secret_refresh_interval` (*int*)

The number of seconds to wait between refreshing cached secret information. The default is 3600.

`secret_cache_hook` (*SecretCacheHook*)

An implementation of the `SecretCacheHook` abstract class. The default value is `None`.

SecretCacheHook

An interface to hook into a [the section called "SecretCache" \(p. 96\)](#) to perform actions on the secrets being stored in the cache.

These are the available methods:

- [put \(p. 98\)](#)
- [get \(p. 99\)](#)

put

Prepares the object for storing in the cache.

Request syntax

```
response = hook.put(  
    obj='secret_object'  
)
```

Parameters

- `obj` (*object*) -- [Required] The secret or object that contains the secret.

Return type

object

get

Derives the object from the cached object.

Request syntax

```
response = hook.get(  
    obj='secret_object'  
)
```

Parameters

- `obj (object)` -- [Required] The secret or object that contains the secret.

Return type

object

@InjectSecretString

This decorator expects a secret ID string and [the section called "SecretCache" \(p. 96\)](#) as the first and second arguments. The decorator returns the secret string value. The secret must contain a string.

```
from aws_secretsmanager_caching import SecretCache  
from aws_secretsmanager_caching import InjectKeywordedSecretString, InjectSecretString  
  
cache = SecretCache()  
  
@InjectSecretString ( 'mysecret' , cache ) def function_to_be_decorated( arg1, arg2,  
    arg3):
```

@InjectKeywordedSecretString

This decorator expects a secret ID string and [the section called "SecretCache" \(p. 96\)](#) as the first and second arguments. The remaining arguments map parameters from the wrapped function to JSON keys in the secret. The secret must contain a string in JSON structure.

For a secret that contains this JSON:

```
{  
    "username": "saanvi",  
    "password": "EXAMPLE-PASSWORD"  
}
```

The following example shows how to extract the JSON values for username and password from the secret.

```
from aws_secretsmanager_caching import SecretCache  
from aws_secretsmanager_caching import InjectKeywordedSecretString, InjectSecretString  
  
cache = SecretCache()  
  
@InjectKeywordedSecretString ( secret_id = 'mysecret' , cache = cache , func_username =  
'username' , func_password = 'password' ) def function_to_be_decorated( func_username,  
    func_password):  
    print( 'Do something with the func_username and func_password parameters')
```

Retrieve AWS Secrets Manager secrets in .NET applications

When you retrieve a secret, you can use the Secrets Manager .NET-based caching component to cache it for future use. Retrieving a cached secret is faster than retrieving it from Secrets Manager. Because there is a cost for calling Secrets Manager APIs, using a cache can reduce your costs.

The cache policy is Least Recently Used (LRU), so when the cache must discard a secret, it discards the least recently used secret. By default, the cache refreshes secrets every hour. You can configure how often the secret is refreshed in the cache, and you can hook into the secret retrieval to add more functionality.

To use the component, you must have the following:

- .NET Framework 4.6.1 or higher, or .NET Standard 2.0 or higher. See [Download .NET](#) on the Microsoft .NET website.
- The AWS SDK for .NET. See [the section called "AWS SDKs" \(p. 8\)](#).

To download the source code, see [Caching client for .NET](#) on GitHub.

To use the cache, first instantiate it, then retrieve your secret by using `GetSecretString` or `GetSecretBinary`. On successive retrievals, the cache returns the cached copy of the secret.

To get the package from NuGet:

```
<ItemGroup>
  <PackageReference Include="AWSSDK.SecretsManager.Caching" Version="1.0.3" />
</ItemGroup>
```

Reference

- [SecretsManagerCache \(p. 101\)](#)
- [SecretCacheConfiguration \(p. 102\)](#)
- [ISecretCacheHook \(p. 103\)](#)

Example Example: Retrieve a secret

The following code example shows a method that retrieves a secret named *MySecret*.

```
using System;
using Amazon.SecretsManager.Extensions.Caching.SecretsManagerCache;

namespace LambdaExample {
    public class CachingExample
    {
        private SecretsManagerCache cache = new SecretsManagerCache();
        private const String MySecretName = "MySecret";

        public async Task<Response> FunctionHandlerAsync(String input, ILambdaContext context)
        {
            String MySecret = await cache.GetSecretString(MySecretName);

            // Use the secret, return success
        }
    }
}
```

```
}  
}
```

SecretsManagerCache

An in-memory cache for secrets requested from Secrets Manager. You use [the section called "GetSecretString" \(p. 102\)](#) or [the section called "GetSecretBinary" \(p. 102\)](#) to retrieve a secret from the cache. You can configure the cache settings by passing in a [the section called "SecretCacheConfiguration" \(p. 102\)](#) object in the constructor.

For more information, including examples, see [the section called ".NET applications" \(p. 100\)](#).

Constructors

```
public SecretsManagerCache()
```

Default constructor for a `SecretsManagerCache` object.

```
public SecretsManagerCache(IAmazonSecretsManager secretsManager)
```

Constructs a new cache using a Secrets Manager client created using the provided [AmazonSecretsManagerClient](#). Use this constructor to customize the Secrets Manager client, for example to use a specific region or endpoint.

Parameters

`secretsManager`

The [AmazonSecretsManagerClient](#) to retrieve secrets from.

```
public SecretsManagerCache(SecretCacheConfiguration config)
```

Constructs a new secret cache using the provided [the section called "SecretCacheConfiguration" \(p. 102\)](#). Use this constructor to configure the cache, for example the number of secrets to cache and how often it refreshes.

Parameters

`config`

A [the section called "SecretCacheConfiguration" \(p. 102\)](#) that contains configuration information for the cache.

```
public SecretsManagerCache(IAmazonSecretsManager secretsManager,  
SecretCacheConfiguration config)
```

Constructs a new cache using a Secrets Manager client created using the provided [AmazonSecretsManagerClient](#) and a [the section called "SecretCacheConfiguration" \(p. 102\)](#). Use this constructor to customize the Secrets Manager client, for example to use a specific region or endpoint as well as configure the cache, for example the number of secrets to cache and how often it refreshes.

Parameters

`secretsManager`

The [AmazonSecretsManagerClient](#) to retrieve secrets from.

`config`

A [the section called "SecretCacheConfiguration" \(p. 102\)](#) that contains configuration information for the cache.

Methods

GetSecretString

```
public async Task<String> GetSecretString(String secretId)
```

Retrieves a string secret from Secrets Manager.

Parameters

secretId

The ARN or name of the secret to retrieve.

GetSecretBinary

```
public async Task<byte[]> GetSecretBinary(String secretId)
```

Retrieves a binary secret from Secrets Manager.

Parameters

secretId

The ARN or name of the secret to retrieve.

RefreshNowAsync

```
public async Task<bool> RefreshNowAsync(String secretId)
```

Requests the secret value from Secrets Manager and updates the cache with any changes. If there is no existing cache entry, creates a new one. Returns `true` if the refresh is successful.

Parameters

secretId

The ARN or name of the secret to retrieve.

GetCachedSecret

```
public SecretCacheItem GetCachedSecret(string secretId)
```

Returns the cache entry for the specified secret if it exists in the cache. Otherwise, retrieves the secret from Secrets Manager and creates a new cache entry.

Parameters

secretId

The ARN or name of the secret to retrieve.

SecretCacheConfiguration

Cache configuration options for a [the section called "SecretsManagerCache" \(p. 101\)](#), such as maximum cache size and Time to Live (TTL) for cached secrets.

Properties

CacheItemTTL

```
public uint CacheItemTTL { get; set; }
```

The TTL of a cache item in milliseconds. The default is 3600000 ms or 1 hour.

MaxCacheSize

```
public ushort MaxCacheSize { get; set; }
```

The maximum cache size. The default is 1024 secrets.

VersionStage

```
public string VersionStage { get; set; }
```

The version of secrets that you want to cache. For more information, see [Secret versions \(p. 11\)](#). The default is "AWSCURRENT".

Client

```
public IAmazonSecretsManager Client { get; set; }
```

The [AmazonSecretsManagerClient](#) to retrieve secrets from. If it is null, the cache instantiates a new client. The default is null.

CacheHook

```
public ISecretCacheHook CacheHook { get; set; }
```

A [the section called "ISecretCacheHook" \(p. 103\)](#).

ISecretCacheHook

An interface to hook into a [the section called "SecretsManagerCache" \(p. 101\)](#) to perform actions on the secrets being stored in the cache.

Methods

Put

```
object Put(object o);
```

Prepare the object for storing in the cache.

Returns the object to store in the cache.

Get

```
object Get(object cachedObject);
```

Derive the object from the cached object.

Returns the object to return from the cache

Retrieve AWS Secrets Manager secrets in Go applications

When you retrieve a secret, you can use the Secrets Manager Go-based caching component to cache it for future use. Retrieving a cached secret is faster than retrieving it from Secrets Manager. Because there is a cost for calling Secrets Manager APIs, using a cache can reduce your costs.

The cache policy is Least Recently Used (LRU), so when the cache must discard a secret, it discards the least recently used secret. By default, the cache refreshes secrets every hour. You can configure how often the secret is refreshed in the cache, and you can hook into the secret retrieval to add more functionality.

To use the component, you must have the following:

- AWS SDK for Go. See [the section called “AWS SDKs” \(p. 8\)](#).

To download the source code, see [Secrets Manager Go caching client](#) on GitHub.

To set up a Go development environment, see [Golang Getting Started](#) on the Go Programming Language website.

Reference

- [type Cache \(p. 104\)](#)
- [type CacheConfig \(p. 106\)](#)
- [type CacheHook \(p. 106\)](#)

Example Example: Retrieve a secret

The following code example shows a Lambda function that retrieves a secret.

```
package main

import (
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-secretsmanager-caching-go/secretcache"
)

var (
    secretCache, _ = secretcache.New()
)

func HandleRequest(secretId string) string {
    result, _ := secretCache.GetSecretString(secretId)

    // Use the secret, return success
}

func main() {
    lambda.Start(HandleRequest)
}
```

type Cache

An in-memory cache for secrets requested from Secrets Manager. You use [the section called “GetSecretString” \(p. 105\)](#) or [the section called “GetSecretBinary” \(p. 105\)](#) to retrieve a secret from the cache.

The following example shows how to configure the cache settings.

```
// Create a custom secretsmanager client
client := getCustomClient()

// Create a custom CacheConfig struct
config := secretcache.CacheConfig{
    MaxCacheSize: secretcache.DefaultMaxCacheSize + 10,
    VersionStage: secretcache.DefaultVersionStage,
    CacheItemTTL: secretcache.DefaultCacheItemTTL,
}

// Instantiate the cache
cache, _ := secretcache.New(
    func( c *secretcache.Cache) { c.CacheConfig = config },
    func( c *secretcache.Cache) { c.Client = client },
)
```

For more information, including examples, see [the section called “Go applications” \(p. 104\)](#).

Methods

New

```
func New(optFns ...func(*Cache)) (*Cache, error)
```

New constructs a secret cache using functional options, uses defaults otherwise. Initializes a SecretsManager Client from a new session. Initializes CacheConfig to default values. Initialises LRU cache with a default max size.

GetSecretString

```
func (c *Cache) GetSecretString(secretId string) (string, error)
```

GetSecretString gets the secret string value from the cache for given secret ID. Returns the secret sting and an error if operation failed.

GetSecretStringWithStage

```
func (c *Cache) GetSecretStringWithStage(secretId string, versionStage string)
(string, error)
```

GetSecretStringWithStage gets the secret string value from the cache for given secret ID and [version stage \(p. 11\)](#). Returns the secret sting and an error if operation failed.

GetSecretBinary

```
func (c *Cache) GetSecretBinary(secretId string) ([]byte, error) {
```

GetSecretBinary gets the secret binary value from the cache for given secret ID. Returns the secret binary and an error if operation failed.

GetSecretBinaryWithStage

```
func (c *Cache) GetSecretBinaryWithStage(secretId string, versionStage string)
([]byte, error)
```

GetSecretBinaryWithStage gets the secret binary value from the cache for given secret ID and [version stage \(p. 11\)](#). Returns the secret binary and an error if operation failed.

type CacheConfig

Cache configuration options for a [Cache \(p. 104\)](#), such as maximum cache size, default [version stage \(p. 11\)](#), and Time to Live (TTL) for cached secrets.

```
type CacheConfig struct {  
  
    // The maximum cache size. The default is 1024 secrets.  
    MaxCacheSize int  
  
    // The TTL of a cache item in nanoseconds. The default is  
    // 3.6e10^12 ns or 1 hour.  
    CacheItemTTL int64  
  
    // The version of secrets that you want to cache. The default  
    // is "AWSCURRENT".  
    VersionStage string  
  
    // Used to hook in-memory cache updates.  
    Hook CacheHook  
}
```

type CacheHook

An interface to hook into a [Cache \(p. 104\)](#) to perform actions on the secret being stored in the cache.

Methods

Put

```
Put(data interface{}) interface{}
```

Prepares the object for storing in the cache.

Get

```
Get(data interface{}) interface{}
```

Derives the object from the cached object.

Use AWS Secrets Manager secrets in AWS Batch

AWS Batch helps you to run batch computing workloads on the AWS Cloud. With AWS Batch, you can inject sensitive data into your jobs by storing your sensitive data in AWS Secrets Manager secrets and then referencing them in your job definition. For more information, see [Specifying sensitive data using Secrets Manager](#).

Retrieve a secret in an AWS CloudFormation resource

With AWS CloudFormation, you can retrieve a secret to use in another AWS CloudFormation resource. A common scenario is to first create a secret with a password generated by Secrets Manager, and

then retrieve the username and password from the secret to use as credentials for a new database. For information about creating secrets with AWS CloudFormation, see [the section called “AWS CloudFormation”](#) (p. 64).

To retrieve a secret in a AWS CloudFormation template, you use a *dynamic reference*.

A dynamic reference for a secret has the following pattern:

```
{{resolve:secretsmanager:secret-id:SecretString:json-key:version-stage:version-id}}
```

secret-id

The name or ARN of the secret. To access a secret in your AWS account, you can use the secret name. To access a secret in a different AWS account, use the ARN of the secret.

json-key (Optional)

The key name of the key-value pair whose value you want to retrieve. If you don't specify a `json-key`, AWS CloudFormation retrieves the entire secret text. This segment may not include the colon character (:).

version-stage (Optional)

The [version](#) (p. 11) of the secret to use. Secrets Manager uses staging labels to keep track of different versions during the rotation process. If you use `version-stage` then don't specify `version-id`. If you don't specify either `version-stage` or `version-id`, then the default is the `AWSCURRENT` version. This segment may not include the colon character (:).

version-id (Optional)

The unique identifier of the version of the secret to use. If you specify `version-id`, then don't specify `version-stage`. If you don't specify either `version-stage` or `version-id`, then the default is the `AWSCURRENT` version. This segment may not include the colon character (:).

For more information, see [Using dynamic references to specify Secrets Manager secrets](#).

Example: Use a secret to set a database password

This example retrieves the `username` and `password` values stored in the `MyRDSSECRET` secret and uses them as the username and password for the Amazon RDS DB instance.

The `MyRDSSECRET` secret value looks like this:

```
{
  "engine": "mysql",
  "username": "admin",
  "password": "EXAMPLE-PASSWORD",
  "host": "my-database-endpoint.us-east-2.rds.amazonaws.com",
  "dbname": "myDatabase",
  "port": "3306"
}
```

For information about creating resources with AWS CloudFormation, see [Learn template basics](#) in the AWS CloudFormation User Guide.

JSON

```
{
```

```

    "MyRDSInstance": {
      "Type": "AWS::RDS::DBInstance",
      "Properties": {
        "DBName": "MyRDSInstance",
        "AllocatedStorage": "20",
        "DBInstanceClass": "db.t2.micro",
        "Engine": "mysql",
        "MasterUsername":
      "{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}",
        "MasterUserPassword":
      "{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}"
      }
    }
  }
}

```

YAML

```

MyRDSInstance:
  Type: 'AWS::RDS::DBInstance'
  Properties:
    DBName: MyRDSInstance
    AllocatedStorage: '20'
    DBInstanceClass: db.t2.micro
    Engine: mysql
    MasterUsername: '{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}'
    MasterUserPassword: '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}'

```

Use AWS Secrets Manager secrets in Amazon Elastic Container Service

Amazon Elastic Container Service (Amazon ECS) is a highly scalable and fast container management service. With Amazon ECS, you can inject sensitive data into your containers by storing it in Secrets Manager secrets and then referencing the secrets in your container definition. Sensitive data stored in Secrets Manager secrets can be exposed to a container as environment variables or as part of the log configuration. For more information, see [Specifying sensitive data using Secrets Manager](#) and [Tutorial: Specifying Sensitive Data Using Secrets Manager Secrets](#).

Use AWS Secrets Manager secrets in Amazon Elastic Kubernetes Service

To show secrets from Secrets Manager as files mounted in [Amazon EKS](#) pods, you can use the AWS Secrets and Configuration Provider (ASCP) for the [Kubernetes Secrets Store CSI Driver](#). The ASCP works with Amazon Elastic Kubernetes Service (Amazon EKS) 1.17+.

With the ASCP, you can store and manage your secrets in Secrets Manager and then retrieve them through your workloads running on Amazon EKS. If your secret contains multiple key/value pairs in JSON format, you can choose which ones to mount in Amazon EKS. The ASCP uses [JMESPath syntax](#) to query the key/value pairs in your secret.

You can use IAM roles and policies to limit access to your secrets to specific Amazon EKS pods in a cluster. The ASCP retrieves the pod identity and exchanges the identity for an IAM role. ASCP assumes

the IAM role of the pod, and then it can retrieve secrets from Secrets Manager that are authorized for that role.

If you use Secrets Manager automatic rotation for your secrets, you can also use the Secrets Store CSI Driver rotation reconciler feature to ensure you are retrieving the latest secret from Secrets Manager. For more information, see [Auto rotation of mounted contents and synced Kubernetes Secrets](#).

For a tutorial about how to use the ASCP, see [the section called "Tutorial" \(p. 111\)](#).

To learn how to integrate Parameter Store with Amazon EKS, see [Use Parameter Store parameters in Amazon Elastic Kubernetes Service](#).

Install the ASCP

The ASCP is available on GitHub in the [secrets-store-csi-provider-aws](#) repository. The repo also contains example YAML files for creating and mounting a secret. You first install the Kubernetes Secrets Store CSI Driver, and then you install the ASCP.

To install the ASCP

1. To install the Secrets Store CSI Driver, run the following commands. For full installation instructions, see [Installation](#) in the Secrets Store CSI Driver Book.

```
helm repo add secrets-store-csi-driver https://kubernetes-sigs.github.io/secrets-store-csi-driver/charts
helm install -n kube-system csi-secrets-store secrets-store-csi-driver/secrets-store-csi-driver
```

2. To install the ASCP, use the YAML file in the GitHub repo deployment directory.

```
kubectl apply -f https://raw.githubusercontent.com/aws/secrets-store-csi-driver-provider-aws/main/deployment/aws-provider-installer.yaml
```

Step 1: Set up access control

To grant your Amazon EKS pod access to secrets in Secrets Manager, you first create a policy that limits access to the secrets that the pod needs to access. The policy must include `secretsmanager:GetSecretValue` and `secretsmanager:DescribeSecret` permission. Then you create an [IAM role for service account](#) and attach the policy to it.

The ASCP retrieves the pod identity and exchanges it for the IAM role. ASCP assumes the IAM role of the pod, which gives it access to the secrets you authorized. Other containers can't access the secrets unless you also associate them with the IAM role.

For information about creating policies, see [the section called "Attach a permissions policy to an identity" \(p. 28\)](#).

For a tutorial about how to use the ASCP, see [the section called "Tutorial" \(p. 111\)](#).

Step 2: Mount secrets in Amazon EKS

To show secrets in Amazon EKS as though they are files on the filesystem, you create a `SecretProviderClass` YAML file that contains information about your secrets and how to display them in the Amazon EKS pod.

The `SecretProviderClass` must be in the same namespace as the Amazon EKS pod it references.

If Amazon EKS does not have internet access, for the provider to access Secrets Manager, you need to set up a [VPC endpoint \(p. 137\)](#).

For a tutorial about how to use the ASCP, see [the section called "Tutorial" \(p. 111\)](#).

SecretProviderClass

The SecretProviderClass YAML has the following format:

```
apiVersion: secrets-store.csi.x-k8s.io/v1alpha1
kind: SecretProviderClass
metadata:
  name: <NAME>
spec:
  provider: aws
  parameters:
```

parameters

Contains the details of the mount request.

objects

A string containing a YAML declaration of the secrets to be mounted. We recommend using a YAML multi-line string or pipe (|) character, as shown in [the section called "Example" \(p. 111\)](#).

objectName

The name or full ARN of the secret. If you use the ARN, you can omit objectType. This becomes the file name of the secret in the Amazon EKS pod unless you specify objectAlias.

jmesPath

(Optional) A map of the keys in the secret to the files to be mounted in Amazon EKS. To use this field, your secret value must be in JSON format. If you use this field, you must include the subfields path and objectAlias.

path

A key from a key/value pair in the JSON of the secret value.

objectAlias

The file name to be mounted in the Amazon EKS pod.

objectType

Required if you don't use a Secrets Manager ARN for objectName. Can be either secretsmanager or ssmparameter.

objectAlias

(Optional) The file name of the secret in the Amazon EKS pod. If you don't specify this field, the objectName appears as the file name.

objectVersion

(Optional) The version ID of the secret. We recommend you don't use this field because you must update it every time you update the secret. By default the most recent version is used.

objectVersionLabel

(Optional) The alias for the version. The default is the most recent version AWSCURRENT. For more information, see [the section called "Version" \(p. 11\)](#).

region

(Optional) The AWS Region of the secret. If you don't use this field, the ASCP looks up the Region from the annotation on the node. This lookup adds overhead to mount requests, so we recommend that you provide the Region for clusters that use large numbers of pods.

pathTranslation

(Optional) A single substitution character to use if the file name (either `objectName` or `objectAlias`) contains the path separator character, such as slash (/) on Linux. If a secret contains the path separator, the ASCP will not be able to create a mounted file with that name. Instead, you can replace the path separator character with a different character by entering it in this field. If you don't use this field, the default is underscore (_), so for example, `My/Path/Secret` mounts as `My_Path_Secret`.

To prevent character substitution, enter the string `False`.

Example

The following example shows a `SecretProviderClass` that mounts six files in Amazon EKS:

1. A secret specified by full ARN.
2. The `username` key/value pair from the same secret.
3. The `password` key/value pair from the same secret.
4. A secret specified by full ARN.
5. A secret specified by name.
6. A specific version of a secret.

```
apiVersion: secrets-store.csi.x-k8s.io/v1alpha1
kind: SecretProviderClass
metadata:
  name: aws-secrets
spec:
  provider: aws
  parameters:
    objects: |
      - objectName: "arn:aws:secretsmanager:us-east-2:
[111122223333]:secret:MySecret-00AACC"
        jmesPath:
          - path: username
            objectAlias: dbusername
          - path: password
            objectAlias: dbpassword
      - objectName: "arn:aws:secretsmanager:us-east-2:
[111122223333]:secret:MySecret2-00AABB"
        - objectName: "MySecret3"
          objectType: "secretsmanager"
        - objectName: "MySecret4"
          objectType: "secretsmanager"
          objectVersionLabel: "AWSCURRENT"
```

Tutorial: Create and mount a secret in an Amazon EKS pod

In this tutorial, you create an example secret in Secrets Manager, and then you mount the secret in an Amazon EKS pod and deploy it.

Before you begin, install the ASCP: [the section called “Install the ASCP” \(p. 109\)](#).

To create and mount a secret

1. Set the AWS Region and the name of your cluster as shell variables so you can use them in bash commands. For **<REGION>**, enter the AWS Region where your Amazon EKS cluster runs. For **<CLUSTERNAME>**, enter the name of your cluster.

```
REGION=<REGION>
CLUSTERNAME=<CLUSTERNAME>
```

2. Create a test secret. For more information, see [Create and manage secrets \(p. 49\)](#).

```
aws --region "$REGION" secretsmanager create-secret --name MySecret --secret-string
'{"username":"lijuan", "password":"hunter2"}'
```

3. Create a resource policy for the pod that limits its access to the secret you created in the previous step. For **<SECRETARN>**, use the ARN of the secret. Save the policy ARN in a shell variable.

```
POLICY_ARN=$(aws --region "$REGION" --query Policy.Arn --output text iam create-policy
--policy-name nginx-deployment-policy --policy-document '{
  "Version": "2012-10-17",
  "Statement": [ {
    "Effect": "Allow",
    "Action": ["secretsmanager:GetSecretValue", "secretsmanager:DescribeSecret"],
    "Resource": [ "<SECRETARN>" ]
  } ]
}')'
```

4. Create an IAM OIDC provider for the cluster if you don't already have one. For more information, see [Create an IAM OIDC provider for your cluster](#).

```
eksctl utils associate-iam-oidc-provider --region="$REGION" --cluster="$CLUSTERNAME" --
approve # Only run this once
```

5. Create the service account the pod uses and associate the resource policy you created in step 3 with that service account. For this tutorial, for the service account name, you use *nginx-deployment-sa*. For more information, see [Create an IAM role for a service account](#).

```
eksctl create iamserviceaccount --name nginx-deployment-sa --region="$REGION" --cluster
"$CLUSTERNAME" --attach-policy-arn "$POLICY_ARN" --approve --override-existing-
serviceaccounts
```

6. Create the `SecretProviderClass` to specify which secret to mount in the pod. The following command uses `ExampleSecretProviderClass.yaml` in the [ASCP GitHub repo examples](#) directory to mount the secret you created in step 1. For information about creating your own `SecretProviderClass`, see [the section called “SecretProviderClass” \(p. 110\)](#).

```
kubectl apply -f https://raw.githubusercontent.com/aws/secrets-store-csi-driver-
provider-aws/main/examples/ExampleSecretProviderClass.yaml
```

7. Deploy your pod. The following command uses `ExampleDeployment.yaml` in the [ASCP GitHub repo examples](#) directory to mount the secret in `/mnt/secrets-store` in the pod.

```
kubectl apply -f https://raw.githubusercontent.com/aws/secrets-store-csi-driver-
provider-aws/main/examples/ExampleDeployment.yaml
```

8. To verify the secret has been mounted properly, use the following command and confirm that your secret value appears.

```
kubectl exec -it $(kubectl get pods | awk '/nginx-deployment/{print $1}' | head -1)
cat /mnt/secrets-store/MySecret; echo
```

The secret value appears.

```
{"username": "lijuan", "password": "hunter2"}
```

Troubleshoot

You can view most errors by describing the pod deployment.

To see error messages for your container

1. Get a list of pod names with the following command. If you aren't using the default namespace, use `-n <NAMESPACE>`.

```
kubectl get pods
```

2. To describe the pod, in the following command, for `<PODID>` use the pod ID from the pods you found in the previous step. If you aren't using the default namespace, use `-n <NAMESPACE>`.

```
kubectl describe pod/<PODID>
```

To see errors for the ASCP

- To find more information in the provider logs, in the following command, for `<PODID>` use the ID of the `csi-secrets-store-provider-aws` pod.

```
kubectl -n kube-system get pods
kubectl -n kube-system logs pod/<PODID>
```

Use AWS Secrets Manager secrets in AWS IoT Greengrass

AWS IoT Greengrass is software that extends cloud capabilities to local devices. This enables devices to collect and analyze data closer to the source of information, react autonomously to local events, and communicate securely with each other on local networks.

AWS IoT Greengrass lets you authenticate with services and applications from Greengrass devices without hard-coding passwords, tokens, or other secrets. You can use AWS Secrets Manager to securely store and manage your secrets in the cloud. AWS IoT Greengrass extends Secrets Manager to Greengrass core devices, so your connectors and Lambda functions can use local secrets to interact with services and applications.

To integrate a secret into a Greengrass group, you create a group resource that references the Secrets Manager secret. This secret resource references the cloud secret by using the associated ARN. To learn how to create, manage, and use secret resources, see [Working with Secret Resources](#) in the AWS IoT Developer Guide.

To deploy secrets to the AWS IoT Greengrass Core, see [Deploy secrets to the AWS IoT Greengrass core](#).

Use AWS Secrets Manager secrets in Parameter Store

AWS Systems Manager Parameter Store provides secure, hierarchical storage for configuration data management and secrets management. You can store data such as passwords, database strings, and license codes as parameter values. However, Parameter Store doesn't provide automatic rotation services for stored secrets. Instead, Parameter Store enables you to store your secret in Secrets Manager, and then reference the secret as a Parameter Store parameter.

When you configure Parameter Store with Secrets Manager, the `secret-id` Parameter Store requires a forward slash (/) before the name-string.

For more information, see [Referencing AWS Secrets Manager Secrets from Parameter Store Parameters](#) in the *AWS Systems Manager User Guide*.

Rotate AWS Secrets Manager secrets

Rotation is the process of periodically updating a secret. When you rotate a secret, you update the credentials in both the secret and the database or service. In Secrets Manager, you can set up automatic rotation for your secrets. Applications that [retrieve the secret](#) from Secrets Manager automatically get the new credentials after rotation.

To turn on automatic rotation, you need administrator permissions. See [the section called “Secrets Manager administrator permissions”](#) (p. 27).

Topics

- [Rotation strategies](#) (p. 115)
- [Automatically rotate an Amazon RDS, Amazon DocumentDB, or Amazon Redshift secret](#) (p. 117)
- [Automatically rotate a secret](#) (p. 119)
- [Schedule expressions in Secrets Manager rotation](#) (p. 120)
- [Rotate a secret immediately](#) (p. 122)
- [How rotation works](#) (p. 122)
- [Network access for the rotation function](#) (p. 123)
- [Permissions for rotation](#) (p. 124)
- [Customize a Lambda rotation function for Secrets Manager](#) (p. 127)
- [Secrets Manager rotation function templates](#) (p. 128)
- [Troubleshoot AWS Secrets Manager rotation of secrets](#) (p. 132)

Rotation strategies

There are two rotation strategies offered by Secrets Manager:

- [the section called “Single user”](#) (p. 115)
- [the section called “Alternating users”](#) (p. 116)

Single user rotation strategy

The single user strategy updates credentials for one user in one secret.

This is the simplest rotation strategy, and it is appropriate for most use cases. You can use single-user rotation for:

- Accessing databases. Database connections are not dropped when a secret rotates, and new connections after rotation use the new credentials.
- Accessing services that allow the user to create one user account, for example with email address as the user name. The service typically allows the user to change the password as often as required, but the user can't create additional users or change their user name.
- Users created as necessary, called *ad-hoc users*.

- Users who enter their password interactively instead of having an application programmatically retrieve it from Secrets Manager. This type of user does not expect to have to change their user name as well as password.

For detailed instructions, see [the section called “Single user rotation” \(p. 17\)](#).

While this type of rotation is happening, there is a short period of time between when the password in the database changes and when the corresponding secret updates. In this time, there is a low risk of the database denying calls that use the rotated credentials. You can mitigate this risk with an [appropriate retry strategy](#).

To use this strategy, the user in your secret must have permission to update their password.

To use the single user rotation strategy

1. Create a secret with the database or service credentials.
2. [Turn on automatic rotation](#) for your secret, and for **Select which secret will be used to perform the rotation**, choose **Use this secret / Single user rotation**.

Alternating users rotation strategy

The alternating users strategy updates credentials for two users in one secret. You create the first user, and rotation clones it to create the second. Rotation updates the original and the clone credentials in an alternating pattern. For example, if the first version is `user/password1`, then the second version has `user_clone/password2`. The third version has `user/password3`, and the fourth version has `user_clone/password4`. Applications that retrieve the secret from Secrets Manager get the existing version of the credentials while rotation creates the new version. Once the new version is ready, rotation switches the staging labels so that applications use the new version. With this strategy, you have two sets of valid credentials at any given time: both `user` and `user_clone` credentials are valid.

Because most users don't have permission to clone themselves, Secrets Manager uses a `superuser` to do the cloning. You provide the credentials for the `superuser` in another secret.

For detailed instructions, see [the section called “Alternating users rotation” \(p. 24\)](#).

This strategy is appropriate for:

- Applications and databases with permission models where one role owns the database tables and a second role for the application has permission to access the tables.
- Applications that require high availability. There is less chance of applications getting a deny during this type of rotation than single user rotation.

If the database or service is hosted on a server farm where the password change takes time to propagate to all member servers, there is a risk of the database denying calls that use the rotated credentials. You can mitigate this risk with an [appropriate retry strategy](#).

To use the alternating users strategy

1. Create a `superuser` with elevated credentials for your database or service. This user must be able to create new users and change their credentials.
2. Create a secret for the `superuser` credentials.
3. Create a `user` who will access your database or service.
4. Create a secret for the `user` credentials.
5. [Turn on automatic rotation](#) for the `user` secret. For **Use separate credentials**, choose **Yes**, and then under **Secrets**, choose the `superuser` secret.

Automatically rotate an Amazon RDS, Amazon DocumentDB, or Amazon Redshift secret

Secrets Manager provides complete rotation templates for Amazon RDS, Amazon DocumentDB, and Amazon Redshift secrets. For other types of secrets, see [the section called “Rotate a secret” \(p. 119\)](#).

Rotation functions for Amazon RDS (except Oracle) and Amazon DocumentDB automatically use Secure Socket Layer (SSL) or Transport Layer Security (TLS) to connect to your database, if it is available. Otherwise they use an unencrypted connection.

Note

If you set up automatic secret rotation before December 20, 2021, your rotation function might be based on an older template that did not support SSL/TLS. See [Determine when your rotation function was created \(p. 135\)](#). If it was created before December 20, 2021, to support connections that use SSL/TLS, you need to [recreate your rotation function \(p. 117\)](#). Edit your secret, and then choose **Edit rotation**. In the dialog box, choose **Create a rotation function** to recreate your rotation function. If you made [customizations \(p. 127\)](#) to your previous rotation function, you must redo them in the new rotation function.

Another way to automatically rotate a secret is to use AWS CloudFormation to create the secret, and include `AWS::SecretsManager::RotationSchedule`. See [the section called “AWS CloudFormation” \(p. 64\)](#).

Before you begin, you need the following:

- A user with credentials to Amazon RDS, Amazon DocumentDB, or Amazon Redshift.
- A rotation strategy. See [the section called “Rotation strategies” \(p. 115\)](#).
- If you use the [the section called “Alternating users” \(p. 116\)](#), you need a separate secret that contains credentials that can update the rotating secret's credentials.

To turn on rotation for an Amazon RDS, Amazon DocumentDB, or Amazon Redshift secret (console)

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. On the **Secrets** page, choose your secret.
3. On the **Secret details** page, in the **Rotation configuration** section, choose **Edit rotation**.
4. In the **Edit rotation configuration** dialog box, do the following:
 - a. Turn on **Automatic rotation**.
 - b. Under **Rotation schedule**, enter your schedule in UTC time zone by doing one of the following:
 - Choose **Schedule expression builder** to build a schedule in a form. Secrets Manager stores your schedule as a **rate()** or **cron()** expression. The rotation window automatically starts at midnight unless you specify a **Start time**.
 - Choose **Schedule expression**, and then do one of the following:
 - Enter the *cron expression* for your schedule, for example, `cron(0 21 L * ? *)`, which rotates the secret on the last day of every month at 9:00 PM UTC+0. A cron expression for Secrets Manager must have **0** in the minutes field because Secrets Manager rotation windows open on the hour. It must have ***** in the year field, because Secrets Manager does not support rotation schedules that are more than a year apart. For more information, see [Schedule expressions \(p. 120\)](#).
 - Enter a *rate expression* for a daily rate, for example, `rate(10 days)`, which rotates the secret every 10 days. The expression must include **rate()**. With a rate expression, the rotation window automatically starts at midnight.

- c. (Optional) For **Window duration**, choose the length of the window during which you want Secrets Manager to rotate your secret, for example **3h** for a three hour window. The window must not go into the next UTC day. The rotation window automatically ends at the end of the day if you don't specify **Window duration**.
- d. (Optional) Choose **Rotate immediately when the secret is stored** to rotate your secret when you save your changes. If you clear the checkbox, then the first rotation will begin on the schedule you set.

If you use [the section called "Alternating users" \(p. 116\)](#), the credentials in the previous version of the secret are still valid and can be used to access the database or service. To meet compliance requirements, you might need to rotate your secrets more often. For example, if your credential lifetime maximum is 90 days, then we recommend you set your rotation interval to 44 days. That way both users' credentials will be updated within 90 days.

- e. Under **Rotation function**, do the following:
 - To have Secrets Manager create a rotation function for you based on the [Rotation function templates \(p. 128\)](#) for your secret, choose **Create a new Lambda function** and enter a name for your new function. Secrets Manager adds "SecretsManager" to the beginning of your function name.
 - To use a rotation function that you or Secrets Manager already created, choose **Use an existing Lambda function**. You can reuse a rotation function you used for another secret if the rotation strategy is the same. The rotation functions listed under **Recommended VPC configurations** have the same VPC and security group as the database, so you don't have to make any changes for the rotation function to be able to make calls to the database.
- f. For **Use separate credentials to rotate this secret**, do one of the following:
 - For the [Single user rotation strategy \(p. 115\)](#), choose **No**.
 - For the [the section called "Alternating users" \(p. 116\)](#), choose **Yes**. Then choose a secret that contains a user with elevated credentials.

For help resolving common rotation issues, see [the section called "Troubleshoot rotation" \(p. 132\)](#).

AWS CLI

To turn on rotation, see [rotate-secret](#).

For Secrets Manager to be able to rotate the secret, you must make sure the JSON matches the [JSON structure of a database secret \(p. 51\)](#). In particular, if you want to use the [Alternating users \(p. 116\)](#) strategy, your secret must contain the ARN of a superuser secret.

You also need a Lambda function that can rotate the secret. You can create this function based on the [the section called "Rotation function templates" \(p. 128\)](#) that Secrets Manager provides. For [Single user \(p. 115\)](#), choose a template for single user rotation. For [Alternating users \(p. 116\)](#), choose a template for alternating users rotation.

To turn on automatic rotation

- In the AWS CLI, the following command turns on automatic rotation. Secrets Manager rotates the secret once immediately, and then on the 1st and 15th day of every month between 4:00 PM and 6:00 PM UTC.

```
aws secretsmanager rotate-secret
  --secret-id MySecret
  --rotation-lambda-arn arn:aws:lambda:us-east-2:123456789012:function:SecretsManagerMyLambdaFunction-alt-users
  --rotation-rules "{\"ScheduleExpression\": \"cron(0 16 1,15 * ? *)\", \"Duration\": \"2h\"}"
```

AWS SDK

To turn on rotation, use the [RotateSecret](#) action. For more information, see [the section called “AWS SDKs” \(p. 8\)](#).

Automatically rotate a secret

Secrets Manager provides complete rotation templates for Amazon RDS, Amazon DocumentDB, and Amazon Redshift secrets. For more information, see [the section called “Rotate DB credentials” \(p. 117\)](#).

For other types of secrets, you create your own rotation function. Secrets Manager provides a [the section called “Generic rotation function template” \(p. 131\)](#) that you can use as a starting point. If you use the Secrets Manager console or AWS Serverless Application Repository console to create your function from the template, then the Lambda execution role is also automatically set up.

Another way to automatically rotate a secret is to use AWS CloudFormation to create the secret, and include `AWS::SecretsManager::RotationSchedule`. See [Automate secret creation in AWS CloudFormation](#).

Before you begin, you need the following:

- A secret with the information you want to rotate, for example credentials for a user of a database or service.

To turn on rotation (console)

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. On the **Secrets** page, choose your secret.
3. On the **Secret details** page, in the **Rotation configuration** section, choose **Edit rotation**. The **Edit rotation configuration** dialog box opens. Do the following:
 - a. Turn on **Automatic rotation**.
 - b. Under **Rotation schedule**, enter your schedule in UTC time zone by doing one of the following:
 - Choose **Schedule expression builder** to build a schedule in a form. Secrets Manager stores your schedule as a **rate()** or **cron()** expression. The rotation window automatically starts at midnight unless you specify a **Start time**.
 - Choose **Schedule expression**, and then do one of the following:
 - Enter the *cron expression* for your schedule, for example, `cron(0 21 L * ? *)`, which rotates the secret on the last day of every month at 9:00 PM UTC+0. A cron expression for Secrets Manager must have **0** in the minutes field because Secrets Manager rotation windows open on the hour. It must have ***** in the year field, because Secrets Manager does not support rotation schedules that are more than a year apart. For more information, see [Schedule expressions \(p. 120\)](#).
 - Enter a *rate expression* for a daily rate, for example, `rate(10 days)`, which rotates the secret every 10 days. The expression must include **rate()**. With a rate expression, the rotation window automatically starts at midnight.
 - c. (Optional) For **Window duration**, choose the length of the window during which you want Secrets Manager to rotate your secret, for example **3h** for a three hour window. The window must not go into the next UTC day. The rotation window automatically ends at the end of the day if you don't specify **Window duration**.
 - d. Under **Rotation function**, do one of the following:

- i. If you already created a rotation function for this type of secret, choose it.
- ii. Otherwise, choose **Create function**. In the Lambda console, create your new rotation function. If you see **Browse serverless app repository**, choose it, choose **Show apps that create custom IAM roles or resource policies**, and then choose **SecretsManagerRotationTemplate**. Otherwise, choose **Author from scratch** and use the [the section called "Generic rotation function template" \(p. 131\)](#) as a starting point for your function. Implement each of the steps described in [the section called "How rotation works" \(p. 122\)](#).

When your function is complete, return to the Secrets Manager console to finish your secret. For **Choose a Lambda function**, choose the refresh button. Then in the list of functions, choose your new function.

- e. Choose **Save**.

For help resolving common rotation issues, see [the section called "Troubleshoot rotation" \(p. 132\)](#).

AWS SDK and AWS CLI

To turn on rotation, see [rotate-secret](#).

AWS SDK

To turn on rotation, use the [RotateSecret](#) action. For more information, see [the section called "AWS SDKs" \(p. 8\)](#).

Schedule expressions in Secrets Manager rotation

When you turn on automatic rotation, you can use a **cron()** or **rate()** expression to set the schedule for rotating your secret. With a rate expression, you can create a rotation schedule that repeats on an interval of days. With a cron expression, you can create rotation schedules that are more detailed than a rotation interval. Secrets Manager rotation schedules use UTC time zone.

Secrets Manager rotates your secret at any time during a *rotation window*. For a `rate()` expression, the rotation window automatically starts at midnight. For a `cron()` expression, the rotation window opens at the time you specify. By default, the rotation window closes at the end of the day. You can set a **Window duration** to shorten the rotation window. The rotation window must not go into the next UTC day, or in other words, the start hour plus the window duration must be less than or equal to 24 hours.

To turn on rotation, see:

- [the section called "Rotate DB credentials" \(p. 117\)](#)
- [the section called "Rotate a secret" \(p. 119\)](#)

Rate expressions

Secrets Manager rate expressions have the following format, where *Value* is a positive integer and *Unit* can be day or days:

```
rate(Value Unit)
```

For example, `rate(8 days)` means the secret is rotated every eight days.

Cron expressions

Cron expressions have the following format:

```
cron(Minutes Hours Day-of-month Month Day-of-week Year)
```

Fields	Values	Wildcards
Minutes	Must be 0	
Hours	0–23	
Day-of-month	1–31	, - * ? / L
Month	1–12 or JAN–DEC	, - * /
Day-of-week	1–7 or SUN–SAT	#, - * ? / L
Year	Must be *	

A cron expression for Secrets Manager must have **0** in the minutes field because Secrets Manager rotation windows open on the hour. It must have ***** in the year field, because Secrets Manager does not support rotation schedules that are more than a year apart.

Wildcards:

- The ***** (asterisk) wildcard includes all values in the field. In the **Days** field, ***** means every day.
- The **,** (comma) wildcard includes additional values. In the **Month** field, **JAN, APR, JUL, OCT** means January, April, July, and October.
- The **-** (dash) wildcard specifies ranges. In the **Day** field, **1–15** means days 1 through 15 of the specified month.
- The **/** (forward slash) wildcard specifies increments. In the **Days** field, **1/10** means every 10 days.
- The **?** (question mark) wildcard specifies one or another. You can't specify the **Day-of-month** and **Day-of-week** fields in the same cron expression. If you specify a value in one of the fields, you must use a **?** (question mark) in the other.
- The **L** wildcard in the **Day-of-month** or **Day-of-week** fields specifies the last day of the month or week. In **Day-of-month**, you can also specify the last named day of the month, for example **SUNL** means the last Sunday of the month.
- The **#** wildcard in the **Day-of-week** field specifies the day of the week within a month. For example, **TUE#3** means the third Tuesday of the month.

Examples

Schedule (UTC)	Expression
Every day at 10:00 AM.	<code>cron(0 10 * * ? *)</code>
Every Saturday at 6:00 PM.	<code>cron(0 18 ? * SAT *)</code>
The first day of every month at 8:00 AM.	<code>cron(0 8 1 * ? *)</code>
Every three months on Sunday at 1:00 AM.	<code>cron(0 1 * 1/3 SUN *)</code>

Schedule (UTC)	Expression
The last day of every month at 5:00 PM.	<code>cron(0 17 L * ? *)</code>
Monday through Friday at 8:00 AM.	<code>cron(0 8 ? * MON-FRI *)</code>
First and 15th day of every month at 4:00 PM.	<code>cron(0 16 1,15 * ? *)</code>
First Sunday of every month at midnight.	<code>cron(0 0 ? * SUN#1 *)</code>

Rotate a secret immediately

You can only rotate a secret that has rotation configured. This can be a secret that currently has automatic rotation turned on, or one that previously had rotation turned on. Turn on automatic rotation for:

- [Rotate DB credentials](#) (p. 117)
- [Rotate a secret](#) (p. 119)

To rotate a secret immediately (console)

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose your secret.
3. On the secret details page, under **Rotation configuration**, choose **Rotate secret immediately**.
4. In the **Rotate secret** dialog box, choose **Rotate**.

AWS SDK and AWS CLI

To rotate a secret immediately, see [rotate-secret](#).

AWS SDK

To rotate a secret immediately, use the [RotateSecret](#) action. For more information, see [the section called "AWS SDKs"](#) (p. 8).

How rotation works

To rotate a secret, Secrets Manager calls a Lambda function according to the schedule you set up. During rotation, Secrets Manager calls the same function several times, each time with different parameters. Secrets Manager invokes the function with the following JSON request structure of parameters:

```
{
  "Step" : "request.type",
  "SecretId" : "string",
  "ClientRequestToken" : "string"
}
```

The rotation function does the work of rotating the secret. There are four steps to rotating a secret, which correspond to four steps in the Lambda rotation function. Secrets Manager uses [staging labels](#) to label secret versions during rotation.

Step 1: Create a new version of the secret (`createSecret`)

The first step of rotation is to create a new version of the secret. Depending on your [rotation strategy](#), the new version can contain a new password, a new username and password, or more secret information. Secrets Manager labels the new version with the staging label `AWSPENDING`.

Step 2: Change the credentials in the database or service (`setSecret`)

Next, rotation changes the credentials in the database or service to match the new credentials in the `AWSPENDING` version of the secret. Depending on your [rotation strategy](#), this step can create a new user with the same permissions as the existing user.

Rotation functions for Amazon RDS (except Oracle) and Amazon DocumentDB automatically use Secure Socket Layer (SSL) or Transport Layer Security (TLS) to connect to your database, if it is available. Otherwise they use an unencrypted connection.

Note

If you set up automatic secret rotation before December 20, 2021, your rotation function might be based on an older template that did not support SSL/TLS. See [Determine when your rotation function was created \(p. 135\)](#). If it was created before December 20, 2021, to support connections that use SSL/TLS, you need to [recreate your rotation function \(p. 117\)](#).

Step 3: Test the new secret version (`testSecret`)

Next, rotation tests the `AWSPENDING` version of the secret by using it to access the database or service. Rotation functions based on [Rotation function templates \(p. 128\)](#) test the new secret by using read access. Depending on the type of access your applications need, you can update the function to include other access such as write access. See [the section called "Customize a rotation function" \(p. 127\)](#).

Step 4: Finish the rotation (`finishSecret`)

Finally, rotation moves the label `AWSCURRENT` from the previous secret version to this version. Secrets Manager adds the `AWSPREVIOUS` staging label to the previous version, so that you retain the last known good version of the secret.

During rotation, Secrets Manager logs events that indicate the state of rotation. For more information, see [the section called "Logging with AWS CloudTrail" \(p. 139\)](#).

After rotation is successful, applications that [Retrieve secrets from AWS Secrets Manager \(p. 86\)](#) from Secrets Manager automatically get the updated credentials. For more details about how each step of rotation works, see [the section called "Rotation function templates" \(p. 128\)](#).

To turn on automatic rotation, see:

- [the section called "Rotate DB credentials" \(p. 117\)](#)
- [the section called "Rotate a secret" \(p. 119\)](#)

Network access for the rotation function

Secrets Manager uses a Lambda function to rotate a secret. To be able to rotate a secret, the Lambda function must be able to access both the secret and the database or service:

Access a secret

Your Lambda rotation function must be able to access a Secrets Manager endpoint. If your Lambda function can access the internet, then you can use a public endpoint. To find an endpoint, see [AWS Secrets Manager endpoints and quotas](#).

If your Lambda function runs in a VPC that doesn't have internet access, we recommend you configure Secrets Manager service private endpoints within your VPC. Your VPC can then intercept requests addressed to the public regional endpoint and redirect them to the private endpoint. For more information, see [VPC endpoint \(p. 137\)](#).

Alternatively, you can enable your Lambda function to access a Secrets Manager public endpoint by adding a [NAT gateway](#) to your VPC, which allows traffic from your VPC to reach the public endpoint. This exposes your VPC to more risk because an IP address for the gateway can be attacked from the public Internet.

Access the database or service

If your database or service is running on an Amazon EC2 instance in a VPC, we recommend that you configure your Lambda function to run in the same VPC. Then the rotation function can communicate directly with your service. For more information, see [Configuring VPC access](#).

To allow the Lambda function to access the database or service, you must make sure that the security groups attached to your Lambda rotation function allow outbound connections to the database or service. You must also make sure that the security groups attached to your database or service allow inbound connections from the Lambda rotation function. For more information, see:

- Amazon RDS: [Controlling access with security groups](#).
- Amazon Redshift: [Managing VPC security groups for a cluster](#).
- Amazon DocumentDB: [Security Group Allows Inbound Connections](#).

Permissions for rotation

Secrets Manager uses a Lambda function to rotate a secret. The Lambda service assumes an [IAM execution role](#) and provides those credentials to the code for the Lambda function when it executes. If you turn on rotation by using the Secrets Manager console, the Lambda function, resource policy, execution role, and execution role inline policies are created for you.

If you create the Lambda function another way, you must make sure it has the correct permissions. You also need to create an execution role and make sure it has the correct permissions.

To turn on automatic rotation, you must have permission to create the IAM execution role and attach a permission policy to it. You need both `iam:CreateRole` and `iam:AttachRolePolicy` permissions.

Warning

Granting an identity both `iam:CreateRole` and `iam:AttachRolePolicy` permissions allows the identity to grant themselves any permissions.

In the resource policy for your Lambda function, we recommend that you include the context key `aws:SourceAccount` to help prevent AWS Lambda from being used as a [confused deputy](#). For some AWS services, to avoid the confused deputy scenario, AWS recommends that you use both the `aws:SourceArn` and `aws:SourceAccount` global condition keys. However, if you include the context key `aws:SourceArn` in your Lambda rotation function policy, the rotation function can only be used to rotate the secret specified by that ARN. We recommend that you include only the context key `aws:SourceAccount` so that you can use the rotation function for multiple secrets.

Lambda function resource policy

Example

The following policy allows Secrets Manager to invoke the Lambda function specified in the Resource. To attach a resource policy to a Lambda function, see [Using resource-based policies for AWS Lambda](#).

```
{
  "Version": "2012-10-17",
```

```
{
  "Id": "default",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "secretsmanager.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "LambdaRotationFunctionARN",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        }
      }
    }
  ]
}
```

Alternately, you can add this permission by running the following AWS CLI command:

```
aws lambda add-permission --function-name ARN_of_lambda_function --principal
secretsmanager.amazonaws.com --action lambda:InvokeFunction --statement-id
SecretsManagerAccess
```

Lambda function execution role inline policy

The following examples show inline policies for Lambda function execution roles. To create an execution role and attach a permissions policy, see [AWS Lambda execution role](#).

Example IAM execution role inline policy for single user rotation strategy

For an [Rotate DB credentials \(p. 117\)](#), Secrets Manager creates the IAM execution role and attaches this policy for you.

The following example policy allows the function to:

- Run Secrets Manager operations for secrets that are configured to use this rotation function.
- Create a new password.
- Set up the required configuration if your database or service runs in a VPC. See [Configuring a Lambda function to access resources in a VPC](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:DescribeSecret",
        "secretsmanager:GetSecretValue",
        "secretsmanager:PutSecretValue",
        "secretsmanager:UpdateSecretVersionStage"
      ],
      "Resource": "SecretARN"
    },
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetRandomPassword"
      ],
      "Resource": "*"
    }
  ]
}
```

```
    },
    {
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DetachNetworkInterface"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

Example IAM execution role inline policy statement for alternating users strategy

For an [Rotate DB credentials \(p. 117\)](#), Secrets Manager creates the IAM execution role and attaches this policy for you.

The following example policy allows the function to:

- Run Secrets Manager operations for secrets that are configured to use this rotation function.
- Retrieve the credentials in the separate secret. Secrets Manager uses the credentials in the separate secret to update the credentials in the rotated secret.
- Create a new password.
- Set up the required configuration if your database or service runs in a VPC. For more information, see [Configuring a Lambda function to access resources in a VPC](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:DescribeSecret",
        "secretsmanager:GetSecretValue",
        "secretsmanager:PutSecretValue",
        "secretsmanager:UpdateSecretVersionStage"
      ],
      "Resource": "SecretARN"
    },
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": "SeparateSecretARN"
    },
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetRandomPassword"
      ],
      "Resource": "*"
    },
    {
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DetachNetworkInterface"
      ]
    }
  ]
}
```

```
    ],  
    "Resource": "*",  
    "Effect": "Allow"  
  }  
]  
}
```

Example IAM execution role inline policy statement for customer managed key

If you use a KMS key other than the AWS managed key `aws/secretsmanager` to encrypt your secret, then you need to grant the Lambda execution role permission to use the key.

The following example shows a statement to add to the execution role policy to allow the function to retrieve the KMS key.

```
{  
  "Effect": "Allow",  
  "Action": [  
    "kms:Decrypt",  
    "kms:GenerateDataKey"  
  ],  
  "Resource": "KMSKeyARN"  
}
```

Customize a Lambda rotation function for Secrets Manager

For [Rotate DB credentials](#) (p. 117), Secrets Manager can create rotation functions for you for the [Single user](#) (p. 115) or [Alternating users](#) (p. 116) rotation strategies. Secrets Manager uses [the section called "Rotation function templates"](#) (p. 128) to create rotation functions.

You can modify those rotation functions, for example, if you need to test that a rotated secret works for more than read-only access, or to create a different rotation strategy. To change or delete the rotation function that rotates your secret, you first need the name of the function. Then you can download it from the AWS Lambda console to edit it.

For information about what Secrets Manager expects in the rotation function, see [the section called "How rotation works"](#) (p. 122) and [Using AWS Lambda with Secrets Manager](#).

To find the rotation function for a secret (console)

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. From the list of secrets, choose your secret.
3. In the **Rotation configuration** section, in the rotation ARN, the part that follows `:function:` is the name of the function.

To find the rotation function for a secret (AWS CLI)

```
$ aws secretsmanager describe-secret --secret-id SecretARN
```

To edit a Lambda function

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Choose your Lambda rotation function.

3. On the **Function code** menu, choose **Export function**.
4. In the **Export your function** dialog box, choose **Download deployment package**.
5. In your development environment, from the downloaded package, open `lambda_function.py`.

Secrets Manager rotation function templates

To create a Lambda rotation function with any of the following templates, we recommend you use the procedures in [the section called “Rotate DB credentials” \(p. 117\)](#) or [the section called “Rotate a secret” \(p. 119\)](#). Secrets Manager includes the required dependencies when you turn on rotation, unless you create your Lambda rotation function by hand. The templates support Python 3.7.

Secrets Manager provides the following rotation function templates:

Topics

- [Amazon RDS databases \(p. 128\)](#)
- [Amazon DocumentDB \(with MongoDB compatibility\) databases \(p. 130\)](#)
- [Amazon Redshift \(p. 131\)](#)
- [Other types of secrets \(p. 131\)](#)

Amazon RDS databases

Topics

- [Amazon RDS MariaDB single user \(p. 128\)](#)
- [Amazon RDS MariaDB alternating users \(p. 128\)](#)
- [Amazon RDS MySQL single user \(p. 129\)](#)
- [Amazon RDS MySQL alternating users \(p. 129\)](#)
- [Amazon RDS Oracle single user \(p. 129\)](#)
- [Amazon RDS Oracle alternating users \(p. 129\)](#)
- [Amazon RDS PostgreSQL single user \(p. 129\)](#)
- [Amazon RDS PostgreSQL alternating users \(p. 130\)](#)
- [Amazon RDS Microsoft SQLServer single user \(p. 130\)](#)
- [Amazon RDS Microsoft SQLServer alternating users \(p. 130\)](#)

Amazon RDS MariaDB single user

- **Template name:** `SecretsManagerRDSMariaDBRotationSingleUser`
- **Supported database/service:** MariaDB database hosted on an Amazon Relational Database Service (Amazon RDS) database instance.
- **Rotation strategy:** [Single user rotation strategy \(p. 115\)](#).
- **SecretString structure:** [the section called “Amazon RDS MariaDB secret structure” \(p. 51\)](#).
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSMariaDBRotationSingleUser/lambda_function.py

Amazon RDS MariaDB alternating users

- **Template name:** `SecretsManagerRDSMariaDBRotationMultiUser`
- **Supported database/service:** MariaDB database hosted on an Amazon RDS database instance.

- **Rotation strategy:** the section called “Alternating users” (p. 116).
- **SecretString structure:** the section called “Amazon RDS MariaDB secret structure” (p. 51).
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSMariaDBRotationMultiUser/lambda_function.py

Amazon RDS MySQL single user

- **Template name:** SecretsManagerRDSMySQLRotationSingleUser
- **Supported database/service:** MySQL database hosted on an Amazon Relational Database Service (Amazon RDS) database instance.
- **Rotation strategy:** the section called “Single user” (p. 115).
- **Expected SecretString structure:** the section called “Amazon RDS MySQL secret structure” (p. 52).
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSMySQLRotationSingleUser/lambda_function.py

Amazon RDS MySQL alternating users

- **Template name:** SecretsManagerRDSMySQLRotationMultiUser
- **Supported database/service:** MySQL database hosted on an Amazon RDS database instance.
- **Rotation strategy:** the section called “Alternating users” (p. 116).
- **Expected SecretString structure:** the section called “Amazon RDS MySQL secret structure” (p. 52).
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSMySQLRotationMultiUser/lambda_function.py

Amazon RDS Oracle single user

- **Template name:** SecretsManagerRDSOracleRotationSingleUser
- **Supported database/service:** Oracle database hosted on an Amazon Relational Database Service (Amazon RDS) database instance.
- **Rotation strategy:** the section called “Single user” (p. 115).
- **Expected SecretString structure:** the section called “Amazon RDS Oracle secret structure” (p. 52).
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSOracleRotationSingleUser/lambda_function.py

Amazon RDS Oracle alternating users

- **Template name:** SecretsManagerRDSOracleRotationMultiUser
- **Supported database/service:** Oracle database hosted on an Amazon RDS database instance.
- **Rotation strategy:** the section called “Alternating users” (p. 116).
- **Expected SecretString structure:** the section called “Amazon RDS Oracle secret structure” (p. 52).
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSOracleRotationMultiUser/lambda_function.py

Amazon RDS PostgreSQL single user

- **Template name:** SecretsManagerRDSPostgreSQLRotationSingleUser

- **Supported database/service:** PostgreSQL database hosted on an Amazon RDS database instance.
- **Rotation strategy:** [Single user rotation strategy](#) (p. 115).
- **Expected SecretString structure:** the section called “Amazon RDS PostgreSQL secret structure” (p. 52).
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSPostgreSQLRotationSingleUser/lambda_function.py

Amazon RDS PostgreSQL alternating users

- **Template name:** SecretsManagerRDSPostgreSQLRotationMultiUser
- **Supported database/service:** PostgreSQL database hosted on an Amazon RDS database instance.
- **Rotation strategy:** the section called “Alternating users” (p. 116).
- **Expected SecretString structure:** the section called “Amazon RDS PostgreSQL secret structure” (p. 52).
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSPostgreSQLRotationMultiUser/lambda_function.py

Amazon RDS Microsoft SQLServer single user

- **Template name:** SecretsManagerRDSSQLServerRotationSingleUser
- **Supported database/service:** Microsoft SQLServer database hosted on an Amazon RDS database instance.
- **Rotation strategy:** the section called “Single user” (p. 115).
- **Expected SecretString structure:** the section called “Amazon RDS Microsoft SQLServer secret structure” (p. 52).
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSSQLServerRotationSingleUser/lambda_function.py

Amazon RDS Microsoft SQLServer alternating users

- **Template name:** SecretsManagerRDSSQLServerRotationMultiUser
- **Supported database/service:** Microsoft SQLServer database hosted on an Amazon RDS database instance.
- **Rotation strategy:** the section called “Alternating users” (p. 116).
- **Expected SecretString structure:** the section called “Amazon RDS Microsoft SQLServer secret structure” (p. 52).
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRDSSQLServerRotationMultiUser/lambda_function.py

Amazon DocumentDB (with MongoDB compatibility) databases

Amazon DocumentDB single user

- **Template name:** SecretsManagerMongoDBRotationSingleUser
- **Supported database/service:** Amazon DocumentDB

- **Rotation strategy:** the section called “Single user” (p. 115).
- **Expected `SecretString` structure:** the section called “Amazon DocumentDB secret structure” (p. 53).
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerMongoDBRotationSingleUser/lambda_function.py

Amazon DocumentDB alternating users

- **Template name:** `SecretsManagerMongoDBRotationMultiUser`
- **Supported database/service:** Amazon DocumentDB
- **Rotation strategy:** the section called “Alternating users” (p. 116).
- **Expected `SecretString` structure:** the section called “Amazon DocumentDB secret structure” (p. 53).
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerMongoDBRotationMultiUser/lambda_function.py

Amazon Redshift

Amazon Redshift single user

- **Template name:** `SecretsManagerRedshiftRotationSingleUser`
- **Supported database/service:** Amazon Redshift
- **Rotation strategy:** the section called “Single user” (p. 115).
- **Expected `SecretString` structure:** the section called “Amazon Redshift secret structure” (p. 53).
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRedshiftRotationSingleUser/lambda_function.py

Amazon Redshift alternating users

- **Template name:** `SecretsManagerRedshiftRotationMultiUser`
- **Supported database/service:** Amazon Redshift
- **Rotation strategy:** the section called “Alternating users” (p. 116).
- **Expected `SecretString` structure:** the section called “Amazon Redshift secret structure” (p. 53).
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRedshiftRotationMultiUser/lambda_function.py

Other types of secrets

Generic rotation function template

- **Template name:** `SecretsManagerRotationTemplate`
- **Supported database/service:** None. You supply the code to interact with whatever service you want.
- **Rotation strategy:** You can use this template to implement your own strategy. Rotation templates have four steps: the section called “How rotation works” (p. 122). To use a rotation function that you created based on this template, see the section called “Rotate a secret” (p. 119).
- **Expected `SecretString` structure:** You define this.
- **Source code:** https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/tree/master/SecretsManagerRotationTemplate/lambda_function.py

Troubleshoot AWS Secrets Manager rotation of secrets

Use the information here to help you diagnose and fix common errors that you might encounter when you're rotating Secrets Manager secrets.

Rotating secrets in AWS Secrets Manager requires you to use a Lambda function that defines how to interact with the database or service that owns the secret.

Topics

- [I want to find the diagnostic logs for my Lambda rotation function \(p. 132\)](#)
- [I get "access denied" when trying to configure rotation for my secret \(p. 132\)](#)
- [My first rotation fails after I enable rotation \(p. 133\)](#)
- [Rotation fails because the secret value is not formatted as expected by the rotation function. \(p. 133\)](#)
- [Secrets Manager says I successfully configured rotation, but the password isn't rotating \(p. 133\)](#)
- [Rotation fails with an "Internal failure" error message \(p. 134\)](#)
- [CloudTrail shows access-denied errors during rotation \(p. 134\)](#)
- [My database requires an SSL/TLS connection but the Lambda rotation function isn't using SSL/TLS \(p. 135\)](#)

I want to find the diagnostic logs for my Lambda rotation function

When the rotation function doesn't operate the way you expect, you should first check the CloudWatch logs. Secrets Manager provides template code for the Lambda rotation function, and this code writes error messages to the CloudWatch log.

To view the CloudWatch logs for your Lambda function

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. From the list of functions, choose the name of the Lambda function associated with your secret.
3. On the **Monitor** tab, choose **Logs**, and then choose **View logs in CloudWatch**.

The CloudWatch console opens and displays the logs for your function.

I get "access denied" when trying to configure rotation for my secret

When you add a Lambda rotation function Amazon Resource Name (ARN) to your secret, Secrets Manager checks the permissions of the function. The role policy for the function must grant the Secrets Manager service principal `secretsmanager.amazonaws.com` permission to invoke the function (`lambda:InvokeFunction`).

You can add this permission by running the following AWS CLI command:

```
aws lambda add-permission --function-name ARN_of_lambda_function --principal
secretsmanager.amazonaws.com --action lambda:InvokeFunction --statement-id
SecretsManagerAccess
```

My first rotation fails after I enable rotation

When you enable rotation for a secret that uses a "master" secret to change the credentials on the secured service, Secrets Manager automatically configures most elements required for rotation. However, Secrets Manager can't automatically grant permission to read the master secret to your Lambda function. You must explicitly grant this permission yourself. Specifically, you grant the permission by adding it to the policy attached to the IAM role attached to your Lambda rotation function. That policy must include the following statement; this is only a statement, not a complete policy. For the complete policy, see the second sample policy in the section [CloudTrail shows access-denied errors during rotation \(p. 134\)](#).

```
{
  "Sid": "AllowAccessToMasterSecret",
  "Effect": "Allow",
  "Action": "secretsmanager:GetSecretValue",
  "Resource": "ARN_of_master_secret"
}
```

This enables the rotation function to retrieve the credentials from the master secret—then use the master secret credentials to change the credentials for the rotating secret.

Rotation fails because the secret value is not formatted as expected by the rotation function.

Rotation might also fail if you don't format the secret value as a JSON structure as expected by the rotation function. The rotation function you use determines the format used. For the details of what each rotation function requires for the secret value, see the **Expected SecretString Value** entry under the relevant rotation function at [Secrets Manager rotation function templates \(p. 128\)](#).

For example, if you use the MySQL Single User rotation function, the SecretString text structure must look like this:

```
{
  "engine": "mysql",
  "host": "<required: instance host name/resolvable DNS name>",
  "username": "<required: username>",
  "password": "<required: password>",
  "dbname": "<optional: database name. If not specified, defaults to None>",
  "port": "<optional: TCP port number. If not specified, defaults to 3306>"
}
```

Secrets Manager says I successfully configured rotation, but the password isn't rotating

This can occur if there are network configuration issues that prevent the Lambda function from communicating with either your secured database/service or the Secrets Manager service endpoint, on the public Internet. If you run your database or service in a VPC, then you use one of two options for configuration:

- Make the database in the VPC publicly accessible with an Amazon EC2 Elastic IP address.
- Configure the Lambda rotation function to operate in the same VPC as the database/service.
- If your VPC doesn't have access to the public Internet, for example, if you don't [configure the VPC with a NAT gateway](#) for access, then you must [configure the VPC with a private service endpoint for Secrets Manager \(p. 123\)](#) accessible from within the VPC.

To determine if this type of configuration issue caused the rotation failure, perform the following steps.

To diagnose connectivity issues between your rotation function and the database or Secrets Manager

1. Open your logs by following the procedure [I want to find the diagnostic logs for my Lambda rotation function \(p. 132\)](#).
2. Examine the log files to look for indications that timeouts occur between either the Lambda function and the AWS Secrets Manager service, or between the Lambda function and the secured database or service.
3. For information about how to configure services and Lambda functions to interoperate within the VPC environment, see the [Amazon Virtual Private Cloud documentation](#) and the [AWS Lambda Developer Guide](#).

Rotation fails with an "Internal failure" error message

When your rotation function generates a new password and attempts to store it in the database as a new set of credentials, you must ensure the password includes only characters valid for the specified database. The attempt to set the password for a user fails if the password includes characters that the database engine doesn't accept. This error appears as an "internal failure". Refer to the database documentation for a list of the characters you can use. Then, exclude all others by using the [ExcludeCharacters](#) parameter in the `GetRandomPassword` API call.

CloudTrail shows access-denied errors during rotation

When you configure rotation, if you let Secrets Manager create the rotation function for you, Secrets Manager automatically provides a policy attached to the function IAM role that grants the appropriate permissions. If you create a custom function, you need to grant the following permissions to the role attached to the function.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:DescribeSecret",
        "secretsmanager:GetRandomPassword",
        "secretsmanager:GetSecretValue",
        "secretsmanager:PutSecretValue",
        "secretsmanager:UpdateSecretVersionStage"
      ],
      "Resource": "*"
    }
  ]
}
```

Also, if your rotation uses separate master secret credentials to rotate this secret, then you must also grant permission to retrieve the secret value from the master secret. For more information, see [My first rotation fails after I enable rotation \(p. 133\)](#). The combined policy might look like this:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToSecretsManagerAPIs",
```

```
    "Effect": "Allow",
    "Action": [
        "secretsmanager:DescribeSecret",
        "secretsmanager:GetRandomPassword",
        "secretsmanager:GetSecretValue",
        "secretsmanager:PutSecretValue",
        "secretsmanager:UpdateSecretVersionStage"
    ],
    "Resource": "*"
  },
  {
    "Sid": "AllowAccessToMasterSecret",
    "Effect": "Allow",
    "Action": "secretsmanager:GetSecretValue",
    "Resource": "MasterSecretArn"
  }
]
```

My database requires an SSL/TLS connection but the Lambda rotation function isn't using SSL/TLS

If your database requires an SSL/TLS connection, but the rotation function uses an unencrypted connection, the rotation function can't connect to the database, and rotation fails. In Amazon CloudWatch, the rotation function logs one of the following errors:

- For single-user rotation:

```
setSecret: Unable to log into database with previous, current, or pending
secret of secret arn SecretArn
```

- For multi-user rotation:

```
setSecret: Unable to log into database using current credentials for secret
SecretArn
```

Rotation functions for Amazon RDS (except Oracle) and Amazon DocumentDB automatically use Secure Socket Layer (SSL) or Transport Layer Security (TLS) to connect to your database, if it is available. Otherwise they use an unencrypted connection.

Note

If you set up automatic secret rotation before December 20, 2021, your rotation function might be based on an older template that did not support SSL/TLS. To support connections that use SSL/TLS, you need to [recreate your rotation function \(p. 117\)](#).

To determine when your rotation function was created

1. In the Secrets Manager console <https://console.aws.amazon.com/secretsmanager/>, open your secret. In the **Rotation configuration** section, under **Lambda rotation function**, you see the **Lambda function ARN**, for example, `arn:aws:lambda:aws-region:123456789012:function:SecretsManagerMyRotationFunction`. Copy the function name from the end of the ARN, in this example `SecretsManagerMyRotationFunction`.
2. In the AWS Lambda console <https://console.aws.amazon.com/lambda/>, under **Functions**, paste your Lambda function name in the search box, choose Enter, and then choose the Lambda function.
3. In the function details page, on the **Configuration** tab, under **Tags**, copy the value next to the key `aws:cloudformation:stack-name`.
4. In the AWS CloudFormation console <https://console.aws.amazon.com/cloudformation>, under **Stacks**, paste the key value in the search box, and then choose Enter.

5. The list of stacks filters so that only the stack that created the Lambda rotation function appears. In the **Created date** column, view the date the stack was created. This is the date the Lambda rotation function was created.

Using an AWS Secrets Manager VPC endpoint

You can establish a private connection between your VPC and Secrets Manager by creating an *interface VPC endpoint*. Interface endpoints are powered by [AWS PrivateLink](#), a technology that enables you to privately access Secrets Manager APIs without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with Secrets Manager APIs. Traffic between your VPC and Secrets Manager does not leave the AWS network.

Each interface endpoint is represented by one or more [Elastic Network Interfaces](#) in your subnets.

For more information, see [Interface VPC endpoints \(AWS PrivateLink\)](#) in the *Amazon VPC User Guide*.

Considerations for Secrets Manager VPC endpoints

Before you set up an interface VPC endpoint for Secrets Manager, ensure that you review [interface endpoint properties and limitations](#) in the *Amazon VPC User Guide*.

Automatic secret rotation uses a Lambda function, and the Lambda function makes requests to both the database and Secrets Manager. When you turn on automatic rotation, Secrets Manager creates the Lambda function in the same VPC as your database. We recommend you create a Secrets Manager endpoint in the same VPC so that requests from the Lambda rotation function to Secrets Manager don't leave the Amazon network. For more information, see [the section called "Network access for rotation" \(p. 123\)](#).

Secrets Manager supports making calls to all of its API actions from your VPC.

You can use AWS CloudTrail logs to audit your use of secrets through the VPC endpoint.

For information about denying access to requests that don't originate from a specified VPC or VPC endpoint, see [the section called "Example: Permissions and VPCs" \(p. 35\)](#).

Creating an interface VPC endpoint for Secrets Manager

You can create a VPC endpoint for the Secrets Manager service using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see [Creating an interface endpoint](#) in the *Amazon VPC User Guide*.

Create a VPC endpoint for Secrets Manager using the following service name:

- `com.amazonaws.region.secretsmanager`

If you enable private DNS for the endpoint, you can make API requests to Secrets Manager using its default DNS name for the Region, for example, `secretsmanager.us-east-1.amazonaws.com`.

For more information, see [Accessing a service through an interface endpoint](#) in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for Secrets Manager

You can attach an endpoint policy to your VPC endpoint that controls access to Secrets Manager. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see [Controlling access to services with VPC endpoints](#) in the *Amazon VPC User Guide*.

Example Enable access to the Secrets Manager endpoint for a specific account

The following example grants access to all users and roles in account 123456789012.

```
{
  "Statement": [
    {
      "Sid": "AccessSpecificAccount",
      "Principal": {"AWS": "123456789012"},
      "Action": "secretsmanager:*",
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Example Enable access to a single secret on the Secrets Manager endpoint

The following example restricts access to only the specified secret.

```
{
  "Statement": [
    {
      "Principal": "*",
      "Action": "secretsmanager:*",
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:us-east-2:111122223333:secret:SecretName-a1b2c3"
      ]
    }
  ]
}
```

Monitor AWS Secrets Manager secrets

AWS provides the following monitoring tools to watch Secrets Manager secrets, report when something is wrong, and take automatic actions when appropriate:

- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. For more information, see [the section called "Logging with AWS CloudTrail"](#) (p. 139).
- *Amazon CloudWatch* monitors your AWS resources and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For more information, see the [Amazon CloudWatch User Guide](#).

Logging AWS Secrets Manager events with AWS CloudTrail

AWS CloudTrail records all [API calls for Secrets Manager](#) as events, including calls from the Secrets Manager console. CloudTrail also captures the following events:

- `RotationAbandoned` event - Secrets Manager removed the `AWSPENDING` label from an existing version of a secret. When you manually create a new version of a secret, you send a message signalling the abandonment of the current ongoing rotation in favor of the new secret version. As a result, Secrets Manager removes the `AWSPENDING` label to allow future rotations to succeed and publish a CloudTrail event to provide awareness of the change.
- `RotationStarted` event - A secret started rotation.
- `RotationSucceeded` event - A successful rotation event.
- `RotationFailed` event - Secret rotation failed.
- `StartSecretVersionDelete` event - a mechanism that notifies you of the start deletion for a secret version.
- `CancelSecretVersionDelete` event - A delete cancellation for a secret version.
- `EndSecretVersionDelete` event - An ending secret version deletion.

You can use the CloudTrail console to view the last 90 days of recorded events. For an ongoing record of events in your AWS account, including events for Secrets Manager, create a trail so that CloudTrail delivers log files to an Amazon S3 bucket. See [Creating a trail for your AWS account](#). You can also configure CloudTrail to receive CloudTrail log files from [multiple AWS accounts](#) and [AWS Regions](#).

You can configure other AWS services to further analyze and act upon the data collected in CloudTrail logs. See [AWS service integrations with CloudTrail logs](#). You can also get notifications when CloudTrail publishes new log files to your Amazon S3 bucket. See [Configuring Amazon SNS notifications for CloudTrail](#).

To retrieve Secrets Manager events from CloudTrail logs (console)

1. Open the CloudTrail console at <https://console.aws.amazon.com/cloudtrail/>.
2. Ensure that the console points to the region where your events occurred. The console shows only those events that occurred in the selected region. Choose the region from the drop-down list in the upper-right corner of the console.
3. In the left-hand navigation pane, choose **Event history**.
4. Choose **Filter** criteria and/or a **Time range** to help you find the event that you're looking for. For example, to see all Secrets Manager events, for **Select attribute**, choose **Event source**. Then, for **Enter event source**, choose **secretsmanager.amazonaws.com**.
5. To see additional details, choose the expand arrow next to event. To see all of the information available, choose **View event**.

AWS CLI or SDK

To retrieve Secrets Manager events from CloudTrail logs (AWS CLI or SDK)

1. Open a command window to run AWS CLI commands.
2. Run a command similar to the following example.

```
$ aws cloudtrail lookup-events --region us-east-1 --lookup-attributes
AttributeKey=EventSource,AttributeValue=secretsmanager.amazonaws.com
{
  "Events": [
    {
      "EventId": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE",
      "EventName": "CreateSecret",
      "EventTime": 1525106994.0,
      "Username": "Administrator",
      "Resources": [],
      "CloudTrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type\":\"IAMUser\", \"principalId\":\"AKIAIOSFODNN7EXAMPLE\", \"arn\":\"arn:aws:iam::123456789012:user/Administrator\", \"accountId\":\"123456789012\", \"accessKeyId\":\"AKIAIOSFODNN7EXAMPLE\", \"userName\":\"Administrator\"}, \"eventTime\":\"2018-04-30T16:49:54Z\", \"eventSource\":\"secretsmanager.amazonaws.com\", \"eventName\":\"CreateSecret\", \"awsRegion\":\"us-east-2\", \"sourceIPAddress\":\"192.168.100.101\", \"userAgent\":\"<useragent string>\", \"requestParameters\":{\"name\":\"MyTestSecret\", \"clientRequestToken\":\"EXAMPLE2-90ab-cdef-fedc-ba987EXAMPLE\"}, \"responseElements\":null, \"requestID\":\"EXAMPLE3-90ab-cdef-fedc-ba987EXAMPLE\", \"eventID\":\"EXAMPLE4-90ab-cdef-fedc-ba987EXAMPLE\", \"eventType\":\"AwsApiCall\", \"recipientAccountId\":\"123456789012\"}"
    }
  ]
}
```

Examples of Secrets Manager log entries

The following example shows a CloudTrail log entry for a sample CreateSecret call:

```
{
```

```
"eventVersion": "1.05",
"userIdentity": {
  "type": "Root",
  "principalId": "123456789012",
  "arn": "arn:aws:iam::123456789012:root",
  "accountId": "123456789012",
  "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "userName": "myusername",
  "sessionContext": {"attributes": {
    "mfaAuthenticated": "false",
    "creationDate": "2018-04-03T17:43:50Z"
  }}
},
"eventTime": "2018-04-03T17:50:55Z",
"eventSource": "secretsmanager.amazonaws.com",
"eventName": "CreateSecret",
"awsRegion": "us-east-2",
"requestParameters": {
  "name": "MyDatabaseSecret",
  "clientRequestToken": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE"
},
"responseElements": null,
"requestID": "EXAMPLE2-90ab-cdef-fedc-ba987EXAMPLE",
"eventID": "EXAMPLE3-90ab-cdef-fedc-ba987EXAMPLE",
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
}
```

The following example shows a CloudTrail log entry for a sample `DeleteSecret` call:

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "Root",
    "principalId": "123456789012",
    "arn": "arn:aws:iam::123456789012:root",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "myusername",
    "sessionContext": {"attributes": {
      "mfaAuthenticated": "false",
      "creationDate": "2018-04-03T17:43:50Z"
    }}
  },
  "eventTime": "2018-04-03T17:51:02Z",
  "eventSource": "secretsmanager.amazonaws.com",
  "eventName": "DeleteSecret",
  "awsRegion": "us-east-2",
  "requestParameters": {
    "recoveryWindowInDays": 30,
    "secretId": "MyDatabaseSecret"
  },
  "responseElements": {
    "name": "MyDatabaseSecret",
    "deletionDate": "May 3, 2018 5:51:02 PM",
    "aRN": "arn:aws:secretsmanager:us-east-2:123456789012:secret:MyDatabaseSecret-a1b2c3"
  },
  "requestID": "EXAMPLE2-90ab-cdef-fedc-ba987EXAMPLE",
  "eventID": "EXAMPLE3-90ab-cdef-fedc-ba987EXAMPLE",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

Monitoring Secrets Manager with Amazon CloudWatch

You can monitor AWS Secrets Manager using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the [Amazon CloudWatch User Guide](#).

For Secrets Manager, you can use CloudWatch to alert you when your request rate for APIs or the number of secrets in your account reaches a specific threshold. You can also use CloudWatch to monitor estimated Secrets Manager charges. For more information, see [Creating a billing alarm to monitor your estimated AWS charges](#).

Topics

- [Secrets Manager metrics and dimensions \(p. 142\)](#)
- [Create alarms to monitor Secrets Manager metrics \(p. 143\)](#)
- [Secrets Manager events \(p. 143\)](#)
- [Amazon CloudWatch Synthetics canaries \(p. 143\)](#)
- [Monitor secrets scheduled for deletion \(p. 143\)](#)

Secrets Manager metrics and dimensions

The `AWS/SecretsManager` namespace includes the following metrics.

Metric	Description
SecretCount	The number of secrets in your account, including secrets that are marked for deletion. Units: Count

The following dimensions are supported for the Secrets Manager metrics.

Dimension	Description
Service	The name of the AWS service containing the resource. For Secrets Manager, the value for this dimension is <code>Secrets Manager</code> .
Type	The type of entity that is being reported. For Secrets Manager, the value for this dimension is <code>Resource</code> .
Resource	The type of resource that is running. For Secrets Manager, the value for this dimension is <code>SecretCount</code> .
Class	None.

Secrets Manager API requests that you can monitor using CloudWatch metrics include `GetSecretValue`, `DescribeSecret`, `ListSecrets`, and others. To find metrics, in the CloudWatch console, choose **All metrics**, and then in the search box, enter your search term, for example **secrets**.

Create alarms to monitor Secrets Manager metrics

You can create a CloudWatch alarm that sends an Amazon SNS message when the value of the metric changes and causes the alarm to change state. An alarm watches a metric over a time period you specify, and performs actions based on the value of the metric relative to a given threshold over a number of time periods. Alarms invoke actions for sustained state changes only. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods.

For more information, see [Using Amazon CloudWatch alarms](#).

Secrets Manager events

Secrets Manager integrates with Amazon EventBridge to notify you of certain events that affect your secrets. With Amazon EventBridge, you can be notified of Secrets Manager events that happen except `Get*` API calls. You can configure EventBridge rules that look for these events and then send new generated events to an Amazon SNS topic that emails or text messages to subscribers.

For more information, see [Creating Amazon EventBridge rules that react to events](#).

Amazon CloudWatch Synthetics canaries

Amazon CloudWatch Synthetics canaries are configurable scripts that run on a schedule to monitor your endpoints and APIs. Canaries follow the same routes and perform the same actions as a customer, which makes it possible for you to continually verify your customer experience even when you don't have any customer traffic on your applications.

For an example of how to integrate Secrets Manager, see [Integrating your canary with other AWS services](#).

Monitor secrets scheduled for deletion

You can use a combination of AWS CloudTrail, Amazon CloudWatch Logs, and Amazon Simple Notification Service (Amazon SNS) to create an alarm that notifies you of any attempts to access a secret pending deletion. If you receive a notification from an alarm, you might want to cancel deletion of the secret to give yourself more time to determine if you really want to delete it. Your investigation might result in the secret being restored because you still need the secret. Alternatively, you might need to update the user with details of the new secret to use.

The following procedures explain how to receive a notification when a request for the `GetSecretValue` operation that results in a specific error message written to your CloudTrail log files. Other API operations can be performed on the secret without triggering the alarm. This CloudWatch alarm detects usage that might indicate a person or application using outdated credentials.

Before you begin these procedures, you must turn on CloudTrail in the AWS Region and account where you intend to monitor AWS Secrets Manager API requests. For instructions, go to [Creating a trail for the first time](#) in the *AWS CloudTrail User Guide*.

Step 1: Configure CloudTrail log file delivery to CloudWatch logs

You must configure delivery of your CloudTrail log files to CloudWatch Logs. You do this so CloudWatch Logs can monitor them for Secrets Manager API requests to retrieve a secret pending deletion.

To configure CloudTrail log file delivery to CloudWatch Logs

1. Open the CloudTrail console at <https://console.aws.amazon.com/cloudtrail/>.

2. On the top navigation bar, choose the AWS Region to monitor secrets.
3. In the left navigation pane, choose **Trails**, and then choose the name of the trail to configure for CloudWatch.
4. On the **Trails Configuration** page, scroll down to the **CloudWatch Logs** section, and then choose the edit icon (✎).
5. For **New or existing log group**, type a name for the log group, such as **CloudTrail/MyCloudWatchLogGroup**.
6. For **IAM role**, you can use the default role named **CloudTrail_CloudWatchLogs_Role**. This role has a default role policy with the required permissions to deliver CloudTrail events to the log group.
7. Choose **Continue** to save your configuration.
8. On the **AWS CloudTrail will deliver CloudTrail events associated with API activity in your account to your CloudWatch Logs log group** page, choose **Allow**.

Step 2: Create the CloudWatch alarm

To receive a notification when a Secrets Manager `GetSecretValue` API operation requests to access a secret pending deletion, you must create a CloudWatch alarm and configure notification.

To create a CloudWatch alarm

1. Sign in to the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. On the top navigation bar, choose the AWS Region where you want to monitor secrets.
3. In the left navigation pane, choose **Logs**.
4. In the list of **Log Groups**, select the check box next to the log group you created in the previous procedure, such as **CloudTrail/MyCloudWatchLogGroup**. Then choose **Create Metric Filter**.
5. For **Filter Pattern**, type or paste the following:

```
{ $.eventName = "GetSecretValue" && $.errorMessage = "*secret because it was marked for deletion*" }
```

Choose **Assign Metric**.

6. On the **Create Metric Filter and Assign a Metric** page, do the following:
 - a. For **Metric Namespace**, type **CloudTrailLogMetrics**.
 - b. For **Metric Name**, type **AttemptsToAccessDeletedSecrets**.
 - c. Choose **Show advanced metric settings**, and then if necessary for **Metric Value**, type **1**.
 - d. Choose **Create Filter**.
7. In the filter box, choose **Create Alarm**.
8. In the **Create Alarm** window, do the following:
 - a. For **Name**, type **AttemptsToAccessDeletedSecretsAlarm**.
 - b. **Whenever**; for **is**;, choose **>=**, and then type **1**.
 - c. Next to **Send notification to**;, do one of the following:
 - To create and use a new Amazon SNS topic, choose **New list**, and then type a new topic name. For **Email list**;, type at least one email address. You can type more than one email address by separating them with commas.
 - To use an existing Amazon SNS topic, choose the name of the topic to use. If a list doesn't exist, choose **Select list**.
 - d. Choose **Create Alarm**.

Step 3: Test the CloudWatch alarm

To test your alarm, create a secret and then schedule it for deletion. Then, try to retrieve the secret value. You shortly receive an email at the address you configured in the alarm. It alerts you to the use of a secret scheduled for deletion.

Compliance validation for AWS Secrets Manager

Third-party auditors assess the security and compliance of AWS Secrets Manager as part of multiple AWS compliance programs. These include SOC, PCI, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see [AWS Services in Scope by Compliance Program](#). For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using Secrets Manager is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and Compliance Quick Start Guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- *AWS Config* assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations. For more information, see [the section called “Audit secrets for compliance by using AWS Config” \(p. 146\)](#).
- *AWS Security Hub* provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

The AWS Foundational Security Best Practices standard is a set of controls that detects when your deployed accounts and resources deviate from security best practices. Security Hub provides a set of controls for Secrets Manager that allows you to continuously evaluate and identify areas of deviation from best practices. For more information, see [AWS Foundational Security Best Practices controls](#).

- *IAM Access Analyzer* analyzes policies, including condition statements in a policy, that allow an external entity to access a secret. For more information, see [Previewing access with Access Analyzer](#).
- *AWS Systems Manager* provides predefined runbooks for Secrets Manager. For more information, see [Systems Manager Automation runbook reference for Secrets Manager](#).

Audit secrets for compliance by using AWS Config

You can use AWS Config to evaluate your secrets and assess how well they comply with your internal practices, industry guidelines, and regulations. You define your internal security and compliance requirements for secrets using AWS Config rules. Then AWS Config can identify secrets that don't conform to your rules. You can also track changes to secret metadata, rotation configuration, the KMS key used for secret encryption, the Lambda rotation function, and tags associated with a secret.

You can receive notifications from Amazon SNS about your secret configurations. For example, you can receive Amazon SNS notifications for a list of secrets not configured for rotation which enables you to drive security best practices for rotating secrets.

If you have secrets in multiple AWS accounts and AWS Regions in your organization, you can aggregate that configuration and compliance data.

Monitoring secrets with AWS Config is supported in all AWS Regions except Asia Pacific (Jakarta).

To add a new rule for your secrets

- Follow the instructions on [Working with AWS Config managed rules](#), and choose one of the following rules:
 - [secretsmanager-rotation-enabled-check](#) — Checks whether rotation is configured for secrets stored in Secrets Manager.
 - [secretsmanager-scheduled-rotation-success-check](#) — Checks whether secrets were successfully rotated. AWS Config also checks if the last rotated date falls within the configured rotation frequency.
 - [secretsmanager-secret-periodic-rotation](#) — Checks whether secrets were rotated within the specified number of days.
 - [secretsmanager-secret-unused](#) — Checks whether secrets were accessed within the specified number of days.
 - [secretsmanager-using-cmk](#) — Checks whether secrets are encrypted using the AWS managed key `aws/secretsmanager` or a customer managed key you created in AWS KMS.

After you save the rule, AWS Config evaluates your secrets every time the metadata of a secret changes. You can configure AWS Config to notify you of changes. For more information, see [Notifications that AWS Config sends to an Amazon SNS topic](#).

Aggregate secrets from your AWS accounts and AWS Regions

You can configure AWS Config Multi-Account Multi-Region Data Aggregator to review configurations of your secrets across all accounts and regions in your organization, and then review your secret configurations and compare to secrets management best practices.

You must enable AWS Config and the AWS Config managed rules specific to secrets across all accounts and regions before you create an aggregator. For more information, see [Use CloudFormation StackSets to provision resources across multiple AWS accounts and Regions](#).

For more information about AWS Config Aggregator, see [Multi-Account Multi-Region Data Aggregation](#) and [Setting Up an Aggregator Using the Console](#) in the AWS Config Developer Guide.

AWS services that use Secrets Manager secrets

The following AWS services integrate with Secrets Manager:

- [How Alexa for Business uses AWS Secrets Manager \(p. 149\)](#)
- [How AWS App2Container uses AWS Secrets Manager \(p. 149\)](#)
- [How Amazon AppFlow uses AWS Secrets Manager \(p. 149\)](#)
- [How AWS AppSync uses AWS Secrets Manager \(p. 149\)](#)
- [How Amazon Athena uses AWS Secrets Manager \(p. 149\)](#)
- [How AWS CodeBuild uses AWS Secrets Manager \(p. 150\)](#)
- [How AWS Directory Service uses AWS Secrets Manager \(p. 150\)](#)
- [How Amazon DocumentDB \(with MongoDB compatibility\) uses AWS Secrets Manager \(p. 150\)](#)
- [How AWS Elemental Live uses AWS Secrets Manager \(p. 150\)](#)
- [How AWS Elemental MediaConnect uses AWS Secrets Manager \(p. 150\)](#)
- [How AWS Elemental MediaConvert uses AWS Secrets Manager \(p. 151\)](#)
- [Find unprotected secrets in your code with Amazon CodeGuru Reviewer \(p. 151\)](#)
- [How AWS Elemental MediaPackage uses AWS Secrets Manager \(p. 151\)](#)
- [How AWS Elemental MediaTailor uses AWS Secrets Manager \(p. 151\)](#)
- [How Amazon EMR uses AWS Secrets Manager \(p. 151\)](#)
- [How Amazon EventBridge uses AWS Secrets Manager \(p. 152\)](#)
- [How Amazon FSx uses AWS Secrets Manager secrets \(p. 152\)](#)
- [How AWS Glue DataBrew uses AWS Secrets Manager \(p. 152\)](#)
- [How AWS Glue Studio uses AWS Secrets Manager \(p. 152\)](#)
- [How AWS IoT SiteWise uses AWS Secrets Manager \(p. 153\)](#)
- [How Amazon Kendra uses AWS Secrets Manager \(p. 153\)](#)
- [How AWS Launch Wizard uses AWS Secrets Manager \(p. 153\)](#)
- [How Amazon Lookout for Metrics uses AWS Secrets Manager \(p. 153\)](#)
- [How Amazon Managed Streaming for Apache Kafka uses AWS Secrets Manager \(p. 153\)](#)
- [How Amazon Managed Workflows for Apache Airflow uses AWS Secrets Manager \(p. 154\)](#)
- [How AWS Migration Hub uses AWS Secrets Manager \(p. 154\)](#)
- [How AWS OpsWorks for Chef Automate uses AWS Secrets Manager \(p. 154\)](#)
- [How Amazon RDS uses AWS Secrets Manager \(p. 154\)](#)
- [How Amazon Redshift uses AWS Secrets Manager \(p. 155\)](#)
- [How Amazon SageMaker uses AWS Secrets Manager \(p. 155\)](#)
- [How AWS Toolkit for JetBrains uses AWS Secrets Manager \(p. 155\)](#)
- [How AWS Transfer Family uses AWS Secrets Manager secrets \(p. 156\)](#)

How Alexa for Business uses AWS Secrets Manager

Alexa for Business makes it easy for you to use Alexa in your organization. Alexa for Business gives you the tools you need to manage Alexa devices, enroll your users, and assign skills, at scale.

To simplify the process of creating and managing network configurations, you can define network profiles. Network profiles are associated with devices and consist of network configuration settings, including the SSID, network security type, network credentials, and description.

When you create a network profile for a password-based Wi-Fi network, Alexa for Business stores your passwords in Secrets Manager. For more information, see [Manage network profiles](#).

How AWS App2Container uses AWS Secrets Manager

AWS App2Container (A2C) is a command line tool to help you lift and shift applications that run in your on-premises data centers or on virtual machines, so that they run in containers that are managed by Amazon ECS, Amazon EKS, or AWS App Runner.

App2Container uses Secrets Manager to manage the credentials for connecting your worker machine to application servers in order to run remote commands. For more information, see [Manage secrets for AWS App2Container](#).

How Amazon AppFlow uses AWS Secrets Manager

Amazon AppFlow is a fully-managed integration service that enables you to securely exchange data between software as a service (SaaS) applications, such as Salesforce, and AWS services, such as Amazon Simple Storage Service (Amazon S3) and Amazon Redshift.

In Amazon AppFlow, when you configure an SaaS application as a source or destination, you create a connection. This includes information required for connecting to the SaaS applications, such as authentication tokens, user names, and passwords. Amazon AppFlow securely stores your connection data in a Secrets Manager secret. For more information, see [Data protection in Amazon AppFlow](#).

How AWS AppSync uses AWS Secrets Manager

AWS AppSync provides a robust, scalable GraphQL interface for application developers to combine data from multiple sources, including Amazon DynamoDB, AWS Lambda, and HTTP APIs.

AWS AppSync uses the CLI command `rds execute-statement` to connect to Amazon RDS using the credentials in a secret. For more information, see [Tutorial: Aurora Serverless](#).

How Amazon Athena uses AWS Secrets Manager

Amazon Athena is an interactive query service that makes it easy to analyze data directly in Amazon Simple Storage Service (Amazon S3) using standard SQL.

Amazon Athena data source connectors can use the Athena Federated Query feature with Secrets Manager secrets to query data. For more information, see [Using Amazon Athena Federated Query](#).

How AWS CodeBuild uses AWS Secrets Manager

AWS CodeBuild is a fully managed build service in the cloud. CodeBuild compiles your source code, runs unit tests, and produces artifacts ready to deploy.

You can store your private registry credentials using Secrets Manager. For more information, see [Private registry with AWS Secrets Manager sample for CodeBuild](#).

How AWS Directory Service uses AWS Secrets Manager

AWS Directory Service provides multiple ways to use Microsoft Active Directory (AD) with other AWS services. You can join an Amazon EC2 instance to your directory using secrets for credentials:

- [Seamlessly join a Linux EC2 instance to your AWS Managed Microsoft AD directory](#)
- [Seamlessly join a Linux EC2 instance to your AD Connector directory](#)
- [Seamlessly join a Linux EC2 instance to your Simple AD directory](#)

How Amazon DocumentDB (with MongoDB compatibility) uses AWS Secrets Manager

In Amazon DocumentDB, users authenticate to a cluster in conjunction with a password. With AWS Secrets Manager, you can replace hardcoded credentials in your code (including passwords) with an API call to Secrets Manager to retrieve the secret programmatically. For more information, see [the section called "Create a database secret" \(p. 49\)](#) and [Managing Amazon DocumentDB Users](#).

How AWS Elemental Live uses AWS Secrets Manager

AWS Elemental Live is a real-time video service that lets you create live outputs for broadcast and streaming delivery.

AWS Elemental Live uses a secret ARN to get a secret that contains an encryption key from Secrets Manager. Elemental Live uses the encryption key to encrypt/decrypt the video. For more information, see [How delivery from AWS Elemental Live to MediaConnect works at runtime](#)

How AWS Elemental MediaConnect uses AWS Secrets Manager

AWS Elemental MediaConnect is a service that makes it easy for broadcasters and other premium video providers to reliably ingest live video into the AWS Cloud and distribute it to multiple destinations inside or outside the AWS Cloud.

You can use static key encryption to protect your sources, outputs, and entitlements, and you store your encryption key in AWS Secrets Manager. For more information, see [Static key encryption in AWS Elemental MediaConnect](#).

How AWS Elemental MediaConvert uses AWS Secrets Manager

AWS Elemental MediaConvert is a file-based video processing service that provides scalable video processing for content owners and distributors with media libraries of any size. To use MediaConvert to encode Kantar watermarks, you use Secrets Manager to store your Kantar credentials. For more information, see [Using Kantar for audio watermarking in AWS Elemental MediaConvert outputs](#).

Find unprotected secrets in your code with Amazon CodeGuru Reviewer

Amazon CodeGuru Reviewer is a service that uses program analysis and machine learning to detect potential defects that are difficult for developers to find and offers suggestions for improving your Java and Python code. CodeGuru Reviewer integrates with Secrets Manager to find unprotected secrets in your code. For the types of secrets it can find, see [Types of secrets detected by CodeGuru Reviewer](#).

Once you've found hardcoded secrets, take action to replace them:

- [the section called "Replace hardcoded DB credentials" \(p. 15\)](#)
- [the section called "Replace hardcoded secrets" \(p. 12\)](#)

How AWS Elemental MediaPackage uses AWS Secrets Manager

AWS Elemental MediaPackage is a just-in-time video packaging and origination service that runs in the AWS Cloud. With MediaPackage, you can deliver highly secure, scalable, and reliable video streams to a wide variety of playback devices and content delivery networks (CDNs). For more information, see [Secrets Manager access for CDN authorization](#).

How AWS Elemental MediaTailor uses AWS Secrets Manager

AWS Elemental MediaTailor is a scalable ad insertion and channel assembly service that runs in the AWS Cloud.

MediaTailor supports Secrets Manager access token authentication to your source locations. With Secrets Manager access token authentication, MediaTailor uses a Secrets Manager secret to authenticate requests to your origin. For more information, see [Configuring AWS Secrets Manager access token authentication](#).

How Amazon EMR uses AWS Secrets Manager

Amazon EMR is a managed cluster platform that simplifies running big data frameworks, such as Apache Hadoop and Apache Spark, on AWS to process and analyze vast amounts of data. By using these

frameworks and related open-source projects, such as Apache Hive and Apache Pig, you can process data for analytics purposes and business intelligence workloads. Additionally, you can use Amazon EMR to transform and move large amounts of data into and out of other AWS data stores and databases, such as Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB.

You can store your private Git-based registry credentials using Secrets Manager. For more information, see [Add a Git-based Repository to Amazon EMR](#).

How Amazon EventBridge uses AWS Secrets Manager

Amazon EventBridge is a serverless event bus service that you can use to connect your applications with data from a variety of sources.

Amazon EventBridge API destinations are HTTP endpoints that you can invoke as the target of an EventBridge rule. When you create an API destination, you specify a connection to use for it, which EventBridge stores in a secret in Secrets Manager. The cost of storing the Secrets Manager secret is included with the charge for using an API destination. For more information, see [API destinations](#).

How Amazon FSx uses AWS Secrets Manager secrets

Amazon FSx for Windows File Server provides fully managed Microsoft Windows file servers, backed by a fully native Windows file system. When you create or manage file shares, you can pass credentials from an AWS Secrets Manager secret. For more information, see [File shares](#) and [Migrating file share configurations to Amazon FSx](#).

How AWS Glue DataBrew uses AWS Secrets Manager

AWS Glue DataBrew is a visual data preparation tool that you can use to clean and normalize data without writing any code. In DataBrew, a set of data transformation steps is called a recipe. DataBrew provides the following recipe steps to perform transformations on personally identifiable information (PII) in a dataset, which use a Secrets Manager secret as an encryption key:

- [DETERMINISTIC_DECRYPT](#)
- [DETERMINISTIC_ENCRYPT](#)
- [CRYPTOGRAPHIC_HASH](#)

How AWS Glue Studio uses AWS Secrets Manager

AWS Glue Studio is a graphical interface that makes it easy to create, run, and monitor extract, transform, and load (ETL) jobs in AWS Glue. You can use OpenSearch as a data store for your extract, transform, and load (ETL) jobs by configuring the Elasticsearch Spark Connector in AWS Glue Studio. To

connect to the OpenSearch cluster, you can use a secret in Secrets Manager. For more information, see [Setting up for AWS Glue Studio](#).

How AWS IoT SiteWise uses AWS Secrets Manager

AWS IoT SiteWise is a managed service that lets you collect, model, analyze, and visualize data from industrial equipment at scale. You can use the AWS IoT SiteWise console to create a gateway. Then add data sources, local servers or industrial equipment that are connected to gateways. If your source requires authentication, use a secret to authenticate. For more information, see [Configuring data source authentication](#).

How Amazon Kendra uses AWS Secrets Manager

Amazon Kendra is a highly accurate and intelligent search service that enables your users to search unstructured and structured data using natural language processing and advanced search algorithms.

You can index documents stored in a database by specifying a secret that contains credentials for the database. For more information, see [Using a database data source](#).

How AWS Launch Wizard uses AWS Secrets Manager

AWS Launch Wizard for Active Directory is a service that applies AWS cloud application best practices to guide you through setting up a new Active Directory infrastructure, or adding domain controllers to an existing infrastructure, either in the AWS Cloud or on premises.

AWS Launch Wizard requires domain administrator credentials to be added to Secrets Manager to join your domain controllers to Active Directory. For more information, see [Set up for AWS Launch Wizard for Active Directory](#).

How Amazon Lookout for Metrics uses AWS Secrets Manager

Amazon Lookout for Metrics is a service that finds anomalies in your data, determines their root causes, and enables you to quickly take action. You can use Amazon Redshift or Amazon RDS as a datasource for an Lookout for Metrics detector. To configure the datasource, you use a secret that contains the database password. For more information, see [Using Amazon RDS with Lookout for Metrics](#) and [Using Amazon Redshift with Lookout for Metrics](#).

How Amazon Managed Streaming for Apache Kafka uses AWS Secrets Manager

Amazon Managed Streaming for Apache Kafka (Amazon MSK) is a fully managed service that enables you to build and run applications that use Apache Kafka to process streaming data. You can control

access to your Amazon MSK clusters using usernames and passwords that are stored and secured using AWS Secrets Manager. For more information, see [Username and password authentication with AWS Secrets Manager](#).

How Amazon Managed Workflows for Apache Airflow uses AWS Secrets Manager

Amazon Managed Workflows for Apache Airflow (MWAA) is a managed orchestration service for [Apache Airflow](#) that makes it easier to setup and operate end-to-end data pipelines in the cloud at scale.

You can configure an Apache Airflow connection using a Secrets Manager secret. For more information, see [Configuring an Apache Airflow connection using a Secrets Manager secret](#) and [Using a secret key in AWS Secrets Manager for an Apache Airflow variable](#).

How AWS Migration Hub uses AWS Secrets Manager

AWS Migration Hub provides a single location to track migration tasks across multiple AWS tools and partner solutions.

AWS Migration Hub Orchestrator simplifies and automates the migration of servers and enterprise applications to AWS. Migration Hub Orchestrator uses a secret for the connection information to your source server. For more information, see:

- [Migrate SAP NetWeaver applications to AWS](#)
- [Rehost applications on Amazon EC2](#)

Migration Hub Strategy Recommendations offers migration and modernization strategy recommendations for viable transformation paths for your applications. Strategy Recommendations can analyze SQL Server databases, using a secret for the connection information. For more information, see [Strategy Recommendations database analysis](#).

How AWS OpsWorks for Chef Automate uses AWS Secrets Manager

AWS OpsWorks is a configuration management service that helps you configure and operate applications in a cloud enterprise by using Puppet or Chef.

When you create a new server in AWS OpsWorks CM, OpsWorks CM stores secrets for the server in Secrets Manager. For more information, see [AWS OpsWorks for Chef Automate: Integration with AWS Secrets Manager](#).

How Amazon RDS uses AWS Secrets Manager

Amazon Relational Database Service (Amazon RDS) is a web service that makes it easier to set up, operate, and scale a relational database in the AWS Cloud. To manage user credentials for Amazon

RDS, we recommend that you use Secrets Manager secrets. For more information, see [the section called “Create a database secret”](#) (p. 49) and [Security best practices for Amazon RDS](#).

When you call the Amazon RDS Data API, you can pass credentials for the database by using a secret in Secrets Manager. For more information, see [Using the Data API for Aurora Serverless](#).

When you use the Amazon RDS query editor to connect to a database, it stores your credentials in a Secrets Manager secret. For more information, see [Using the query editor for Aurora Serverless](#).

How Amazon Redshift uses AWS Secrets Manager

Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud. To manage user credentials for Amazon Redshift, we recommend you use Secrets Manager secrets. For more information, see [the section called “Create a database secret”](#) (p. 49) and [Storing database credentials in AWS Secrets Manager](#).

When you call the Amazon Redshift Data API, you can pass credentials for the cluster by using a secret in Secrets Manager. For more information, see [Using the Amazon Redshift Data API](#).

When you use the Amazon Redshift query editor to connect to a cluster, it stores your credentials in a Secrets Manager secret. For more information, see [Working with query editor](#).

How Amazon SageMaker uses AWS Secrets Manager

SageMaker is a fully managed machine learning service. With SageMaker, data scientists and developers can quickly and easily build and train machine learning models, and then directly deploy them into a production-ready hosted environment. It provides an integrated Jupyter authoring notebook instance for easy access to your data sources for exploration and analysis, so you don't have to manage servers.

You can associate Git repositories with your Jupyter notebook instances to save your notebooks in a source control environment that persists even if you stop or delete your notebook instance. You can manage your private repositories credentials using Secrets Manager. For more information, see [Associate Git Repositories with Amazon SageMaker Notebook Instances](#).

To import data from Databricks, Data Wrangler stores your JDBC URL in Secrets Manager. For more information, see [Import data from Databricks \(JDBC\)](#).

To import data from Snowflake, Data Wrangler stores your credentials in a Secrets Manager secret. For more information, see [Import data from Snowflake](#).

How AWS Toolkit for JetBrains uses AWS Secrets Manager

The AWS Toolkit for JetBrains is an open source plugin for the integrated development environments (IDEs) from JetBrains. The toolkit makes it easier for developers to develop, debug, and deploy serverless applications that use AWS. When connecting to an Amazon Redshift cluster using the toolkit, you can authenticate using a Secrets Manager secret. For more information, see [Accessing Amazon Redshift clusters](#).

How AWS Transfer Family uses AWS Secrets Manager secrets

AWS Transfer Family is a secure transfer service that enables you to transfer files into and out of AWS storage services.

To authenticate Transfer Family users, you can use AWS Secrets Manager as an identity provider. For more information, see [Working with custom identity providers](#).

Security in AWS Secrets Manager

Security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture built to meet the requirements of the most security-sensitive organizations.

You and AWS share the responsibility for security. The [shared responsibility model](#) describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS Compliance Programs](#). To learn about the compliance programs that apply to AWS Secrets Manager, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your AWS service determines your responsibility. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

Topics

- [Secrets Manager best practices](#) (p. 157)
- [Mitigate the risks of using the AWS CLI to store your secrets](#) (p. 158)
- [Data protection in AWS Secrets Manager](#) (p. 159)
- [Secret encryption and decryption](#) (p. 161)
- [Infrastructure security in AWS Secrets Manager](#) (p. 167)
- [Resiliency in AWS Secrets Manager](#) (p. 167)

Secrets Manager best practices

The following recommendations help you to more securely use AWS Secrets Manager:

Add retries to your application

Your AWS client might see calls to Secrets Manager fail due to rate limiting. When you exceed an API request quota, Secrets Manager throttles the request. To respond, use a backoff and retry strategy. See [the section called "Add retries to your application"](#) (p. 173).

Mitigate the risks of logging and debugging your Lambda function

When you create a Lambda rotation function, be cautious about including debugging or logging statements in your function. These statements can cause information in your function to be written to Amazon CloudWatch, so make sure the log doesn't include any sensitive data from the secret. If you do include these statements in your code for testing and debugging, make sure you remove them before using the code in production. Also remove any logs that include sensitive information collected during development.

The Lambda functions for [supported databases](#) (p. 3) don't include logging and debug statements.

Mitigate the risks of using the AWS CLI to store your secrets

When you use the AWS CLI and enter commands in a command shell, there is a risk of the command history being accessed or utilities having access to your command parameters. See [the section called "Mitigate the risks of using the AWS CLI to store your secrets"](#) (p. 158).

Run everything in a VPC

We recommend that you run as much of your infrastructure as possible on private networks that are not accessible from the public internet. See [VPC endpoint \(p. 137\)](#).

Rotate secrets on a schedule

If you don't change your secrets for a long period of time, the secrets become more likely to be compromised. We recommend that you rotate your secrets every 30 days. See [Rotate AWS Secrets Manager secrets \(p. 115\)](#)

Monitor your secrets

Monitor your secrets and log any changes to them. You can use the logs if you need to investigate any unexpected usage or change, and then you can roll back unwanted changes. You can also set automated checks for inappropriate usage of secrets and any attempts to delete secrets. See [Monitor AWS Secrets Manager secrets \(p. 139\)](#).

Use Secrets Manager to provide credentials to Lambda functions

Use Secrets Manager to securely provide database credentials to Lambda functions without hardcoding the secrets in code or passing them through environmental variables. See [How to securely provide database credentials to Lambda functions by using AWS Secrets Manager](#).

More resources on best practices

For more resources, see [Security Pillar - AWS Well-Architected Framework](#).

Mitigate the risks of using the AWS CLI to store your secrets

When you use the AWS Command Line Interface (AWS CLI) to invoke AWS operations, you enter those commands in a command shell. For example, you can use the Windows command prompt or Windows PowerShell, or the Bash or Z shell, among others. Many of these command shells include functionality designed to increase productivity. But this functionality can be used to compromise your secrets. For example, in most shells, you can use the up arrow key to see the last entered command. The *command history* feature can be exploited by anyone who accesses your unsecured session. Also, other utilities that work in the background might have access to your command parameters, with the intended goal of helping you perform your tasks more efficiently. To mitigate such risks, ensure you take the following steps:

- Always lock your computer when you walk away from your console.
- Uninstall or disable console utilities you don't need or no longer use.
- Ensure the shell or the remote access program, if you are using one or the other, don't log typed commands .
- Use techniques to pass parameters not captured by the shell command history. The following example shows how you can type the secret text into a text file, and then pass the file to the AWS Secrets Manager command and immediately destroy the file. This means the typical shell history doesn't capture the secret text.

The following example shows typical Linux commands but your shell might require slightly different commands:

```
$ touch secret.txt
# Creates an empty text file
$ chmod go-rx secret.txt
# Restricts access to the file to only the user
```

```
$ cat > secret.txt
# Redirects standard input (STDIN) to the text file
ThisIsMyTopSecretPassword^D
# Everything the user types from this point up to the CTRL-D (^D) is saved in the
file
$ aws secretsmanager create-secret --name TestSecret --secret-string file://secret.txt
# The Secrets Manager command takes the --secret-string parameter from the contents
of the file
$ shred -u secret.txt
# The file is destroyed so it can no longer be accessed.
```

After you run these commands, you should be able to use the up and down arrows to scroll through the command history and see that the secret text isn't displayed on any line.

Important

By default, you can't perform an equivalent technique in Windows unless you first reduce the size of the command history buffer to 1.

To configure the Windows Command Prompt to have only 1 command history buffer of 1 command

1. Open an Administrator command prompt (**Run as administrator**).
2. Choose the icon in the upper left, and then choose **Properties**.
3. On the **Options** tab, set **Buffer Size** and **Number of Buffers** both to 1, and then choose **OK**.
4. Whenever you have to type a command you don't want in the history, immediately follow it with one other command, such as:

```
echo.
```

This ensures you flush the sensitive command.

For the Windows Command Prompt shell, you can download the [SysInternals SDelete](#) tool, and then use commands similar to the following:

```
C:\> echo. 2> secret.txt
# Creates an empty file
C:\> icacls secret.txt /remove "BUILTIN\Administrators" "NT AUTHORITY\SYSTEM" /
inheritance:r # Restricts access to the file to only the owner
C:\> copy con secret.txt /y
# Redirects the keyboard to text file, suppressing prompt to overwrite
THIS IS MY TOP SECRET PASSWORD^Z
# Everything the user types from this point up to the CTRL-Z (^Z) is saved in the file
C:\> aws secretsmanager create-secret --name TestSecret --secret-string file://secret.txt
# The Secrets Manager command takes the --secret-string parameter from the contents of
the file
C:\> sdelete secret.txt
# The file is destroyed so it can no longer be accessed.
```

Data protection in AWS Secrets Manager

The AWS [shared responsibility model](#) applies to data protection in AWS Secrets Manager. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For

more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a **Name** field. This includes when you work with Secrets Manager or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption at rest

Secrets Manager uses encryption via AWS Key Management Service (AWS KMS) to protect the confidentiality of data at rest. AWS KMS provides a key storage and encryption service used by many AWS services. Secrets Manager associates every secret with a KMS key. The associated KMS key can either be the Secrets Manager AWS managed key for the account, or you can create your own customer managed key in AWS KMS. For more information, see [the section called "Secret encryption and decryption"](#) (p. 161).

Encryption in transit

Secrets Manager provides secure and private endpoints for encrypting data in transit. The secure and private endpoints allows AWS to protect the integrity of API requests to Secrets Manager. AWS requires API calls be signed by the caller using X.509 certificates and/or a Secrets Manager Secret Access Key. This requirement is stated in the [Signature Version 4 Signing Process](#) (Sigv4).

If you use the AWS Command Line Interface (AWS CLI) or any of the AWS SDKs to make calls to AWS, you configure the access key to use. Then those tools automatically use the access key to sign the requests for you.

Encryption key management

When Secrets Manager needs to encrypt a new version of the protected secret data, Secrets Manager sends a request to AWS KMS to generate a new data key from the KMS key. Secrets Manager uses this data key for [envelope encryption](#). Secrets Manager stores the encrypted data key with the encrypted secret. When the secret needs to be decrypted, Secrets Manager asks AWS KMS to decrypt the data key. Secrets Manager then uses the decrypted data key to decrypt the encrypted secret. Secrets Manager never stores the data key in unencrypted form and removes the key from memory as soon as possible. For more information, see [the section called "Secret encryption and decryption"](#) (p. 161).

Inter-network traffic privacy

AWS offers options for maintaining privacy when routing traffic through known and private network routes.

Traffic between service and on-premises clients and applications

You have two connectivity options between your private network and AWS Secrets Manager:

- An AWS Site-to-Site VPN connection. For more information, see [What is AWS Site-to-Site VPN?](#)
- An AWS Direct Connect connection. For more information, see [What is AWS Direct Connect?](#)

Traffic between AWS resources in the same Region

If want to secure traffic between Secrets Manager and API clients in AWS, set up an [AWS PrivateLink](#) to privately access Secrets Manager API endpoints.

Secret encryption and decryption

Secrets Manager uses [envelope encryption](#) with AWS KMS [keys](#) and [data keys](#) to protect each secret value. Whenever the secret value in a secret changes, Secrets Manager generates a new data key to protect it. The data key is encrypted under a KMS key and stored in the metadata of the secret. To decrypt the secret, Secrets Manager first decrypts the encrypted data key using the KMS key in AWS KMS.

Secrets Manager does not use the KMS key to encrypt the secret value directly. Instead, it uses the KMS key to generate and encrypt a 256-bit Advanced Encryption Standard (AES) symmetric [data key](#), and uses the data key to encrypt the secret value. Secrets Manager uses the plaintext data key to encrypt the secret value outside of AWS KMS, and then removes it from memory. It stores the encrypted copy of the data key in the metadata of the secret.

When you create a secret, you can choose any symmetric encryption customer managed key in the AWS account and Region, or you can use the AWS managed key for Secrets Manager (`aws/secretsmanager`). In the console, if you choose the default value for the encryption key, Secrets Manager creates the AWS managed key `aws/secretsmanager`, if it doesn't already exist, and associates it with the secret. You can use the same KMS key or different KMS keys for each secret in your account. Secrets Manager supports only [symmetric encryption KMS keys](#).

You can change the encryption key for a secret in the console or in the AWS CLI or an AWS SDK with [UpdateSecret](#). When you change the encryption key, Secrets Manager re-encrypts versions of the secret that have the staging labels `AWSCURRENT`, `AWSPENDING`, and `AWSPREVIOUS` under the new encryption key. When the secret value changes, Secrets Manager also encrypts it under the new key. You can use the old key or the new one to decrypt the secret when you retrieve it.

To find the KMS key associated with a secret, view the secret in the console or call [ListSecrets](#) or [DescribeSecret](#). When the secret is associated with the AWS managed key for Secrets Manager (`aws/secretsmanager`), these operations do not return a KMS key identifier.

Topics

- [Encryption and decryption processes \(p. 162\)](#)
- [How Secrets Manager uses your KMS key \(p. 162\)](#)
- [Permissions for the KMS key \(p. 163\)](#)

- [Secrets Manager encryption context](#) (p. 164)
- [Monitor Secrets Manager interaction with AWS KMS](#) (p. 165)

Encryption and decryption processes

To encrypt the secret value in a secret, Secrets Manager uses the following process.

1. Secrets Manager calls the AWS KMS [GenerateDataKey](#) operation with the ID of the KMS key for the secret and a request for a 256-bit AES symmetric key. AWS KMS returns a plaintext data key and a copy of that data key encrypted under the KMS key.
2. Secrets Manager uses the plaintext data key and the Advanced Encryption Standard (AES) algorithm to encrypt the secret value outside of AWS KMS. It removes the plaintext key from memory as soon as possible after using it.
3. Secrets Manager stores the encrypted data key in the metadata of the secret so it is available to decrypt the secret value. However, none of the Secrets Manager APIs return the encrypted secret or the encrypted data key.

To decrypt an encrypted secret value:

1. Secrets Manager calls the AWS KMS [Decrypt](#) operation and passes in the encrypted data key.
2. AWS KMS uses the KMS key for the secret to decrypt the data key. It returns the plaintext data key.
3. Secrets Manager uses the plaintext data key to decrypt the secret value. Then it removes the data key from memory as soon as possible.

How Secrets Manager uses your KMS key

Secrets Manager uses the KMS key that is associated with a secret to generate a data key for each secret value. Secrets Manager also uses the KMS key to decrypt that data key when it needs to decrypt the encrypted secret value. You can track the requests and responses in AWS CloudTrail events, [Amazon CloudWatch Logs](#), and audit trails.

The following Secrets Manager operations trigger a request to use your KMS key.

GenerateDataKey

Secrets Manager calls the AWS KMS [GenerateDataKey](#) operation in response to the following Secrets Manager operations.

- [CreateSecret](#) – If the new secret includes a secret value, Secrets Manager requests a new data key to encrypt it.
- [PutSecretValue](#) – Secrets Manager requests a new data key to encrypt the specified secret value.
- [UpdateSecret](#) – If you change the secret value or the KMS key, Secrets Manager requests a new data key to encrypt the new secret value.

Note

The [RotateSecret](#) operation does not call [GenerateDataKey](#), because it does not change the secret value. However, if the Lambda function that [RotateSecret](#) invokes changes the secret value, its call to the [PutSecretValue](#) operation triggers a [GenerateDataKey](#) request.

Decrypt

To decrypt an encrypted secret value, Secrets Manager calls the AWS KMS [Decrypt](#) operation to decrypt the encrypted data key in the secret. Then, it uses the plaintext data key to decrypt the encrypted secret value.

Secrets Manager calls the [Decrypt](#) operation in response to the following Secrets Manager operations.

- [GetSecretValue](#) – Secrets Manager decrypts the secret value before returning it to the caller.
- [PutSecretValue](#) and [UpdateSecret](#) – Most [PutSecretValue](#) and [UpdateSecret](#) requests do not trigger a [Decrypt](#) operation. However, when a [PutSecretValue](#) or [UpdateSecret](#) request attempts to change the secret value in an existing version of a secret, Secrets Manager decrypts the existing secret value and compares it to the secret value in the request to confirm that they are the same. This action ensures that Secrets Manager operations are idempotent.

Validating access to the KMS key

When you establish or change the KMS key that is associated with secret, Secrets Manager calls the [GenerateDataKey](#) and [Decrypt](#) operations with the specified KMS key. These calls confirm that the caller has permission to use the KMS key for these operation. Secrets Manager discards the results of these operations; it does not use them in any cryptographic operation.

You can identify these validation calls because the value of the [SecretVersionId](#) key [encryption context](#) in these requests is [RequestToValidateKeyAccess](#).

Note

In the past, Secrets Manager validation calls did not include an encryption context. You might find calls with no encryption context in older AWS CloudTrail logs.

Permissions for the KMS key

When Secrets Manager uses a KMS key in cryptographic operations, it acts on behalf of the user who is creating or changing the secret value in the secret.

To use the KMS key for a secret on your behalf, the user must have the following permissions. You can specify these required permissions in an IAM policy or key policy.

- `kms:GenerateDataKey`
- `kms:Decrypt`

To allow the KMS key to be used only for requests that originate in Secrets Manager, you can use the [kms:ViaService condition key](#) with the `secretsmanager.<Region>.amazonaws.com` value.

You can also use the keys or values in the [encryption context](#) as a condition for using the KMS key for cryptographic operations. For example, you can use a [string condition operator](#) in an IAM or key policy document, or use a [grant constraint](#) in a grant.

Key policy of the AWS managed key (aws/secretsmanager)

The key policy for the AWS managed key for Secrets Manager (`aws/secretsmanager`) gives users permission to use the KMS key for specified operations only when Secrets Manager makes the request on the user's behalf. The key policy does not allow any user to use the KMS key directly.

This key policy, like the policies of all [AWS managed keys](#), is established by the service. You cannot change the key policy, but you can view it at any time. For details, see [Viewing a key policy](#).

The policy statements in the key policy have the following effect:

- Allow users in the account to use the KMS key for cryptographic operations only when the request comes from Secrets Manager on their behalf. The `kms:ViaService` condition key enforces this restriction.
- Allows the AWS account to create IAM policies that allow users to view KMS key properties and revoke grants.

- Although Secrets Manager does not use grants to gain access to the KMS key, the policy also allows Secrets Manager to [create grants](#) for the KMS key on the user's behalf and allows the account to [revoke any grant](#) that allows Secrets Manager to use the KMS key. These are standard elements of policy document for an AWS managed key.

The following is a key policy for an example AWS managed key for Secrets Manager.

```
{
  "Version" : "2012-10-17",
  "Id" : "auto-secretsmanager-1",
  "Statement" : [ {
    "Sid" : "Allow access through AWS Secrets Manager for all principals in the account
that are authorized to use AWS S
ecrets Manager",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "*"
    },
    "Action" : [ "kms:Decrypt", "kms:ReEncrypt*", "kms:GenerateDataKey*",
"kms:CreateGrant", "kms:Describ
eKey" ],
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "kms:ViaService" : "secretsmanager.us-west-2.amazonaws.com",
        "kms:CallerAccount" : "111122223333"
      }
    }
  }, {
    "Sid" : "Allow direct access to key metadata to the account",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*", "kms:RevokeGrant" ],
    "Resource" : "*"
  } ]
}
```

Secrets Manager encryption context

An [encryption context](#) is a set of key-value pairs that contain arbitrary nonsecret data. When you include an encryption context in a request to encrypt data, AWS KMS cryptographically binds the encryption context to the encrypted data. To decrypt the data, you must pass in the same encryption context.

In its [GenerateDataKey](#) and [Decrypt](#) requests to AWS KMS, Secrets Manager uses an encryption context with two name-value pairs that identify the secret and its version, as shown in the following example. The names do not vary, but combined encryption context values will be different for each secret value.

```
"encryptionContext": {
  "SecretARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:test-secret-a1b2c3",
  "SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987SECRET1"
}
```

You can use the encryption context to identify these cryptographic operation in audit records and logs, such as [AWS CloudTrail](#) and Amazon CloudWatch Logs, and as a condition for authorization in policies and grants.

The Secrets Manager encryption context consists of two name-value pairs.

- **SecretARN** – The first name–value pair identifies the secret. The key is `SecretARN`. The value is the Amazon Resource Name (ARN) of the secret.

```
"SecretARN": "ARN of an Secrets Manager secret"
```

For example, if the ARN of the secret is `arn:aws:secretsmanager:us-east-2:111122223333:secret:test-secret-a1b2c3`, the encryption context would include the following pair.

```
"SecretARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:test-secret-a1b2c3"
```

- **SecretVersionId** – The second name–value pair identifies the version of the secret. The key is `SecretVersionId`. The value is the version ID.

```
"SecretVersionId": "<version-id>"
```

For example, if the version ID of the secret is `EXAMPLE1-90ab-cdef-fedc-ba987SECRET1`, the encryption context would include the following pair.

```
"SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987SECRET1"
```

When you establish or change the KMS key for a secret, Secrets Manager sends [GenerateDataKey](#) and [Decrypt](#) requests to AWS KMS to validate that the caller has permission to use the KMS key for these operations. It discards the responses; it does not use them on the secret value.

In these validation requests, the value of the `SecretARN` is the actual ARN of the secret, but the `SecretVersionId` value is `RequestToValidateKeyAccess`, as shown in the following example encryption context. This special value helps you to identify validation requests in logs and audit trails.

```
"encryptionContext": {  
  "SecretARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:test-secret-a1b2c3",  
  "SecretVersionId": "RequestToValidateKeyAccess"  
}
```

Note

In the past, Secrets Manager validation requests did not include an encryption context. You might find calls with no encryption context in older AWS CloudTrail logs.

Monitor Secrets Manager interaction with AWS KMS

You can use AWS CloudTrail and Amazon CloudWatch Logs to track the requests that Secrets Manager sends to AWS KMS on your behalf. For information about monitoring the use of secrets, see [Monitor secrets](#) (p. 139).

GenerateDataKey

When you [create or change](#) the secret value in a secret, Secrets Manager sends a [GenerateDataKey](#) request to AWS KMS that specifies the KMS key for the secret.

The event that records the `GenerateDataKey` operation is similar to the following example event. The request is invoked by `secretsmanager.amazonaws.com`. The parameters include the Amazon Resource Name (ARN) of the KMS key for the secret, a key specifier that requires a 256-bit key, and the [encryption context](#) that identifies the secret and version.

```
{
```

```
"eventVersion": "1.05",
"userIdentity": {
  "type": "IAMUser",
  "principalId": "AROAIQDTESTANDEXAMPLE:user01",
  "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
  "accountId": "111122223333",
  "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "sessionContext": {
    "attributes": {
      "mfaAuthenticated": "false",
      "creationDate": "2018-05-31T23:23:41Z"
    }
  },
  "invokedBy": "secretsmanager.amazonaws.com"
},
"eventTime": "2018-05-31T23:23:41Z",
"eventSource": "kms.amazonaws.com",
"eventName": "GenerateDataKey",
"awsRegion": "us-east-2",
"sourceIPAddress": "secretsmanager.amazonaws.com",
"userAgent": "secretsmanager.amazonaws.com",
"requestParameters": {
  "keyId": "arn:aws:kms:us-east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
  "keySpec": "AES_256",
  "encryptionContext": {
    "SecretARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:test-secret-alb2c3",
    "SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987SECRET1"
  }
},
"responseElements": null,
"requestID": "a7d4dd6f-6529-11e8-9881-67744a270888",
"eventID": "af7476b6-62d7-42c2-bc02-5ce86c21ed36",
"readOnly": true,
"resources": [
  {
    "ARN": "arn:aws:kms:us-east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "accountId": "111122223333",
    "type": "AWS::KMS::Key"
  }
],
"eventType": "AwsApiCall",
"recipientAccountId": "111122223333"
}
```

Decrypt

Whenever you [get or change](#) the secret value of a secret, Secrets Manager sends a [Decrypt](#) request to AWS KMS to decrypt the encrypted data key.

The event that records the Decrypt operation is similar to the following example event. The user is the principal in your AWS account who is accessing the table. The parameters include the encrypted table key (as a ciphertext blob) and the [encryption context](#) that identifies the table and the AWS account. AWS KMS derives the ID of the KMS key from the ciphertext.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AROAIQDTESTANDEXAMPLE:user01",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
    "accountId": "111122223333",
```

```
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2018-05-31T23:36:09Z"
      }
    },
    "invokedBy": "secretsmanager.amazonaws.com"
  },
  "eventTime": "2018-05-31T23:36:09Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "Decrypt",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "secretsmanager.amazonaws.com",
  "userAgent": "secretsmanager.amazonaws.com",
  "requestParameters": {
    "encryptionContext": {
      "SecretARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:test-secret-alb2c3",
      "SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987SECRET1"
    }
  },
  "responseElements": null,
  "requestID": "658c6a08-652b-11e8-a6d4-ffee2046048a",
  "eventID": "f333ec5c-7fc1-46b1-b985-cbda13719611",
  "readOnly": true,
  "resources": [
    {
      "ARN": "arn:aws:kms:us-east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "accountId": "111122223333",
      "type": "AWS::KMS::Key"
    }
  ],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

Infrastructure security in AWS Secrets Manager

As a managed service, AWS Secrets Manager is protected by the AWS global network security procedures described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use AWS published API calls to access Secrets Manager through the network. Clients must support Transport Layer Security (TLS) 1.1. We recommend TLS 1.2. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

Resiliency in AWS Secrets Manager

AWS builds the global infrastructure around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which connect with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability

Zones allow you to be more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information on resiliency and disaster recovery, refer to [Reliability Pillar - AWS Well-Architected Framework](#).

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

Troubleshooting AWS Secrets Manager

Use the information here to help you diagnose and fix issues that you might encounter when you're working with Secrets Manager.

For issues related to rotation, see [the section called "Troubleshoot rotation" \(p. 132\)](#).

Topics

- ["Access denied" messages when sending requests to Secrets Manager \(p. 169\)](#)
- ["Access denied" for temporary security credentials \(p. 169\)](#)
- [Changes I make aren't always immediately visible. \(p. 170\)](#)
- ["Cannot generate a data key with an asymmetric KMS key" when creating a secret \(p. 170\)](#)
- [An AWS CLI or AWS SDK operation can't find my secret from a partial ARN \(p. 170\)](#)

"Access denied" messages when sending requests to Secrets Manager

Verify that you have permissions to call the operation and resource you requested. An administrator must grant permissions by attaching an IAM policy to your IAM user, or to a group that you're a member of. If the policy statements that grant those permissions include any conditions, such as time-of-day or IP address restrictions, you also must meet those requirements when you send the request. For information about viewing or modifying policies for an IAM user, group, or role, see [Working with Policies](#) in the *IAM User Guide*. For information about permissions required for Secrets Manager, see [Authentication and access control \(p. 27\)](#).

If you're signing API requests manually, without using the [AWS SDKs](#), verify you correctly [signed the request](#).

"Access denied" for temporary security credentials

Verify the IAM user or role you're using to make the request has the correct permissions. Permissions for temporary security credentials derive from an IAM user or role. This means the permissions are limited to those granted to the IAM user or role. For more information about how permissions for temporary security credentials are determined, see [Controlling Permissions for Temporary Security Credentials](#) in the *IAM User Guide*.

Verify that your requests are signed correctly and that the request is well-formed. For details, see the [toolkit documentation](#) for your chosen SDK, or [Using Temporary Security Credentials to Request Access to AWS Resources](#) in the *IAM User Guide*.

Verify that your temporary security credentials haven't expired. For more information, see [Requesting Temporary Security Credentials](#) in the *IAM User Guide*.

For information about permissions required for Secrets Manager, see [Authentication and access control \(p. 27\)](#).

Changes I make aren't always immediately visible.

Secrets Manager uses a distributed computing model called [eventual consistency](#). Any change that you make in Secrets Manager (or other AWS services) takes time to become visible from all possible endpoints. Some of the delay results from the time it takes to send the data from server to server, from replication zone to replication zone, and from region to region around the world. Secrets Manager also uses caching to improve performance, but in some cases this can add time. The change might not be visible until the previously cached data times out.

Design your global applications to account for these potential delays. Also, ensure that they work as expected, even when a change made in one location isn't instantly visible at another.

For more information about how some other AWS services are affected by eventual consistency, see:

- [Managing data consistency](#) in the *Amazon Redshift Database Developer Guide*
- [Amazon S3 Data Consistency Model](#) in the *Amazon Simple Storage Service User Guide*
- [Ensuring Consistency When Using Amazon S3 and Amazon EMR for ETL Workflows](#) in the AWS Big Data Blog
- [Amazon EC2 Eventual Consistency](#) in the *Amazon EC2 API Reference*

"Cannot generate a data key with an asymmetric KMS key" when creating a secret

Secrets Manager uses a [symmetric encryption KMS key](#) associated with a secret to generate a data key for each secret value. You can't use an asymmetric KMS key. Verify you are using a symmetric encryption KMS key instead of an asymmetric KMS key. For instructions, see [Identifying asymmetric KMS keys](#).

An AWS CLI or AWS SDK operation can't find my secret from a partial ARN

In many cases, Secrets Manager can find your secret from part of an ARN rather than the full ARN. However, if your secret's name ends in a hyphen followed by six characters, Secrets Manager might not be able to find the secret from only part of an ARN. Instead, we recommend that you use the complete ARN.

Secrets Manager constructs an ARN for a secret with Region, account, secret name, and then a hyphen and six more characters, as follows:

```
arn:aws:secretsmanager:us-east-2:111122223333:secret:SecretName-abcdef
```

If your secret name ends with a hyphen and six characters, using only part of the ARN can appear to Secrets Manager as though you are specifying a full ARN. For example, you might have a secret named MySecret-abcdef with the ARN

```
arn:aws:secretsmanager:us-east-2:111122223333:secret:MySecret-abcdef-nutBrk
```

If you call the following operation, which only uses part of the secret ARN, then Secrets Manager might not find the secret.

```
$ aws secretsmanager describe-secret --secret-id arn:aws:secretsmanager:us-east-2:111122223333:secret:MySecret-abcdef
```

AWS Secrets Manager quotas

This section specifies quotas for AWS Secrets Manager. For Secrets Manager quotas for resources, see [the section called “Maximum quotas” \(p. 172\)](#). For Secrets Manager APIs, read APIs have high TPS quotas, and control plane APIs that are less frequently called have lower TPS quotas. For more information, see [the section called “Rate quotas” \(p. 172\)](#).

For information about **Service Endpoints**, see [AWS Secrets Manager endpoints and quotas](#) which includes regional service endpoints. You may operate multiple regions in your account, such as US East (N. Virginia) Region and US West (N. California) Region, and each quota is specific to each region.

Secret name constraints

Secrets Manager has the following constraints:

- Secret names must use Unicode characters.
- Secret names contain 1-512 characters.

Maximum quotas

You can operate multiple AWS Regions in your account, and each quota is per AWS Region.

Entity	Quota
Secrets	500,000
Versions of a secret	~100
Staging labels attached across all versions of a secret	20
Versions attached to a label at the same time	1
Length of a secret	65,536 bytes
Length of a resource-based policy - JSON text	20,480 characters

Rate quotas

You can operate multiple AWS Regions in your account, and each quota is per AWS Region.

Request type	Quota (per second)
DescribeSecret and GetSecretValue, combined	5,000

Request type	Quota (per second)
CreateSecret	50
DeleteSecret	50
GetRandomPassword	50
ListSecrets and ListSecretVersionIds, combined	50
DeleteResourcePolicy, GetResourcePolicy, PutResourcePolicy, and ValidateResourcePolicy, combined	50
PutSecretValue, RemoveRegionsFromReplication, ReplicateSecretToRegions, StopReplicationToReplica, UpdateSecret, and UpdateSecretVersionStage, combined	50
RestoreSecret	50
RotateSecret and CancelRotateSecret, combined	50
TagResource and UntagResource, combined	50

We recommend you avoid calling `PutSecretValue` or `UpdateSecret` at a sustained rate of more than once every 10 minutes. When you call `PutSecretValue` or `UpdateSecret` to update the secret value, Secrets Manager creates a new version of the secret. Secrets Manager removes outdated versions when there are more than 100, but it does not remove versions created less than 24 hours ago. If you update the secret value more than once every 10 minutes, you create more versions than Secrets Manager removes, and you will reach the quota for secret versions.

Add retries to your application

Your AWS client might see calls to Secrets Manager fail due to unexpected issues on the client side. Or calls might fail due to rate limiting from Secrets Manager. When you exceed an API request quota, Secrets Manager throttles the request. It rejects an otherwise valid request and returns a throttling error. For both kinds of failures, we recommend you retry the call after a brief waiting period. This is called a [backoff and retry strategy](#).

If you experience the following errors, you might want to add retries to your application code:

Transient errors and exceptions

- `RequestTimeout`
- `RequestTimeoutException`
- `PriorRequestNotComplete`
- `ConnectionError`
- `HTTPClientError`

Service-side throttling and limit errors and exceptions

- `Throttling`
- `ThrottlingException`
- `ThrottledException`
- `RequestThrottledException`
- `TooManyRequestsException`
- `ProvisionedThroughputExceededException`
- `TransactionInProgressException`
- `RequestLimitExceeded`
- `BandwidthLimitExceeded`
- `LimitExceededException`
- `RequestThrottled`
- `SlowDown`

For more information, as well as example code, on retries, exponential backoff, and jitter, see the following resources:

- [Exponential Backoff and Jitter](#)
- [Timeouts, retries and backoff with jitter](#)
- [Error retries and exponential backoff in AWS.](#)

Cross-account requests

When an application in one AWS account uses a secret owned by a different account, it's known as a *cross-account request*. For cross-account requests, Secrets Manager throttles the account of the identity that makes the requests, not the account that owns the secret. For example, if an identity from account A uses a secret in account B, the secret use applies only to the quotas in account A.