# AWS Secrets Manager

## User Guide

# AWS Secrets Manager: User Guide

# Table of Contents

# What is AWS Secrets Manager?

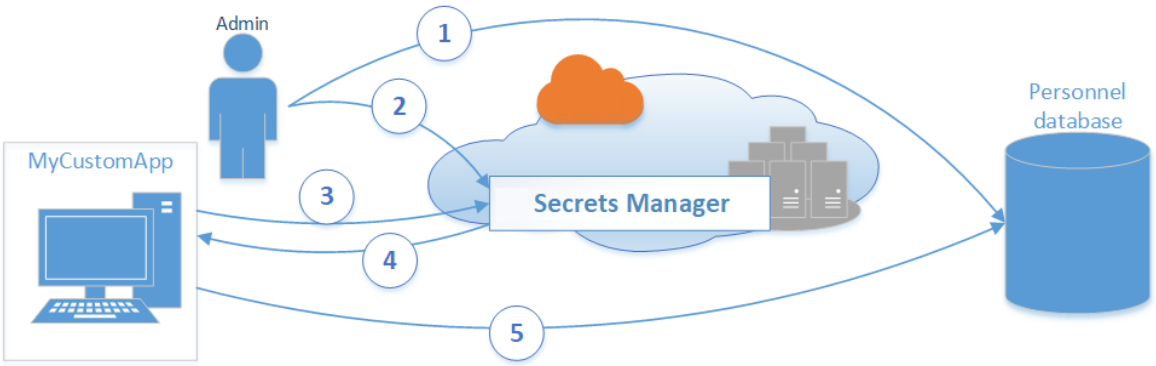
In the past, when you created a custom application to retrieve information from a database, you typically embedded the credentials, the secret, for accessing the database directly in the application. When the time came to rotate the credentials, you had to do more than just create new credentials. You 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. 9).

## Basic 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 your AWS Secrets Manager secrets (p. 68).

# Features of 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

For Amazon RDS, Amazon DocumentDB, or Amazon Redshift secret (p. 70), Secrets Manager can create rotation functions for you for the Single user (p. 68) or Alternating users (p. 69) rotation strategies.

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. 73) 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`. Use Python 3.7 to customize it.

.
  ```

## 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

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.

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

Amazon Web Services (AWS) achieved the Outsourced Service Provider's Audit Report (OSPAR) attestation. AWS alignment with the Association of Banks in Singapore (ABS) Guidelines on Control Objectives and Procedures for Outsourced Service Providers (ABS Guidelines) demonstrates to customers AWS commitment to meeting the high expectations for cloud service providers set by the financial services industry in Singapore. For more information, see OSPAR Resources

# Access Secrets Manager

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

**AWS Management Console**

You can manage your secrets using the browser-based The Secrets Manager console and perform almost any task related to your secrets by using the console.

Currently, you can't perform the following task in the console:

- *Store binary data in a secret.* The console currently stores data only in the `SecretString` field of the secret, and does not use the `SecretBinary` field. To store binary data, you must currently use the AWS CLI or one of the AWS SDKs.

**AWS 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: the AWS Command Line Interface (AWS CLI) and the AWS Tools for Windows PowerShell. For information about installing and using the AWS CLI, see the AWS Command Line Interface User Guide. For information about installing and using the Tools for Windows PowerShell, see the AWS Tools for Windows PowerShell User Guide.

**AWS SDKs**

The AWS SDKs consist of libraries and sample code for various programming languages and platforms, for example, Java, Python, Ruby, .NET, iOS and Android, and others. The SDKs include tasks such as cryptographically signing requests, managing errors, and retrying requests automatically. For more information about the AWS SDKs, including how to download and install them, see Tools for Amazon Web Services.

**Secrets Manager HTTPS Query API**

The Secrets Manager 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. When you use the HTTPS API, you must include code to digitally sign requests by using your credentials. For more information, see Calling the API by Making HTTP Query Requests and the AWS Secrets Manager API Reference.

> **Note**
> We recommend using the SDK specific to the programming language you prefer instead of using the HTTPS Query API. The SDK performs many useful tasks you perform manually. The SDKs automatically sign your requests and convert the response into a structure syntactically appropriate to your language. Use the HTTPS Query API only when an SDK is unavailable.

# Pricing for 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.

# Get started with AWS Secrets Manager

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

## Secret

In Secrets Manager, a *secret* consists of a set of credentials, user name and password, and the connection details to access a database or other service. You can also store any other type of secret information in a secret as text or binary. To store multiple 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"   : "My-P@ssw0rd!F0r+Th3_Acc0unt",
  "dbname"     : "MyDatabase",
  "engine"     : "mysql"
}
```

A secret has metadata:

- 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. 112).
- Information about how to rotate the secret, if you set up rotation. See the section called “Rotation” (p. 10).

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. 25).

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. 10).

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 “Multi-Region secrets” (p. 52).

To create a secret, see the section called “Create a secret” (p. 43).

# 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. 68).

# 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**



# Tutorial: Create and retrieve a secret

In this tutorial, you create a secret and store it in AWS Secrets Manager. You can then retrieve the secret using the AWS Management Console or the AWS CLI.

Users new to Secrets Manager can benefit from enrolling in the 30 day free trial and not receive billing for the activity performed in this tutorial.

**Step 1: Create and store your secret in AWS Secrets Manager (p. 11)**

In this step, you create a secret and provide the basic information required by AWS Secrets Manager.

**Step 2: Retrieve your secret from AWS Secrets Manager (p. 13)**

Next, you use the Secrets Manager console and the AWS CLI to retrieve the secret.

## Prerequisites

This tutorial assumes you can access an AWS account, and you can sign in to AWS as an IAM user with permissions to create and retrieve secrets in the AWS Secrets Manager console, or use equivalent commands in the AWS CLI. For more information on configuring IAM users, refer to the IAM documentation.

## Step 1: Create and store your secret in AWS Secrets Manager

Secrets Manager Console

### Creating and storing your secret from the console

1. Sign in to the AWS Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. On either the service introduction page or the **Secrets** list page, choose **Store a new secret**.
3. On the **Store a new secret** page, choose **Other type of secret**. You choose this type of secret because your secret doesn't apply to a database.
4. Under **Specify key/value pairs to be stored in the secret**, in the first field, type **`MyFirstSecret`**. To configure a password, add a value in the next field.

You provide your secret information, such as credentials and connection details, as key name and value string pairs. For example, you could specify “UserName” as a key name and the user sign-in name as the value.

5. For **Select the encryption key**, choose **DefaultEncryptionKey**. This is the AWS managed key (`aws/secretsmanager`), and there is no cost for using it. If you choose to create a customer managed key in AWS KMS, then AWS charges you at the standard AWS KMS rate.
6. Choose **Next**.
7. Under **Secret name**, type a name for the secret in the text field. You must use only alphanumeric characters and the characters /_+=.@-.

   For example, you can use a secret name such as **`tutorials/MyFirstSecret`**. This stores your secret in the virtual folder **tutorials** with the value **MyFirstSecret**. We recommend naming secrets in a hierarchical manner which makes managing your secrets easier.
8. In the **Description** field, type a description of the secret.

   For **Description**, type, for example, **`Create Secret`**
9. In the **Tags** section, add desired tags in the **Key** and **Value - optional** text fields.

   For this tutorial, you can leave tags blank. However, we recommend using tags as a best practice to help identify secrets.
10. Choose **Next**.
11. In this tutorial, choose **Disable automatic rotation**, and then choose **Next**.

    **Note**
    The next tutorial describes rotating a secret.
12. On the **Review** page, you can check your secret settings. Also, be sure to review the **Sample code** section with cut-and-paste–enabled code you can add to your own applications and use this secret to retrieve the credentials. Each tab displays the code in different programming languages.
13. To save your changes, choose **Store**.

    Secrets Manager console returns to the list of secrets in your account with your new secret now included in the list.

Secrets Manager CLI

1. Open a command prompt to run the AWS CLI. If you haven't installed the AWS CLI yet, see Installing the AWS Command Line Interface.
2. **Creating your secret**

   Type the following command and parameters:

```
$ aws secretsmanager create-secret --name tutorials/MyFirstSecret
          --description "Basic Create Secret" --secret-string S3@tt13R0cks
```

The output of the command displays the following information:

```
{
    "ARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:tutorials/
MyFirstSecret-rzM8Ja",
    "Name": "MyFirstSecret",
    "VersionId": "35e07aa2-684d-42fd-b076-3b3f6a19c6dc"
```

```
}
```

## Step 2: Retrieve your secret from AWS Secrets Manager

In this step, you retrieve the secret by using the Secrets Manager console and the AWS CLI.

### Retrieving your secret in the AWS Secrets Manager console

1. If not already logged into the console, go to the console at https://console.aws.amazon.com/secretsmanager/ and log into the Secrets Manager service.
2. On the **Secrets** list page, choose the name of the new secret you created.

   Secrets Manager displays the **Secrets details** page for your secret.
3. In the **Secret value** section, choose **Retrieve secret value**.
4. You can view your secret as either key-value pairs, or as a JSON text structure.

### To retrieve your secret by using the AWS Secrets Manager CLI

1. Open a command prompt to run the AWS CLI. If you haven't installed the AWS CLI yet, see Installing the AWS Command Line Interface.
2. Using credentials with permissions to access your secret, type the following command and parameters.

   **To view all of the details of your secret except the encrypted text:**

```
$ aws secretsmanager describe-secret --secret-id tutorials/MyFirstSecret
{
    "ARN": "arn:aws::secretsmanager:us-east-2:111122223333:secret:tutorials/
MyFirstSecret-jiObOV",
    "Name": "tutorials/MyFirstSecret",
    "Description": "Basic Create Secret",
    "LastChangedDate": 1522680794.8,
    "LastAccessedDate": 1522627200.0,
    "VersionIdsToStages": {
        "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE": [
            "AWSCURRENT"
        ]
    }
}
```

   Review the **VersionIdsToStages** response value. The output contains a list of all active versions of the secret and the staging labels attached to each version. In this tutorial, you should see a single version ID (a UUID type value) mapping to a single staging label `AWSCURRENT`.

   **To view the encrypted text in your secret:**

```
$ aws secretsmanager get-secret-value --secret-id tutorials/MyFirstSecret --version-
stage AWSCURRENT
{
    "ARN": "arn:aws:secretsmanager:us-east-2:111122223333:secret:tutorials/
MyFirstSecret-jiObOV",
    "Name": "tutorials/MyFirstSecret",
    "VersionId": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE",
    "SecretString": "S3@ttl3R0cks",
    "VersionStages": [
        "AWSCURRENT"
    ],
```

```
    "CreatedDate": 1522680764.668
}
```

If you want details for a version with a different staging label than `AWSCURRENT`, you must include the `--version-stage` parameter in the previous command. Secrets Manager uses `AWSCURRENT` as the default value.

The rest of the output includes the JSON version of your secret value in the `SecretString` response field.

**Summary**

This tutorial demonstrated how easily you can create a simple secret, and to retrieve the secret value when you need it. For another tutorial on creating a secret and configuring automatic rotation, see Tutorial: Rotate a secret for an AWS database (p. 14).

# Tutorial: Rotate a secret for an AWS database

In this tutorial, you create a secret for an AWS database and configure the secret to rotate on a schedule. You trigger one rotation manually, and then confirm that the new version of the secret continues to provide access.

**Configure a test MySQL database (p. 15)**

In this step, create a test database in Amazon Relational Database Service (Amazon RDS). For this tutorial, the test database runs MySQL.

**Step 2: Create your secret (p. 16)**

Next, use the Secrets Manager console to create your secret and populate the secret with the initial user name and password for your MySQL database. Test the secret by using the returned credentials to sign in to the database.

**Step 3: Validate your initial secret (p. 17)**

In Step 3, use your new secret to test the credentials and ensure you can use them to connect to your database.

**Step 4: Configure rotation for your secret (p. 18)**

In Step 4, enable rotation for the secret and perform the initial rotation.

**Step 5: Verify successful rotation (p. 18)**

In this step, after the initial rotation completes, repeat the validation steps to show that the new credentials generated during rotation continue to allow you to access the database.

**Step 6: Clean up (p. 19)**

In the final step, remove the Amazon RDS database instance and the secret to avoid incurring any unnecessary costs.

## Prerequisites

The tutorial assumes you can access an AWS account, and you can sign into AWS as a user with full permissions to configure AWS Secrets Manager and Amazon RDS, either using the console or the equivalent commands in the AWS CLI.

The tutorial uses a MySQL client tool, MySQLWorkbench, to interact with the database, configure users, and check status. The tutorial includes the installation instructions at the appropriate point in the following steps.

The database configured in this tutorial allows access to the public internet on port 3306, again for simplicity in setup for the tutorial. To complete this tutorial, you must be able to access the MySQL database from your internet-connected computer by using the MySQL client tool, MySQLWorkbench. We recommend you follow the guidance in the Lambda and Amazon EC2 VPC documentation to configure production servers securely.

> **Important**
> For rotation to work, your network environment must permit the Lambda rotation function to communicate with your database and the Secrets Manager service. Because this tutorial configures your database with public internet access, Lambda automatically configures your rotation function to access the database through the public IP address. If you block public internet access to your database instance, then you must configure the Lambda function to run in the same VPC as the database instance. Then you must either configure your VPC with a private Secrets Manager endpoint (p. 74), or configure the VPC with public internet access by using a NAT gateway, so that the Lambda rotation function can access the public Secrets Manager endpoint.

# Required permissions

To successfully run this tutorial, you must have all of the permissions associated with the SecretsManagerReadWrite AWS managed policy. You must also have permission to create an IAM role and attach a permission policy to the role. You can grant either the IAMFullAccess AWS managed policy, or explicitly assign `iam:CreateRole` and `iam:AttachRolePolicy`.

> **Warning**
> The `iam:CreateRole` and `iam:AttachRolePolicy` permit a user to grant themselves any permissions, so grant these policies only to trusted users in an account.

# Configure a test MySQL database

1. For this part of the tutorial, sign in to your account and configure a MySQL database in Amazon RDS.
2. Perform the following steps:

   a. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
   b. From the **Dashboard**, scroll down to the **Create database** section, and choose **Create database**.
   c. Refer to the Amazon RDS tutorial, Creating a MySQL DB instance, for the latest information on setting up an RDS database.

      Use the following information when creating your database:

      - DB instance identifier: **`MyTestDatabaseInstance`**.
      - Master username: **`adminuser`**.
      - Master password: Type a secure initial password, and retype the password in the **Confirm password**box. *Be sure to remember this password.* You need it when you create your secret in Step 2.

        > **Note**
        > Database creation may take up to 20 minutes before the DB instance becomes available.

   d. From the list of available databases under **RDS > Databases**, choose your database and then choose **Modify**.

e. In the **Network and Security** section, set the **Public accessibility** to **Yes**.

f. In the **Backup** section, set the **Backup retention period** to **0 days** to disable backups.

g. Leave the rest of the settings at the default values.

h. Choose **Continue**.

i. In the **Scheduling of modifications** section, choose **Apply immediately** and then **Modify DB Instance**.

j. When the **Summary** section displays **Available** under **Info**, refresh the page, and then scroll down to the **Connectivity and security** section.

k. In the **Security** section, choose the **default** under **VPC security groups**. The console for Amazon EC2 opens and displays the configured **Security Groups**.

l. Choose **Inbound rules** and then choose **Edit Inbound Rules**.

m. Under **Source type**, choose **Anywhere**, and choose **Save rules**.

**Note**
To configure the tutorial correctly, use these settings at a minimum. If you require a private VPC, then the Lambda function must be configured to run in that VPC. Next, you must either configure your VPC with a private Secrets Manager endpoint (p. 74) or configure the VPC with public internet access by using a NAT gateway. These configurations allow the Lambda rotation function to access the public Secrets Manager endpoint.

# Step 2: Create your secret

In this step, you create a secret in Secrets Manager, and populate the secret with your test details, which include database and the credentials of your master user.

### To create your secret

a. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.

b. Ensure you set your console to the same region as you created the Amazon RDS MySQL database in the previous step.

c. Choose **Store a new secret**.

d. On the **Store a new secret** page, in the **Select secret type** section, choose **Credentials for RDS database**.

e. For **User name**, type `adminuser` to match the name of the master user you previously provided in Step 1.3.

f. For **Password**, type the same password that you provided for **adminuser** when you created your database.

g. For **Select the encryption key**, leave it set to **DefaultEncryptionKey**. There is a charge for creating new KMS keys.

h. For **Select which RDS database this secret will access**, and choose the instance **MyTestDatabaseInstance** you created in Step 1. Choose **Next**.

i. In the **Secret name and description** section, for **Secret name**, type `MyTestDatabaseMasterSecret`. Choose **Next**.

j. In the **Configure automatic rotation** section, disable rotation for now. Choose **Next**.

k. In the **Review** section, verify your details, and then choose **Store**.

Secrets Manager returns to the list of secrets, which now includes your new secret.

# Step 3: Validate your initial secret

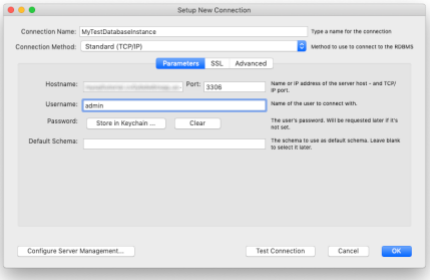
Before you configure your secret to rotate automatically, you should verify you have the correct information in your secret and can connect to the database. This tutorial describes how to install a GUI-based application, **MySQL Workbench** to test the connection. Download the client appropriate to your operating system.

At the very least, you can retrieve the secret by using either the AWS CLI or the Secrets Manager console. Then cut and paste the user name and password into the MySQL database client.

To test your database your database connection

a. After installing the MySQLWorkBench client software, open the MySQLWorkbench client to display the **Welcome to MySQLWorkbench** interface.


b. In the **MySQL Connections**, choose the **+** icon to display **Setup New Connection**.




c. For Connection Name, enter **`MyTestDatabaseInstance`**.

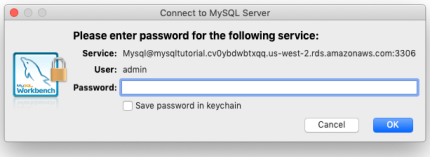
d. For the **Hostname**, enter the endpoint of the database, such as **`MyTestDatabase.`*`hostname`*`.region.rds.amazonaws.com`**.

   You can find the endpoint on the details page for your database. In the Amazon RDS console, choose **Databases** section of the **RDS > Databases > MyTestDatabaseInstance**.

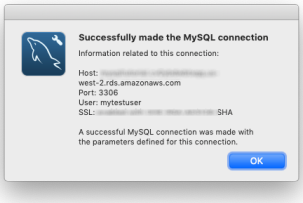
e. Leave the **Port** at the default value, 3306.

f. In the **Username** field, type the username you created for the database, **`adminuser`**.

g. Choose **Test Connection** and enter the database password in the **Password** field.




h. If configured correctly, MySQLWorkbench displays a successful connection message.




i. Choose **OK**.

**Troubleshooting tip**
If the MySQLWorkbench client fails to connect to the database, you should check the security group attached to the VPC with the database. The default rules in a security group enable all outbound traffic, but the rules block all inbound traffic except for the traffic you explicitly allow by defining a rule. If you run your computer on the public internet then your security group must enable inbound traffic from the Internet to the TCP port you configured your database communication, typically port 3306. If you configure MySQL to use a different TCP port, ensure you update the security rule to match.

# Step 4: Configure rotation for your secret

After you validate the initial credentials in your secret, you can configure and start your first rotation.

To configure secret rotation

a. In the Secrets Manager console, choose the secret **MyTestDatabaseMasterSecret**.
b. On the **Secret details** page, in the **Rotation configuration** section, choose **Edit rotation**.
c. On the **Edit rotation configuration** page, choose **Enable automatic rotation**.
d. For **Select rotation interval**, choose **30 days**.
e. Under **Select the secret will be used to perform the rotation**, choose **Use this secret**.
f. Choose **Save**. Secrets Manager begins to configure rotation for your secret, including creating the Lambda rotation function and attaching a role enabling Secrets Manager to invoke the function.
g. Stay on the console page with the **Rotation is being configured** message, until the message changes to **Your secret MyTestDatabaseMasterSecret has been successfully stored and secret rotation is enabled.**

# Step 5: Verify successful rotation

After you rotate the secret, you can confirm the new credentials in the secret work to connect with your database.

a. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
b. Choose your secret **MyTestDatabaseMasterSecret**.
c. Choose **Retrieve secret value**.
d. Locate the **password** field.

   If you successfully rotated the secret, the password should change to something similar to `E4%I)rj)vmpRg)U}++=}GHAnNDD1v0cJ` instead of the original secret.
e. To access your database, open MySQLWorkbench and choose the connection **MyTestDatabase**.
f. When prompted for the password, copy the password from Secrets Manager and paste into the **Password** field. Choose **OK**.

You can successfully access the database with the new password and validate that rotating secrets works.

## Step 6: Clean up

**Important**
If you intend to also perform the tutorial Tutorial: Rotate a user secret with a master secret (p. 19), don't perform these steps until you complete the tutorial.

Because databases and secrets can incur charges on your AWS bill, you should remove the database instance and the secret you created in this tutorial after you finish experimenting with the tutorial.

### Deleting the secret

a. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
b. In the list of secrets, choose the **MyTestDatabaseSecret** secret you created for this tutorial.
c. Choose **Actions**, then choose **Delete secret**.
d. In the **Schedule secret deletion** dialog box, for **Enter a waiting period**, type **7**, the minimum value allowed.
e. Choose **Schedule deletion**.

After the number of days in the recovery window elapses, Secrets Manager removes the secret permanently.

### To delete the database instance

a. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
b. In the navigation pane, choose **Instances**.
c. In the list of available instances, choose the **MyTestDatabaseInstance** instance you created for this tutorial.
d. Choose **Instance actions**, and then choose **Delete**.
e. On the **Delete DB Instance** page, in the **Options** section, for **Create final snapshot**, choose **No**.
f. Select the acknowledgement of losing all of your data, and then choose **Delete**.

# Tutorial: Rotate a user secret with a master secret

This tutorial builds on the tasks you completed in the first tutorial: Tutorial: Rotate a secret for an AWS database (p. 14). Complete the previous tutorials before beginning this one.

In this tutorial, you use the secret you already created as the master user for the database. You create a new limited user, and create a secret for the user. You then configure rotation for the user secret by using the credentials in the master secret. The Lambda rotation function for a master secret clones the first user, and then alternates between the users, changing the password for each in turn.

**Step 1: Create a new user for your database and a user secret (p. 20)**

First, create a new limited-permissions user on your Amazon RDS MySQL database and store those credentials in a new secret.

**Step 2: Validate your initial secret (p. 21)**

In step 2, confirm that you can access the database as your new user by using the credentials stored in the secret.

**Step 3: Configure rotation for your secret (p. 21)**

In step 3, configure rotation for your user secret. You specify the master secret to use for granting access to the rotation function.

**Step 4: Verify successful rotation (p. 21)**

In this step, rotate the secret twice to show the secret retrieves working credentials from two alternating users with access the database.

**Step 5: Clean up (p. 22)**

In the final step, remove the Amazon RDS database instance and the secrets you created to avoid incurring any unnecessary costs.

# Prerequisites

- This tutorial assumes you have access to an AWS account, and you can sign in to AWS as a user with full permissions to configure AWS Secrets Manager and Amazon RDS, in either the console or by using the equivalent commands in the AWS CLI.
- You must complete the steps in the tutorial Tutorial: Rotate a secret for an AWS database (p. 14), without deleting the database and user as described in the final section and provides you with the following items:
  - An Amazon RDS MySQL database called **MyTestDatabase** running in an instance called **MyTestDatabaseInstance**.
  - A master user named **adminuser** with administrative permissions.
  - A secret named **adminuser** with the credentials stored in Secrets Manager.

# Step 1: Create a new user for your database and a user secret

From the original tutorial, you have an Amazon RDS MySQL database with a single admin *master* user. You also have a secret you can use to retrieve the latest credentials for the master user. In this step, you create a new, limited user and store the credentials in a secret. This secret could be used by, for example, a mobile app querying for information from the database. The user doesn't require any other permissions.

a. Open your MySQLWorkbench and choose your **MyTestDatabaseInstance** connection and log into the database.

b. In the **Query 1** interface, enter `CREATE USER mytestuser IDENTIFIED BY 'userpassword';`.

c. Run the query using the  icon.

Review **Action Output** to see a successful completion of the query.

d. Delete the previous query from **Query 1**.

e. Enter the following text in **Query 1**: `GRANT SELECT on *.* TO mytestuser`, and run the query.

f. Review **Action Output** to see a successful completion of the query.

## Create a new secret using the console

a. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.

b. On the page with the list of secrets in your account, choose **Store a new secret**.
c. On the **Create Secret** page, in the **Select secret type** section, choose **Credentials for RDS database**.
d. For **User name**, type **`mytestuser`** to match the name of the user you created in Step 1b.
e. For **Password**, type the same password you provided for **mytestuser** in Step 1b.
f. For **Select the encryption key**, leave it set to **DefaultEncryptionKey**.
g. For **Select which RDS database this secret will access**, choose the instance **MyTestDatabaseInstance** you created in the previous tutorial.
h. Choose **Next**.
i. In the **Secret name and description** section, for **Secret name** type **`MyTestDatabaseUserSecret`**.
j. In the **Configure automatic rotation** section, leave rotation disabled for now. Choose **Next**.
k. In the **Review** section, verify your details and then choose **Store**.

You return to the list of secrets, which now includes your new secret.

# Step 2: Validate your initial secret

Before you configure your secret to rotate automatically, you should verify you have the correct information in your secret and can connect to the database. In the previous tutorial, you installed the MySQLWorkbench client component. Continue to use the MySQLWorkbench client in this tutorial.

You can retrieve the secret by using either the AWS CLI or the Secrets Manager console. Cut and paste the user name and password into your MySQL database client.

a. Open your MySQLWorkbench and choose the **+** to create a new connection.
b. For the **Connection Name**, enter **`MyTestUser`**.
c. Copy and paste your MySQL host name into **Hostname**. Leave the port at `3306`
d. For the **Username**, enter **`mytestuser`**, and choose **Test Connection**.
e. MySQLWorkbench returns a message indicating you successfully connected to the database.

# Step 3: Configure rotation for your secret

After you validate the initial credentials in your secret, you can configure and start your first rotation.

a. In the Secrets Manager console, choose the secret **MyTestDatabaseUserSecret**.
b. On the secret details page, in the **Rotation configuration** section, choose **Edit rotation**.
c. On the **Edit rotation configuration** page, choose **Enable automatic rotation**.
d. For **Select rotation interval**, choose **30 days**.
e. Choose a Lambda function from the list.
f. Under **Select which secret will be used to perform the rotation**, choose **Use a secret that I have previously stored in AWS Secrets Manager**.
g. In the list of secrets that appears, choose **MyTestDatabaseMasterSecret**.
h. Choose **Save**.

# Step 4: Verify successful rotation

Now you have rotated the secret, you can confirm that the new credentials in the secret work to connect with your database.

a. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.

b. On the **Secrets** list page, choose the name of your user secret.
c. Choose **Retrieve secret value** and view your current password. The secret value should be either the original password or a new one created by a successful rotation. If you still see the original password, close the **Secret value** section and reopen it until the secret value successfully changes. This step might take a few minutes.
d. When the new secret appears, copy and paste it into the MySQLWorkbench **MyTestUser**connection.
e. MySQL Workbench returns a message indicating you successfully logged into the database with the new secret.

# Step 5: Clean up

Because databases and secrets can incur charges on your AWS bill, you should remove the database instance and the secret you created in this tutorial.

### Deleting the secret

a. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
b. In the list of secrets, choose the **MyTestDatabaseSecret** secret you created for this tutorial.
c. Choose **Actions**, and then choose **Delete secret**.
d. In the **Schedule secret deletion** dialog box, for **Enter a waiting period**, type **7**, the minimum value allowed.
e. Choose **Schedule deletion**.

   After the number of days in the recovery window elapses, Secrets Manager removes the secret permanently.

### To delete the database instance

a. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
b. In the navigation pane, choose **Instances**.
c. In the list of available instances, choose the **MyTestDatabaseInstance** instance you created for this tutorial.
d. Choose **Instance actions**, and then choose **Delete**.
e. On the **Delete DB Instance** page, in the **Options** section, for **Create final snapshot**, choose **No**.
f. Select the acknowledgement that you lose all of your data, and then choose **Delete**.

# Secrets Manager best practices

The following recommendations help you to more securely use AWS Secrets Manager:

**Protect additional sensitive information**

A secret often includes more information than a user name and password, such as password hints. See the section called “Protect additional information” (p. 45).

**Improve performance by using client-side caching**

To use your secrets most efficiently, cache your secrets on the client and update the cache when the secret changes. See Cache secrets to improve performance (p. 66).

**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. 129).

**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. 109).

**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 the section called “Amazon VPC” (p. 107).

**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 your AWS Secrets Manager secrets (p. 68)

**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 the use of your AWS Secrets Manager secrets (p. 90).

**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.

# 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. 26)
- the section called “Attach a permissions policy to a secret” (p. 26)

## 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. 74).

## 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. 114).

# 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.

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.

See the section called “Permissions policy examples” (p. 30).

**To add or remove permissions on an identity**

- Do one of the following:
  - To use the console, see Adding IAM identity permissions (console).
  - To use the AWS CLI, see Adding IAM identity permissions (AWS CLI)
  - To use the AWS API, see Adding IAM identity permissions (AWS API)

# 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. 30).

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-a1b2c3",
  "Name": "MyAwesomeAppSecret",
  "ResourcePolicy": "{\"Version\":\"2012-10-17\",\"Statement\":{\"Effect\":\"Allow\",
\"Principal\":{\"AWS\":\"arn:aws:iam::111122223333:root\",\"arn:aws:iam::444455556666:root
\"},\"Action\":[\"secretsmanager:GetSecret\",\"secretsmanager:GetSecretValue\"],\"Resource
\":\"*\"}}"
}
```

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-a1b2c3",
  "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. 30) 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-a1b2c3",
  "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. 30) to get started writing your policy.

- C++
- Java

- PHP
- Python
- Ruby
- Node.js

# 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 |
|---|---|---|
| SecretsManagerReadWrite | 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. 74). | 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. 29).
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. 46).

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. 26).

```
{
  "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. 26). 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.

  You can grant permissions to an application that retrieves a secret from Secrets Manager. 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.

  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.
- `Action`: what they can do. See the section called "Secrets Manager actions" (p. 36).
- `Resource`: which secrets they can access. See the section called "Secrets Manager resources" (p. 39).

  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. 26).

To attach a policy to an identity, see the section called “Attach a permissions policy to an identity” (p. 26).

**Topics**



# 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. 26) and the section called “Attach a permissions policy to an identity” (p. 26).

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. 26).

```
{
  "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. 26). 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. 26).

```
{
  "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. 26).

```
{
  "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. 26).

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>a1b2c3"`.

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. 26).

```
{
  "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. 42) and *VPC endpoints* (p. 84).

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. 26).

```
{
  "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. 26).

```
{
  "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. 26).

```
{
  "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. 26).

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

```
    }
}
```

# 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. 30).

## Secrets Manager actions

The following table shows the actions and related permissions for Secrets Manager. For a description of each condition key, see the section called "Condition keys" (p. 40).

| Action and permission | Description | Access level | Condition keys |
|---|---|---|---|
| CancelRotateSecret<br><br>`secretsmanager:CancelRotateSecret` | Enables the user to cancel an in-progress secret rotation. | Write | secretsmanager:SecretId<br><br>secretsmanager:resource/AllowRotationLambdaArn<br><br>secretsmanager:ResourceTag/tag-key |
| CreateSecret<br><br>`secretsmanager:CreateSecret` | Enables the user to create a secret that stores encrypted data that can be queried and rotated. | Write | secretsmanager:Name<br><br>secretsmanager:Description<br><br>secretsmanager:KmsKeyId<br><br>aws:RequestTag/tag-key<br><br>aws:TagKeys<br><br>secretsmanager:ResourceTag/tag-key |
| DeleteResourcePolicy<br><br>`secretsmanager:DeleteResourcePolicy` | Enables the user to delete the resource policy attached to a secret. | Permissions management | secretsmanager:SecretId<br><br>secretsmanager:resource/AllowRotationLambdaArn<br><br>secretsmanager:ResourceTag/tag-key |
| DeleteSecret<br><br>`secretsmanager:DeleteSecret` | Enables the user to delete a secret. | Write | secretsmanager:SecretId<br><br>secretsmanager:resource/AllowRotationLambdaArn<br><br>secretsmanager:RecoveryWindowInDa<br><br>secretsmanager:ForceDeleteWithoutRe |

| Action and permission | Description | Access level | Condition keys |
|---|---|---|---|
| | | | secretsmanager:ResourceTag/tag-key |
| DescribeSecret<br>`secretsmanager:DescribeSecret` | Enables the user to retrieve the metadata about a secret, but not the encrypted data. | Read | secretsmanager:SecretId<br>secretsmanager:resource/AllowRotationLambdaArn<br>secretsmanager:ResourceTag/tag-key |
| GetRandomPassword<br>`secretsmanager:GetRandomPassword` | Enables the user to generate a random string for use in password creation. | Read | |
| GetResourcePolicy<br>`secretsmanager:GetResourcePolicy` | Enables the user to get the resource policy attached to a secret. | Read | secretsmanager:SecretId<br>secretsmanager:resource/AllowRotationLambdaArn<br>secretsmanager:ResourceTag/tag-key |
| GetSecretValue<br>`secretsmanager:GetSecretValue` | Enables the user to retrieve and decrypt the encrypted data. | Read | secretsmanager:SecretId<br>secretsmanager:VersionId<br>secretsmanager:VersionStage<br>secretsmanager:resource/AllowRotationLambdaArn<br>secretsmanager:ResourceTag/tag-key |
| ListSecretVersionIds<br>`secretsmanager:ListSecretVersionIds` | Enables the user to list the available versions of a secret. | Read | secretsmanager:SecretId<br>secretsmanager:resource/AllowRotationLambdaArn<br>secretsmanager:ResourceTag/tag-key |
| ListSecrets<br>`secretsmanager:ListSecrets` | Enables the user to list the available secrets. | List | |
| PutResourcePolicy<br>`secretsmanager:PutResourcePolicy` | Enables the user to attach a resource policy to a secret. | Permissions management | secretsmanager:SecretId<br>secretsmanager:resource/AllowRotationLambdaArn<br>secretsmanager:ResourceTag/tag-key<br>secretsmanager:BlockPublicPolicy |

| Action and permission | Description | Access level | Condition keys |
|---|---|---|---|
| PutSecretValue<br><br>`secretsmanager:PutSecretValue` | Enables the user to create a new version of the secret with new encrypted data. | Write | secretsmanager:SecretId<br><br>secretsmanager:resource/AllowRotationLambdaArn<br><br>secretsmanager:ResourceTag/tag-key |
| RemoveRegionsFromReplication<br><br>`secretsmanager:RemoveRegionsFromReplication` | Remove regions from replication. | Write | secretsmanager:SecretId<br><br>secretsmanager:resource/AllowRotationLambdaArn<br><br>secretsmanager:ResourceTag/tag-key |
| ReplicateSecretToRegions<br><br>`secretsmanager:ReplicateSecretToRegions` | Converts an existing secret to a multi-Region secret and begins replicating the secret to a list of new regions. | Write | secretsmanager:SecretId<br><br>secretsmanager:resource/AllowRotationLambdaArn<br><br>secretsmanager:ResourceTag/tag-key |
| RestoreSecret<br><br>`secretsmanager:RestoreSecret` | Enables the user to cancel deletion of a secret. | Write | secretsmanager:SecretId<br><br>secretsmanager:resource/AllowRotationLambdaArn<br><br>secretsmanager:ResourceTag/tag-key |
| RotateSecret<br><br>`secretsmanager:RotateSecret` | Enables the user to start rotation of a secret. | Write | secretsmanager:SecretId<br><br>secretsmanager:RotationLambdaARN<br><br>secretsmanager:resource/AllowRotationLambdaArn<br><br>secretsmanager:ResourceTag/tag-key |
| StopReplicationToReplica<br><br>`secretsmanager:StopReplicationToReplica` | Removes the secret from replication and promotes the secret to a regional secret in the replica Region. | Write | secretsmanager:SecretId<br><br>secretsmanager:resource/AllowRotationLambdaArn<br><br>secretsmanager:ResourceTag/tag-key |

| Action and permission | Description | Access level | Condition keys |
|---|---|---|---|
| TagResource<br>`secretsmanager:TagResource` | Enables the user to add tags to a secret. | Tagging | secretsmanager:SecretId<br>aws:RequestTag/tag-key<br>aws:TagKeys<br>secretsmanager:resource/AllowRotationLambdaArn<br>secretsmanager:ResourceTag/tag-key |
| UntagResource<br>`secretsmanager:UntagResource` | Enables the user to remove tags from a secret. | Tagging | secretsmanager:SecretId<br>aws:TagKeys<br>secretsmanager:resource/AllowRotationLambdaArn<br>secretsmanager:ResourceTag/tag-key |
| UpdateSecret<br>`secretsmanager:UpdateSecret` | Enables the user to update a secret with new metadata or with a new version of the encrypted data. | Write | secretsmanager:SecretId<br>secretsmanager:Description<br>secretsmanager:KmsKeyId<br>secretsmanager:resource/AllowRotationLambdaArn<br>secretsmanager:ResourceTag/tag-key |
| UpdateSecretVersionStage<br>`secretsmanager:UpdateSecretVersionStage` | Enables the user to move a staging label from one secret to another. | Write | secretsmanager:SecretId<br>secretsmanager:VersionStage<br>secretsmanager:resource/AllowRotationLambdaArn<br>secretsmanager:ResourceTag/tag-key |
| ValidateResourcePolicy<br>`secretsmanager:ValidateResourcePolicy` | Enables the user to validate a resource policy before attaching policy. | Permissions management | secretsmanager:SecretId<br>secretsmanager:resource/AllowRotationLambdaArn<br>secretsmanager:ResourceTag/tag-key |

# Secrets Manager resources

| Resource type | ARN format |
| --- | --- |
| Secret | arn:aws:secretsmanager:*<Region>*:*<AccountId>*:secret:*SecretName*-*6RandomCharacters* |

Secrets Manager constructs the last part of the 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

Condition keys in Secrets Manager correspond to the request parameters of an API call. You can use them to allow or block requests based on the parameter value. For more information, see IAM JSON policy elements: Condition.

You can also use AWS global condition context keys.

The following table shows the condition keys for Secrets Manager.

| Condition key | Description | Type |
| --- | --- | --- |
| aws:RequestTag/tag-key | Filters access by a key present in the request that the user makes to the Secrets Manager. | String |
| aws:TagKeys | Filters access by the list of all the tag key namespresent in the request the user makes to the Secrets Manager service. | String |
| secretsmanager:resource/AllowRotationLambdaArn | Filters the request based on the ARN of the Lambda rotation function attached to the target resource of the request. This enables you to restrict access to only those secrets with a rotation Lambda ARN matching this value. Secrets without rotation enabled or with a different rotation Lambda ARN do not match. | ARN |
| secretsmanager:BlockPublicPolicy | Evaluates your resource policy assigned to a secret and blocks any policies that allow public access. | Boolean |
| secretsmanager:Description | Filters the request based on the Description parameter in the request. | String |
| secretsmanager:ForceDeleteWithoutRecovery | Filters the request based if the delete specifies no recovery window. This enables you to effectively disable this feature. | Boolean |
| secretsmanager:KmsKeyId | Filters the request based on the KmsKeyId parameter of the request. This enables you to limit which keys can be used in a request. | String |

| Condition key | Description | Type |
|---|---|---|
| secretsmanager:Name | Filters the request based on Name parameter value of the request. This enables you to restrict a secret name to only those matching this value. | String |
| secretsmanager:RecoveryWindowInDays | Filters the request based on the recovery window specified. Enables you to enforce that recovery windows occur an approved number of days. | Long |
| secretsmanager:ResourceTag/<tag-key> | Filters the request based on a tag attached to the secret. Replace <tag-key> with the actual tag name. You can then use condition operators to ensure the presence of the tag, and contains the requested value. | String |
| secretsmanager:RotationLambdaArn | Filters the request based on the `RotationLambdaARN` parameter. This enables you to restrict which Lambda rotation functions can be used with a secret. The key can be used with both CreateSecret and the operations modifying existing secrets. | ARN |
| secretsmanager:SecretId | Filters the request based on the ARN for the secret provided in the SecretId parameter. This enables you to limit which secrets can be accessed by a request. | ARN |
| secretsmanager:VersionId | Filters the request based on the VersionId parameter of the request. This enables you to restrict which versions of a secret can be accessed. | String |
| secretsmanager:VersionStage | Filters the request based on the staging labels identified in the `VersionStage` parameter of a request. We recommend you do not use this key, because if you use this key, requests must pass in a staging label to compare to this policy. | String |

# 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. 42).

# 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. 33).

- .
- `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 endpoints* (p. 84).

# Create and manage secrets with AWS Secrets Manager

This section describes how to create, update, retrieve, search, and delete secrets by using AWS Secrets Manager.

**Topics**



## Create a secret

A *secret* is a set of credentials, such as a user name and password, that you store in an encrypted form in Secrets Manager. The secret also includes the connection information to access a database or other service, which Secrets Manager doesn't encrypt.

You control access to the secret with IAM permission policies, which means that only authorized users can access or modify the secret. Applications which access the database or other service use an IAM user or role, so you grant permission to that user or role to access the secret. You can do this by resource or by identity:

- You can attach a resource-based policy to the secret and then in the policy, list the users or roles that have access. For more information, see the section called “Attach a permissions policy to a secret” (p. 26).
- You can attach an identity-based policy to a user or role, and then in the policy, list the secrets that the identity can access. For more information, see the section called “Attach a permissions policy to an identity” (p. 26).

To create a secret, you need the permissions granted by the **SecretsManagerReadWrite** AWS managed policy. For more information, see AWS managed policy (p. 28).

**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 **Select secret type** page, do the following:

    a. For **Select secret type**, do one of the following:

- To store database credentials, choose **Amazon RDS**, **Amazon DocumentDB**, **Amazon Redshift**, or **Other database**, and then enter the credentials you want to store.
- To store non-database credentials or other information, choose **Other type of secrets**, and then in **Specify the key/value pairs to be stored in this secret**, do one of the following:
  - In **Key/value pairs**, enter the secret you want to store in **Key** and **Value** pairs. You can add as many pairs as you need. For example, you can specify **Key** **`Username`**, and then for **Value** enter the user name. Add a second **Key** **`Password`**, and then for **Value** enter the password. You can add pairs for **`Database name`**, **`Server address`**, **`TCP port`**, and so on.
  - On the **Plaintext** tab, enter your secret in any format.

b. For **Encryption key**, choose the AWS KMS key that Secrets Manager uses to encrypt the protected text in the secret:
   - Choose **DefaultEncryptionKey** to use the AWS managed key for Secrets Manager. There is no cost for using this key.
   - Choose another KMS key from the list. You must have the following permissions: `kms:Encrypt`, `kms:Decrypt`, and `kms:GenerateDataKey`.
   - Choose **Add new key** to go to the AWS KMS console to create a customer managed key. You must have `kms:CreateKey` permission. You will be charged for KMS keys that you create.

c. If you chose database credentials in step 3a, for **Database**, enter your database connection information.

d. Choose **Next**.

4. On the **Secret name and description** page, do the following:

   a. For **Secret name**, enter a name for your secret, for example **`MyAppSecret`** or **`development/TestSecret`**. Use slashes to create a hierarchy for your secrets.

   b. (Optional) For **Description**, enter information to help you remember the purpose of this secret.

   c. (Optional) In the **Tags** section, add tags to your secret. For tagging strategies, see the section called “Tag your secrets” (p. 63). Don't store sensitive information in tags because they aren't encrypted.

   d. (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. 26).

   e. (Optional) In **Replicate secret**, to replicate your secret to another AWS Region, choose **Replicate secret to other Regions**. You can replicate your secret now or come back and replicate it later. For more information, see Multi-Region secrets (p. 52).

   f. Choose **Next**.

5. (Optional) On the **Configure automatic 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. 68). 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.

If you want Secrets Manager to rotate the secret, your secret must be in the format described in Rotation function templates (p. 77). Otherwise, you can store your secret in any format.

### To create a secret that uses the AWS managed key for Secrets Manager

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

```
{
      "username": "saanvi",
      "password": "aDM4N3*!8TT"
}
```

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

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

The following shows the output.

```
{
    "SecretARN": "arn:aws:secretsmanager:Region:AccountId:secret:production/
MyAwesomeAppSecret-AbCdEf",
    "SecretName": "production/MyAwesomeAppSecret",
    "SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE"
}
```

**To create a secret that uses a customer managed key**

- In your AWS CLI command, include the `KmsKeyId` parameter, as shown in the following example.

```
aws secretsmanager create-secret --name production/MyAwesomeAppSecret --secret-string
 file://mycreds.json --kms-key-id MyKMSKey
```

## AWS SDK

To create a secret by using one of the AWS SDKs, use the `CreateSecret` action. For more information, see:

- C++
- Java
- PHP
- Python
- Ruby
- Node.js

# Protect additional sensitive information

A secret often includes several pieces of information besides the user name and password. Depending on the database, service, or website, you can choose to include additional sensitive data. This data can include password hints, or question-and-answer pairs you can use to recover your password.

Ensure you protect any information that might be used to gain access to the credentials in the secret as securely as the credentials themselves. Don't store this type of information in the `Description` or any other non-encrypted part of the secret.

Instead, store all such sensitive information as part of the encrypted secret value, either in the `SecretString` or `SecretBinary` field. You can store up to 65536 bytes in the secret. In the

`SecretString` field, the text usually takes the form of JSON key-value string pairs, as shown in the following example:

```
{
  "engine": "mysql",
  "username": "user1",
  "password": "i29wwX!%9wFV",
  "host": "my-database-endpoint.us-east-1.rds.amazonaws.com",
  "dbname": "myDatabase",
  "port": "3306"
}
```

# 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. 68).

### 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. In 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. 112).
   - To update tags, in the **Tags** section, choose **Edit**. See the section called “Tag your 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. 68).
   - To update permissions for your secret, choose Edit permissions. See the section called “Attach a permissions policy to a secret” (p. 26).

## 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 your 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":"a different 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.

**API/SDK:** `UpdateSecret`, `PutSecretValue`

- C++
- Java

- PHP
- Python
- Ruby
- Node.js

# Enhanced search capabilities for secrets in Secrets Manager

Secrets Manager provides an easier method to manage and search secrets based on attributes such as secret names, description, tag keys, and tag values by introducing an enhanced search functionality supported through the AWS Management Console, AWS API, and AWS CLI.

If you store thousands of secrets in Secrets Manager, searching secrets by name may not return enough information about the secrets. However, using additional search attributes when you search provides in depth information about the secrets. You can apply multiple filters to the search criteria.

You may also want to read the AWS Security BlogIdentify, arrange, and manage secrets easily using enhanced search in AWS Secrets Manager.

You can search for a secret using these attributes:

- **Name** - a matching prefix case-sensitive phrase query using part or all of the secret name to return relevant results.

  For example, searching for **`MySecret`** returns results if you named the secret, MySecret. The search does not find secrets named **`mysecret`** or other variations.
- **Description** - a matching phrase case-insensitive query to specify a phrase located in the description to filter matching descriptions.

  For example, searching for **`MyDescription`** returns results with MyDescription, mydescription, Mydescription, or myDescription.
- **Tag keys** - a matching prefix case-sensitive query that searches for the existence of tag key. Secrets Manager returns only tags matching the specified prefix.

  For instance, searching for **`MyKey`** returns only secrets with a Tag Key value of MyKey, not mykey or other variations.
- **Tag values** a matching prefix case-sensitive query that searches for the existence of a tag value. Secrets Manager returns only tags matching the specified prefix.

  For instance, searching for **`MyValue`** returns only secrets with a Tag Value of MyValue, not myvalue or other variations.

## Search without filters

Using the Search feature without specifying filters returns different search results for secrets. When you specify a search keyword, the search returns all secrets with that keyword in the name, tag key, tag value, or description and ignores case-sensitivity in the results.

If you use a full text search without any filters, the search feature ignores special characters, such as space, /, _, =, #, and only searches for matching numbers and alphabet.

Secrets Manager provides this type of search behavior by default.

**Note**
Secrets Manager is a regional service and the search returns matches stored in the selected region.

### To search for a secret (console)

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Click in the **Search** field and choose the filter(s) to use in your search:
   - **Name**
   - **Description**
   - **Tag Keys**
   - **Tag Value**
3. Type your keyword in the **Search** field and press **Enter**. You may use multiple filters for your search criteria.

### To search for secrets (AWS CLI or SDK)

- Use the following commands to search for a secret stored in AWS Secrets Manager:
  - **API/SDK:** `ListSecrets`
    - C++
    - Java
    - PHP
    - Python
    - Ruby
    - Node.js
  - **AWS CLI:** `list-secrets`

    **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"
    }
]
}
```

# 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 also can't directly delete a version of a secret. Instead, you remove all staging labels from the secret using the AWS CLI or AWS SDK. This marks the secret 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 secret versions scheduled for deletion (p. 94).

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. On the **Secrets** page, choose **Preferences** (  ).
2. In the Preferences dialog box, select **Show disabled secrets**, and then choose **Save**

## AWS CLI

To delete a secret by using the AWS CLI, 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.

### 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 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. For more information, see:

- AC++
- Java
- PHP
- Python
- Ruby
- Node.js

# 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:

- C++
- Java
- PHP
- Python
- Ruby
- Node.js

# Create and manage multi-Region Secrets Manager secrets

Secrets Manager lets you easily replicate your secrets in multiple AWS Regions to support applications spread across those Regions as well as disaster recovery scenarios. You can create a primary secret in one AWS Region, and then replicate the secret to all AWS Regions where the application uses the secret. Secrets Manager securely replicates the primary secret for use in specified AWS Regions without the overhead of managing a complex custom solution for this functionality.

Secrets Manager enables the easy lifecycle management of multi-Region secrets by replicating the primary secret and the associated metadata to the replica secrets. Replicated secrets have a common name across all Regions to enable you to easily find multi-Region secrets and begin using them with minimal changes to your application. Secrets Manager integrates with AWS Key Management Service (AWS KMS) to encrypt every version of every secret with a unique data key protected by a KMS key. This integration protects your secrets under encryption keys that never leave AWS KMS unencrypted.

Secrets Manager replicates all encrypted secret data and metadata such as tags, resource policies and secret updates such as rotation across the specified Regions.

You can set up a single, tag-based IAM policy to provision access to the secret in all Regions.

You can also use this feature to replicate secrets and read them in required Regions to meet Regional access and low latency requirements of your multi-Region applications. Use multi-Region secrets to support your disaster recovery strategies by converting any secret replica to a stand-alone secret and set it up for replication independently.

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 cannot replicate a secret in your enabled AWS Regions to a specialized Region or replicate secrets from a specialized region to a commercial region.

**Enable Regions in AWS**

An AWS Region is a collection of AWS resources in a geographic area. Each AWS Region is isolated and independent of the other Regions. Regions provide fault tolerance, stability, and resilience, and can also reduce latency. They enable you to create redundant resources that remain available and unaffected by a Regional outage.

If a Region is disabled by default, you must enable it before you can create and manage resources. The following Regions are disabled by default:

- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Europe (Milan)
- Middle East (Bahrain)

When you enable a Region, AWS performs actions to prepare your account in that Region, such as distributing your IAM resources to the Region. This process takes a few minutes for most accounts, but this can take several hours. You cannot use the Region until this process is complete.

For more information on enabling Regions, see Managing AWS Regions.

You can manage multi-Region secrets using the Secrets Manager console, API, or CLI or provision them using AWS CloudFormation templates.

**Topics**



# Configure primary and replica secrets

Before you can configure multi-Region secrets, you must enable the regions where you want to set up the replica secrets. For more information, see Managing AWS Regions.

### Minimum permissions

To create a secret in the console, you must have these permissions:

- The permissions granted by the **SecretsManagerReadWrite** AWS managed policy.
- The permissions granted by the **IAMFullAccess** AWS managed policy – required only if you enable rotation for the secret.
- `kms:CreateKey` – required only if you want Secrets Manager to create a customer managed key.

- `kms:Encrypt` – required only if you use a customer managed key to encrypt your secret instead of the AWS managed key (`aws/secretsmanager`) for your account. You don't need this permission to use `aws/secretsmanager`.
- `kms:Decrypt` – required only if you use a customer managed key to encrypt your secret instead of the AWS managed key (`aws/secretsmanager`) for your account. You don't need this permission to use `aws/secretsmanager`.
- `kms:GenerateDataKey` – required only if you use a custom AWS KMS key to encrypt your secret instead of the AWS managed key (`aws/secretsmanager`) for your account. You don't need this permission to use `aws/secretsmanager`.

### 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 to other Regions**.
   - If your secret is replicated, in the **Replicate secret** section, choose **Add more regions**.
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. You can choose the same key as the primary secret or a different one.
   c. (Optional)To add another Region, choose **Add more regions**.
   d. Choose Complete adding regions.

If replication fails, see the section called “Retry secret replication” (p. 54).

### To replicate a secret (AWS CLI or SDK)

- Use the following commands to create a secret and configure a replica secret:
  - **API/SDK:** `ReplicateSecretToRegions`
  - • C++
    - Java
    - PHP
    - Python
    - Ruby
    - Node.js
  - **AWS CLI:** `replicate-secret-to-regions`

    The following example replicates a secret in US West (Oregon) to US East (N. Virginia).

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

    If replication fails, see the section called “Retry secret replication” (p. 54).

## Retry secret replication

Secret replication can fail for the following reasons:

- A secret with the same name already exists in a Region.
- You do not have sufficient permissions on the encryption key.
- You have not enabled the Region where you want a replicated secret.
- Other failures as described in the displayed banner.

Before retrying the failed replication, resolve the failure reason.

To retry replication, use the following steps:

1. Choose the secret you want to retry replication.
2. In the **Replicate Secret** section, choose the Region with the **Replication Status** Failed.
3. From the **Actions** menu, choose **Retry Replication**.

If the replication fails due to a name conflict in the replica secret Region, you can choose to overwrite the existing secret in the replica Region.

Choose the checkbox to overwrite the secret, and enter the AWS Region.

**Replica secret information**

After creating a primary secret with a replica secret, you can view the following replica secret information in the Secrets Manager console:

- **Region** - Displays the Region of the replica secret.
- **Region Replication Status** - Displays the replication status.
  - **In Sync** - Secret replication successful in the target Region.
  - **In Progress** - Secret replication in progress.
  - **Failed** - Secret with the same name exists in the selected Region.
  - **Failed** - No permissions available on the KMS key to complete the replication.
  - **Failed** - Secret replication failed due to a network error.
  - **Failed** - You have not enabled the region where the replication occurs.
- **Secret ARN** - Displays the replica Amazon Resource Number (ARN).
- **Encryption Key** - Displays the type of encryption key used to encrypt the replica secret.
- **Last Accessed Date** - Displays the date you last accessed the replica secret.

## Create a replica secret from an existing secret

You can set up an existing secret for replication across multiple Regions. You can keep specific secrets regional by setting up an IAM policy that restricts permissions to the replication APIs.

Before you can add replicas in Regions to an existing secret, you must enable the Regions for the replica secrets. For more information, see Managing AWS Regions.

**Using the Secrets Manager console**

1. Log in to the Secrets Manager at https://console.aws.amazon.com/secretsmanager/.
2. Choose an existing secret from the list of available secrets and display **Secret details**
3. Choose **Replicate secret to other regions**.
4. Select the AWS Region from the list.
5. Select the encryption key used to encrypt the secret. You can use the same encryption key or a different encryption key.
6. Choose **Complete adding regions**.

# Manage multi-Region secrets in Secrets Manager

**Topics**



## Delete a replica secret

You can delete each replica secret from a Region by removing the replica secret from the primary secret, and then deleting it.

> **Note**
> You cannot delete a primary secret if a replica secret exists for it.

To delete a replica secret, use the following steps:

1. Log into the Secrets Manager at https://console.aws.amazon.com/secretsmanager/.
2. Choose the primary secret of the replica secret to display the primary secret details.
3. In the **Replicate Secret** section, choose the replica secret.
4. From the **Actions** menu, choose **Delete Replica**.

## Edit an encryption key

When you change the encryption key associated with your replica secret, all future secret versions use the new key. Be sure your applications have `kms:Decrypt` permissions on the new key.

To edit the replica secret encryption key, use the following steps:

1. Log in to the Secrets Manager at https://console.aws.amazon.com/secretsmanager/.
2. Choose the primary secret that you want to edit the replica secret encryption key.
3. In the **Replication Configuration** section, choose the replica secret.
4. From the **Actions** menu, choose **Edit encryption key**.
5. Choose to edit the existing encryption key or add a new key.
6. Type the name of the AWS Region into the **AWS Region** field.

   By typing in the name of the AWS Region, you confirm changing the encryption key.
7. Choose **Re-encrypt Secret**.

## Add additional Regions to a replica secret

To add more AWS Regions to a replica secret, use the following steps:

1. Log in to the Secrets Manager at https://console.aws.amazon.com/secretsmanager/.
2. Choose the primary secret of the replica secret to display the primary secret details.
3. In the **Replicate Secret** section, choose the replica secret.

4. Choose **Add more Regions**.
5. Enter an additional Region in the **AWS Region** field.
6. After you enter a Region, you can change the encryption key or leave as default.
7. You can choose **Add more Regions**.

## Rotate multi-Region secrets

If you configure your primary secret for rotation, Secrets Manager executes the secret rotation in the primary Region and the new secret value propagates to all of the associated replica secrets. You do not have manage rotation individually for all of the replica secrets.

## Promote a replica secret to a standalone secret

You may want to promote a replica secret to a standalone secret in the following scenarios:

- Implementing disaster recovery - You can promote a replica secret to a standalone instance as a disaster recovery solution if the primary secret becomes unavailable.
- Enabling secret rotation - If you decide that you want to enable rotation, promote the secret to a standalone secret and configure rotation.

For any other changes, to a replica secret, promote it to a standalone secret and add the changes.

Attempting to change any of the replica secret parameters generates an error message indicating failure to update the secret. You must promote the replica secret before changing any parameters.

To promote a replica secret, choose **Promote to standalone secret**.

Promoting a replica secret severs the relationship of the replica secret to the primary secret. Any changes to the primary secret no longer replicate to the standalone secret.

> **Note**
> Be sure to update the corresponding applications to use the standalone secret.

## Find multi-Region secrets

You can find multi-Region secrets using the following search criteria:

- **Replicated Secrets: Primary Secrets** - search for all secrets with the parameter **primary** in your Region.
- **Replicated Secrets: Replica Secrets** - search for all replica secrets in a Region.

  You cannot search across Regions for replica secrets. For instance, if you log into the console in the **us-east-2** region and search for replica secrets, the search only returns replica secrets in **us-east-2**.
- **Replicated Secrets: Not Replicated Secrets** - search for secrets without primary or replica labels in a Region.

# Automate secret creation in AWS CloudFormation

You can use AWS CloudFormation to create and reference secrets from within your AWS CloudFormation stack template. You can create a secret and then reference it from another part of the template. For example, you can retrieve the user name and password from the new secret and then use that to define the user name and password for a new database. You can create and attach resource-based policies to a secret. You can also configure rotation by defining a Lambda function in your template and associating the function with your new secret as its rotation Lambda function.

You create an AWS CloudFormation template in either JSON or YAML. AWS CloudFormation processes the template and builds the resources that are defined in the template. You can use templates to create a new copy of your infrastructure whenever you need it. For example, you can duplicate your test infrastructure to create the public version. You can also share the infrastructure as a simple text file, so other people can replicate the resources.

Secrets Manager provides the following resource types that you can use to create secrets in an AWS CloudFormation template:

- **AWS::SecretsManager::Secret** – Creates a secret and stores it in Secrets Manager. You can specify a password or Secrets Manager can generate one for you. You can also create an empty secret and then update it later using the parameter `SecretString`.
- **AWS::SecretsManager::ResourcePolicy** – Creates a resource-based policy and attaches it to the secret. A resource-based policy controls who can perform actions on the secret.
- **AWS::SecretsManager::RotationSchedule** – Configures a secret to perform automatic periodic rotation using the specified Lambda rotation function.
- **AWS::SecretsManager::SecretTargetAttachment** – Configures the secret with the details about the service or database that Secrets Manager needs to rotate the secret. For example, for an Amazon RDS DB instance, Secrets Manager adds the connection details and database engine type as entries in the `SecureString` property of the secret.

You can also use the AWS Cloud Development Kit (CDK). For more information, see AWS Secrets Manager Construct Library.

# Examples

The following example templates create a secret and an Amazon RDS MySQL DB 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 and configures the secret to automatically rotate every 30 days.

**Note**
The JSON specification doesn't support comments. See the the section called “YAML” (p. 61) version later on this page for comments.

## 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"
            }
        }
    },
    "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"
            }
        }
    },
    "MyRDSInstanceRotationSecret": {
        "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"
                        }
                    ]
                }
            },
            "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"
                            ]
                        }
                    ]
                }
            },
            "MyDBSubnetGroup": {
                "Type": "AWS::RDS::DBSubnetGroup",
                "Properties": {
                    "DBSubnetGroupDescription": "Test Group",
                    "SubnetIds": [
                        {
                            "Ref": "TestSubnet01"
                        },
                        {
                            "Ref": "TestSubnet02"
                        }
                    ]
                }
            },
            "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": {
                        "AutomaticallyAfterDays": 30
                    }
                }
            }
        }
    }
```

## 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: 16
       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:
       AutomaticallyAfterDays: 30
```

# Tag your secrets

Several AWS services enable you to add *tags* to your resources, and Secrets Manager allows you tag your secrets. Secrets Manager defines a tag as a simple label consisting of a customer-defined key and an optional value. You can use the tags to make it easy to manage, search, and filter the resources in your AWS account. When you tag your secrets, be sure to follow these guidelines:

- Use a standardized naming scheme across all of your resources. Remember tags are case sensitive.
- Create tag sets enabling you to perform the following:

- **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. 35).
- **Cost allocation and tracking** – You can group and categorize your AWS bills by tags. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.
- **Automation** – You can use tags to filter resources for automation activities. 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.
- **AWS Management Console** – Some AWS service consoles enable you to organize the displayed resources according to the tags, and to sort and filter by tags. 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*.

Use tags creatively to manage your secrets. Remember you must never store sensitive information for a secret in a tag.

You can tag your secrets when you create them (p. 43) or when you edit them (p. 46).

For more information, see AWS Tagging Strategies on the *AWS Answers* website.

### 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.

### To change tags for your secret (AWS CLI)

- 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.

### To change tags for your secret (AWS SDK)

- Use `TagResource` or `UntagResource`.

For more information, see:

- C++
- Java
- PHP
- Python
- Ruby
- Node.js

### To find secrets with a specific tag (AWS CLI)

- Use list-secrets. The following example finds secrets with the `tag` **`costcenter`** and the `value` **`12345`**:

```
$ aws secretsmanager list-secrets --filters Key=tag-key,Values=costcenter Key=tag-
value,Values=12345
```

# Retrieve secrets

With Secrets Manager, you can programmatically and securely retrieve your secrets in your applications. You can also retrieve your secrets by using the console or the AWS CLI.

To retrieve a secret in the console, you must have these permissions:

- `secretsmanager:ListSecrets` – Use to navigate to the secret to retrieve.
- `secretsmanager:DescribeSecret` — Use to retrieve the non-encrypted parts of the secret.
- `secretsmanager:GetSecretValue` – Use to retrieve the encrypted part of the secret.
- `kms:Decrypt` – Required only if you used a customer managed key instead of the AWS managed key (`aws/secretsmanager`) to encrypt your secret.

**To retrieve a 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 **Secret value** section, choose **Retrieve secret value**.
4. Do one of the following:
   - Choose **Secret key/value** to see the credentials as individual keys and values.
   - Choose **Plaintext** to see the JSON text string that Secrets Manager encrypts and stores.

# Retrieve secrets programmatically

You can use the following commands to retrieve a secret stored in AWS Secrets Manager:

- **API/SDK:** `GetSecretValue`
  - C++
  - Java
  - PHP
  - Python
  - Ruby
  - Node.js
- **AWS CLI:** `get-secret-value`

You identify the secret by the name or ARN. You can include the version, but if you don't specify a version, Secrets Manager defaults to the version with the staging label `AWSCURRENT`. Secrets Manager returns the contents of the secret text in the response parameters `PlaintextString`. If you stored binary data in the secret, Secrets Manager also returns `Plaintext`, a byte array. Secrets Manager uses the last modified date for the `CreatedDate` output.

# Cache secrets to improve performance

To efficiently use your secret, you must not simply retrieve the secret value every time you want to use it. You should also include code to perform the following operations:

- Your application should cache a secret after retrieving it. Then, instead of retrieving the secret across the network from Secrets Manager every time you need it, use the cached value.
- Whenever your code receives a network or AWS service error, retry your requests using an algorithm that implements an exponential backoff and retry with jitter, as described at Exponential Backoff And Jitter in the *AWS Architecture Blog*.

By writing your code to perform those tasks, you get the following benefits:

- **Reduced costs** – For secrets you use often, retrieving them from the cache reduces the number of API calls your applications make to Secrets Manager. Since Secrets Manager charges a fee per API call, this can significantly reduce your costs.
- **Improved performance** – Retrieving the secret from memory instead of sending a request over the Internet to Secrets Manager can dramatically improve the performance of your applications.
- **Improved availability** – Transient network errors can be much less of a problem when you retrieve your secret information from the cache.

Secrets Manager has created a client-side component to implement these best practices and simplify your creation of code to access secrets stored in AWS Secrets Manager. You install the client and then call the secret, providing the identifier of the secret you want the code to use. Secrets Manager also requires the AWS credentials you use to call Secrets Manager contain the `secretsmanager:DescribeSecret` and `secretsmanager:GetSecretValue` permissions on the secret. Those credentials would typically be associated with an IAM role. The role might be assigned to you at sign-on time if you use SAML federation or web identity (OIDC) federation. If you run your application on an Amazon EC2 instance, then your administrator can assign an IAM role by creating an instance profile.

The component comes in the following forms:

- Client-side libraries in Java, Python, Go, and .NET you interact with the secret instead of directly calling the `GetSecretValue` operation.
- A database driver component compliant with Java Database Connectivity (JDBC). This component is a wrapper around the true JDBC driver that adds the functionality described above.

For instructions to download and use one of the following links:

- Java-based caching client component
- JDBC-compatible database connector component
- Python caching client
- Caching client for .NET
- Go caching client

# Rotate your 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. 25).

**Topics**



## Rotation strategies

There are two rotation strategies offered by Secrets Manager:



### Single user rotation strategy

The single user strategy updates credentials for one user in one secret. See the section called "Tutorial: Rotate a secret for an AWS database" (p. 14).

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.

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.

Each subsequent version of a secret updates the other user. For example, if the first version has `user1/password1`, then the second version has `user2/password2`. The third version has `user1/password3`, and the fourth version has `user2/password4`. You have two sets of valid credentials at any given time: both the current and previous credentials are valid. See the section called "Tutorial: Rotate a user secret with a master secret" (p. 19).

Applications continue to use 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.

A separate secret contains credentials for an administrator or superuser who can create the second user and update both users' credentials.

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 this strategy, you need a separate secret with credentials for an administrator or superuser who has permissions to create a user and change password on both users. The first rotation clones this user to create the alternate user to ensure that both users have the same permissions.

### To use the alternating users strategy

1. Create a user 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 elevated user's credentials.

3. Create a user who will access your database or service.
4. Create a secret for the user's credentials.
5. Turn on automatic rotation for your user's secret, and for **Select which secret will be used to perform the rotation**, choose **Use a secret I previously stored / Multi-user rotation** and then choose the elevated user 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 “Other type of secret” (p. 71).

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.

There are two the section called “Rotation strategies” (p. 68) available as rotation templates: single user and alternating users. You can also Customize a rotation function (p. 76).

Before you begin, you need the following:

- A user with credentials to the database or service.
- A rotation strategy. See the section called “Rotation strategies” (p. 68).
- If you use the the section called “Alternating users” (p. 69), 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. Choose **Enable automatic rotation**.
   b. For **Select rotation interval**, choose the number of days to keep the secret before rotating it.

      If you use the section called “Alternating users” (p. 69), 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.
   c. Do one of the following:
      - To have Secrets Manager create a rotation function for you based on the Rotation function templates (p. 77) 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.

d. For **Select which secret will be used to perform the rotation**, do one of the following:
   - For the Single user rotation strategy (p. 68), choose **Use this secret / Single user rotation**.
   - For the the section called "Alternating users" (p. 69), choose **Use a secret I previously stored / Multi-user rotation**.

For help resolving common rotation issues, see the section called "Troubleshoot rotation" (p. 122).

## AWS SDK and AWS CLI

To turn on rotation, see:

- **API/SDK:** RotateSecret
- **AWS CLI:** rotate-secret

# Automatically rotate another type of secret

Secrets Manager provides complete rotation templates for Amazon RDS, Amazon DocumentDB, and Amazon Redshift secrets. For more information, see the section called "Amazon RDS, Amazon DocumentDB, or Amazon Redshift secret" (p. 70).

For other types of secrets, you create your own rotation function. Secrets Manager provides a the section called "Generic rotation function template" (p. 83) 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. Choose **Enable automatic rotation**.
   b. For **Select rotation interval**, choose the number of days to keep the secret before rotating it.
   c. For **Choose a Lambda 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**. The Lambda console opens to the **Create function** page.
      iii. Choose **Browse serverless app repository**, in the search box enter **`SecretsManagerRotationTemplate`**, choose **Show apps that create custom IAM roles**, and then choose the **SecretsManagerRotationTemplate** card.

iv. Edit **Application settings** as follows:

   A. For **Application name**, enter the appliation name for AWS Serverless Application Repository.
   B. For **functionName**, enter the name of the Lambda function. This is the name you see in the Secrets Manager console.
   C. For **invokingServicePrincipal**, keep **`secretsmanager.amazonaws.com`**.
   D. For **endpoint**, enter the endpoint for your secret's Region. See AWS Secrets Manager endpoints and quotas.
   E. (Optional) For **excludeCharacters**, enter characters that you don't want to use in rotated passwords.
   F. (Optional) For **kmsKeyArn**, enter the KMS key that encrypts the secret. If you don't enter a key ARN, Secrets Manager uses the AWS managed key `aws/secretsmanager`.
   G. (Optional) If your rotation strategy uses an elevated user to change credentials for another user, for **masterSecretArn**, enter the ARN of the secret that contains the elevated credentials. See the section called “Alternating users” (p. 69).
   H. (Optional) If your rotation strategy uses an elevated user to change credentials for another user, for **masterSecretKmsKeyArn**, enter the KMS key that encrypts the secret. If you don't enter a key ARN, Secrets Manager uses the AWS managed key `aws/secretsmanager`.
   I. (Optional) For **vpcSecurityGroupIds**, enter the VPC security group that your database or service uses. See the section called “Network access for rotation” (p. 74).
   J. (Optional) For **vpcSubnetIds**, enter the VPC subnets that your database or service uses. See the section called “Network access for rotation” (p. 74).
   K. Choose **I acknowledge that this app creates custom IAM roles and resource policies**, because this app creates a Lambda execution role that Secrets Manager uses to run the Lambda function.
   L. Choose **Deploy**.
   M. After the function deploys, under **Resources**, choose **SecretsManagerRotationTemplate** to open the function for editing. Implement each of the steps described in the section called “How rotation works” (p. 73). Use Python 3.7.

      When your function is complete, return to the Secrets Manager console to finish your secret.

v. For **Choose a Lambda function**, choose the refresh button. Then in the list of functions, choose your new function.
vi. Choose **Save**.

For help resolving common rotation issues, see the section called “Troubleshoot rotation” (p. 122).

## AWS SDK and AWS CLI

To turn on rotation, see:

- **API/SDK:** RotateSecret
- **AWS CLI:** rotate-secret

# Rotate a secret now

You can only rotate a secret that has automatic rotation turned on. Turn on automatic rotation for:

- Amazon RDS, Amazon DocumentDB, or Amazon Redshift secret (p. 70)
- Other type of secret (p. 71)

**To rotate a secret now (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 now, see:

- **API/SDK:** RotateSecret
- **AWS CLI:** rotate-secret

# 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. 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:

**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.

**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. 77) 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. 76).

**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 Secrets Manager non-API events (p. 90).

After rotation is successful, applications that ??? (p. 66) from Secrets Manager automatically get the updated credentials. For more details about how each step of rotation works, see the the section called “Rotation function templates” (p. 77).

To turn on automatic rotation, see:

- the section called “Amazon RDS, Amazon DocumentDB, or Amazon Redshift secret” (p. 70)
- the section called “Other type of secret” (p. 71)

# 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 endpoints (p. 84).

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.

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 the Lambda rotation function

Secrets Manager uses a Lambda function to rotate a secret. The Lambda function has a resource policy that allows Secrets Manager to invoke it.

Secrets Manager calls the Lambda function by invoking an IAM execution role attached to the Lambda function. Permissions for the Lambda function are granted through the IAM execution role as inline policies.

- For an Amazon RDS, Amazon DocumentDB, or Amazon Redshift secret (p. 70), Secrets Manager creates the role for you and attaches permissions to it.
- For an Other type of secret (p. 71), if you create the Lambda function through the Secrets Manager console or the AWS Serverless Application Repository console, the role is created for you. If you create the Lambda function another way, you need to also create an execution role and make sure it has the correct permissons.

For information about permissions to access secrets, see the section called “Permissions to access secrets” (p. 25).

### Example Lambda function resource policy

The following policy allows Secrets Manager to assume the role by identifying the service as the `Principal`: `secretsmanager.amazonaws.com`, and it allows Secrets Manager to invoke the Lambda function in the `Resource`.

```
{
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE",
      "Effect": "Allow",
      "Principal": {
        "Service": "secretsmanager.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "LambdaRotationFunctionARN"
    }
  ]
}
```

### Example IAM execution role inline policy for single user rotation strategy

For an the section called “Amazon RDS, Amazon DocumentDB, or Amazon Redshift secret” (p. 70), Secrets Manager creates the IAM execution role and attaches this policy for you.

The following example policy does the following:

- Allows the rotation function to run Secrets Manager operations for secrets that are configured to use this rotation function.
- Allows the function to create a new password.
- Allows Lambda to set up the required configuration if you 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",
            "Condition": {
                "StringEquals": {
                    "secretsmanager:resource/AllowRotationLambdaArn":
 "LambdaRotationFunctionARN"
                }
            }
        },
        {
            "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 the section called “Amazon RDS, Amazon DocumentDB, or Amazon Redshift secret” (p. 70), Secrets Manager creates the IAM execution role and attaches this policy for you.

For the alternating users strategy, the following example shows a statement to add to the execution role policy. This statement allows the function to retrieve the credentials in the separate secret. Secrets Manager uses the credentials in the separate secret to update the credentials in the rotated secret.

```
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "SeparateSecretARN"
        }
```

**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": "*"
        }
```

# Customize a Lambda rotation function for Secrets Manager

For Amazon RDS, Amazon DocumentDB, or Amazon Redshift secret (p. 70), Secrets Manager can create rotation functions for you for the Single user (p. 68) or Alternating users (p. 69) rotation strategies.

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. 73) 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`. Use Python 3.7 to customize it.

# 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 "Amazon RDS, Amazon DocumentDB, or Amazon Redshift secret" (p. 70) or the section called "Other type of secret" (p. 71). 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:

**Contents**
  ```

# Amazon RDS databases

## Amazon RDS MariaDB single user

- **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. 68).
- **Expected `SecretString` structure:**

```
{
  "engine": "mariadb",
  "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>"
}
```

- **Source code**

## Amazon RDS MariaDB alternating users

- **Name:** SecretsManagerRDSMariaDBRotationMultiUser
- **Supported database/service:** MariaDB database hosted on an Amazon RDS database instance.
- **Rotation strategy:** Alternating users rotation strategy (p. 69).
- **Expected `SecretString` structure:**

```
{
  "engine": "mariadb",
  "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>",
  "masterarn": "<required: the ARN of the elevated secret used to create 2nd user and
 change passwords>"
}
```

- **Source code**

## Amazon RDS MySQL single user

- **Name:** SecretsManagerRDSMySQLRotationSingleUser
- **Supported database/service:** MySQL database hosted on an Amazon Relational Database Service (Amazon RDS) database instance.
- **Rotation strategy:** Single user rotation strategy (p. 68).
- **Expected `SecretString` structure:**

```
{
  "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>"
}
```

- **Source code**

## Amazon RDS MySQL alternating users

- **Name:** SecretsManagerRDSMySQLRotationMultiUser
- **Supported database/service:** MySQL database hosted on an Amazon RDS database instance.
- **Rotation strategy:** Alternating users rotation strategy (p. 69).
- **Expected `SecretString` structure:**

```
{
  "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>",
  "masterarn": "<required: the ARN of the elevated secret used to create 2nd user and
 change passwords>"
}
```

- **Source code**

## Amazon RDS Oracle single user

- **Name:** SecretsManagerRDSOracleRotationSingleUser
- **Supported database/service:** Oracle database hosted on an Amazon Relational Database Service (Amazon RDS) database instance.
- **Rotation strategy:** Single user rotation strategy (p. 68).
- **Expected `SecretString` 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>"
```

```
}
```

- **Source code**

## Amazon RDS Oracle alternating users

- **Name:** SecretsManagerRDSOracleRotationMultiUser
- **Supported database/service:** Oracle database hosted on an Amazon RDS database instance.
- **Rotation strategy:** Alternating users rotation strategy (p. 69).
- **Expected `SecretString` 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>",
  "masterarn": "<required: the ARN of the elevated secret used to create 2nd user and
 change passwords>"
}
```

- **Source code**

## Amazon RDS PostgreSQL single user

- **Name:** SecretsManagerRDSPostgreSQLRotationSingleUser
- **Supported database/service:** PostgreSQL database hosted on an Amazon RDS database instance.
- **Rotation strategy:** Single user rotation strategy (p. 68).
- **Expected `SecretString` structure:**

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

- **Source code**

## Amazon RDS PostgreSQL alternating users

- **Name:** SecretsManagerRDSPostgreSQLRotationMultiUser
- **Supported database/service:** PostgreSQL database hosted on an Amazon RDS database instance.
- **Rotation strategy:** Alternating users rotation strategy (p. 69).
- **Expected `SecretString` structure:**

```
{
  "engine": "postgres",
  "host": "<required: instance host name/resolvable DNS name>",
  "username": "<required: username>",
  "password": "<required: password>",
```

```
  "dbname": "<optional: database name. If not specified, defaults to 'postgres'>",
  "port": "<optional: TCP port number. If not specified, defaults to 5432>",
  "masterarn": "<required: the ARN of the elevated secret used to create 2nd user and
 change passwords>"
}
```

- **Source code**

## Amazon RDS Microsoft SQLServer single user

- **Name:** SecretsManagerRDSSQLServerRotationSingleUser
- **Supported database/service:** Microsoft SQLServer database hosted on an Amazon RDS database instance.
- **Rotation strategy:** Single user rotation strategy (p. 68).
- **Expected `SecretString` structure:**

```
{
  "engine": "sqlserver",
  "host": "<required: instance host name/resolvable DNS name>",
  "username": "<required: username>",
  "password": "<required: password>",
  "dbname": "<optional: database name. If not specified, defaults to 'master'>",
  "port": "<optional: TCP port number. If not specified, defaults to 1433>"
}
```

- **Source code**

## Amazon RDS Microsoft SQLServer alternating users

- **Name:** SecretsManagerRDSSQLServerRotationMultiUser
- **Supported database/service:** Microsoft SQLServer database hosted on an Amazon RDS database instance.
- **Rotation strategy:** Alternating users rotation strategy (p. 69).
- **Expected `SecretString` structure:**

```
{
  "engine": "sqlserver",
  "host": "<required: instance host name/resolvable DNS name>",
  "username": "<required: username>",
  "password": "<required: password>",
  "dbname": "<optional: database name. If not specified, defaults to 'master'>",
  "port": "<optional: TCP port number. If not specified, defaults to 1433>",
  "masterarn": "<required: the ARN of the elevated secret used to create 2nd user and
 change passwords>"
}
```

- **Source code**

# Amazon DocumentDB databases

## Amazon DocumentDB MongoDB single user

- **Name:** SecretsManagerMongoDBRotationSingleUser
- **Supported database/service:** MongoDB database version 3.2 or 3.4.

- **Rotation strategy:** Single user rotation strategy (p. 68).
- **Expected `SecretString` structure:**

```
{
  "engine": "mongo",
  "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 27017>"
}
```

- **Source code:** Source code

## Amazon DocumentDB MongoDB alternating users

- **Name:** SecretsManagerMongoDBRotationMultiUser
- **Supported database or service:** MongoDB database version 3.2 or 3.4.
- **Rotation strategy:** Alternating users rotation strategy (p. 69).
- **Expected `SecretString` structure:**

```
{
  "engine": "mongo",
  "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 27017>",
  "masterarn": "<required: the ARN of the elevated secret used to create 2nd user and
 change passwords>"
}
```

- Source code

# Amazon Redshift

## Amazon Redshift single user

```
arn:aws:serverlessrepo:us-east-2:123456789012:applications/
SecretsManagerRDSMySQLRotationSingleUser
```

- **Name:** SecretsManagerRedshiftRotationSingleUser
- **Supported database/service:** Amazon Redshift
- **Rotation strategy:** Single user rotation strategy (p. 68).
- **Expected `SecretString` structure:**

```
{
  "engine": "redshift",
  "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 5439>"
}
```

- **Source code**

## Amazon Redshift primary user

- **Name:** SecretsManagerRedshiftRotationMultiUser
- **Supported database/service:** Amazon Redshift
- **Rotation strategy:** Alternating users rotation strategy (p. 69).
- **Expected `SecretString` structure:**

```
{
  "engine": "redshift",
  "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 5439",
  "masterarn": "<required: the elevated secret ARN used to create 2nd user and change
 passwords"
}
```

- **Source code**

# Other types of secrets

## Generic rotation function 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. See the section called "Other type of secret" (p. 71).
- **Expected `SecretString` structure:** You define this.
- **Source code:** **Source code**

# Using Secrets Manager with VPC endpoints

Instead of connecting your VPC to an internet, you can connect directly to Secrets Manager through a private endpoint you configure within your VPC. When you use a VPC service endpoint, communication between your VPC and Secrets Manager occurs entirely within the AWS network, and requires no public Internet access.

Secrets Manager supports Amazon VPC interface endpoints provided by AWS PrivateLink. Each VPC endpoint is represented by one or more elastic network interfaces with private IP addresses in your VPC subnets.

For information about permissions policies and VPCs, see the section called "Example: Permissions and VPCs" (p. 33).

The VPC interface endpoint connects your VPC directly to Secrets Manager without a NAT device, VPN connection, or AWS Direct Connect connection. The instances in your VPC don't require public IP addresses to communicate with Secrets Manager.

For your Lambda rotation function to find the private endpoint, perform one of the following steps:

- You can manually specify the VPC endpoint in Secrets Manager API operations and AWS CLI commands. For example, the following command uses the **endpoint-url** parameter to specify a VPC endpoint in an AWS CLI command to Secrets Manager. For more information, see AWS CLI command line options.

```
$ aws secretsmanager list-secrets --endpoint-url https://
vpce-1234a5678b9012c-12345678.secretsmanager.us-west-2.vpce.amazonaws.com
```

- If you enable private DNS hostnames for your VPC private endpoint, you don't need to specify the endpoint URL. The standard Secrets Manager DNS hostname the Secrets Manager CLI and SDKs use by default (`https://secretsmanager.<region>.amazonaws.com`) automatically resolves to your VPC endpoint.

You can also use AWS CloudTrail logs to audit your use of secrets through the VPC endpoint. And you can use the conditions in IAM and secret resource-based policies to deny access to any request that doesn't originate from a specified VPC or VPC endpoint.

> **Note**
> Use caution when creating IAM and key policies based on your VPC endpoint. If a policy statement requires the requests originate from a particular VPC or VPC endpoint, then requests from other AWS services interacting with the secret on your behalf might fail. For help, see VPC endpoint conditions (p. 42).

**Regions**

Secrets Manager supports VPC endpoints in all AWS Regions where both Amazon VPC and Secrets Manager are available.

For more information, see VPC Endpoints.

**To create a Secrets Manager VPC private endpoint**

Follow the steps under one of the following tabs:

Using the AWS Management Console

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. On the navigation bar, use the region selector to choose the region.
3. In the navigation pane, choose **Endpoints**. In the main pane, choose **Create Endpoint**.
4. For **Service category**, choose **AWS services**.
5. In the **Service Name** list, choose the entry for the Secrets Manager interface endpoint in the region. For example, in the US East (N.Virginia) Region, the entry name is `com.amazonaws.us-east-1.secretsmanager`.
6. For **VPC**, choose your VPC.
7. For **Subnets**, choose a subnet from each Availability Zone to include.

   The VPC endpoint can span multiple Availability Zones. AWS creates an elastic network interface for the VPC endpoint in each subnet that you choose. Each network interface has a DNS hostname and a private IP address.
8. By default, AWS enables the **Enable Private DNS Name** option, the standard Secrets Manager DNS hostname (`https://secretsmanager.<region>.amazonaws.com`) automatically resolves to your VPC endpoint. This option makes it easier to use the VPC endpoint. The Secrets Manager CLI and SDKs use the standard DNS hostname by default, so you don't need to specify the VPC endpoint URL in applications and commands.

   This feature works only when you set the `enableDnsHostnames` and `enableDnsSupport` attributes of your VPC to `true` , the default values. To set these attributes, update DNS support for your VPC.
9. For **Security group**, select or create a security group.

   You can use security groups to control access to your endpoint, much like you would use a firewall.
10. Choose **Create endpoint**.

The results display the VPC endpoint, including the VPC endpoint ID and the DNS names you use to connect to your VPC endpoint.

You can also use the Amazon VPC tools to view and manage your endpoint. This includes creating a notification for an endpoint, changing properties of the endpoint, and deleting the endpoint. For instructions, see Interface VPC Endpoints.

Using the AWS CLI or SDK Operations

You can use the create-vpc-endpoint command in the AWS CLI to create a VPC endpoint that connects to Secrets Manager.

Be sure to use `interface` as the VPC endpoint type. Also, use a service name value that includes `secretsmanager` and the region where you located your VPC.

The command doesn't include the `PrivateDnsNames` parameter because the VPC defaults to the value `true`. To disable the option, you can include the parameter with a value of `false`. Private DNS names are available only when the `enableDnsHostnames` and `enableDnsSupport` attributes of your VPC are set to `true`. To set these attributes, use the ModifyVpcAttribute API.

The following diagram shows the general syntax of the command.

```
aws ec2 create-vpc-endpoint  --vpc-id <vpc id> \
                             --vpc-endpoint-type Interface \
                             --service-name com.amazonaws.<region>.secretsmanager \
                             --subnet-ids <subnet id> \
                             --security-group-id <security group id>
```

For example, the following command creates a VPC endpoint in the VPC with VPC ID `vpc-1a2b3c4d`, which is in the `us-west-2` Region. It specifies just one subnet ID to represent the Availability Zones, but you can specify many. VPC endpoints require the security group ID.

The output includes the VPC endpoint ID and DNS names you can use to connect to your new VPC endpoint.

```
$ aws ec2 create-vpc-endpoint  --vpc-id vpc-1a2b3c4d \
                               --vpc-endpoint-type Interface \
                               --service-name com.amazonaws.us-west-2.secretsmanager \
                               --subnet-ids subnet-e5f6a7b8c9 \
                               --security-group-id sg-1a2b3c4d
{
  "VpcEndpoint": {
      "PolicyDocument": "{\n  \"Statement\": [\n    {\n      \"Action\": \"*\", \n
  \"Effect\": \"Allow\", \n      \"Principal\": \"*\", \n      \"Resource\": \"*\"\n
  }\n  ]\n}",
      "VpcId": "vpc-1a2b3c4d",
      "NetworkInterfaceIds": [
          "eni-abcdef12"
      ],
      "SubnetIds": [
          "subnet-e5f6a7b8c9"
      ],
      "PrivateDnsEnabled": true,
      "State": "pending",
      "ServiceName": "com.amazonaws.us-west-2.secretsmanager",
      "RouteTableIds": [],
      "Groups": [
          {
              "GroupName": "default",
              "GroupId": "sg-1a2b3c4d"
          }
      ],
      "VpcEndpointId": "vpce-1234a5678b9012c",
      "VpcEndpointType": "Interface",
      "CreationTimestamp": "2018-06-12T20:14:41.240Z",
      "DnsEntries": [
          {
              "HostedZoneId": "Z7HUB22UULQXV",
              "DnsName": "vpce-1234a5678b9012c-12345678.secretsmanager.us-
west-2.vpce.amazonaws.com"
          },
          {
              "HostedZoneId": "Z7HUB22UULQXV",
              "DnsName": "vpce-1234a5678b9012c-12345678-us-west-2a.secretsmanager.us-
west-2.vpce.amazonaws.com"
          },
          {
              "HostedZoneId": "Z1K56Z6FNPJRR",
              "DnsName": "secretsmanager.us-west-2.amazonaws.com"
          }
      ]
  }
```

```
}
```

# Create an endpoint policy for your Secrets Manager VPC endpoint

Once you create a Secrets Manager VPC endpoint, you can attach an endpoint policy to control secrets-related activity on the endpoint. For example, you can attach an endpoint policy to define the performed Secrets Manager actions, actions performed on the secrets, the IAM users or roles performing these actions, and the accounts accessed through the VPC endpoint. For additional information about endpoint policies, including a list of the AWS services supporting endpoint policies, see Using VPC Endpoint policies.

To add a policy to your secret, on the **Secret details** page, choose **Edit permissions**.

> **Note**
> AWS does not share VPC endpoints across AWS services. If you use VPC endpoints for multiple AWS services, such as Secrets Manager and Amazon S3, you must attach a distinct policy to each endpoint.

**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"
                      ]
            }
          ]
      }
```

# Connecting to a Secrets Manager VPC private endpoint

Because, by default, VPC automatically enables private DNS names when you create a VPC private endpoint, you don't need to do anything other than use the standard endpoint DNS name for your region. The endpoint DNS name automatically resolves to the correct endpoint within your VPC:

```
https://secretsmanager.<region>.amazonaws.com
```

The AWS CLI and SDKs use this hostname by default, so you can begin using the VPC endpoint without changing anything in your scripts and application.

If you don't enable private DNS names, you can still connect to the endpoint by using the full DNS name.

For example, this list-secrets command uses the `endpoint-url` parameter to specify the VPC private endpoint. To use a command like this, replace the example VPC private endpoint ID with one in your account.

```
aws secretsmanager list-secrets --endpoint-url https://
vpce-1234a5678b9012c-12345678.secretsmanager.us-west-2.vpce.amazonaws.com
```

# Audit the use of your Secrets Manager VPC endpoint

When a request to Secrets Manager uses a VPC endpoint, the VPC endpoint ID appears in the AWS CloudTrail log (p. 90) entry that records the request. You can use the endpoint ID to audit the use of your Secrets Manager VPC endpoint.

For example, this sample log entry records a `GenerateDataKey` request that used the VPC endpoint. In this example, the `vpcEndpointId` field appears at the end of the log entry. For brevity, many irrelevant parts of the example have been omitted.

```
     {
"eventVersion":"1.05",
"userIdentity":
     {
            "type": "IAMUser",
            "arn": "arn:aws:iam::123456789012:user/Anika",
            "accountId": "123456789012",
            "userName": "Anika"
           },
"eventTime":"2018-01-16T05:46:57Z",
"eventSource":"secretsmanager.amazonaws.com",
"eventName":"GetSecretValue",
"awsRegion":"us-west-2",
"sourceIPAddress":"172.01.01.001",
"userAgent":"aws-cli/1.14.23 Python/2.7.12 Linux/4.9.75-25.55.amzn1.x86_64
botocore/1.8.27",
"requestID":"EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE",
"eventID":"EXAMPLE2-90ab-cdef-fedc-ba987EXAMPLE",
"readOnly":true,
"eventType":"AwsApiCall",
```

```
"recipientAccountId":"123456789012",
"vpcEndpointId": "vpce-1234a5678b9012c"
     }
```

# Monitor the use of your AWS Secrets Manager secrets

You can use CloudTrail and CloudWatch to monitor activity related to your secrets. CloudTrail captures API activity for your AWS resources by any AWS service and writes the activity to log files in your Amazon S3 buckets. CloudWatch enables you to create rules to monitor those log files and trigger actions when activities of interest occur. For example, a text message can alert you whenever someone creates a new secret, or when a secret rotates successfully. You could also create an alert for when a client attempts to use a deprecated version of a secret instead of the current version. This can help with troubleshooting.

As a best practice, you should monitor your secrets to ensure usage of your secrets and log any changes to them. This helps you to ensure that any unexpected usage or change can be investigated, and unwanted changes can be rolled back. AWS Secrets Manager currently supports two AWS services that enable you to monitor your organization and activity.

**Topics**



# Log AWS Secrets Manager API calls with AWS CloudTrail

AWS Secrets Manager integrates with AWS CloudTrail, a service that provides a record of actions taken by a user, role or an AWS service in Secrets Manager. CloudTrail captures all API calls for Secrets Manager as events, including calls from the Secrets Manager console and from code calls to the Secrets Manager APIs. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Secrets Manager. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request sent to Secrets Manager, the IP address of the request, who sent the request, when the time of the request, and additional details.

To learn more about CloudTrail see the AWS CloudTrail User Guide.

## Log AWS Secrets Manager non-API events

In addition to logging AWS API calls, CloudTrail captures other related events that might have a security or compliance impact on your AWS account or might help you troubleshoot operational problems. CloudTrail records these events as non-API service events.

Secrets Manager has the following non-API service events:

- `RotationAbandoned` event - a mechanism to inform you that the Secrets Manager service 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 mechanism that notifies you of a secret starting rotation.

- `RotationSucceeded` event - a mechanism that notifies you of a successful rotation event.
- `RotationFailed` event - a mechanism to inform you that secret rotation failed for an application.
- `StartSecretVersionDelete` event - a mechanism that notifies you of the start deletion for a secret version.
- `CancelSecretVersionDelete` event - a mechanism that notifies you of a delete cancellation for a secret version.
- `EndSecretVersionDelete` event - a mechanism that notifies you of an ending secret version deletion.

# Secrets Manager information in CloudTrail

When you create your AWS account, AWS enables CloudTrail. When activity occurs in Secrets Manager, CloudTrail records the activity in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Secrets Manager, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all regions. The trail logs events from all regions in the AWS partition and delivers the log files to the Amazon S3 bucket you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

CloudTrail logs all Secrets Manager actions and documents the actions in the AWS Secrets Manager API Reference. For example, calls to the `CreateSecret`, `GetSecretValue` and `RotateSecret` sections generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- If root or IAM user credentials generated the request.
- If temporary security credentials for a role or federated user generated the request.
- If another AWS service generated the request.

For notification of log file delivery, you can configure CloudTrail to publish Amazon SNS notifications. For more information, see Configuring Amazon SNS Notifications for CloudTrail.

You also can aggregate AWS Secrets Manager log files from multiple AWS Regions and multiple AWS accounts into a single Amazon S3 bucket.

For more information, see Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts.

# Retrieve Secrets Manager log file entries

You can retrieve individual events from CloudTrail by using any of the following techniques:

**To retrieve Secrets Manager events from CloudTrail logs**

Using the AWS Management Console

The CloudTrail console enables you to view events that occurred within the past 90 days.

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**.

Using the AWS CLI or SDK Operations

1. Open a command window to run AWS CLI commands.
2. Run a command similar to the following example. For readability here, the output displays as word-wrapped, but the real output doesn't.

```
$ 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\"}"
        }
    ]
}
```

# Secrets Manager log file entries

A trail enables delivery of events as log files to a specified Amazon S3 bucket. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information

about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files does not collect an ordered stack trace of the public API calls, so they do not appear in any specific order.

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"
}
```

# Amazon CloudWatch Events

Secrets Manager can work with CloudWatch Events to trigger alerts when administrator-specified operations occur in an organization. For example, because of the sensitivity of such operations, administrators might want to be warned of deleted secrets or secret rotation. You might want an alert if anyone tries to use a secret version in the waiting period to be deleted. You can configure CloudWatch Events rules that look for these operations and then send the generated events to administrator defined "targets". A target could be an Amazon SNS topic that emails or text messages to subscribers. You can also create a simple AWS Lambda function triggered by the event, which logs the details of the operation for your later review.

To learn more about CloudWatch Events, including how to configure and enable it, see the Amazon CloudWatch Events User Guide.

## Monitor secret versions 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 version of 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 version of 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 ManagerAPI requests. For instructions, go to Creating a trail for the first time in the *AWS CloudTrail User Guide*.

**Steps**

- Part 1: Configure CloudTrail log file delivery to CloudWatch logs (p. 94)
- Part 2: Create the CloudWatch alarm (p. 95)
- Part 3: Monitor CloudWatch for deleted secrets (p. 96)

### Part 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 version of 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**.

## Part 2: Create the CloudWatch alarm

To receive a notification when a Secrets Manager `GetSecretValue` API operation requests to access a version of a secret pending deletion, you must create a CloudWatch alarm and configure notification.

### Creating a CloudWatch alarm to monitors usage of a version of a secret pending deletion

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**.

### Part 3: Monitor CloudWatch for deleted secrets

You have created your alarm. To test it, create a version of a secret and then schedule it for deletion. Then, try to retrieve the secret value. You shortly receive an email at the address configured in the alarm. It alerts you to the use of a secret version scheduled for deletion.

# Monitor Secrets Manager secrets using AWS Config

AWS Secrets Manager integrates with AWS Config and provides easier tracking of secret changes in Secrets Manager. You can use two additional managed AWS Config rules for defining your organizational internal guidelines on secrets management best practices. Also, you can quickly identify secrets that don't conform to your security rules as well as receive notifications from Amazon SNS about your secrets configuration changes. 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.

When you leverage the AWS Config Multi-Account Multi-Region Data Aggregator, you can view a list of secrets and verify conformity across all accounts and regions in your entire organization, and by doing so, create secrets management best practices across your organization from a single location.

AWS Config allows you to assess, audit, and evaluate configurations across your AWS resources by continually monitoring and recording your AWS resource configurations, and also allows you to automate the evaluation of recorded configurations against desired configurations. Using AWS Config, you can perform the following tasks:

- Review changes in configurations and relationships between AWS resources.
- Track detailed resource configuration histories.
- Determine your overall compliance with configurations specified in your internal guidelines.

If you have multi-account and multi-region AWS resources in AWS Config, you can view configuration and compliance data from multiple accounts and regions into a single account. As a result, you can identify noncompliant resources across multiple accounts..

By tracking your secrets with AWS Config, a secret becomes an AWS Config supported resource, and you can track changes to the secret metadata, such as secret description and rotation configuration, relationship to other AWS sources such as the KMS key used for secret encryption, Lambda function used for secret rotation, as well as attributes such as tags associated with the secrets.

You can also leverage AWS Managed Config Rules to evaluate if your secrets configuration complies with your organization's security and compliance requirements, and identify secrets that don't conform to these standards.

**Topics**



## AWS Config supported rules for Secrets Manager

When you use AWS Config to evaluate your resource configurations, you can assess how well your resource configurations comply with internal practices, industry guidelines, and regulations. AWS Config supports the following rules for Secrets Manager:

- `secretsmanager-rotation-enabled-check` — Checks if you configured rotation for secrets stored in Secrets Manager. AWS Config verifies you configured the secrets for rotation. This rule also supports the `maximumAllowedRotationFrequency`, which if specified, compares the frequency configuration of the secret to the value set in the parameter.
- `secretsmanager-scheduled-rotation-success-check`— Checks if Secrets Manager successfully rotates secrets. AWS Config verifies the rule and checks if the last rotated date falls within the configured rotation frequency.

  For more information about AWS Config and rules, see the AWS Config product documentation.
- `secretsmanager-using-cmk` — Checks if AWS encrypts all secrets using the AWS managed key `aws/secretsmanager` or a customer managed key you created in AWS KMS.
- `secretsmanager-secret-unused`— Checks if Secrets Manager accessed secrets within a specified number of days.
- `secretsmanager-secret-periodic-rotation`— Checks if AWS rotated secrets within the past specified number of days.

## AWS Config supported resources for Secrets Manager

AWS resources are entities you create and manage using the AWS Management Console, the AWS Command Line Interface (AWS CLI), the AWS SDKs, or AWS partner tools. AWS Config refers to each resource using a unique identifier, such as the resource ID or an Amazon Resource Name (ARN).

AWS Config supports the Secrets Manager resource, `AWS::SecretsManager::Secret`. For more information, see AWS Config Developer Guide.

# Best practices using AWS Config

Secrets Manager rotates secrets as part of a best practice security plan. Using the managed rule, `secretsmanager-rotation-enabled-check`, AWS Config verifies rotation of secrets in Secrets Manager.

For more information about rule configuration in AWS Config, see AWS Config Rules in the AWS Config Developer Guide.

# Configure AWS Config Secrets Manager rules

> **Important**
> If using the AWS Config console for the first time, see Setting Up AWS Config (Console).

The Rules page provides initial AWS managed rules you can add to your account. After set up, AWS Config evaluates your AWS Secrets Manager resources against your selected rules. You can update the rules and create additional managed rules after set up.

1. Log into the AWS Config console at https://console.aws.amazon.com/config/.
2. Choose **Settings**. Be sure you enable the parameter **Recording is on**.
3. Choose **Rules**.
4. In the **Rules** section, choose **Add Rule**.
5. Type **`secretsmanager-rotation-enabled-check`** in the filter field.
6. To configure the `secretsmanager-rotation-enabled-check` rule, choose **Rules** from the console panel, and then choose **Add rule**.
7. Locate the rule, `secretsmanager-rotation-enabled-check` using the search function.
8. Create a unique name for the rule such as *MySecretsRotationRule*.

9. Specify a **Remedial Action** to receive notification about noncompliant secrets using a Amazon SNS topic.
10. Specify a topic for the Amazon SNS notification.
11. Choose **Save** to store the rule in AWS Config

Once you save the rule, AWS Config evaluates your secrets every time the metadata of a secret changes. If changes occur, you receive an Amazon SNS notification about noncompliant secrets. You can also view the results from the **Rules** or **Dashboard** of AWS Config.

If you choose the **secretsmanager-rotation-check-mySecretsRotationRule** from the list of rules, then AWS Config displays a list of secrets in your account not configured for rotation. Because you identified the secrets, you can begin implementation of your best practices for secret rotation.

- See AWS Config documentation on the `secretsmanager-rotation-enabled-check`.
- See AWS Config documentation on the `secretsmanager-scheduled-rotation-enabled-success-check`.

## View Secrets Manager rules in AWS Config

To view your Secrets Manager rules in AWS Config Developer Guide, log into the AWS Config console and choose **Rules**. Choose the desired rule and perfom any of the following options:

- **View details**
- **Edit Rule**
- **Actions**

For more information about viewing your rules, see AWS Config Developer Guide.

# Configure AWS Config Multi-Account Multi-Region Data Aggregator for secrets management best practices

You 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.

> **Note**
> 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 in the AWS Config Developer Guide.

You create an Aggregator from your organization's management account or from any member account in your organization. Use the following steps to create an AWS Config Aggregator:

1. Log into the AWS Config console at https://console.aws.amazon.com/config/.
2. Choose **Aggregators** and then **Add Aggregator**.
3. In the **Allow data replication** section, check the **Allow AWS Config to replicate data**. You must enable this to provide the management account access to the resource configuration and compliance details for all of the accounts and regions in your organization.

4. Type a unique name, consisting of up to 64 alphanumeric characters, for your aggregator in the **Aggregator name** field.
5. In the **Select source accounts** section, choose **Add my organization**, and then click **Choose IAM role**.
6. Under **Regions**, select all regions applicable to your organization and then choose **Save**.

To view a dashboard of all secrets in your organization, choose the name of your aggregator. You can now see all of the secrets across all accounts and regions.

For more information on configuring AWS Config aggregators, see Setting Up an Aggregator Using the Console.

Review the data to identify all noncompliant secrets in your organization. Work with individual account and secret owners to apply secrets management best practices such as ensuring rotation for all secrets. Ensure all secrets meet your best practices policies for rotation frequency and you proactively identify unused secrets and schedule them for deletion.

**Tip**
More information about Secrets Manager and AWS Config can be found in the following AWS Config documentation:

- List of AWS Config Managed Rules
- AWS Config Supported AWS Resource Types and Resource Relationships
- Notifications that AWS Config Sends to an Amazon SNS topic

# AWS services integrated with AWS Secrets Manager

AWS Secrets Manager works with the following services:



## Store AWS CodeBuild registry credentials with 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. CodeBuild eliminates the need to provision, manage, and scale your own build servers. It provides prepackaged build environments for popular programming languages and build tools such as Apache Maven, Gradle, and more. You can also customize build environments in CodeBuild to use your own build tools. CodeBuild scales automatically to meet peak build requests.

You can store your private registry credentials using Secrets Manager. See Private registry with AWS Secrets Manager sample for CodeBuild.

## Integrate Secrets Manager with Amazon Elastic Container Service

Amazon ECS enables you to inject sensitive data into your containers by storing your sensitive data in Secrets Manager secrets and then referencing them 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 a complete description of the integration, see Specifying Sensitive Data Secrets.

Tutorial: Specifying Sensitive Data Using Secrets Manager Secrets

# Store Amazon EMR registry credentials with 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. See Add a Git-based Repository to Amazon EMR.

# Integrate Secrets Manager with AWS Fargate

AWS Fargate is a technology that you can use with Amazon ECS to run containers without managing servers or clusters of Amazon ECS instances. With AWS Fargate, you no longer have to provision, configure, or scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing.

When you run your Amazon Amazon ECS tasks and services with the Fargate launch type or a Fargate capacity provider, you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application. Each Fargate task has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task

You can configure Fargate interfaces to allow retrieval of secrets from Secrets Manager. For more information, see Fargate Task Networking

# Integrate Secrets Manager with AWS IoT Greengrass

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. For example, the Twilio Notifications connector uses a locally stored authentication token.

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 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. 105).

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://raw.githubusercontent.com/kubernetes-sigs/secrets-store-csi-driver/master/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. 26).

For a tutorial about how to use the ASCP, see the section called "Tutorial" (p. 105).

# 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.

For a tutorial about how to use the ASCP, see the section called "Tutorial" (p. 105).

# `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. 104).

**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. 10).

**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. 102).

**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 the section called “Create a secret” (p. 43).

   ```
   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. 103).

```
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>
```

# Retrieving your secrets with the Parameter Store APIs

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*.

# Managing SageMaker repository credentials with 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. It also provides common machine learning algorithms that are optimized to run efficiently against extremely large data in a distributed environment. With native support for bring-your-own-algorithms and frameworks, SageMaker offers flexible distributed training options that adjust to your specific workflows. Deploy a model into a secure and scalable environment by launching it with a single click from the SageMaker console.

You can manage your private repositories credentials using Secrets Manager.

For more information, see Associate Git Repositories with Amazon SageMaker Notebook Instances.

# Using Security Hub for security best practices in Secrets Manager

AWS Security Hub provides you with a comprehensive view of your security state in AWS and helps you check your environment against security industry standards and best practices.

Security Hub collects security data from across AWS accounts, services, and supported third-party partner products and helps you analyze your security trends and identify the highest priority security issues.

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.

See AWS Foundational Security Best Practices controls.

# Running everything in a VPC

Whenever possible, we recommend that you run as much of your infrastructure on private networks not accessible from the public internet. To do this, host your servers and services in a virtual private cloud (VPC) provided by Amazon VPC. AWS provides a virtualized private network accessible only to the resources in your account. The public internet can't view or access, unless you explicitly configure it with

access, for example with a NAT gateway. For information about Amazon VPC, see the Amazon VPC User Guide.

To enable secret rotation within a VPC environment, perform these steps:

1. Configure your Lambda rotation function to run within the same VPC as the database server or service with a rotated secret. For more information, see Configuring a Lambda Function to Access Resources in an Amazon VPC in the *AWS Lambda Developer Guide*.
2. The Lambda rotation function, now running from within your VPC, must be able to access a Secrets Manager service endpoint. If the VPC has no direct Internet connectivity, then you can configure your VPC with a private Secrets Manager endpoint accessible by all of the resources in your VPC. For details, see *VPC endpoints* (p. 84).

# Integrating Zelkova with Secrets Manager resource policies

Zelkova uses automated reasoning to analyze policies and the future consequences of policies. This includes AWS Identity and Access Management (IAM) policies, Amazon Simple Storage Service (Amazon S3) policies, Secrets Manager resource policies, and other resource policies. These policies dictate who can (or can't) perform actions on which resources. Because Zelkova uses automated reasoning, you no longer have to think about what questions you need to ask about your policies. Using fancy math, as mentioned above, Zelkova automatically derives the questions and answers you should ask about your policies, and improves your confidence in your security configuration(s).

For more information about Zelkova, see How AWS uses automated reasoning to help you achieve security at scale on the AWS Security Blog.

# 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**



## 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^Z
      # 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. 112).

## 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. 112).

## 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 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 KMS keys. For help determining whether a KMS key is symmetric or asymmetric, see Identifying symmetric and asymmetric 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

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 the update changes the secret value, 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 the 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:Encrypt", "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 your secrets* (p. 90).

### 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": "AROAIGDTESTANDEXAMPLE: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-a1b2c3",
            "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": "AROAIGDTESTANDEXAMPLE: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-a1b2c3",
            "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 or later. We recommend TLS 1.2 or later. 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.

# 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.
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

# Troubleshoot AWS Secrets Manager

If you encounter issues when working with AWS Secrets Manager, consult the topics in this section.

**Topics**



# Troubleshoot general issues

Use the information here to help you diagnose and fix access-denied or other common issues that you might encounter when you're working with AWS Secrets Manager.

**Topics**



## I receive an "access denied" message when I send a request to AWS 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*.
- If you're signing API requests manually, without using the AWS SDKs, verify you correctly signed the request.

## I receive an "access denied" message when I send a request with 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*.

# Changes I make aren't always immediately visible.

As a service accessed through computers in data centers around the world, AWS 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 this, consult the following resources:

- 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*
-

# I receive a “cannot generate a data key with an asymmetric KMS key” message when creating a secret.

Verify you are using a symmetric KMS key instead of an asymmetric KMS key. Secrets Manager uses a symmetric KMS key 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. You cannot use an asymmetric KMS key at this time.

# An AWS CLI or AWS SDK operation can't find my secret from a partial ARN.

If your secret's name ends in a hyphen followed by six characters, Secrets Manager might not be able to find the secret from a partial ARN. Secrets Manager adds a hyphen and six random characters to ARNs, so if your secret ends in the same pattern, Secrets Manager assumes you are specifying a complete ARN. Instead, use the full ARN for `SecretId`.

For example, if your secret name is `MySecret-abcdef`, with the ARN `arn:aws:secretsmanager:us-east-2:111122223333:secret:MySecret-abcdef-nutBrk`, and you call the following operation, then Secrets Manager might not find the secret.

```
aws secretsmanager describe-secret --secret-id arn:aws:secretsmanager:us-
east-2:111122223333:secret:MySecret-abcdef
```

# 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.

**Common Rotation Errors**



## 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 can't predict when rotation will start

You can predict only the date of the next rotation, not the time.

Secrets Manager schedules the next rotation when the previous one is complete. Secrets Manager schedules the date by adding the rotation interval (number of days) to the actual date of the last rotation. The service chooses the hour within that 24-hour date window randomly. The minute is also chosen somewhat randomly, but is weighted towards the top of the hour and influenced by a variety of factors that help distribute load.

# 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. 124).

```
{
    "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. 77).

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. 74) 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. 122).
2. Examine the log files to look for indications that timeouts occurr 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. 123). 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"
        }
    ]
}
```

# Call the API by sending HTTPS query requests

This section contains general information about using the Query API for AWS Secrets Manager. For details about the API operations and errors, see the AWS Secrets Manager API Reference.

> **Note**
> Instead of sending direct calls to the AWS Secrets Manager Query API, you can use one of the AWS SDKs. The AWS SDKs consist of libraries and sample code for various programming languages and platforms such as Java, Ruby, .NET, iOS, Android, and more. The SDKs provide a convenient way to create programmatic access to Secrets Manager and AWS. For example, the SDKs perform tasks such as cryptographically signing requests, managing errors, and retrying requests automatically. For information about the AWS SDKs, including how to download and install the packages, see Tools for Amazon Web Services.

The Query API for AWS Secrets Manager lets you call service operations. Query API requests are HTTPS requests that must contain an `Action` parameter to indicate the operation to be performed. AWS Secrets Manager supports GET and POST requests for all operations. The API doesn't require you to use GET for some operations and POST for others. However, GET requests are subject to the limitation size of a URL. Although this limit depends on the browser , a typical limit is 2048 bytes. Therefore, for Query API requests that require larger sizes, you must use a POST request.

The API returns the response in an XML document. For details about the response, see the individual API description pages in the AWS Organizations API Reference.

**Topics**



## Endpoints

AWS Secrets Manager has endpoints in most AWS Regions. For the complete list, see the endpoint list for AWS Secrets Manager in the *AWS General Reference*.

For more information about AWS Regions and endpoints for all services, see Regions and Endpoints, also in the *AWS General Reference*.

## HTTPS required

Because the Query API returns sensitive information such as security credentials, you must use HTTPS to encrypt all API requests.

See the *AWS General Reference.*

# Sign API requests for Secrets Manager

You must sign API requests by using an access key ID and a secret access key. We strongly recommend you don't use your AWS account root user credentials for everyday work with Secrets Manager. Instead, you can use the credentials for an IAM user, or temporary credentials like those you use with an IAM role.

To sign your API requests, you must use AWS Signature Version 4. For information about using Signature Version 4, see Signature Version 4 Signing Process in the *AWS General Reference*.

For more information, see the following:

- AWS Security Credentials. Provides general information about the types of credentials you can use to access AWS.
- IAM Best Practices. Offers suggestions for using the IAM service to help secure your AWS resources, including those in Secrets Manager.
- Temporary Credentials. Describes how to create and use temporary security credentials.

# AWS Secrets Manager quotas

This section specifies quotas for AWS Secrets Manager.

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 | 40,000 |
| 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 |
| `CreateSecret` | 50 |
| `DeleteSecret` | 50 |

| Request type | Quota (per second) |
|---|---|
| `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.

# Document history for AWS Secrets Manager

The following table describes major documentation updates for AWS Secrets Manager.

- **API version: 2017-10-17**

| update-history-change | update-history-description | update-history-date |
| --- | --- | --- |
| Kubernetes CSI plugin for managing secrets (p. 131) | Added instructions for implementing Kubernetes plugin for secrets management. | April 22, 2021 |
| Multi-Region secrets support | Added support for AWS replicating secrets across regions to support cross region applications. | February 25, 2021 |
| Added three additional AWS Config Rules for Secrets Manager. | Three new rules to check for unused secrets, rotated secrets, and secrets with customer managed keys. | February 25, 2021 |
| Added information about Security Hub controls for Secrets Manager and security best practices. (p. 131) | Security Hub provides security controls to check for automatic rotation of secrets and successful rotation of secrets. | September 18, 2020 |
| Updated CloudFormation examples to use the Secrets Manager Transform. | When creating a Secrets Manager secret and database with roation using the CloudFormation template, you have the option of using `Transform: AWS::SecretsManager-2020-07-23` which allows you to create a hosted Lambda function. | July 23, 2020 |
| Enhanced search capabilities for secrets. | You can search for secrets using name, description, tag key, and tag value. Secrets Manager allows multiple filters for finding secrets. | July 9, 2020 |
| Added the ability to attach resource-based policies to secrets using the Secrets Manager console. | You can add, modify, and delete resource-based policies using the console. Also, Secrets Manager validates the policies. | July 9, 2020 |
| Changed the Rotate Secret tutorial to include a link to Amazon RDS. | To keep the tutorial steps up to date in the guide, a link to the Amazon RDS documentation | May 12, 2020 |

| | | |
|---|---|---|
| | replaced the steps to set up a test database. | |
| Added FedRAMP compliance for Secrets Manager. | Added FedRAMP logo and information on compliance with Secrets Manager. | May 12, 2020 |
| Added AWS Config with Secrets Manager and added more information on CloudFormation. (p. 131) | Added documentation for using AWS Config with Secrets Manager. | April 16, 2020 |
| Replaced CloudFormation templates with shorter and easier to use templates. (p. 131) | Templates now use only 60 lines of code to create CloudFormation configurations. | November 20, 2019 |
| Added documentation for endpoint policies | You can now use an endpoint policy to control secrets-related activity on your Secrets Manager VPC endpoint. Added section for creating an endpoint policy for Secrets Manager VPC endpoint. Also created a distinct reference article for all VPC endpoint content. | July 25, 2019 |
| Added Python, Go, and .NET caching clients | Added links to GitHub where you acquire the caching clients for Python, Go, and .NET. | May 9, 2019 |
| Added secret types for Amazon Redshift and Amazon DocumentDB | Added Amazon Redshift and Amazon DocumentDB databases to the secret types. | March 7, 2019 |
| Updated supported databases | Added the full list of supported databases on Amazon RDS for rotational support, including Microsoft SQL Server, Oracle and more. | December 2, 2018 |
| Compliance with PCI and ISO | Included the PCI and ISO standards in the compliance standards section. | December 1, 2018 |
| Use existing Lambda rotation functions with your secrets | When you enable rotation for a secret in the Secrets Manager console, you can now choose an existing Lambda function in addition to being able to create new functions. | November 15, 2018 |
| Tag your secrets using the Secrets Manager console | You can now include tags when create and modify your secrets using the Secrets Manager console. | November 15, 2018 |

| | | |
|---|---|---|
| Create secrets programmatically with CloudFormation | You can now create secrets by defining it in a CloudFormation template. If the secret is associated with one of the fully supported databases, then you can also generate the credentials dynamically during the processing of the template, configure the database to use those credentials and store them in a secret that is configured to automatically rotate. | November 12, 2018 |
| Delete a secret without a recovery window | You can now delete secrets without specifying a recovery window. This enables you to 'clean up' unneeded secrets without having to wait a minimum of seven days. | August 9, 2018 |
| Private VPC service endpoints | You can now configure private service endpoints for Secrets Manager within your VPCs. This enables you to call Secrets Manager API operations from within a VPC without requiring connection to the public internet. | July 11, 2018 |
| Resource-based policies | You can now attach IAM permission policies directly to a secret to determine who can access that secret. This also enables cross-account access because you can specify other AWS accounts in the `Principal` element of a resource-based policy. | June 26, 2018 |
| Compliance with HIPAA | Secrets Manager is now available as a HIPAA-eligible service. | June 4, 2018 |
| Initial release of service | Documentation provided for the initial release of AWS Secrets Manager. | April 4, 2018 |