AWS Secrets Manager

User Guide

AWS Secrets Manager: User Guide

Table of Contents

What Is AWS Secrets Manager?

AWS provides the service AWS Secrets Manager for easier management of secrets. *Secrets* can be database credentials, passwords, third-party API keys, and even arbitrary text. You can store and control access to these secrets centrally by using the Secrets Manager console, the Secrets Manager command line interface (CLI), or the Secrets Manager API and SDKs.

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 have chosen 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.

Getting Started with Secrets Manager

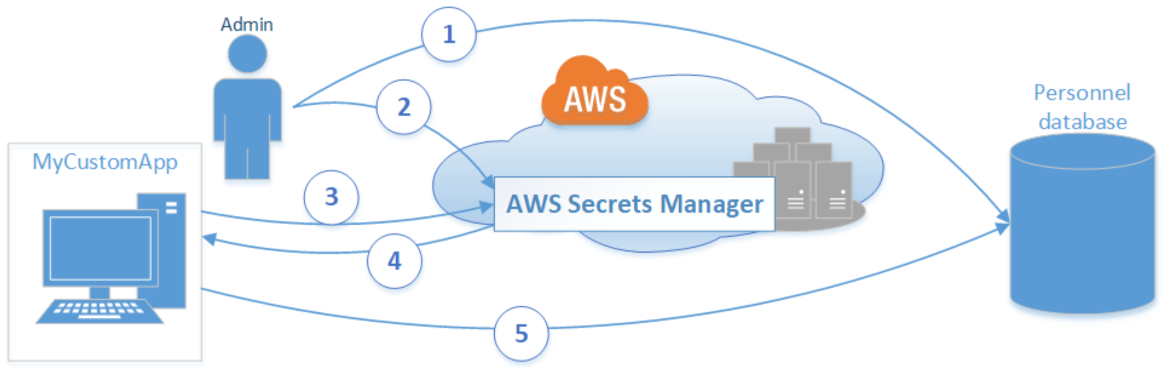
For a list of terms and concepts you need to understand to make full use of Secrets Manager, see Key Terms and Concepts for AWS Secrets Manager (p. 8).

Typical users of Secrets Manager can have one or more of the following roles:

- Secrets Manager administrator – Administers the Secrets Manager service. Grants permissions to individuals who can then perform the other roles listed here.
- Database or service administrator – Administers the database or service with secrets stored in Secrets Manager. Determines and configures the rotation and expiration settings for their secrets.
- Application developer – Creates the application, and then configures the application to request the appropriate credentials from Secrets Manager.

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. 4) 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 Rotating Your AWS Secrets Manager Secrets (p. 71).

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. It tracks these different versions by using *staging labels (p. 11)*.

Storing 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 AWS KMS customer master key (CMK) associated with the secret
- Any attached AWS tags

Encrypting 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 an AWS KMS CMK. It can be either the default CMK for Secrets Manager for the account, or a customer-created CMK.

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 specified CMK. 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 that use the 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 Rotating 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 Overview of the Lambda Rotation Function (p. 104).

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.

Accessing 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 `SecureBinary` 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. For the current complete pricing list, see AWS Secrets Manager Pricing.

AWS KMS – Custom Encryption Keys

If you create your own customer master keys by using AWS KMS to encrypt your secrets, AWS charges you at the current AWS KMS rate. However, you can use the "default" key created by AWS Secrets Manager for your account for free. For more information about the cost of customer-created AWS KMS keys, see AWS Key Management Service Pricing.

AWS CloudTrail Logging – Storage and Notification

If you enable AWS CloudTrail on your account, you can obtain logs of API calls AWS Secrets Manager sends out. Secrets Manager logs all events as management events. There are no data events. There's no additional charge for capturing a single trail in AWS CloudTrail to capture 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 the AWS CloudTrail pricing page.

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.

Getting Started with AWS Secrets Manager

To get started using AWS Secrets Manager, we recommend you review the following topics:

- Terms and Concepts (p. 8) - Basic concepts and Secrets Manager terms used throughout the APIs, SDK commands, and the console interface.
- Tutorials (p. 12) - Step-by-step procedures taking you through common scenarios to teach you how to perform basic tasks in Secrets Manager.

Key Terms and Concepts for AWS Secrets Manager

The following terms and concepts are important for understanding AWS Secrets Manager and how it works.

Secret

In Secrets Manager, a secret consists of a set of credentials, user name and password, and the connection details used to access a secured service (p. 10). You want to store these securely, and ensure only authorized users can access them. Secrets Manager always stores the secret text in an encrypted form and encrypts the secret in transit.

Secrets Manager uses IAM permission policies to ensure only authorized users can access or modify the secret. You can attach these policies to users or roles, and specify which secrets the users can access. For more details about controlling access to your secrets, see Authentication and Access Control for AWS Secrets Manager (p. 34).

When storing credentials, different secured services might require different pieces of information. Secrets Manager provides this flexibility by storing the secret as key-value pairs of text strings. If you choose a database supported by Secrets Manager , Secrets Manager defines the key-value pairs according to the requirements of the rotation function for the chosen database. Secrets Manager formats the pairs as JSON text. If you choose some other service or database Secrets Manager doesn't provide the Lambda function for, then you can specify your secret as a user-defined JSON key-value pairs.

The resulting stored encrypted secret text might then resemble the following example:

```
{
  "host" : "ProdServer-01.databases.example.com",
  "port" : "8888",
  "username"   : "administrator",
  "password"   : "My-P@ssw0rd!F0r+Th3_Acc0unt",
  "dbname"     : "MyDatabase",
  "engine"     : "mysql"
}
```

If you use the command-line tools or the API, you can also store binary data in the secret. The Secrets Manager console does not support binary data .

Secrets Manager can automatically rotate your secret for you on a specified schedule. You can rotate credentials without interrupting the service if you choose to store a complete set of credentials for a user or account, instead of only the password. If you change or rotate only the password, then the old password immediately becomes obsolete, and clients must immediately start using the new password or fail. If you can instead create a new user with a new password, or at least alternate between two users, then the old user and password can continue to operate side by side with the new one, until you choose to deprecate the old one. This gives you a window of time when all of your clients can continue to work while you test and validate the new credentials. After your new credentials pass testing, you commit all of your clients to using the new credentials and remove the old credentials.

Supported Databases

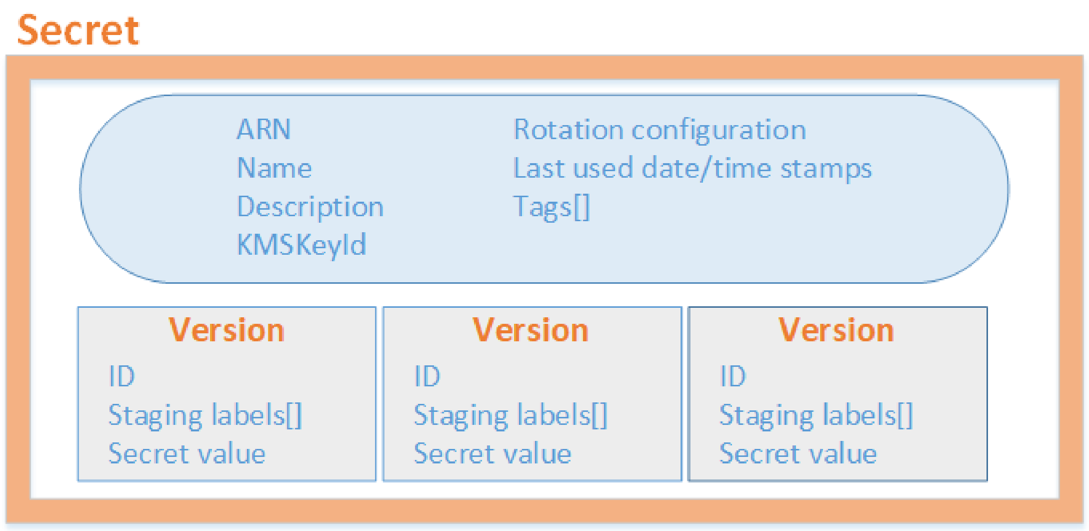
If you use the Secrets Manager console and specify a secret for one of the databases that Secrets Manager natively supports (p. 4), then Secrets Manager manages all of the structure and parsing for you. The console prompts you for the details for the specific type of database . Secrets Manager then constructs the necessary structure, stores the information, and then parses the information back into easy-to-understand text information when you retrieve it.

Other Databases or Services

If you instead specify the secret for a "custom" database or service, then you control what you do with the secret text after you retrieve it and how you interpret it . The Secrets Manager console accepts your secret as key-value strings, and automatically converts them into a JSON structure for storage. If you retrieve the secret in the console, Secrets Manager automatically parses the secret back into key-value text strings for you to view. If you retrieve the secret programmatically, then you can use an appropriate JSON parsing library, available for almost every programming language, to parse the secret in any way useful to you. If a secret requires more than per-secret limit of 65,536 bytes you could split your key-value pairs between two secrets and concatenate them back together when you retrieve them.

Basic Structure of a Secrets Manager Secret

In Secrets Manager, a secret contains not only the encrypted secret text, but also several metadata elements that describe the secret and define how Secrets Manager should handle the secret:



- **Metadata – Details about the secret**
  - Basic information includes the name of the secret, a description, and the Amazon Resource Name (ARN) to serve as a unique identifier.
  - The ARN of the AWS Key Management Service (AWS KMS) key Secrets Manager uses to encrypt and decrypt the protected text in the secret. If you don't provide this information, Secrets Manager uses the default AWS KMS key for the account.
  - Information about how frequently to rotate (p. 10) the key and what Lambda function to use to perform the rotation.

  - A user-provided set of tags. You can attach tags as key-value pairs to AWS resources for organizing, logical grouping, and cost allocation.
- **Versions – A collection of one or more versions (p. 11) of the encrypted secret text**
  - Although you typically only have one version of the secret active at a time, multiple versions can exist while you rotate a secret on the database or service. Whenever you change the secret, Secrets Manager creates a new version.
  - Each version holds a copy of the encrypted secret value.
  - Each version can have one or more staging labels (p. 11) attached identifying the stage of the secret rotation cycle.

Secured Service

Secrets Manager defines a secured service as a database or other service running on a network server, with access controlled by the credentials stored in the secret. The secured service can refer to a single server or a large group of servers sharing the same access method. You need the secret to successfully access the secured service. The secret contains all of the information a client needs to access the secured service. This guide uses the term "secured service" as a generic term to represent all of the different types of databases and services with secrets that can be protected by AWS Secrets Manager.

Rotation

Secrets Manager defines rotation as the process where you periodically change the secret to make it more difficult for an attacker to access the secured service. With Secrets Manager, you don't have to manually change the secret and update it on all of your clients. Instead, Secrets Manager uses an AWS Lambda function to perform all of the steps of rotation for you on a regular schedule.

Suppose you have a large set of clients all running an application accessing a database, the secured service (p. 10). Instead of hardcoding the credentials into your application, the application sends a request to Secrets Manager and receive the secret details whenever needed. When you rotate the secret, the Lambda rotation function automatically performs the following steps:

1. The rotation function contacts the secured service authentication system and creates a new set of credentials to access the database. The credentials typically consist of a user name, a password, and connection details, but can vary from system to system. Secrets Manager stores these new credentials as the secret text in a new version (p. 11) of the secret with the `AWSPENDING` staging label attached.
2. The rotation function then tests the `AWSPENDING` version of the secret to ensure that the credentials work, and grants the required level of access to the secured service.
3. If the tests succeed, the rotation function then moves the label `AWSCURRENT` to the new version to mark it as the default version. Then, all of the clients start using this version of the secret instead of the old version. The function also assigns the label `AWSPREVIOUS` to the old version, which marks it as the "last known good" version. The version with the `AWSPREVIOUS` staging label now has no label, and therefore deprecated.

You can trigger the Lambda rotation function manually when you choose **Rotate secret** in the console, or you can trigger it automatically every ***n*** days by specifying a rotation schedule. If you use one of the AWS databases that Secrets Manager natively supports (p. 4), then Secrets Manager provides a Lambda function to rotate the database credentials. This function performs basic rotation for you automatically or you can customize the function to support an advanced custom rotation strategy.

If you choose to create a secret for a custom service, then you must create the Lambda function . In the code of the function, you determine how to compose the JSON structure and parse it in your function.

For any service or database your secret uses, the Lambda rotation function for the secret must be able to access both your database or service and a Secrets Manager service endpoint. If the Lambda rotation function and database or service reside in a VPC provided by Amazon VPC, then you must configure the VPC with either a VPC service endpoint for Secrets Manager (p. 74), or direct Internet connectivity by using a NAT gateway, to allow access to the public Secrets Manager service endpoint.

For more information about rotation, see Rotating Your AWS Secrets Manager Secrets (p. 71).

Staging Labels

Secrets Manager uses staging labels, a simple text string, to enable you to identify different versions (p. 11) of a secret during rotation (p. 10). Whenever you query for the encrypted secret value, you can specify the version of the secret to retrieve. If you don't specify a version either by version ID or staging label, Secrets Manager defaults to the version with the staging label `AWSCURRENT` attached. Secrets Manager always attaches the staging label `AWSCURRENT` to one version of the secret. See the brief introduction to rotation (p. 10) for an example of how this works.

A version of a secret can have from 0 to 20 staging labels attached.

A staging label can be attached to only one version of a secret at a time. Two versions of the secret can't have the same staging label. When you attach a staging label to a version and a different version exists with the same label, you must also specify the version to remove the label, or Secrets Manager returns an error.

One version of the secret must **always** have the staging label `AWSCURRENT`, and the API operation enforces this behavior. The Lambda rotation functions provided by Secrets Manager automatically maintain the `AWSPENDING`, `AWSCURRENT`, and `AWSPREVIOUS` labels on the appropriate versions.

Versioning

Multiple versions of a secret exist to support rotation of a secret (p. 10). Secrets Manager distinguishes between different versions by the staging labels (p. 11). For most scenarios, you don't worry about versions of the secret. Secrets Manager and the provided Lambda rotation function manage these details for you. However, if you create a Lambda rotation function, your code must manage multiple versions of a secret and move the staging labels between versions appropriately. Versions also have a unique identifier (typically a UUID value) that always stays with the same version, unlike staging labels you can move between versions.

Configure your clients to always request for the default version of the secret with the `AWSCURRENT` label attached. Other versions can exist, but you only access other version by requesting a specific version ID or staging label. If you request the secret value and you don't specify either a version ID or a staging label, then by default you see the version with the staging label `AWSCURRENT`.

During rotation, Secrets Manager creates a new version of the secret and attaches the staging label `AWSPENDING`. The rotation function uses the `AWSPENDING` version to identify the version until after the version passes testing. After the rotation function verifies the new credentials work, Secrets Manager moves the label `AWSPREVIOUS` to the older version with `AWSCURRENT`, moves the label `AWSCURRENT` to the newer `AWSPENDING` version, and finally removes `AWSPENDING`.

For more information about how staging labels work to support rotation, see Rotating Your AWS Secrets Manager Secrets (p. 71).

Each version maintained for a secret contains the following elements:

- A unique ID for the version.
- A collection of staging labels used to identify the version, unique within the secret. Secrets Manager deprecates versions with no staging labels and these versions may be subject to deletion.

- The encrypted and stored secret text.

Whenever you query for the encrypted 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 defaults to the version with the staging label `AWSCURRENT` attached. Secrets Manager always attaches the staging label `AWSCURRENT` to one version of the secret.

AWS Secrets Manager Tutorials

Use the tutorials in this section to learn how to perform tasks using AWS Secrets Manager.

Tutorial: Creating and Retrieving a Secret (p. 12)

Get up and running with step-by-step instructions to create a secret and then retrieve it. This tutorial does not require a pre-configured AWS database.

Tutorial: Rotating a Secret for an AWS Database (p. 15)

Create a secret for an Amazon RDS MySQL database. Then rotate the secret used to access the database, and configure the secret to rotate on a schedule.

Tutorial: Rotating a User Secret with a Master Secret (p. 21)

Use the previous tutorial's secret as a *master* secret that can rotate a separate *user* secret for an Amazon RDS MySQL database. Then rotate the user secret by signing in as the master secret and alternating users.

Tutorial: Creating and Retrieving a Secret

In this tutorial, you create a secret and store it in AWS Secrets Manager. You then retrieve it in both the AWS Management Console and the AWS CLI. Since you simply create and store the secret, this tutorial doesn't require a database in conjunction with the secret.

Step 1: Create and Store Your Secret in AWS Secrets Manager (p. 12)

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

Step 2: Retrieving Your Secret from AWS Secrets Manager (p. 13)

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

Prerequisites

This tutorial assumes you have access to 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.

Step 1: Create and Store Your Secret in AWS Secrets Manager

In this step, you sign in as an IAM user and create a secret.

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**.
4. Under **Specify key/value pairs to be stored in the secret**, in the first field, type **`tutorials/MyFirstTutorialSecret`**. This stores your secret in the virtual folder **tutorials** with the value **MyFirstTutorialSecret**. You can leave the second field blank.
5. Choose **Plaintext** to see the JSON version of the secret text stored in the `SecretString` field of the secret.
6. For **Select the encryption key**, choose **DefaultEncryptionKey**. AWS KMS doesn't charge a fee if you use the default AWS managed key Secrets Manager creates in your account. If you choose to use a custom KMS key, then you can be charged at the standard AWS KMS rate.
7. Choose **Next**.
8. Under **Secret name**, type a name for the secret in the text field. You must use only alphanumeric characters and the characters /_+=.@-.
9. In the **Description** field, type a description of the secret.

   For **Description**, type, for example, **`The secret I created for the first tutorial.`**
10. In the **Tags** section, add desired tags in the **Key** and **Value - optional** text fields.

    For this tutorial, you can leave tags blank.
11. Choose **Next**.
12. In this tutorial, skip configuring rotation, so choose **Disable automatic rotation**, and then choose **Next**.
13. On the **Review** page, you can check your selected 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 has the code in different programming languages.
14. 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.

Step 1: Creating and Storing Your Secret in AWS Secrets Manager using the 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 each of the following commands.
3. **Creating your secret**

```
$ aws secretsmanager create-secret --name tutorials/MyFirstTutorialSecret
          --description "The secret I created for the first tutorial"
```

The output of the command displays the following information:

```
{
  "ARN": "&region-arn;secretsmanager:us-west-2:123456789012:secret:tutorials/
MyFirstTutorialSecret-a1b2c3",
  "Name": "tutorials/MyFirstTutorialSecret",
  }
```

Step 2: Retrieving 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 each of the following commands.

   To see all of the details of your secret except the encrypted text:

```
$ aws secretsmanager describe-secret --secret-id tutorials/MyFirstTutorialSecret
{
    "ARN": "&region-arn;secretsmanager:region:123456789012:secret:tutorials/
MyFirstTutorialSecret-jiObOV",
    "Name": "tutorials/MyFirstTutorialSecret",
    "Description": "My First 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 of the active versions of the secret and the staging labels attached to each version. In this tutorial, you should see one version ID (a UUID type value) maps to a single staging label `AWSCURRENT`.

   To see the encrypted text in your secret:

```
$ aws secretsmanager get-secret-value --secret-id tutorials/MyFirstTutorialSecret --
version-stage AWSCURRENT
{
    "ARN": "&region-arn;secretsmanager:region:123456789012:secret:tutorials/
MyFirstTutorialSecret-jiObOV",
    "Name": "tutorials/MyFirstTutorialSecret",
    "VersionId": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE",
    "SecretString": "{\"username\":\"myserviceusername\",\"password\":
\"MyVerySecureP@ssw0rd!\"}",
    "VersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": 1522680764.668
}
```

   You need to include the `--version-stage` parameter in the previous command only if you want details for a version with a different staging label than `AWSCURRENT`. Secrets Manager uses `AWSCURRENT` as the default value.

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

Summary

This tutorial demonstrated how easy you can create a simple secret, and to retrieve the secret value when you need the value. For another tutorial on creating a secret and configuring automatic rotation, see Tutorial: Rotating a Secret for an AWS Database (p. 15).

Tutorial: Rotating a Secret for an AWS Database

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

Step 1: Set Up a Test Database (p. 16)

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. 17)

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. 18)

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. 19)

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

Step 5: Verify Successful Rotation (p. 20)

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. 21)

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

Prerequisites

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

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

The tutorial also uses a Linux JSON parsing tool called **jq**. To download the tool, see jq on the GitHub website.

Important
For the sake of simplicity, this tutorial uses **jq** to parse the secret value into environment variables to allow for easy command line manipulation. This is NOT a security best practice for

a production environment. In a production environment, we recommend that you do not store passwords in environment variables.
Also, 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. To configure production servers securely, we recommend you follow the guidance in the Lambda and Amazon EC2 VPC documentation.

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 instead 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` enable a user to grant themselves any permissions, so grant these policies only to trusted users in an account.

Step 1: Set Up a Test Database

In this step, sign in to your account and set up a MySQL database in Amazon RDS.

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

   Refer to the Amazon RDS tutorial at 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 repeat the password in the Confirm password box. *Be sure to remember this password.* You need it when you create your secret in Step 2.
   - Set the **Public accessibility** to `Yes`.

     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 so that the Lambda rotation function can access the public Secrets Manager endpoint.
2. Choose **Launch DB instance**.

3. Choose **View DB instance summary**, and wait until the instance becomes active before proceeding with the next step. This step can take several minutes to complete.
4. When the **Summary** section **DB instance status** shows **Available**, refresh the page, and then scroll down to the **Connect** section.
5. In the **Connect** section, choose the name of the security group with the type **CIDR/IP - Inbound**. This opens the Amazon EC2 console for that security group.
6. At the bottom of the page, choose the **Inbound** tab, and then choose **Edit**.

   One existing rule enables access to the database from the current VPC. You need to enable access to the database from the public Internet so that your MySQL client tool and your hypothetical customers running applications can all access this database.

   > **Warning**
   > The following step allows Internet access your test database. Secrets Manager uses this configuration for this tutorial. For a secure setup, block public access to your database instance. Then, configure the Lambda function to run in the same VPC as your database instance. You must then 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. This enables the Lambda rotation function to access a Secrets Manager endpoint. When you've completed the tutorials on this database, we recommend you remove the rule you create in the next step.
7. Under **Source**, change it from **Custom** to **Anywhere**. Leave the Port Range at **3306**, the default port for MySQL.
8. Choose **Save**.

Step 2: Create Your Secret

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

To create your secret

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Ensure you set your console to the same region as you created the Amazon RDS MySQL database in the previous step.
3. Choose **Store a new secret**.
4. On the **Create Secret** page, in the **Select secret type** section, choose **Credentials for RDS database**.
5. For **User name**, type **`adminuser`** to match the name of the master user you previously provided in Step 1.3.
6. For **Password**, type the same password you provided for **adminuser** in Step 1.3.
7. For **Select the encryption key**, leave it set to **DefaultEncryptionKey**. If you use a custom master key (CMK) instead of the default for the account, then you can be charged for the use of the custom CMK.
8. For **Select which RDS database this secret will access**, choose the instance **MyTestDatabaseInstance** you created in Step 1.
9. Choose **Next**.
10. In the **Secret name and description** section, for **Secret name**, type **`MyTestDatabaseMasterSecret`**.
11. In the **Configure automatic rotation** section, disable rotation for now. Choose **Next**.
12. 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 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 be used to connect to the database. In this tutorial, we describe how to install a Linux-based MySQL client component. Then run a Linux command to retrieve the secret and connect to the database using the credentials configured in the secret.

If you run this tutorial on another platform, you might have to translate these commands into equivalents for your platform. 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.

1. Install a MySQL client. For example:

```
$ sudo yum install mysql
...
Installed:
  mysql.noarch 0:5.5-1.3.amzn1

Dependency Installed:
  mysql55.x86_64 0:5.5.24-1.24.amzn1                                      mysql55-
common.x86_64 0:5.5.24-1.24.amzn1

Complete!
$
```

2. Run commands to retrieve the secret and store them temporarily.

```
$ secret=$(aws secretsmanager get-secret-value --secret-id MyTestDatabaseMasterSecret |
 jq .SecretString | jq fromjson)
$ user=$(echo $secret | jq -r .username)
$ password=$(echo $secret | jq -r .password)
$ endpoint=$(echo $secret | jq -r .host)
$ port=$(echo $secret | jq -r .port)
```

   The first command retrieves the secret from Secrets Manager. Then the command retrieves the `SecretString` field out of the JSON response and stores it in an environment variable. Each of the following commands parses out one additional credential piece or connection detail, and stores it in a separate environment variable.

3. Run a command using the parsed details to access your database.

```
$ mysql -h $endpoint -u $user -P $port -p$password
Warning: Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 44
Server version: 5.6.39 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Troubleshooting tip:
If the `mysql` command fails to connect to the database, then 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.

4. At the `mysql>` prompt, run a command to verify your connection to the database by using the credentials from the secret. Ensure that the **Current user** in the output contains the user specified in your secret.

```
mysql> status;
--------------
/path/mysql_real  Ver 14.14 Distrib 5.6.39, for Linux (x86_64) using  EditLine wrapper

Connection id:          48
Current database:
Current user:           adminuser@192-168-1-1.example.com
SSL:                    Not in use
Current pager:          less
Using outfile:          ''
Using delimiter:        ;
Server version:         5.6.39 MySQL Community Server (GPL)
Protocol version:       10
Connection:
 mytestdatabaseinstance.randomcharacters.region.rds.amazonaws.com via TCP/IP
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:    utf8
Conn.  characterset:    utf8
TCP port:               3306
Uptime:                 2 hours 52 min 24 sec

Threads: 2  Questions: 15249  Slow queries: 0  Opens: 319  Flush tables: 1  Open
 tables: 80  Queries per second avg: 1.474
--------------
```

5. Close the connection with the following command:

```
mysql> quit
Bye
```

Step 4: Configure Rotation for Your Secret

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

1. In the Secrets Manager console, choose the secret **MyTestDatabaseMasterSecret**.
2. On the **Secret details** page, in the **Rotation configuration** section, choose **Edit rotation**.
3. On the **Edit rotation configuration** page, choose **Enable automatic rotation**.
4. For **Select rotation interval**, choose **30 days**.
5. Under **Select which secret will be used to perform the rotation**, choose **Use this secret**.
6. Choose **Save**. Secrets Manager begins to configure rotation for your secret, including creating the Lambda rotation function and attaching a role that enables Secrets Manager to invoke the function.

7. 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.

1. Run the commands to retrieve the secret and store them temporarily in environment variables.

```
$ secret=$(aws secretsmanager get-secret-value --secret-id MyTestDatabaseMasterSecret |
 jq .SecretString | jq fromjson)
$ user=$(echo $secret | jq -r .username)
$ password=$(echo $secret | jq -r .password)
$ endpoint=$(echo $secret | jq -r .host)
$ port=$(echo $secret | jq -r .port)
```

2. Using the same MySQL client you installed previously, run the command using the parsed details to access your database.

```
$ mysql -h $endpoint -u $user -P $port -p$password
Warning: Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 44
Server version: 5.6.39 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

3. At the `mysql>` prompt, run a command verifying your connection to the database by using the credentials from the secret.

```
mysql> status;
--------------
/path/mysql_real  Ver 14.14 Distrib 5.6.39, for Linux (x86_64) using  EditLine wrapper

Connection id:          48
Current database:
Current user:           adminuser@192-168-1-1.example.com
SSL:                    Not in use
Current pager:          less
Using outfile:          ''
Using delimiter:        ;
Server version:         5.6.39 MySQL Community Server (GPL)
Protocol version:       10
Connection:
 mytestdatabaseinstance.randomcharacters.region.rds.amazonaws.com via TCP/IP
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:    utf8
Conn.  characterset:    utf8
TCP port:               3306
Uptime:                 2 hours 52 min 24 sec
```

```
Threads: 2  Questions: 15249  Slow queries: 0  Opens: 319  Flush tables: 1  Open
 tables: 80  Queries per second avg: 1.474
--------------
```

4. Ensure that the **Current user** in the output matches the user specified in your secret.
5. Close the connection with the following command:

```
mysql> quit
Bye
```

Step 6: Clean Up

Important
If you intend to also perform the tutorial Tutorial: Rotating a User Secret with a Master Secret (p. 21), 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

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

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

To delete the database instance

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Instances**.
3. In the list of available instances, choose the **MyTestDatabaseInstance** instance you created for this tutorial.
4. Choose **Instance actions**, and then choose **Delete**.
5. On the **Delete DB Instance** page, in the **Options** section, for **Create final snapshot**, choose **No**.
6. Select the acknowledgement all of your data will be lost, and then choose **Delete**.

Tutorial: Rotating a User Secret with a Master Secret

This tutorial builds on the tasks you completed in the first tutorial: Tutorial: Rotating a Secret for an AWS Database (p. 15). 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: Creating a New User for Your Database and a User Secret (p. 22)

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. 23)

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. 24)

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. 25)

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. 26)

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 assumes 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: Rotating a Secret for an AWS Database (p. 15), without deleting the database and user as described in the final section. That provides you with the following:
  - An Amazon RDS MySQL database called **MyTestDatabase** running in an instance called **MyTestDatabaseInstance**.
  - A master user named **adminuser** with administrative permissions.
  - A secret named **MyTestDatabaseMasterSecret** with the credentials for **adminuser** stored in it.

Step 1: Creating 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 query 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 that needs to query information from the database. The user doesn't need anything other than read permissions.

a. Open a command prompt where you can run the AWS CLI and your MySQL client, as installed in the previous tutorial. Run the following commands to retrieve the master secret. Then use the secret to sign in to the MySQL server.

```
$ secret=$(aws secretsmanager get-secret-value --secret-id MyTestDatabaseMasterSecret |
 jq .SecretString | jq fromjson)
$ user=$(echo $secret | jq -r .username)
$ password=$(echo $secret | jq -r .password)
$ endpoint=$(echo $secret | jq -r .host)
$ port=$(echo $secret | jq -r .port)
$ mysql -h $endpoint -u $user -P $port -p$password
```

b. Now run the following commands at the `mysql>` prompt to create a new, restricted-permissions user. Replace the sample password with your password, and use it for the following steps.

```
mysql> CREATE USER mytestuser IDENTIFIED BY 'ReplaceThisWithASecurePassword';
Query OK, 0 rows affected (0.07 sec)
```

```
mysql> GRANT SELECT on *.* TO mytestuser;
Query OK, 0 rows affected (0.08 sec)
```

c. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
d. On the page with the list of secrets in your account, choose **Store a new secret**.
e. On the **Create Secret** page, in the **Select secret type** section, choose **Credentials for RDS database**.
f. For **User name**, type **`mytestuser`** to match the name of the user you created in Step 1b.
g. For **Password**, type the same password you provided for **mytestuser** in Step 1b.
h. For **Select the encryption key**, leave it set to **DefaultEncryptionKey**.
i. For **Select which RDS database this secret will access**, choose the instance **MyTestDatabaseInstance** you created in the previous tutorial.
j. Choose **Next**.
k. In the **Secret name and description** section, for **Secret name** type **`MyTestDatabaseUserSecret`**.
l. In the **Configure automatic rotation** section, leave rotation disabled for now. Choose **Next**.
m. 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 a Linux-based MySQL client component. Continue to use the MySQL client in this tutorial.

If you run this tutorial on another platform, you might have to translate these commands into equivalents for your platform. At the very least, 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. Run commands to retrieve the secret and store them temporarily.

```
$ secret=$(aws secretsmanager get-secret-value --secret-id MyTestDatabaseUserSecret |
 jq .SecretString | jq fromjson)
$ user=$(echo $secret | jq -r .username)
$ password=$(echo $secret | jq -r .password)
$ endpoint=$(echo $secret | jq -r .host)
$ port=$(echo $secret | jq -r .port)
```

   The first command retrieves the secret from Secrets Manager, and then retrieves the `SecretString` field from the JSON response and stores it in an environment variable. Each of the following commands parses out one additional credential piece or connection detail, and stores it in a separate environment variable.

b. Run a command using the parsed details to access your database.

```
$ mysql -h $endpoint -u $user -P $port -p$password
Warning: Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 44
Server version: 5.6.39 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

c. At the `mysql>` prompt, run a command verifying your connection to the database by using the credentials from the secret. Ensure the **Current user** in the output matches the user specified in your secret.

```
mysql> status;
--------------
/path/mysql_real  Ver 14.14 Distrib 5.6.39, for Linux (x86_64) using  EditLine wrapper

Connection id:          48
Current database:
Current user:           mytestuser@192-168-1-1.example.com
SSL:                    Not in use
Current pager:          less
Using outfile:          ''
Using delimiter:        ;
Server version:         5.6.39 MySQL Community Server (GPL)
Protocol version:       10
Connection:             mytestdatabaseinstance.randomcharacters.region.rds.amazonaws.com
 via TCP/IP
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:    utf8
Conn.  characterset:    utf8
TCP port:               3306
Uptime:                 2 hours 52 min 24 sec

Threads: 2  Questions: 15249  Slow queries: 0  Opens: 319  Flush tables: 1  Open tables:
 80  Queries per second avg: 1.474
--------------
```

d. Close the connection with the following command:

```
mysql> quit
Bye
```

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. Under **Select which secret will be used to perform the rotation**, choose **Use a secret that I have previously stored in AWS Secrets Manager**.
f. In the list of secrets that appears, choose **MyTestDatabaseMasterSecret**.

g. Choose **Save**.

   Rotation configuration begins, but the initial rotation fails because of one step that you must, for security purposes, perform yourself.

h. Copy the Amazon Resource Name (ARN) of the master secret from the message to your clipboard.

i. Choose the link on the word **role** in the message. This opens the role details page in the IAM console, for the role attached to the Lambda rotation function Secrets Manager created for you.

j. On the **Permissions** tab, choose **Add inline policy**, and then set the following values:

   - For **Service**, choose **Secrets Manager**.
   - For **Actions**, choose **GetSecretValue**.
   - For **Resources**, choose **Add ARN** next to the **secret** resource type entry.
   - In the **Add ARN(s)** dialog box, paste the ARN of the master secret you copied previously, and then choose **Add**.

k. Choose **Review policy**, and for **Name**, type **`AccessToMasterSecret`**.

l. Choose **Create policy**.

   Note
   As an alternative to using the Visual Editor as described in the previous steps, you can copy and paste the following statement into an existing or new policy:

```
{ "Effect": "Allow", "Action": [ "secretsmanager:GetSecretValue" ], "Resource":
 "<ARN of the master secret>" }
```

m. Return to the AWS Secrets Manager console.

n. On the **Secrets** list page, choose the name of your user secret.

o. 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. After it changes, move on to the next step.

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. Run the commands to retrieve the secret, and store them temporarily in environment variables.

```
$ secret=$(aws secretsmanager get-secret-value --secret-id MyTestDatabaseUserSecret |
 jq .SecretString | jq fromjson)
$ user=$(echo $secret | jq -r .username)
$ password=$(echo $secret | jq -r .password)
$ endpoint=$(echo $secret | jq -r .host)
$ port=$(echo $secret | jq -r .port)
```

b. Using the same MySQL client you installed previously, run the command that uses the parsed details to access your database.

```
$ mysql -h $endpoint -u $user -P $port -p$password
Warning: Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 44
Server version: 5.6.39 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
```

```
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

c. At the `mysql>` prompt, run a command that verifies your connection to the database by using the credentials from the secret. Check that the **Current user** in the output matches either the user specified in your first user secret, or the same user name suffixed with `_clone`. This shows the rotation has successfully cloned your original user and alternated it with a new version of the secret.

```
mysql> status;
--------------
/path/mysql_real  Ver 14.14 Distrib 5.6.39, for Linux (x86_64) using  EditLine wrapper

Connection id:          48
Current database:
Current user:           mytestuser_clone@192-168-1-1.example.com
SSL:                    Not in use
Current pager:          less
Using outfile:          ''
Using delimiter:        ;
Server version:         5.6.39 MySQL Community Server (GPL)
Protocol version:       10
Connection:             mytestdatabaseinstance.randomcharacters.region.rds.amazonaws.com
 via TCP/IP
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:    utf8
Conn.  characterset:    utf8
TCP port:               3306
Uptime:                 2 hours 52 min 24 sec

Threads: 2  Questions: 15249  Slow queries: 0  Opens: 319  Flush tables: 1  Open tables:
 80  Queries per second avg: 1.474
--------------
```

d. Close the connection with the following command:

```
mysql> quit
Bye
```

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**.

AWS Secrets Manager Best Practices

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

Topics

Protecting 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"
}
```

Improving Performance by Using the AWS provided Client-side Caching Components

To use your secrets most efficiently, you should not simply retrieve the secret value from Secrets Manager every time you need to use the credentials. Instead, use a supported Secrets Manager client component to cache your secrets and update them only when required because of rotation. AWS has created such client components for you and made them available as open-source. For more information, see Using the AWS-developed Open Source Client-side Caching Components (p. 64)

Mitigating the Risks of Logging and Debugging Your Lambda Function

When you create a custom Lambda rotation function to support your Secrets Manager secret, be careful about including debugging or logging statements in your function. These statements can cause information in your function to be written to CloudWatch. Ensure the logging of information to CloudWatch doesn't include any of the sensitive data stored in the encrypted secret value. Also, if you do choose to include any such statements in your code during development for testing and debugging purposes, make sure you remove those lines from the code before using the code in production. Also, remember to remove any logs including sensitive information collected during development after you no longer need it.

The Lambda functions provided by AWS and supported databases (p. 4) don't include logging and debug statements.

Mitigating 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 shll 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.
```

Cross-Account Access – Should I Specify a User/Role or the Account?

When you want to use a resource-based policy attached to a secret give access to an IAM principal in a different AWS account, you have two options:

- **Specify only the other account ID** – In the `Principal` element of the statement, you specify the Amazon Resource Name (ARN) of the "foreign" account root. This enables the administrator of the foreign account to delegate access to roles in the foreign account. The administrator must then assign IAM permission policies to the role or roles that require access to the secret.

```
"Principal": {"AWS": arn:aws:iam::AccountId:root}
```

- **Specify the exact user or role in the other account** – You specify an exact user or role ARN as the `Principal` of a secret-based policy. You get the ARN from the administrator of the other account. Only the specific single user or role in the account can access the resource.

```
"Principal": [
    {"AWS": "arn:aws:iam::AccountId:role/MyCrossAccountRole"},
    {"AWS": "arn:aws:iam::AccountId:user/MyUserName"}
]
```

As a best practice, we recommend you specify only the account in the secret-based policy, for the following reasons:

- In both cases, you trust the administrator of the other account. In the first case, you trust the administrator to ensure only authorized individuals have access to the IAM user, or can assume the specified role. This creates essentially the same level of trust as specifying only the account ID. You trust the account and the administrator. When you specify only the account ID, you give the administrator the flexibility to manage users.
- When you specify an exact role, IAM internally converts the role to a "principal ID" unique to the role. If you delete the role and recreate it with the same name, the role receives a new principal ID. This means the new role doesn't automatically obtain access to the resource. IAM provides this functionality for security reasons, and means that an accidental delete and restore can result in "broken" access.

When you grant permissions to only the account root, that set of permissions then becomes the limit of the options the administrator of the account can delegate to users and roles. The administrator can't grant a permission to the resource if you didn't grant it to the account first.

Important
If you choose to grant cross-account access directly to the secret without using a role, then the secret must be encrypted by using a custom AWS KMS customer master key (CMK). A principal from a different account must be granted permission to both the secret and the custom AWS KMS CMK.

Running Everything in a VPC

Whenever possible, you should 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 acccess, unless you explicitly configure it with access. For example, you could add a NAT gateway. For complete 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 Configuring Your Network to Support Rotating Secrets (p. 74).

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 managing, searching, and filtering 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.
  - **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. 54) or when you edit them (p. 58).

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

Rotating 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. As more users get access to a secret, it may be possible someone has mishandled and leaked it to an unauthorized entity. Secrets can be leaked through logs and cache data. They can be shared for debugging purposes and not changed or revoked once the debugging completes. For all these reasons, secrets should be rotated frequently.

Configure Secrets Manager to rotate your secrets frequently to avoid using the same secrets for your applications.

See Rotating Your AWS Secrets Manager Secrets (p. 71)

Auditing Access to Secrets

As a best practice, you should monitor your secrets for usage and log any changes to them. This helps you to ensure you can investigate any unexpected usage or change, and unwanted changes can be rolled back.

You can monitor access to secrets using AWS CloudTrail to log all activity for Secrets Manager. In addition, you should set automated checks for inappropriate usage of secrets.

See Monitoring the Use of Your AWS Secrets Manager Secrets (p. 128)

Monitoring Your Secrets with AWS Config

AWS Secrets Manager integrates with AWS Config and provides easier tracking of secret changes in Secrets Manager. AWS Config can help you define 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.

See Monitoring Secrets Manager Secrets Using AWS Config (p. 144)

Authentication and Access Control for AWS Secrets Manager

Access to AWS Secrets Manager requires AWS credentials. Those credentials must contain permission to access the AWS resources you want to access, such as your Secrets Manager secrets. The following sections provide details on how you can use AWS Identity and Access Management (IAM) policies to help secure access to your secrets and control who can access and administer them.

Access to secrets must be tightly controlled because the secrets contain extremely sensitive information . By using the permissions capabilities of AWS and IAM permission policies, you can control which users or services have access to your secrets. You can specify which API, CLI, and console operations the user can perform on the authorized secrets. By taking advantage of the granular access features in the IAM policy language, you can choose to limit the user to only a subset of your secrets—or even to one individual secret—by using tags as filters. You can also restrict a user to specific versions of a secret by using staging labels as filters.

You can also determine who can *manage* which secrets. You specify who updates or modifies the secrets and the associated metadata. If you have administrator permissions to the AWS Secrets Manager service, you can delegate access to Secrets Manager tasks by granting permissions to others.

You can attach permission policies to your users, groups, and roles, and specify the secrets the attached identities can access. Secrets Manager refers to this process as *identity-based policies*. Alternatively, you can attach a permission policy directly to the secret and specify access to it. Secrets Manager refers to this process a *resource-based policy*. Either way, those policies specify the actions each principal can perform on secrets.

For general information about IAM permissions policies, see Overview of IAM Policies in the *IAM User Guide*.

For the permissions available specifically for use with AWS Secrets Manager, see Actions, Resources, and Context Keys You Can Use in an IAM Policy or Secret Policy for AWS Secrets Manager (p. 161).

The following sections describe how to manage permissions for AWS Secrets Manager. We recommend you read the overview first.

- Overview of Managing Access Permissions to Your Secrets Manager Secrets (p. 34)
- Using Identity-based Policies (IAM Policies) for Secrets Manager (p. 42)
- Using Resource-based Policies for Secrets Manager (p. 48)

Overview of Managing Access Permissions to Your Secrets Manager Secrets

An AWS account owns all of AWS resources, including the secrets you store in AWS Secrets Manager. AWS contains permissions policies to create or access the resources.. An account administrator can control access to AWS resources by attaching permissions policies directly to the resources, the secrets, or to the IAM identities, users, groups, and roles, needing to access the resources.

Note
An *administrator*, or administrator user, has administrator permissions. This typically means they have permissions to perform all operations on all resources for a service. For more information, see IAM Best Practices in the *IAM User Guide*.

An account administrator for Secrets Manager can perform administrative tasks, including delegating administrator permissions to other IAM users or roles in the account. To do this, you attach an IAM permissions policy to an IAM user, group, or role. By default, a user has no permissions at all, which means the user has an *implicit deny*. The policy you attach overrides the implicit deny with an *explicit allow* specifying actions the user can perform, and on which resources they can perform the actions . The policy can be attached to users, groups, and roles within the account. If you grant the permissions to a role, the role can be assumed by users in other accounts in the organization.

Topics

Authentication

AWS defines authentication as the process of establishing an identity representing you in the services you access. Typically, an administrator assigns an identity to you, and you access the identity by providing credentials with your request, such as a user name and password—or by encrypting your request with an AWS access key.

AWS supports the following types of identities:

- **AWS account root user** – When you sign up for AWS, you provide an email address and password for your AWS account. AWS uses this information as the *credentials* for your account root user, and provides complete access to all of your AWS resources.

  Important
  For security reasons, we recommend you use the root user credentials only to create an administrator user, an IAM user with full permissions to your AWS account. Then you can use this administrator user to create other IAM users and roles with limited, specific permissions for a job or role. For more information, see Create Individual IAM Users (IAM Best Practices) and Creating An Admin User and Group in the *IAM User Guide*.
- **IAM user** – IAM user has an identity within your AWS account with specific permissions, for example, accessing a Secrets Manager secret. You can use an IAM user name and password to sign in to secure AWS web pages such as the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  In addition to a user name and password, you can also create access keys for each user to enable t access to AWS services programmatically, through one of the AWS SDKsor the command line tools. The SDKs and command line tools use the access keys to cryptographically sign API requests. If you don't use the AWS tools, you must sign API requests yourself. AWS KMS supports *Signature Version 4*, an AWS protocol for authenticating API requests. For more information about authenticating API requests, see Signature Version 4 Signing Process in the *AWS General Reference*.
- **IAM role** – You can create an IAM role in your account with specific permissions. Similar to an IAM user, the IAM role does not associate with a specific person. An IAM role enables you to obtain temporary access keys to access AWS services and resources programmatically. AWS roles become useful in the following situations:
  - **Federated user access** – Instead of creating an IAM user, you can use pre-existing user identities from AWS Directory Service, your enterprise user directory, or a web identity provider (IdP). These are known as *federated users*. Identity providers associate IAM roles with federated users.. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.

- **Cross-account access** – You can use an IAM role in your AWS account to allow another AWS account permission to access your account resources. For an example, see Tutorial: Delegate Access Across AWS Accounts Using IAM Roles in the *IAM User Guide*.
- **AWS service access** – You can use an IAM role in your account to grant an AWS service permission to access your account resources. For example, you can create a role that allows Amazon Redshift to access an S3 bucket on your behalf, and then load data stored in the S3 bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.
- **Applications running on EC2 instances** – Instead of storing access keys on an EC2 instance, for use by applications running on the instance and sending AWS API requests, you can use an IAM role to provide temporary access keys for these applications. To assign an IAM role to an EC2 instance, you create an *instance profile*, and then attach the profile when you launch the instance. An instance profile contains the IAM role, and enables applications running on the EC2 instance to obtain temporary access keys. For more information, see Using Roles for Applications on Amazon EC2 in the *IAM User Guide*.

After you sign in with an identity, you then use access control and authorization, to establish tasks you, through your identity, can perform.

Access Control and Authorization

You can have a valid identity with credentials to authenticate your requests, but you also need *permissions* to make Secrets Manager API requests to create, manage, or use Secrets Manager resources. For example, you must have permissions to create a secret, to manage the secret, to retrieve the secret, as well as other tasks. The following sections describe managing permissions for Secrets Manager.

Secrets Manager Policies: Combining Resources and Actions

This section discusses how Secrets Manager concepts map to the IAM equivalent concepts.

Policies

Secrets Manager grants permissions , as in almost all AWS services, by creating and attaching permission policies. Policies have two basic types:

- **Identity-based policies** – Attached directly to a user, group, or role. The policy specifies the allowed tasks for the attached identity. The attached user automatically and implicitly becomes the `Principal` of the policy. You can specify the `Actions` the identity can perform, and the `Resources` the identity can perform actions. The policies enable you to perform the following actions:
  - Grant access to multiple resources to share with the identity.
  - Control access to APIs for nonexistent resources, such as the various `Create*` operations.
  - Grant access to an IAM group to a resource.

- **Resource-based policies** – Attached directly to a resource—in this case, a secret. The policy specifies access to the secret and the actions the user performs on the secret. The attached secret automatically and implicitly becomes the `Resource` of the policy. You can specify the `Principals`accessing the secret and the `Actions` the principals can perform. The policies enable you to perform the following actions:
  - Grant access to multiple principals (users or roles) to a single secret. Note that you *can't* specify an IAM group as a principal in a resource-based policy. Only users and roles can be principals.
  - Grant access to users or roles in other AWS accounts by specifying the account IDs in the `Principal` element of a policy statement. If you require a "cross account" access to a secret , then this might be one of the primary reasons to use a resource-based policy.

Which type should you use? Although their functionality does overlap a great deal, scenarios exist where one has a distinct advantage over the other, and both should have a place in your security strategy. While either policy type can work, choose the type that simplifies your maintenance for the policies as people come and go from your organization.

Important
When an IAM principal in one account accesses a secret in a different account, the secret must be encrypted using a custom AWS KMS customer master key (CMK). A secret encrypted with the default Secrets Manager CMK for the account can be decrypted only by principals in that account. A principal from a different account must be granted permission to both the secret and to the custom AWS KMS CMK.
As an alternative, you can grant cross-account access to a user and assume an IAM role in the same account as the secret. Because the role exists in the same account as the secret, the role can access secrets encrypted with the default AWS KMS key for the account.

Resources

In Secrets Manager, you control access to your **secrets**,

Each secret has a unique Amazon Resource Name (ARN) associated with it. You can control access to a secret by specifying the ARN in the `Resource` element of an IAM permission policy. The ARN of a secret has the following format:

```
arn:aws:secretsmanager:<region>:<account-id>:secret:optional-path/secret-name-6-random-
characters
```

Understanding Resource Ownership

The AWS account owns the resources you create in the account, regardless of who created the resources. Specifically, the resource owner is the AWS account of the root user, IAM user, or IAM role with credentials to authenticate the resource creation request. The following examples illustrate how this works:

- If you sign in as the root user account to create a secret, your AWS account becomes the owner of the resource.
- If you create an IAM user in your account and grant the user permissions to create a secret, the user can create a secret. However, the AWS account of the user owns the secret.
- If you create an IAM role in your account with permissions to create a secret, anyone who can assume the role can create a secret. The AWS account of the role, not the assuming user, owns the secret.

Actions vs. Operations

Secrets Manager provides a set of *operations* (API calls and CLI commands) to work with secrets. The operations enable you to perform actions such as creating, listing, accessing, modifying, or deleting secrets. These operations correspond to policy *actions* you can use to grant or deny access to that operation. In most cases, a one-to-one relationship exists between API operations and the actions you can assign in a policy. To control access to an operation, specify the corresponding action in the `Action` element of an IAM policy. For a list of allowed Secrets Manager actions used in a policy, see Actions, Resources, and Context Keys You Can Use in an IAM Policy or Secret Policy for AWS Secrets Manager (p. 161).

- When you combine both an `Action` element and a `Resource` element in an *identity*-based permission policy `Statement`, then you control both the performed actions and the resources. The limits apply to the user, group, or role of the attached the policy. .
- When you combine both an `Action` element and a `Principal` element in a *resource*-based permission policy `Statement`, then you control both actions that can be performed—and the users,

groups, or roles (the principals) performing those actions. The limits apply to the secret with the attached policy.

When you want to grant permissions to *create new secrets*, use an identity-based policy attached to your users, groups, or roles. This technique can also be helpful when you want to manage multiple secrets together in a similar manner.

When you want to grant permissions to *access existing secrets*, you can use a resource-based policy attached to the secret.

Managing Access to Resources with Policies

A *permissions policy* describes *who* can perform *which actions* on *which resources*. The following section explains the available options for creating permissions policies, and also briefly describes the elements making up a policy, and the two types of policies you can create for Secrets Manager.

> **Note**
> This section discusses using IAM in the context of Secrets Manager, but does not provide detailed information about the IAM service. For complete IAM documentation, see the IAM User Guide. For information about IAM policy syntax and descriptions, see the AWS IAM Policy Reference in the *IAM User Guide*.

You should understand you use either secret-based or identity-based permission policies for Secrets Manager . Secrets Manager groups together all of the applied policies and processes as one large policy. This basic rule structure controls the interaction:

Explicit Deny >> Explicit Allow >> Implicit Deny (default)

For a request to perform an AWS operation on an AWS resource, the following rules apply:

- **If any statement in any policy with an explicit "deny" matches the request action and resource**: The explicit deny overrides everything else and blocks the specified actions on the specified resources .
- **If no explicit "deny", but a statement with an explicit "allow" matches the request action and resource**: The explicit allow applies, which grants actions in that statement access to the resources in that statement.
- **If no statement with an explicit "deny" and no statement with an explicit "allow" matches the request action and resource**: AWS *implicitly* denies the request by default.

You should understand that an *implicit* deny can be overridden with an explicit allow. An *explicit* deny can't be overridden.

Effective Permissions When Multiple Policies Apply

Policy A	Policy B	Effective Permissions
Allows	Silent	Access allowed
Allows	Allows	Access allowed
Allows	Denies	Access denied
Silent	Silent	Access denied
Silent	Allows	Access allowed
Silent	Denies	Access denied

Policy A	Policy B	Effective Permissions
Denies	Silent	Access denied
Denies	Allows	Access denied
Denies	Denies	Access denied

Policy A and Policy B can be of any type: an IAM identity-based policy attached to a user or role, or a resource-based policy attached to the secret. You can include the effect of additional policies by taking the effective permission of the first two policies (the results) and treating that like a new Policy A. Then add the results of the third policy as Policy B, and repeat this for each additional policy that applies.

Topics

AWS Managed Policies

AWS addresses many common use cases by providing standalone IAM policies created and administered by AWS. These *managed policies* grant necessary permissions for common use cases so you can avoid investigating the necessary permissions. For more information, see AWS Managed Policies in the *IAM User Guide*.

Secrets Manager has a specific AWS managed policy, which you can attach to users in your account:

- **SecretsManagerReadWrite**–This policy can be attached to IAM users and roles to administer Secrets Manager. The policy grants full permissions to the Secrets Manager service , and limited permissions to other services, such as AWS KMS, Amazon CloudFront, AWS Serverless Application Repository, and AWS Lambda.

  You can view this policy in the AWS Management Console.

  > **Important**
  > For security reasons, this managed policy does ***not*** include the IAM permissions needed to configure rotation. You must explicitly grant permissions separately. Typically, you attach the IAMFullAccess managed policy to a Secrets Manager administrator to enable them to configure rotation.

The Secrets Manager team maintains the AWS managed policy which can be an important advantage of using an AWS managed policy. If the rotation process expands to include new functionality that requires additional permissions, Secrets Manager adds those permissions to the managed policy at that time. Your rotation functions automatically receive the new permissions so that they continue to operate as expected without any interruption.

Specifying Policy Statement Elements

The following section contains a brief overview of IAM permission policies from the perspective of Secrets Manager. For more detail about IAM policy syntax, see the AWS IAM Policy Reference in the *IAM User Guide*.

Secrets Manager defines a set of API operations to interact with or manipulate a secret in some way. To grant permissions for these operations, Secrets Manager defines a set of corresponding actions (p. 161) you can specify in a policy. For example, Secrets Manager defines actions to work on a secret, such as `CreateSecret`, `GetSecretValue`, `ListSecrets`, and `RotateSecret`.

A policy document must have a `Version` element. We recommend always using the latest version to ensure you can use all of the available features. As of this writing, the only available version is `2012-10-17` (the latest version).

In addition, a secret policy document must have one `Statement` element with one or more statements in an array. Each statement can consist of up to six elements:

- **Sid** – (Optional) You can use the `Sid` as a statement identifier, an arbitrary string identifying the statement.
- **Effect** – (Required) Use this keyword to specify if the policy statement allows or denies the action on the resource. If you don't explicitly allow access to a resource, access is *implicitly* denied. You also can explicitly deny access to a resource. You might do this to ensure that a user can't perform the specified action on the specified resource, even if a different policy grants access. You should understand that multiple statements overlap, explicit deny in a statement overrides any other statements that explicitly allow. Explicit allow statements override the implicit deny present by default.
- **Action** – (Required) Use this keyword to identify the actions you want to allow or deny. These actions usually, but not always, correspond one-to-one with the available operations. For example, depending on the specified `Effect`, `secretsmanager:PutSecretValue` either allows or denies the user permissions to perform the Secrets Manager `PutSecretValue` operation.
- **Resource** – (Required) In an identity-based policy attached to a user, group, or role, you use this keyword to specify the Amazon Resource Name (ARN) of the resource for the applicable policy statement. If you don't want the statement to restrict access to a specific resource, then you can use "*", and the resulting statement restricts only actions. In a resource-based policy attached to a secret, the resource must *always* be "*".
- **Principal** – (Required in a resource-based policy only) For resource-based policies attached directly to a secret, you specify the user, role, account, service, or other entity you want to receive permissions. This element isn't valid in an identity-based policy. In identity-based policies, the user or role with the attached policy automatically and implicitly becomes the principal.
- **Condition** – (Optional) Use this keyword to specify additional conditions that must be true for the statement to "match" and the `Effect` to apply. For more information, see IAM JSON Policy Elements: Condition.

Identity-based Policies

You can attach policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – To grant a user permissions to create a secret, you can attach a permissions policy directly to the user or to a group the user belongs to (recommended).
- **Attach a permissions policy to a role** – You can attach a permissions policy to an IAM role to grant access to a secret to anyone assuming the role. This includes users in the following scenarios:
  - **Federated users** – You can grant permissions to users authenticated by an identity system other than IAM. For example, you can associate IAM roles to mobile app users signing in using Amazon Cognito. The role grants the app temporary credentials with the permissions in the role permission policy. Those permissions can include access to a secret. For more information, see What is Amazon Cognito? in the *Amazon Cognito Developer Guide*.
  - **Applications running on EC2 instances** – You can grant permissions to applications running on an Amazon EC2 instance by attaching an IAM role to the instance. When an application in the instance invokes an AWS API, the application can get AWS temporary credentials from the instance metadata. These temporary credentials associate with a role, and limited by the role permissions policy. Those permissions can include access to a secret.

- **Cross-account access** – An administrator in Account A can create a role to grant permissions to a user in a different Account B. For example:
  1. The Account A administrator creates an IAM role and attaches a permissions policy to the role. The policy grants access to secrets in account A and specifies the actions users perform on the secrets.
  2. The Account A administrator attaches a trust policy to the role that identifies the account ID of Account B in the `Principal` element to specify who assumes the role.
  3. The Account B administrator can then delegate permissions to any users in account B to enable them to assume account A role. Doing this allows users in account B to access the secrets in the first account.

For more information about using IAM to delegate permissions, see Access Management in the *IAM User Guide*.

The following example policy can be attached to a user, group, or role. The policy allows the affected user or role to perform the `DescribeSecret` and `GetSecretValue` operations in your account only on secrets with a name beginning with the path "TestEnv/". The policy also restricts the user or role to retrieving only the version of the secret with the staging label `AWSCURRENT` attached. Replace the *`<region>`* and *`<account_id>`* placeholders in the following examples with your actual values.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
           "Sid" : "Stmt1DescribeSecret",
           "Effect": "Allow",
           "Action": [ "secretsmanager:DescribeSecret" ],
           "Resource": "arn:aws:secretsmanager:<region>:<account_id>:secret:TestEnv/*"
        },
        {
            "Sid" : "Stmt2GetSecretValue",
            "Effect": "Allow",
            "Action": [ "secretsmanager:GetSecretValue" ],
            "Resource": "arn:aws:secretsmanager:<region>:<account_id>:secret:TestEnv/*",
            "Condition" : {
                "ForAnyValue:StringLike" : {
                    "secretsmanager:VersionStage" : "AWSCURRENT"
                }
            }
        }
    ]
}
```

Because a secret version can have multiple staging labels attached, you need to use the IAM policy language's "set operators" to compare them in a policy. In the previous example, `ForAnyValue:StringLike` states if any one of the staging labels attached to the secret version under evaluation matches the string "`AWSCURRENT`", then the statement matches, and applies the string `Effect`.

For more example identity-based policies, see Using Identity-based Policies (IAM Policies) for Secrets Manager (p. 42). For more information about users, groups, roles, and permissions, see Identities (Users, Groups, and Roles) in the *IAM User Guide*.

Resource-based Policies

Each Secrets Manager secret can have one resource-based permissions policy,a *secret policy*, attached to it. You can use a secret policy to grant cross-account permissions as an alternative to using identity-based policies with IAM roles. For example, you can grant permissions to a user in Account B to access your secret in Account A by adding permissions to the secret policy, and identifying the user in Account B as a `Principal`, instead of creating an IAM role.

The following example describes a Secrets Manager secret policy with one statement. The statement allows the administrator of account 123456789012 to delegate permissions to users and roles in that account. The policy limits the permissions the administrator can delegate to the `secretsmanager:GetSecretValue` action on a single secret named "prod/ServerA-a1b2c3". The condition ensures the user can retrieve only the current version of the secret.

```
{
    "Version" : "2012-10-17",
    "Statement" : [
        {
            "Effect": "Allow",
            "Principal": {"AWS": "arn:aws:iam::123456789012:root" },
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:<region>:<account_id>:secret:prod/ServerA-
a1b2c3",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "secretsmanager:VersionStage" : "AWSCURRENT"
                }
            }
        }
    ]
}
```

Important
When an IAM principal from one account accesses a secret in a different account, the secret must be encrypted by using a custom AWS KMS CMK. A secret encrypted with the default Secrets Manager CMK for the account can be decrypted only by principals in that account. A principal from a different account must be granted permission to both the secret and to the custom AWS KMS CMK.
As an alternative, you can grant the user cross-account access to assume an IAM role in the same account as the secret. Because the role exists in the same account as the secret, the role can access secrets encrypted with the default AWS KMS key for the account.

Because a secret version can have multiple staging labels attached, you need to use the IAM policy language's set operators to compare them in a policy. In the previous example, `ForAnyValue:StringLike` states that if any one of the labels attached to the version under evaluation matches "`AWSCURRENT`", then the statement matches, and applies the `Effect`.

For more example resource-based policies with Secrets Manager, see Using Resource-based Policies for Secrets Manager (p. 48). For additional information about using resource-based policies instead of IAM roles (identity-based policies), see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.

Using Identity-based Policies (IAM Policies) for Secrets Manager

As the administrator of an account, you can control access to the AWS resources in your account by attaching permissions policies to IAM identities such as users, groups, and roles. When granting permissions, you decide the permissions, the resources, and the specific actions to allow on those resources. If you grant permissions to a role, users in other accounts you specify can assume that role.

By default, a user or role has no permissions of any kind. You must explicitly grant any permissions by using a policy. If you don't explicitly grant permission, then the permission implicitly denies. If you explicitly deny a permission, then that overrules any other policy allowing it. In other words, a user has only permissions explicitly granted and not explicitly denied.

For an overview of the basic elements for policies, see Managing Access to Resources with Policies (p. 38).

Topics

AWS Managed Policy for Secrets Manager

AWS Secrets Manager provides the following AWS managed policy to allow granting permissions easier. Choose the link to view the policy in the IAM console.

- SecretsManagerReadWrite – This policy grants the access required to administer Secrets Manager, except the policy doesn't include the IAM permissions required to create roles and attach policies to those roles. For those permissions, attach the IAMFullAccess managed policy. For instructions, see the following section Granting Full Secrets Manager Administrator Permissions to a User (p. 43).

Granting Full Secrets Manager Administrator Permissions to a User

To be a Secrets Manager administrator, you must have permissions in several services. We recommended you don't enable Secrets Manager as an end-user service enabling your users to create and manage their secrets. The permissions required to enable rotation grant significant permissions standard users shouldn't have. Instead, trusted administrators manage the Secrets Manager service. The end user no longer handles the credentials directly or embed them in code.

> **Warning**
> When you enable rotation, Secrets Manager creates a Lambda function and attaches an IAM role to the function. This requires several IAM permissions granted only to trusted individuals. Therefore, the managed policy for Secrets Manager *does not* include these IAM permissions. Instead, you must explicitly choose to assign the `IAMFullAccess` managed policy, in addition to the `SecretsManagerReadWrite` managed policy to create a full Secrets Manager administrator.
> Granting access with only the `SecretsManagerReadWrite` policy enables an IAM user to create and manage secrets, but the user can't create and configure the Lambda rotation functions required to enable rotation.

Complete the following steps to grant full Secrets Manager administrator permissions to an IAM user, group, or role in your account. In this example, you don't create a new policy. Instead, you attach an AWS managed policy preconfigured with the permissions.

To grant full administrator permissions to an IAM user, group, or role

1. Sign in to the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/ as a user with permissions to attach IAM policies to other IAM users.

In the IAM console, navigate to **Policies**.

2. For **Filter Policies: Policy type**, choose **AWS managed**, and then in the **Search** box, start typing **`SecretsManagerReadWrite`** until you can see the policy in the list.
3. Choose the **SecretsManagerReadWrite** policy name.
4. Choose the **Policy actions** , and then choose **Attach**.
5. Check the box next to the users, groups, or roles you want to add as a Secrets Manager administrator.
6. Choose **Attach policy**.
7. Repeat steps 1 through 6 to also attach the **IAMFullAccess** policy.

The selected users, groups, and roles can immediately begin performing tasks in Secrets Manager.

Granting Read Access to One Secret

When you write an application to use Secrets Manager to retrieve and use a secret, you only need to grant the application a very limited set of permissions. Secrets Manager only requires the permission for the action that allows retrieval of the encrypted secret value with the credentials. You specify the Amazon Resource Name (ARN) of the secret to restrict access to only the one secret.

Open the IAM console at https://console.aws.amazon.com/iam/.

Choose **Create Policy** and add the following code:

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "secretsmanager:GetSecretValue",
        "Resource": "<arn-of-the-secret-the-app-needs-to-access>"
    }
}
```

For a list of all the permissions available to assign in an IAM policy, see Actions, Resources, and Context Keys You Can Use in an IAM Policy or Secret Policy for AWS Secrets Manager (p. 161).

Limiting Access to Specific Actions

If you want to grant limited permissions instead of full permissions, you can create a policy that lists individual permissions you want to allow in the `Action` element of the IAM permissions policy. As shown in the following example, you can use wildcard (*) characters to grant only the `Describe*`, `Get*`, and `List*` permissions, which essentially provides read-only access to your secrets.

Open the IAM console at https://console.aws.amazon.com/iam/.

Choose **Create Policy** and add the following code:

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

For a list of all the permissions available to assign in an IAM policy, see Actions, Resources, and Context Keys You Can Use in an IAM Policy or Secret Policy for AWS Secrets Manager (p. 161).

Limiting Access to Specific Secrets

In addition to restricting access to specific actions, you also can restrict access to specific secrets in your account. The `Resource` elements in the previous examples all specify the wildcard character ("*"), which means "any resource this action can interact". Instead, you can replace the "*" with the ARN of specific secrets you want to allow access.

Example Example: Granting permissions to a single secret by name

The first statement of the following policy grants the user read access to the metadata about all of the secrets in the account. However, the second statement allows the user to perform any Secrets Manager actions on only a single, specified secret by name—or on any secret beginning with the string "`another_secret_name-`" followed by exactly 6 characters:

Open the IAM console at https://console.aws.amazon.com/iam/.

Choose **Create Policy** and add the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:List*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "secretsmanager:*",
            "Resource": [
                "arn:aws:secretsmanager:<region>:<account-id-
number>:secret:a_specific_secret_name-a1b2c3",
                "arn:aws:secretsmanager:<region>:<account-id-
number>:secret:another_secret_name-??????"
            ]
        }
    ]
}
```

Using '??????' as a wildcard to match the 6 random characters assigned by Secrets Manager avoids a problem that occurs if you use the '*' wildcard instead. If you use the syntax "`another_secret_name-*`", Secrets Manager matches not just the intended secret with the 6 random characters, but also matches "`another_secret_name-`*`<anything-here>`*`a1b2c3`". Using the '??????' syntax enables you to securely grant permissions to a secret that doesn't yet exist. This happens because you can predict all of the parts of the ARN except those 6 random characters. 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.

You get the ARN for the secret from the Secrets Manager console (on the **Details** page for a secret), or by calling the `List*` APIs. The user or group you apply this policy to can perform any action (`"secretsmanager:*"`) on only the two secrets identified by the ARN in the example.

If you don't care about the region or account that owns a secret, you must specify a wildcard character * , not an empty field, for the region and account ID number fields of the ARN.

For more information about the ARNs for various resources, see Resources You Can Reference in an IAM Policy or Secret Policy (p. 163).

Limiting Access to Secrets with Specific Staging Labels or Tags

In any of the previous examples, you called out actions, resources, and principals explicitly by name or ARN only. You can also refine access to include only those secrets with metadata contains a certain tag key and value, or a secret with a specific label.

Example: Granting permission to a secret containing metadata with a certain tag key and value

The following policy, when attached to a user, group, or role, allows the user to use `DescribeSecret` on any secret in the current account with metadata containing a tag with the key "ServerName" and the value "ServerABC".

Open the IAM console at https://console.aws.amazon.com/iam/.

Choose **Create Policy** and add the following code:

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

Example: Granting permission to the version of the secret with a certain staging label

The following policy, when attached to a user, group, or role, allows the user to use `GetSecretValue` on any secret with a name that begins with `Prod`—and only for the version with the staging label `AWSCURRENT` attached.

Open the IAM console at https://console.aws.amazon.com/iam/.

Choose **Create Policy** and add the following code:

```
{
    "Policy": {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": "secretsmanager:GetSecret",
                "Resource": "arn:aws:secretsmanager:*:*:secret:Prod*",
                "Condition": { "ForAnyValue:StringEquals": { "secretsmanager:VersionStage":
 "AWSCURRENT" } }
```

```
            }
        ]
    }
}
```

Because a version of a secret can have multiple staging labels attached, you need to use the IAM policy language's set operators to compare them. In the previous example, `ForAnyValue:StringLike` states if any one of the staging labels attached to the version under evaluation matches the string `AWSCURRENT`, then the statement matches, and applies the `Effect`.

Granting a Rotation Function Permission to Access a Separate Master Secret

When you create a Lambda rotation function using the provided AWS Serverless Application Repository template, either by using the console or by using AWS CLI commands, a default policy attached to the function role controls the function actions. By default, this policy grants access *only* to secrets with this Lambda function configured as the secret rotation function.

If the credentials in the secret don't allow users permission to change their password on the secured database or service, then you need to use a separate set of credentials with elevated permissions (a *superuser*) to change this secret credentials during rotation. Secrets Manager stores the superuser credentials in a separate "master" secret. Then, when you rotate your secret, the Lambda rotation function signs in to the database or service with the master credentials to change or update the secret credentials. If you choose to implement this strategy, then you must add an additional statement to the role policy attached to the function that grants access to this master secret, in addition to the main secret.

If you use the console to configure a rotation with a strategy using a master secret, then you can select the master secret when you configure rotation for the secret.

Important
You must have the **GetSecretValue** permission for the master secret to select it in the console.

After you configure rotation in the console, you must manually perform the following steps to grant the Lambda function access to the master secret.

To grant a Lambda rotation function access to a master secret

Follow the steps in one of the following tabs:

Using the console

1. When you finish creating a secret with rotation enabled or editing your secret to enable rotation, the console displays a message similar to the following:

```
Your secret MyNewSecret has been successfully stored [and secret rotation is
 enabled].
To finish configuring rotation, you need to grant the role MyLambdaFunctionRole
 permission to retrieve the secret <ARN of master secret>.
```

2. Copy the ARN of the master secret in the message to your clipboard. You use it in a later step.
3. The role name in the preceding message provides a link to the IAM console, and navigates directly to the role for you. Choose that link.
4. On the **Permissions** tab, there can be one or two inline policies. Secrets Manager names one as `SecretsManager`*`<Name of Template>`*`0`. The template contains the EC2-related permissions required when both the rotation function and you run your secured service in a VPC, and not directly accessible from the internet. The other template named `SecretsManager`*`<Name of Template>`*`1`.contains the permissions to enable the rotation function to call Secrets Manager

operations. Open the policy (the one ending with "1") by choosing the expand arrow to the left of the policy and examining the permissions.

5. Choose **Edit policy**, and then perform the steps in one of the following tabs:

   Using the IAM Visual Editor

   On the **Visual editor** tab, choose **Add additional permissions**, and then set the following values:

   - For **Service**, choose **Secrets Manager**.
   - For **Actions**, choose **GetSecretValue**.
   - For **Resources**, choose **Add ARN** next to the **secret** resource type entry.
   - In the **Add ARN(s)** dialog box, paste the ARN of the master secret you copied previously.

   Using the JSON editor

   On the **JSON** tab, examine the top of the script. Between lines 3 (`"Statement": [`) and line 4 (`{`), enter the following lines:

```
        {
            "Action": "secretsmanager:GetSecretValue",
            "Resource":
"arn:aws:secretsmanager:region:123456789012:secret:MyDatabaseMasterSecret",
            "Effect": "Allow"
        },
```

6. After you finish editing the policy, choose **Review policy**, and then **Save changes**.
7. You can now close the IAM console and return to the Secrets Manager console.

Using Resource-based Policies for Secrets Manager

You control access to secrets in AWS Secrets Manager using secret resource-based policies. AWS defines a *secret* as a *resource* in Secrets Manager. You, as the account administrator, control access to a resource in an AWS service. . You can add permissions to the policy attached to a secret. Secrets Manager refers to permissions policies attached directly to secrets as *resource-based policies*. You can use resource-based policies to manage secret access and management permissions.

A resource-based policy has an advantage over identity-based policies because a resource-based policy enables you to grant access to principals from different accounts. See the second example in the following section.

For an overview of the basic elements for policies, see Managing Access to Resources with Policies (p. 38).

For information about alternative, identity-based permission policies, see Using Identity-based Policies (IAM Policies) for Secrets Manager (p. 42).

Topics

Controlling Access to Secrets for Principals

When you use a resource-based policy with Secrets Manager attached directly to a secret, the `Resource` automatically and implicitly becomes the secret with the attached policy.. You can now specify the `Principal` element. The `Principal` element enables you to specify the IAM users, groups, roles, accounts, or AWS service principals can access this secret, and the actions they can perform on this secret. For more information about all the different ways to specify a `Principal`, see Principal in the *IAM User Guide*.

Example: Granting permission to a user in the same account as the secret

The following policy, when attached directly to a secret as part of the metadata, grants the user Anaya permission to run any Secrets Manager operation on the secret..

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:*",
            "Principal": {"AWS": "arn:aws:iam::123456789012:user/anaya"},
            "Resource": "*"
        }
    ]
}
```

Example: Granting permission to authorized users in a different account

The following policy, when attached directly to a secret as part of the metadata, grants the IAM user Mateo in the account 123456789012 access to read any version of a specific secret:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {"AWS": "arn:aws:iam::123456789012:user/mateo" },
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "*"
        }
    ]
}
```

Granting Read-Only Access to a Role

A common Secrets Manager scenario may be an application running on an Amazon EC2 instance and needs access to a database to perform required tasks. The application must retrieve the database credentials from Secrets Manager. To send a request to Secrets Manager, like any other AWS service, you must have AWS credentials with permissions to perform the request. You can achieve this by creating an IAM role attached to the EC2 instance profile. For more information, see IAM Roles for Amazon EC2 in the Amazon EC2 User Guide for Linux Instances and specifically the section Retrieving Security Credentials from Instance Metadata.

If you then attach the following example resource-based policy to the secret, any requests to retrieve the secret work only if the requester uses credentials associated with the role, and if the request asks only for the current version of the secret:

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::123456789012:role/EC2RoleToAccessSecrets"},
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "secretsmanager:VersionStage" : "AWSCURRENT"
        }
      }
    }
  ]
}
```

Determining Access to a Secret

To determine the full extent of users with access to a secret in AWS Secrets Manager, you must examine the secret policy, if one exists, and all AWS Identity and Access Management (IAM) policies attached to either the IAM user and the groups, or the IAM role. You might do this to determine the scope of potential usage of a secret, or to help you meet compliance or auditing requirements. The following topics help you generate a complete list of the AWS principals and identities with current access to a secret.

Topics

Understanding Policy Evaluation

When authorizing access to a secret, Secrets Manager evaluates the secret policy attached to the secret, and all IAM 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*.

For example, assume you have two secrets and three users, all in the same AWS account. The secrets and users have the following policies:

- Secret 1 doesn't have a secret policy.
- Secret 2 has a secret policy that allows access to the user principals Ana and Carlos.
- Ana has no IAM policy that references Secret 1 or Secret 2.
- Bob's IAM policy allows all Secrets Manager actions for all secrets.
- Carlos' IAM policy denies all Secrets Manager actions for all secrets.

Secrets Manager determines the following access:

- Ana can access only Secret 2. She doesn't have a policy attached to her user, but Secret 2 explicitly grants her access.
- Bob can access both Secret 1 and Secret 2 because Bob has an IAM policy that allows access to all secrets in the account, and no explicit "deny" override access..
- Carlos can't access Secret 1 or Secret 2 because Secrets Manager denies all actions in his IAM policy. The explicit deny in Carlos' IAM policy overrides the explicit "allow" in the secret policy attached to Secret 2.

Secrets Manager adds all policy statements from either the secret or the identity that "allow" access. Any explicit deny overrides any allow to the overlapping action and resource.

Examining the Resource Policy

You can view the secret policy document attached to a secret by using the `GetResourcePolicy` operation, and examining the `ResourcePolicy` response element.

Examine the resource policy document and take note of all principals specified in each policy statement `Principal` element. IAM users, IAM roles, and AWS accounts in the `Principal`elements all have access to this secret.

Example Policy Statement 1

```
{
    "Sid": "Allow users or roles in account 123456789012 who are delegated access by that
 account's administrator to have read access to the secret",
    "Effect": "Allow",
    "Principal":  {"AWS": "arn:aws:iam::123456789012:root"},
    "Action": [
        "secretsmanager:List*",
        "secretsmanager:Describe*",
        "secretsmanager:Get*"
    ],
    "Resource": "*"
}
```

In the preceding policy statement, `&region-arn;iam::111122223333:root` refers to the AWS account 111122223333 and allows the administrator of that account to grant access to any users or roles in the account.

You must also examine all IAM policies in all AWS accounts listed as principals to determine if the policies allow access to this secret.

Example Policy Statement 2

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:*",
            "Principal": {"AWS": "arn:aws:iam::123456789012:user/anaya"},
            "Resource": "*"
        }
    ]
}
```

In the preceding policy statement, `&region-arn;iam::111122223333:user/anaya` refers to the IAM user named Anaya in AWS account 111122223333. This user can perform all Secrets Manager actions.

Example Policy Statement 3

```
{
    "Sid": "Allow an app associated with an &IAM; role to only read the current version of
 a secret",
    "Effect": "Allow",
    "Principal":  {"AWS": "arn:aws:iam::123456789012:role/EncryptionApp" },
    "Action": ["secretsmanager:GetSecretValue"],
    "Condition": { "ForAnyValue:StringEquals": {"secretsmanager:VersionStage":
 "AWSCURRENT" } },
    "Resource": "*"
}
```

In the preceding policy statement, `&region-arn;iam::123456789012:role/EncryptionApp` refers to the IAM role named EncryptionApp in AWS account 123456789012. Principals assuming this role can perform the one action listed in the policy statement. This action obtain the details for the current version of the secret.

To learn all the different ways you can specify a principal in a secret policy document, see Specifying a Principal in the *IAM User Guide*.

To learn more about Secrets Manager secret policies, see Resource-based Policies (p. 41).

Examining IAM Policies

In addition to the secret policy, you can also use IAM policies in combination with the secret policy to allow access to a secret. For more information about how IAM policies and secret policies work together, see Understanding Policy Evaluation (p. 50).

To determine the identities with current access to a secret through IAM policies, you can use the browser-based IAM Policy Simulator tool, or you can send requests to the IAM policy simulator API.

Examining the effect of policies

Follow the steps under one of the following tabs:

Using the web-based Policy Simulator

You can use the IAM Policy Simulator to help you learn which principals have access to a secret through an IAM policy.

1. Sign in to the AWS Management Console, and then open the IAM Policy Simulator at https://policysim.aws.amazon.com/.
2. In the **Users, Groups, and Roles** pane, choose the user, group, or role with policies you want to simulate.
3. (Optional) Clear the check box next to any policies you want to omit from the simulation. To simulate all policies, leave all policies selected.
4. In the **Policy Simulator** pane, do the following:
   a. For **Select service**, choose **AWS Secrets Manager**.
   b. To simulate specific Secrets Manager actions, for **Select actions**, choose the actions to simulate. To simulate all Secrets Manager actions, choose **Select All**.
5. (Optional) The Policy Simulator simulates access to all secrets by default. To simulate access to a specific secret, select **Simulation Settings**, and then type the Amazon Resource Name (ARN) of the secret to simulate.
6. Select **Run Simulation**.

You can view the results of the simulation in the **Results** section. Repeat steps 2 through 6 for every IAM user, group, and role in the AWS account.

Using the Policy Simulator CLI or SDK operations

You can use the IAM API to examine IAM policies programmatically. The following steps provide a general overview of how to do this:

1. For each AWS account listed as a principal in the secret policy, that is, each *root user account* listed in this format: `"Principal": {"AWS": "arn:aws:iam::`*`accountnumber`*`:root"}`, use the ListUsers and ListRoles operations in the IAM API to retrieve a list of every IAM user and role in the account.

2. For each IAM user and role in the list, use the SimulatePrincipalPolicy operation in the IAM API, and send the following parameters:
   - For `PolicySourceArn`, specify the ARN of a user or role from your list. You can specify only one `PolicySourceArn` for each `SimulatePrincipalPolicy` API request. So you must call this API multiple times, once for each IAM user and role in your list.
   - For the `ActionNames` list, specify every Secrets Manager API action to simulate. To simulate all Secrets Manager API actions, use `secretsmanager:*`. To test individual Secrets Manager API actions, precede each API action with "`secretsmanager:`"—for example, "`secretsmanager:ListSecrets`". For a complete list of all Secrets Manager API actions, see Actions in the *AWS Key Management Service API Reference*.
   - (Optional) To determine if the IAM users or roles have access to specific secrets, use the `ResourceArns` parameter to specify a list of the ARNs of the secrets. To determine whether the IAM users or roles have access to any secret, don't use the `ResourceArns` parameter.

IAM responds to each `SimulatePrincipalPolicy` API request with an evaluation decision: `allowed`, `explicitDeny`, or `implicitDeny`. For each response that contains an evaluation decision of `allowed`, the response also contains the name of the specific Secrets Manager API action allowed and, if applicable, the ARN of the secret used in the evaluation.

Creating and Managing Secrets with AWS Secrets Manager

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

Topics

Creating a Basic Secret

AWS Secrets Manager enables you to store basic secrets with a minimum of effort. A basic secret consists of a minimum of metadata and a single encrypted secret value. Secrets Manager stores the secret and automatically labels it with `AWSCURRENT`.

Creating a basic secret

Follow the steps under one of the following tabs:

Using the Secrets Manager console

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

1. Sign in to the AWS Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Choose **Store a new secret**.

3. In the **Select secret type** section, specify the type of secret you want to create by choosing one of the following options. Then supply the required information.
4. For **Secret name**, type an optional path and name, such as **`production/MyAwesomeAppSecret`** or **`development/TestSecret`**. Notice that the use of the slash character enables you to structure your secrets into a hierarchy, such as grouping by the deployment environment, which you might find useful for organizing and managing your secrets at scale. You can optionally add a description to help you remember the purpose of this secret.

   The secret name must be ASCII letters, digits, or any of the following characters: /_+=.@-

   Note
   If you add a secret to the Systems Manager Parameter Store, you must add a forward slash to the directory structure. For more information, see AWS Systems Manager documentation on Organizing Parameters into Hierarchies.
5. (Optional) In the **Tags** section, you can add one or more tags to your secret. A tag consists of a key and a value you define. Tags assist with managing your AWS resources. You can create tags to associate resources with your organization's structure, such as Key="Department" and Value="Accounting". This can help with cost allocation and tracking. You assign tags to group resources together by the application using them (Key="AppName" and Value="HRDatabase"). You create tags for almost any purpose. Each resource, like a secret, can have several tags attached. For more information, see AWS Tagging Strategies on the *AWS Answers* website.

   Important
   Do not store sensitive information about a secret in the tags. Store sensitive information only in the secret value (the `SecretString` or `SecretBinary` fields) of the secret where encryption protects the information..
6. After you complete the **Name**, **Description**, and any **Tags**, choose **Next**.
7. (Optional) At this point, you can configure rotation for your secret. Because you created a "basic" secret without rotation, leave the option as **Disable automatic rotation**, and then choose **Next**.

   For information about configuring rotation on new or existing secrets, see Rotating Your AWS Secrets Manager Secrets (p. 71).
8. Review your settings, and then choose **Store secret** to save everything you entered as a new secret in Secrets Manager.

Amazon RDS

Use this type of secret for one of the supported database services (p. 4) Secrets Manager provides full rotation support with a preconfigured Lambda rotation function. You specify only the authentication credentials because Secrets Manager determines other parameters by querying the database instance.

1. Type the user name and password to allow access to the database. Choose a user with only the permissions required by the customer accessing this secret.
2. Choose the AWS KMS encryption key you want to use to encrypt the protected text in the secret. If you don't choose one, Secrets Manager checks for a default key for the account, and uses the default key if it exists. If a default key does not exist, Secrets Manager creates one for you automatically. You can also choose **Add new key** to create a custom CMK specifically for this secret. To create your own AWS KMS CMK, you must have permission to create CMKs in your account.
3. Choose the database instance from the list. Secrets Manager retrieves the connection details about the database by querying the chosen instance.

Amazon Redshift

Use this type of secret for an Amazon Redshift cluster. You only specify the authentication credentials because Secrets Manager determines other parameters by querying the database instance.

1. Type the user name and password to allow access to the database.
2. Choose the AWS KMS encryption key you want to use to encrypt the protected text in the secret. If you don't choose one, Secrets Manager checks for a default key for the account, and uses the default key if it exists. If a default key does not exist, Secrets Manager creates one for you automatically. You can also choose **Add new key** to create a custom CMK specifically for this secret. To create your own AWS KMS CMK, you must have permissions to create CMKs in your account.
3. Choose the correct database engine.
4. Specify the connection details by typing the database server IP address, database name, and TCP port number.

DocumentDB database

Use this type of secret for a DocumentDB database. You only specify the authentication credentials because Secrets Manager determines other parameters by querying the database instance.

1. Type the user name and password to allow access to the database.
2. Choose the AWS KMS encryption key you want to use to encrypt the protected text in the secret. If you don't choose one, Secrets Manager checks for a default key for the account, and uses it if it exists. If a default key doesn't exist, Secrets Manager creates one for you automatically. You can also choose **Add new key** to create a custom CMK specifically for this secret. To create your own AWS KMS CMK, you must have permissions to create CMKs in your account.
3. Choose the correct database engine.
4. Specify the connection details by typing the database server IP address, database name, and TCP port number.

Other databases

Secrets Manager supports other types of databases and uses this secret for those types. However, you must provide additional information about the database. To rotate this secret, you must write a custom Lambda rotation function to parse the secret and interact with the service to rotate the secret .

1. Type the user name and password to allow access to the database.
2. Choose the AWS KMS encryption key you want to use to encrypt the protected text in the secret. If you don't choose one, Secrets Manager checks for a default key for the account, and uses it if it exists. If a default key doesn't exist, Secrets Manager creates one for you automatically. You can also choose **Add new key** to create a custom CMK specifically for this secret. To create your own AWS KMS CMK, you must have permissions to create CMKs in your account.
3. Choose the type of database engine to run your database.
4. Specify the connection details by typing the database server IP address, database name, and TCP port number.

Other type of secret

Secrets Manager can be configured for other databases or services and uses this secret for them. You must supply the structure and details of your secret. To rotate this secret, you must write a custom

Lambda rotation function parse the secret and interact with the service to rotate the secret on your behalf.

1. Specify the details of your custom secret as **Key** and **Value** pairs. For example, you can specify a key of UserName, and then supply the appropriate user name as the value. Add a second key with the name of Password and the password text as the value. You could also add entries for `Database name`, `Server address`, `TCP port`, and so on. You can add as many pairs as you need to store the information you require.

   Alternatively, you can choose the **Plaintext** tab and enter the secret value in any format.
2. Choose the AWS KMS encryption key you want to use to encrypt the protected text in the secret. If you don't choose one, Secrets Manager checks for a default key in the account, and uses it if it exists. If a default key doesn't exist, Secrets Manager creates one for you automatically. You can also choose **Add new key** to create a custom CMK specifically for this secret. To create your own AWS KMS CMK, you must have permissions to create CMKs in your account.

Using the AWS CLI or AWS SDK operations

You use the following commands to create a basic secret in Secrets Manager:

- **API/SDK:** `CreateSecret`
- **AWS CLI:** `create-secret`

Example

An example of an AWS CLI command to perform the equivalent of the console-based secret configuration. This command assumes you've placed your secret, such as this example JSON text structure `{"username":"anika","password":"aDM4N3*!8TT"}`, in a file named `mycreds.json`.

```
$ aws secretsmanager create-secret --name production/MyAwesomeAppSecret --secret-string
 file://mycreds.json
{
    "SecretARN": "arn:aws:secretsmanager:region:accountid:secret:production/
MyAwesomeAppSecret-AbCdEf",
    "SecretName": "production/MyAwesomeAppSecret",
    "SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE"
}
```

Important
You can create a basic secret using any desired format for `SecretString`. For example, you could use a simple JSON key-value pair, or , `{"username":"someuser", "password":"securepassword"}` However, if you want to enable rotation for this secret later, then you must use the specific structure expected by the rotation function used with this secret. For the details of each required rotation function to work with the secret value, see the **Expected SecretString Value** entry under the relevant rotation function at AWS Templates You Can Use to Create Lambda Rotation Functions (p. 152).

Secrets Manager does not require The `ClientRequestToken` parameter because you use the AWS CLI, which automatically generates and supplies one for you. When you use the default Secrets Manager CMK for the account, you don't need the `KmsKeyId` parameter . When you use the Secrets Manager console and the `SecretString` , you can't use `SecretBinary`. Secrets Manager reserves the `SecretType` for use by the console.

In a working environment, where your customers use an application that uses the secret to access a database, you might still need to grant permissions to the IAM user or role the application uses to access the secret. You can do this by attaching a resource-based policy directly to the secret, and listing the user or role in the `Principal` element. Or you can attach a policy to the user or role that identifies the secret in the `Resource` element.

Modifying a Secret

You can modify some elements of a secret after you create it. In the console, you can edit the description, edit or attach a new resource-based policy to modify permissions to the secret, change the AWS KMS customer master key (CMK) used to encrypt and decrypt the protected secret information, and edit or add or remove tags.

You can also change the value of the encrypted secret information. However, we recommend you use rotation to update secret values that contain credentials. Rotation doesn't just update the secret. Rotation also modifies the credentials on the protected database or service to match those in the secret. This keep the secrets automatically synchronized so when clients request a secret value, they always retrieve a working set of credentials.

This section includes procedures and commands describing how to modify the following elements of a secret:

- Encrypted secret value (p. 58)
- Description (p. 60)
- AWS KMS encryption key (p. 61)
- Tags (p. 61)
- Permission policy (p. 68)

Modifying the encrypted secret value stored in a secret

Follow the steps under one of the following tabs:

Important

- Updating the secret in this manner doesn't change the credentials on the protected server. If you want the credentials on the server to stay in sync with the credentials stored in the secret value, we recommend you enable rotation. An AWS Lambda function changes both the credentials on the server and those in the secret to match, and tests that the updated credentials work. For more information, see Rotating Your AWS Secrets Manager Secrets (p. 71).
- You can create a basic secret using a desired format for the `SecretString`. For example, you could use a simple JSON key-value pair. For example, `{"username":"someuser", "password":"securepassword"}` However, if you later want to enable rotation for this secret, you must use a specific structure for the secret. Secrets Manager determines the exact format by the rotation function you want to use with this secret. For the details of what each rotation function requires to work with the secret value, see the **Expected SecretString Value** entry under the relevant rotation function at Rotating AWS Secrets Manager Secrets for Other Databases or Services.

Using the Secrets Manager console

When you update the encrypted secret value in a secret, you create a *new version* of the secret. The new version automatically receives the staging label `AWSCURRENT`. You can still access the old version by querying for the staging label AWSPREVIOUS. From the CLI, use the `get-version-ids` command or use the API GETVERSIONIDS to compare the modified secret to the original secret.

The output for the two commands displays a `CreatedDate` field, a valid parameter since Secrets Manager creates a new version for the modified secret.

Note
Any time the staging label AWSCURRENT moves from one version to another, Secrets Manager automatically moves the staging label AWSPREVIOUS to the version previously labeled AWSCURRENT.

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. From the list of secrets, choose the name of the secret with the secret value to modify.
3. In the **Secret value** section, choose **Retrieve secret value**.
4. With the secret value now displayed, choose `Edit`.
5. Update the values as appropriate, and then choose **Save**.

Using the AWS CLI or AWS SDK operations

You can use the following commands to update the encrypted secret value stored in the secret. When you update the encrypted secret value in a secret, you create a new version of the secret.

Important
You can update a basic secret using a desired format for `SecretString`. For example, you could use a simple JSON key-value pair. For example, `{"username":"someuser", "password":"securepassword"}` However, if you later want to enable rotation for this secret then you must use the specific structure expected by the rotation function you use with this secret. For the details of what each rotation function requires to work with the secret value, see the **Expected SecretString Value** entry under the relevant rotation function at Rotating AWS Secrets Manager Secrets for Other Databases or Services..

Note
`UpdateSecret` automatically moves the staging label AWSCURRENT to the new version of the secret.
`PutSecretValue` *does not* automatically move staging labels. However, the command does add AWSCURRENT if this command creates the first version of the secret. Otherwise, the command only attaches or moves those labels you explicitly request with the `VersionStages` parameter.
Any time the staging label AWSCURRENT moves from one version to another, Secrets Manager automatically moves the staging label AWSPREVIOUS to the version AWSCURRENT and removes the staging label AWSPREVIOUS.

- **API/SDK:** `UpdateSecret`, `PutSecretValue`
- **AWS CLI:** `update-secret`, `put-secret-value`

Example

The following example AWS CLI command changes the secret value for a secret. This results in the creation of a new version. Secrets Manager automatically moves the AWSCURRENT staging label to the new version. Also, Secrets Manager automatically moves the AWSPREVIOUS staging label to the older version with the staging label AWSCURRENT .

```
$ aws secretsmanager update-secret --secret-id production/MyAwesomeAppSecret --secret-
string '{"username":"anika","password":"a different password"}'
{
    "SecretARN": "arn:aws:secretsmanager:us-west-2:123456789012:secret:production/
MyAwesomeAppSecret-AbCdEf",
    "SecretName": "production/MyAwesomeAppSecret",
    "VersionId": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE"
}
```

The `VersionId` in the output contains the unique secret version ID for the new version of the secret. You can manually provide the value by using the `ClientRequestToken` parameter. If you don't specify the value, the SDK or AWS CLI generates a random UUID value for you.

Modifying the description of a secret

Follow the steps under one of the following tabs:

Using the console

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. From the list of secrets, choose the name of the secret to modify.
3. In the **Secrets details** section, choose **Actions**, and then choose **Edit description**.
4. Enter a new description or edit the existing text, and then choose **Save**.

Using the AWS CLI or AWS SDK operations

You can use the following commands to modify the description of a secret in AWS Secrets Manager:

- **API/SDK:** `UpdateSecret`
- **AWS CLI:** `update-secret`

Example

The following example AWS CLI command adds or replaces the description with the one provided by 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:region:accountid:secret:production/
MyAwesomeAppSecret-AbCdEf",
    "Name": "production/MyAwesomeAppSecret",
    "VersionId": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE"
}
```

Example

To view the changes to the secret, use the command `describe-secret`:

```
aws secretsmanager describe-secret --secret-id production/MyAwesomAppSecret
        {
          "ARN": "arn:aws:secretsmanager:region:accountid:secret:production/
MyAwesomeAppSecret-AbCdEf",
          "Name": "production/MyAwesomeAppSecret",
          "Description": "This is the description I want to attach to the secret",
          "LastChangedDate": 1579542853.77,
          "LastAccessedDate": 1579478400.0,
```

Modifying the AWS KMS encryption key used by a secret

Follow the steps under one of the following tabs:

Using the Secrets Manager console

Important
If you change the encryption key used by a secret, you must update the secret value with `UpdateSecret` or `PutSecretValue` at least once before you disable or delete the first CMK. Updating the secret value decrypts the secret using the old CMK and re-encrypts the secret with the new CMK. If you disable or delete the first CMK before this update, Secrets Manager cannot decrypt the key and you lose the contents of the secret unless you can re-enable the CMK.

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. In the list of secrets, choose the name of the secret to modify.
3. In the **Secrets details** section, choose **Actions**, and then choose **Edit encryption key**.
4. Choose the AWS KMS encryption key to encrypt and decrypt later versions of your secret. Then choose **Save**.

Using the AWS CLI or AWS SDK operations

Important
If you change the encryption key used by a secret, you must update the secret value with `UpdateSecret` or `PutSecretValue` at least once before you disable or delete the first CMK. Updating the secret value decrypts the secret using the old CMK and re-encrypts the secret with the new CMK. If you disable or delete the first CMK before this update, Secrets Manager cannot decrypt the key and you lose the contents of the secret unless you can re-enable the CMK.

You can use the following commands to modify the AWS KMS encryption key used by the secret. You must specify the CMK by the Amazon Resource Name (ARN).

- **API/SDK:** `UpdateSecret`
- **AWS CLI:** `update-secret`

Example

The following example AWS CLI command adds or replaces the AWS KMS CMK used for all encryption and decryption operations in this secret from this time on.

```
$ aws secretsmanager update-secret --secret-id production/MyAwesomeAppSecret --kms-key-
id arn:aws:kms:region:123456789012:key/EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE
```

Modifying the tags attached to a secret

Follow the steps under one of the following tabs:

Using the console

Tag key names and values are case sensitive. Only one tag on a secret can have a given key name.

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. In the list of secrets, choose the name of the secret to modify.
3. In the **Tags** section, choose **Edit**.
4. Secrets Manager displays any existing tags. You can type over any of the key names or values.
5. You can remove any existing row by choosing **Remove tag**.
6. If you need to add another key-value pair, choose **Add tag**, and then enter your new key name and the associated value.
7. When you finish making changes, choose **Save**.

Using the AWS CLI or AWS SDK operations

You can use the following commands to add or remove the tags attached to a secret in Secrets Manager. Key names and values are case sensitive. Only one tag on a secret can have a given key name. To edit an existing tag, add a tag with the same key name but with a different value. That doesn't add a new key-value pair. Instead, Secrets Manager updates the value in the existing pair. To change a key name, you must remove the first key and add a second with the new name.

- **API/SDK:** `TagResource`, `UntagResource`
- **AWS CLI:** `tag-resource`, `untag-resource`

Example

The following example AWS CLI command adds or replaces the tags with those provided by the `--tags` parameter. 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"},{"Key":"environment","Value":"production"}]'
```

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

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.

Retrieving the Secret Value

Secrets Manager enables you to programmatically and securely retrieve your secrets in your custom applications. However, you can also retrieve your secrets by using the console or the CLI tools.

This section includes procedures and commands describing how to retrieve the secret value of a secret.

Retrieving a secret value

Follow the steps on one of the following tabs:

Using the Secrets Manager console

Minimum permissions
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 custom AWS KMS customer master key (CMK) to encrypt your secret.

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. From the list of secrets in your account, choose the name of the secret to view.

   The **Secret details** page appears. The page displays all of the chosen secret configuration details except for the encrypted secret text.
3. In the **Secret value** section, choose **Retrieve secret value**.
4. Choose **Secret key/value** to see the credentials parsed out as individual keys and values. Choose **Plaintext** to see the JSON text string encrypted and stored.

Using the AWS CLI or AWS SDK operations

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

- **API/SDK:** `GetSecretValue`
- **AWS CLI:** `get-secret-value`

You must identify the secret by the friendly name or Amazon Resource Name (ARN), and specify the version of the secret. 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` and, if you stored any binary data in the secret, `Plaintext`, which returns a byte array.

Example

The following example of the AWS CLI command decrypts and retrieves the encrypted secret information from the default version of the secret named "MyTestDatabase". Secrets Manager uses *the last modified date* for the `CreatedDate` output.

```
$ aws secretsmanager get-secret-value --secret-id development/MyTestDatabase
{
    "ARN": "arn:aws:secretsmanager:region:accountid:secret:development/
MyTestDatabase-AbCdEf",
    "Name": "development/MyTestDatabase",
    "VersionId": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE",
    "SecretString": "{\"ServerName\":\"MyDBServer\",\"UserName\":\"Anaya\",\"Password
\":\"MyT0pSecretP@ssw0rd\"}",
    "SecretVersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": 1510089380.309
}
```

Using the AWS-developed Open Source Client-side Caching Components

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

Deleting and Restoring 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. By using the CLI (p. 66), you can delete a secret without a recovery window.

You also can't *directly* delete a version of a secret. Instead, you remove all staging labels from the secret. This marks the secret as deprecated, and enables Secrets Manager to automatically delete the version in the background.

This section includes procedures and commands describing deleting a and restoring a deleted secret:

- Delete a secret and all of the versions (p. 65)
- Restore a secret scheduled for deletion (p. 66)
- Delete a version of a secret (p. 68)

Deleting a secret and all of the versions

Follow the steps under one of the following tabs:

Using the Secrets Manager console

When you delete a secret, Secrets Manager immediately deprecates it. However, Secrets Manager doesn't actually delete the secret until the number of days specified in the recovery window has elapsed. You can't access a deprecated secret. If you need to access a secret scheduled for deletion, you must restore the secret. Then you can access the secret and the encrypted secret information.

You can permanently delete a secret without any recovery window, by using the AWS CLI or AWS SDKs. However, you can't do this in the Secrets Manager console.

Minimum permissions

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

- `secretsmanager:ListSecrets` – Used to navigate to the secret you want to delete.
- `secretsmanager:DeleteSecret` – Used to deprecate the secret and schedule it for permanent deletion.

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Navigate to the list of secrets you currently manage in Secrets Manager, and choose the name of the secret you to delete.
3. In the **Secret details** section, choose **Delete secret**.
4. In the **Schedule secret deletion** dialog box, specify the number of days in the recovery window. This represents 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**.
6. If you enabled the option to show deleted items in the console, then the secret continues to display. You can optionally choose to view the **Deleted date** field in the list.

   a. Choose the **Preferences** icon (the gear ⚙) in the upper-right corner.
   b. Choose **Show secrets scheduled for deletion**.
   c. Enable the switch for **Deleted on**.
   d. Choose **Save**.

   Your deleted secrets now appear in the list, with the date you deleted each one.

Using the AWS CLI or AWS SDK operations

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

- **API/SDK:** `DeleteSecret`
- **AWS CLI:** `delete-secret`

You have to identify the secret to delete by the friendly name or Amazon Resource Name (ARN) in the `SecretId` field.

Example

The following example of the AWS CLI command deprecates the secret named "MyTestDatabase" and schedules the secret for deletion after a recovery window of 14 days.

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

At any time after the date and time specified in the `DeletionDate` field, AWS Secrets Manager permanently deletes the secret.

You can also delete a secret immediately with no waiting time.

Important
A secret deleted with the `ForceDeleteWithoutRecovery` parameter cannot be recovered with the `RestoreSecret` operation.

Example

The following example of the AWS CLI command immediately deletes the secret without a recovery window. The `DeletionDate` response field shows the current date and time instead of a future time.

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

Handy tip
If you don't know 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 Monitoring Secret Versions Scheduled for Deletion (p. 131).

Restoring a secret scheduled for deletion

Follow the steps under one of the following tabs:

Using the Secrets Manager console

Secrets Manager considers a secret scheduled for deletion deprecated and no longer directly accessed. 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.

Minimum permissions

To restore a secret and the metadata in the console, you must have these permissions:

- `secretsmanager:ListSecrets` – Use to navigate to the secret you want to restore.
- `secretsmanager:RestoreSecret` – Use to delete any versions still associated with the secret.

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Navigate to the list of secrets you currently manage in Secrets Manager.
3. To view deleted secrets, you must enable this ability in the console. If you haven't enabled this feature, perform these steps:
4. a. Choose the **Preferences** icon, the gear ⚙, in the upper-right corner.
   b. Choose **Show secrets scheduled for deletion**.
   c. Enable the switch for **Deleted on**.
   d. Choose **Save**.

   Your deleted secrets now appear in the list with the date you deleted each one.
5. Choose the name of the deleted secret to restore.
6. In the **Secret details** section, choose **Cancel deletion**.
7. In the **Cancel secret deletion** confirmation dialog box, choose **Cancel deletion**.

   AWS Secrets Manager removes the `DeletionDate` field from the secret. This cancels the scheduled deletion and restores access to the secret.

Using the AWS CLI or AWS SDK operations

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

- **API/SDK:** `RestoreSecret`
- **AWS CLI:** `restore-secret`

You must identify the secret you want to restore by the friendly name or ARN in the `SecretId` field.

Example

The following example of the AWS CLI command 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:region:accountid:secret:development/
MyTestDatabase-AbCdEf",
    "Name": "development/MyTestDatabase"
}
```

Deleting one version of a secret

Follow the steps under one of the following tabs:

Using the AWS CLI or AWS SDK operations

You can't delete one version of a secret using the Secrets Manager console. You must use the AWS CLI or AWS API.

You can't directly delete a version of a secret. Instead, you remove all of the staging labels, which effectively marks the secret as deprecated. Secrets Manager can then delete it in the background.

You can use the following commands to deprecate a version of a secret stored in AWS Secrets Manager:

- **API/SDK:** `UpdateSecretVersionStage`
- **AWS CLI:** `update-secret-version-stage`

You must identify the secret by the friendly name or ARN. You also specify the staging labels you want to add, move, or remove.

You can specify `FromSecretVersionId` and `MoveToSecretId` in the following combinations:

- `FromSecretVersionId` only: This deletes staging labels completely from the specified version.
- `MoveToVersionId` only: This adds the staging labels to the specified version. If other versions have any of the staging labels already attached, Secrets Manager automatically removes the labels from those versions.
- `MoveToVersionId` and `RemoveFromVersionId`: These explicitly move a label. The staging label must be present on the `RemoveFromVersionId` version of the secret, or an error occurs.

Example

The following example of the AWS CLI command removes the `AWSPREVIOUS` staging label from a version of the secret named "MyTestDatabase". You can retrieve the version ID of the version you want to delete by using the ListSecretVersionIds command.

```
$ 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:region:accountid:secret:development/
MyTestDatabase-AbCdEf",
    "Name": "development/MyTestDatabase"
}
```

Managing a Resource-based Policy for a Secret

This section describes how to attach, retrieve, and remove resource-based policies from secrets.

Topics

- Deleting a Resource-based Policy from a Secret (p. 70)

AWS Secrets Manager enables you to attach a permissions policy directly to the secret as part of the authorization strategy for a secret. Secrets Manager provides the functionality in addition to access from policies you attach to the users, groups, or roles requiring access the secret. You can retrieve a resource-based policy to review it at a later date, and you can delete the resource-based policy when you no longer need it.

Important
You can only manage secret policies from the CLI or the API. The Secrets Manager console does not provide this function.

Attaching a Resource-based Policy to a Secret

Attaching a permission policy to a secret

For details about constructing a resource-based policy, see Overview of Managing Access Permissions to Your Secrets Manager Secrets (p. 34) and Using Resource-based Policies for Secrets Manager (p. 48).

Use the following steps to attach a policy to a secret.

Using the AWS CLI or AWS SDK operations

You can use the following command to attach or modify the policy document to grant or deny access to the specified secret. The policy document must be formatted as JSON structured text. For more information, see Using Resource-based Policies for Secrets Manager (p. 48). We recommend you store your policy document as a text file, and then reference the file in the parameter of the command.

- **API/SDK:** `PutResourcePolicy`
- **CLI:** `put-resource-policy`

Example

The following example CLI command attaches or replaces the resource-based permission policy currently attached to the secret. Secrets Manager retrieves policy document from the file `secretpolicy.json`.

```
$ aws secretsmanager put-resource-policy --secret-id production/MyAwesomeAppSecret --
resource-policy file://secretpolicy.json
{
  "ARN": "arn:aws:secretsmanager:us-west-2:123456789012:secret:production/
MyAwesomeAppSecret-a1b2c3",
  "Name": "MyAwesomeAppSecret"
}
```

Retrieving a Resource-based Policy from a Secret

Retrieving a permission policy attached to a secret

Use the following steps to get the text of a resource-based policy attached to a secret.

Using the AWS CLI or AWS SDK operations

You can use the following command to retrieve the policy document currently attached to the specified secret.

- **API/SDK:** `GetResourcePolicy`
- **CLI:** `get-resource-policy`

Example

The following example CLI command retrieves the text for the resource-based permission policy currently attached to the secret.

```
$ aws secretsmanager get-resource-policy --secret-id production/MyAwesomeAppSecret
{
  "ARN": "arn:aws:secretsmanager:us-west-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\":\"*\"}}"
}
```

Deleting a Resource-based Policy from a Secret

Deleting the permission policy attached to a secret

Use the following steps to delete the resource-based policy currently attached to the specified secret.

Using the AWS CLI or AWS SDK operations

You can use the following command to delete a resource-based policy currently attached to the specified secret.

- **API/SDK:** `DeleteResourcePolicy`
- **CLI:** `delete-resource-policy`

Example

The following example CLI command deletes the resource-based permission policy currently attached to the secret.

```
$ aws secretsmanager delete-resource-policy --secret-id production/MyAwesomeAppSecret
{
  "ARN": "arn:aws:secretsmanager:us-west-2:123456789012:secret:production/
MyAwesomeAppSecret-a1b2c3",
  "Name": "production/MyAwesomeAppSecret"
}
```

Rotating Your AWS Secrets Manager Secrets

You can configure AWS Secrets Manager to automatically rotate the secret for a secured service or database. Secrets Manager natively knows how to rotate secrets for supported Amazon RDS databases (p. 4). However, Secrets Manager also can enable you to rotate secrets for other databases or third-party services. Because each service or database can have a unique way of configuring secrets, Secrets Manager uses a Lambda function you can customize to work with a selected database or service. You customize the Lambda function to implement the service-specific details of rotating a secret.

When you enable rotation for a secret by using the **Credentials for RDS database**, **Credentials for Redshift cluster**, or **Credentials for DocumentDB database** secret type, Secrets Manager provides a Lambda rotation function for you and populates the function automatically with the Amazon Resource Name (ARN) in the secret. You provide the details for this to work other than to provide a few details. For example, you specify the secret with permissions to rotate the credentials, and how often you want to rotate the secret.

When you enable rotation for a secret with **Credentials for other database** or some **Other type of secret**, you must provide the code for the Lambda function. The code includes the commands required to interact with your secured service to update or add credentials.

Topics

Permissions Required to Automatically Rotate Secrets

When you use the AWS Secrets Manager console to configure rotation for a secret for one of the fully supported databases (p. 4), the console configures almost all parameters for you. But if you create a function or opt to do anything manually for other reasons, you also might have to manually configure the permissions for that part of the rotation.

Permissions for Users Configuring Rotation vs. Users Triggering Rotation

Secrets Manager requires two separate sets of permissions for ***user*** operations using secret rotation:

- **Permissions required to configure rotation** – Secrets Manager assigns permissions to a trusted user to configure secret rotation. For more information, see Granting Full Secrets Manager Administrator Permissions to a User.

- **Permissions required to rotate the secret** – An IAM user only needs the permission, `secretsmanager:RotateSecret`, to trigger rotation after configuration. After rotation starts, the Lambda rotation function takes over and uses the attached IAM role and the permissions to authorize the operations performed during rotation, including any required AWS KMS operations.

The rest of this topic discusses the permissions for the Lambda rotation function to successfully rotate a secret.

Permissions Associated with the Lambda Rotation Function

AWS Secrets Manager uses a Lambda function to implement the code to rotate the credentials in a secret.

Secrets Manager service invokes the the Lambda function. The service does this by invoking an IAM role attached to the Lambda function. Secrets Manager has two actions for this:

- **The *trust policy* specifying who can assume the role**. You must configure this policy to allow Secrets Manager to assume the role, as identified by the service principal: `secretsmanager.amazonaws.com`. You can view this policy in the Lambda console on the function details page by clicking the key icon in the **Designer** section. The details appear in the **Function policy** section. This policy should look similar to the following example:

```
{
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE",
      "Effect": "Allow",
      "Principal": {
        "Service": "secretsmanager.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "<arn of the Lambda function that this trust policy is attached to -
 must match exactly>"
    }
  ]
}
```

  For security reasons, the trust policy created by Secrets Manager includes a `Resource` element and includes the Amazon Resource Name (ARN) of the Lambda function. Anyone who assumes the role can invoke *only* the Lambda function associated with the role.
- **The *permissions policy* for the role specifying the role of the assumer** You must configure this policy to specify the permissions Secrets Manager can use when assuming the role by invoking the function. Secrets Manager has two different policies available, depending on the rotation strategy to implement.
  - **Single user rotation**: The following example describes a function that rotates a secret by signing in with the credentials stored in the secret, and changing the password.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "secretsmanager:UpdateSecretVersionStage"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "secretsmanager:resource/AllowRotationLambdaArn": "<lambda_arn>"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword"
            ],
            "Resource": "*"
        },
        {
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

The first statement in the single-user example grants permission for the function to run Secrets Manager operations. However, the `Condition` element restricts this to only secrets configured with this Lambda function ARN as the secret rotation Lambda function.

The second statement allows one additional Secrets Manager operation that doesn't require the condition.

The third statement enables Lambda to set up the required configuration when you specify running your database or service in a VPC. For more information, see Configuring a Lambda Function to Access Resources in an Amazon VPC in the *AWS Lambda Developer Guide*.

- **Master user rotation**: The following example describes a function that rotates a secret by signing in using a separate **master secret** with credentials containing elevated permissions. If you use one of the rotation strategies alternate between two users, Secrets Manager requires this function.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "secretsmanager:UpdateSecretVersionStage"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "secretsmanager:resource/AllowRotationLambdaArn": "<lambda_arn>"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword"
            ],
            "Resource": "*"
        },
        {
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "<master_arn>"
        }
    ]
}
```

In addition to the same three statements as the previous single-user policy, this policy adds a fourth statement. The fourth statement enables the function to retrieve the credentials in the master secret. Secrets Manager uses the credentials in the master secret to sign in to the secured database and update the credentials in the rotated secret.

Configuring Your Network to Support Rotating Secrets

To successfully rotate your secrets, the Lambda rotation function must be able to communicate with both the protected database or service, and the AWS Secrets Manager service. The rotation function sends requests to your database or other service to update a user password with a new value. The function also calls Secrets Manager API operations to retrieve and update the secrets involved in the rotation process. If your Amazon RDS instance or other secret-protected service runs in a virtual private cloud (VPC) provided by Amazon VPC, you must take the following high-level steps to enable the required connectivity.

- **Configure your Lambda rotation function to enable communications between the function and the database instance.** If you use one of the database services (p. 4) fully supported by Secrets Manager, then the AWS CloudFormation template creating your function determines the public accessibility of your database instance.
  - If your protected service runs in a VPC and ***not*** publicly accessible, then the AWS CloudFormation template configures the Lambda rotation function to run in the same VPC. In this scenario, the rotation function can communicate with the protected service directly within the VPC.
  - If you run your protected service as a publicly accessible resource, in a VPC or not, then the AWS CloudFormation template configures the Lambda rotation function **not** to run in a VPC. In this scenario, the Lambda rotation function communicates with the protected service through the publicly accessible connection point.

  If you configure your rotation function manually and want to add it to a VPC, then on the function **Details** page scroll down to the **Networking** section and choose the appropriate **VPC** from the list.
- **Configure your VPC to enable communications between the Lambda rotation function running in a VPC and the Secrets Manager service endpoint.** By default, the Secrets Manager endpoints run on the public Internet. If you run your Lambda rotation function and protected database or service in a VPC, then you must perform one of the following steps:

- You can enable your Lambda function to access the public Secrets Manager endpoint by adding a NAT gateway to your VPC. This enables traffic that originates in your VPC to reach the public Secrets Manager endpoint. *This does expose your VPC to a level of risk* because an IP address for the gateway can be attacked from the public Internet.
- You can configure Secrets Manager service endpoints directly within your VPC. This configures your VPC to intercept any request addressed to the public regional endpoint, and redirect the VPC to the private service endpoint running within your VPC. For more details, see Connecting to Secrets Manager Through a VPC Endpoint.

Rotating Secrets for Supported Amazon RDS Databases

You can configure AWS Secrets Manager to automatically rotate the secret for an Amazon RDS database. Secrets Manager uses a Lambda function Secrets Manager provides.

Supported Amazon RDS Databases

For the purposes of the Amazon RDS options in Secrets Manager, a supported database means when you enable rotation, Secrets Manager provides a complete, ready-to-run Lambda rotation function designed for that database. For a complete list of the supported databases, see the section called “Databases with Fully Configured and Ready-to-Use Rotation Support” (p. 4). For any other type of database, you can still rotate your secret. However, you must use the workflow for **Other database**. For those instructions, see Rotating AWS Secrets Manager Secrets for Other Databases or Services (p. 98).

When you enable rotation for a secret with **Credentials for RDS database** as the secret type, Secrets Manager automatically creates and configures a Lambda rotation function for you. Then Secrets Manager equips your secret with the Amazon Resource Name (ARN) of the function. Secrets Manager creates the IAM role associated with the function and configures the role with all of the required permissions. Alternatively, if you use the same rotation strategy with another secret, and you want to use the same rotation with your new secret, you can specify the ARN of the existing function and use it for both secrets.

If you run your Amazon RDS DB instance in a VPC provided by Amazon VPC and the VPC doesn't have public Internet access, then Secrets Manager also configures the Lambda function to run within that VPC. Secrets Manager also requires that the Lambda rotation function must be able to access a Secrets Manager service endpoint to call the required API operations. If one or more of your resources in the VPC must communicate with the Internet, then you can configure the VPC with a NAT gateway to enable the Lambda rotation function to query the public Secrets Manager service endpoint. If you have no requirement to communicate with the Internet, you can configure the VPC with a private Secrets Manager service endpoint (p. 74) accessible from within the VPC.

Otherwise, you typically only need to provide a few details to determine the template used to construct the Lambda function:

- **Specify the secret with credentials with permissions to rotate the secret**: Sometimes the user can change their password. Other times, the user has very restricted permissions and can't change their password. Instead, you must use the credentials for a different administrator or "super" user to change the user credentials.

  You must specify which secret the rotation function uses to rotate the credentials on the secured database:

  - **Use this secret**: Choose this option if the credentials in the current secret have permissions in the database to change the password. Choosing this option causes Secrets Manager to implement a Lambda function with a rotation strategy to change the password for a single user with each rotation. For more information about this rotation strategy, see Rotating AWS Secrets Manager Secrets for One User with a Single Password (p. 107).

> **Considerations**
> Secrets Manager provides this as a "lower availability" option. Sign-in failures can occur between the moment when rotation removes the old password and the moment when the updated password becomes accessible as the new version of the secret. This window of time may be very short—on the order of a second or less. If you choose this option, make sure your client applications implement an appropriate "backoff and retry with jitter" strategy in the code. The applications should generate an error only if sign-in fails several times over a longer period of time.

- **Use a secret that I have previously stored in AWS Secrets Manager**: Choose this option if the credentials in the current secret contain more restrictive permissions and can't be used to update the credentials on the secured service. Or choose this if you require high availability for the secret. To choose this option, create a separate "master" secret and credentials with permission to create and update credentials on the secured service. Then choose that master secret from the list. Choosing this option causes Secrets Manager to implement a Lambda function. This Lambda function has a rotation strategy that clones the initial user found in the secret. Then the function alternates between the two users with each rotation, and updates the password for the user becoming active. For more information about this rotation strategy, see Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110).

> **Considerations**
> Secrets Manager provides this option as the "high availability" option because the old version of the secret continues to operate and handle service requests while preparing and testing the new version. Secrets Manager doesn't deprecate the old version until after the clients switch to the new version. No downtime occurs while changing between versions. This option requires the Lambda function to clone the permissions of the original user and apply them to the new user. The function then alternates between the two users with each rotation.
> If you need to change the permissions granted to the users, ensure you change permissions for both users.

- **Customize the function**: You can customize the Lambda rotation function provided by Secrets Manager to meet your organizational requirements. For example, you could extend the **testSecret** phase of the function (p. 106) to test the new version with application-specific checks to ensure the new secret works as expected. For instructions, see Customizing the Lambda Rotation Function Provided by Secrets Manager (p. 82).

Topics

Enabling Rotation for an Amazon RDS Database Secret

You can enable rotation for a secret with credentials for a supported Amazon RDS database (p. 4) by using the AWS Secrets Manager console (p. 77), the AWS CLI, or one of the AWS SDKs.

> **Warning**
> Enabling rotation causes the secret to rotate once immediately when you save the secret. Before you enable rotation, be sure you update all of your applications using this secret credentials to retrieve the secret from Secrets Manager. The original credentials might not be usable after the initial rotation. Any applications you fail to update break as soon as the old credentials become invalid.

Prerequisites: Network Requirements to Enable Rotation

To successfully enable rotation, you must correctly configure your network environment.

- **The Lambda function must be able to communicate with the database.** If you run your RDS database instance in a VPC, we recommend you configure your Lambda function to run in the same VPC. This enables direct connectivity between the rotation function and your service. To configure this, on the Lambda function details page, scroll down to the **Network** section and choose the **VPC** from the drop-down list to match the one your instance. You must also make sure the EC2 security groups attached to your instance enable communication between the instance and Lambda.
- **The Lambda function must be able to communicate with the Secrets Manager service endpoint.** If your Lambda rotation function can access an internet, either because you haven't configured the function to run in a VPC, or because the VPC has an attached NAT gateway, then you can use any of the available public endpoints for Secrets Manager. Alternatively, if you configure your Lambda function to run in a VPC without an internet access, then you can configure the VPC with a private Secrets Manager service endpoint (p. 74).

Enabling and configuring rotation for a supported Amazon RDS database secret

Follow the steps under one of the following tabs:

Using the AWS Management Console

Minimum permissions
To enable and configure rotation in the console, you must have permissions provided by the following managed policies:

- `SecretsManagerReadWrite` – Provides all of the Secrets Manager, Lambda, and AWS CloudFormation permissions.
- `IAMFullAccess` – Provides the IAM permissions required to create a role and attach a permission policy to it.

1. Sign in to the AWS Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Choose the name of the secret you want to enable rotation.
3. In the **Configure automatic rotation** section, choose **Enable automatic rotation**. This enables the other controls in this section.
4. For **Select rotation interval**, choose one of the predefined values—or choose **Custom**, and then type the number of days you want between rotations. If rotating your secret to meet compliance requirements, then we recommend you set this value to at least 1 day less than the compliance-mandated interval.

   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.

   Note
   If you use the Lambda function provided by Secrets Manager to alternate between two users, the console uses this template if you choose the second "master secret" option in the next step, then you should set your rotation period to one-half of your compliance-specified minimum interval. Secrets Manager keeps the old credentials available, if not actively used, for one additional rotation cycle. Secrets Manager invalidates the old credentials only after updating the user with a new password after the second rotation.

If you modify the rotation function to immediately invalidate the old credentials after the new secret becomes active, then you can extend the rotation interval to your full compliance-mandated minimum. Leaving the old credentials active for one additional cycle with the `AWSPREVIOUS` staging label provides a *last known* good set of credentials you can use for fast recovery. If something happens to break the current credentials, you can simply move the `AWSCURRENT` staging label to the version with the `AWSPREVIOUS` label. Then your customers should be able to access the resource again. For more information, see Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110).

5. Choose one of the following options:
   - **You want to create a new Lambda rotation function**
     a. Choose **Create a new Lambda function to perform rotation**.
     b. For **Lambda function name**, enter the name to assign to the Lambda function Secrets Manager creates for you.
     c. Specify the secret with credentials the rotation function can use. The credentials must contain permissions to update the user name and password on the protected database.
        - **Use this secret**: Choose this option if the credentials in this secret have permission in the database to change the password. Choosing this option causes Secrets Manager to create a Lambda function rotating secrets with a single user and changes the password with each rotation.

          Considerations
          Secrets Manager provides this option as a "lower availability" option. Sign-in failures can occur between the moment when rotation removes the old password and the moment when the updated password becomes accessible as the new version of the secret. This window of time may be very short—on the order of a second or less. If you choose this option, make sure your client applications implement an appropriate "backoff and retry with jitter" strategy in their code. The applications should generate an error only if sign-in fails several times over a longer period of time.
        - **Use a secret that I have previously stored in AWS Secrets Manager**: Choose this option if the credentials in the current secret don't have permissions to update, or if you require high availability for the secret. To choose this option, you must create a separate **master secret** with credentials with permissions to update the secret credentials. Then choose the master secret from the list.

          Choosing this option causes Secrets Manager to create a Lambda function that rotates secrets by creating a new user and password with each rotation, and deprecating the old user.

          Considerations
          Secrets Manager provides this option as the "high availability" option because the old version of the secret continues to operate and handle service requests while preparing and testing the new version. Secrets Manager doesn't deprecate the old version until after the clients switch to the new version. No downtime occurs while changing between versions.
          This option requires the Lambda function to clone the permissions of the original user and apply them to the new user. The function then alternates between the two users with each rotation.
          If you need to change the permissions granted to the users, ensure you change permissions for both users.
   - **You want to use a Lambda function you already created for another secret**
     a. Choose **Use an existing Lambda function to perform rotation**.
     b. Choose the Lambda function from the drop-down list.
     c. Specify the type of rotation function:

   - **Single-user rotation**: A rotation function for a secret that stores credentials for a user with permissions to change their password. Secrets Manager creates this is the type of function when you choose the option **Use this secret**.
   - **Multi-user rotation**: A rotation function for a secret that stores credentials for a user that can't change their password. The function requires a separate master secret that stores credentials for a user with permission to change the credentials for this secret user. Secrets Manager creates this type of function when you choose the option **Use a secret that I have previously stored in AWS Secrets Manager**.

   d. If you specified the second "master secret" option, then you must also choose the secret to provide the master user credentials. The credentials in the master secret must have permission to update the credentials stored in this secret.

6. Choose **Save** to store your changes and to trigger the initial rotation of the secret.
7. If you chose to rotate your secret with a separate master secret, then you must manually grant your Lambda rotation function permission to access the master secret. Follow these instructions:

   a. When rotation configuration completes, the following message might appear at the top of the page:

   Your secret `<secret name>` has been successfully stored and secret rotation is enabled. To finish configuring rotation, you need to provide the role permissions to access the value of the secret `<Amazon Resource Name (ARN) of your master secret>`.

   If this appears, then you must manually modify the policy for the role to grant the rotation function `secretsmanager:GetSecretValue` permission to access the master secret. Secrets Manager can't do this for you for security reasons. Rotation of the secret fails until you complete the following steps if the rotation can't access the master secret.

   b. Copy the Amazon Resource Name (ARN) from the message to your clipboard.

   c. Choose the link on the word "role" in the message. This opens the IAM console to the role details page for the role attached to the Lambda rotation function Secrets Manager created for you.

   d. On the **Permissions** tab, choose **Add inline policy**, and then set the following values:

   - For **Service**, choose **Secrets Manager**.
   - For **Actions**, choose **GetSecretValue**.
   - For **Resources**, choose **Add ARN** next to the **secret** resource type entry.
   - In the **Add ARN(s)** dialog box, paste the ARN of the master secret you copied previously.

   e. Choose **Review policy**, and then choose **Create policy**.

   Note
   As an alternative to using the Visual Editor as described in the previous steps, you can paste the following statement into an existing or new policy:

```
{
    "Effect": "Allow",
    "Action": "secretsmanager:GetSecretValue",
    "Resource": "<ARN of the master secret>"
}
```

   f. Return to the AWS Secrets Manager console.

If you don't have an ARN for a Lambda function assigned to the secret, Secrets Manager creates the function, assigns all required permissions, and configures the function to work with your database. Secrets Manager counts down the number of days specified in the rotation interval. When the interval reaches zero, Secrets Manager rotates the secret again and resets the interval for the next cycle. This continues until you disable rotation.

Using the AWS CLI or SDK Operations

Minimum permissions
To enable and configure rotation in the console, you must have the permissions provided by the following managed policies:

- `SecretsManagerReadWrite` – Provides all of the Secrets Manager, Lambda, and AWS CloudFormation permissions.
- `IAMFullAccess` – Provides the IAM permissions required to create a role and attach a permission policy to it.

You can use the following Secrets Manager commands to configure rotation for an existing secret for a supported Amazon RDS database:

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

You also need to use commands from AWS CloudFormation and AWS Lambda. For more information about the following commands, see the documentation for those services.

Important
The rotation function determines the exact format of the secret value used in your secret by the rotation function using this secret. 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 AWS Templates You Can Use to Create Lambda Rotation Functions (p. 152).

Creating a Lambda rotation function using an AWS Serverless Application Repository template

The following example of a AWS CLI session performing the equivalent of the console-based rotation configuration described in the **Using the AWS Management Console** tab. You create the function by using an AWS CloudFormation change set. Then you configure the resulting function with the required permissions. Finally, you configure the secret with the ARN of the completed function, and rotate once to test it.

The following example uses the generic template, so the template uses the last ARN shown earlier.

The first command sets up an AWS CloudFormation change set based on the template provided by Secrets Manager.

If your database or service resides in a VPC provided by Amazon VPC, then you must include the fourth of the following commands to configure the function to communicate with that VPC. If you don't have a VPC, then you can skip that command.

Also, if your VPC doesn't have access to the public Internet, then you must configure your VPC with a private service endpoint for Secrets Manager. The fifth of the following commands does that.

You use the `--application-id` parameter to specify which template to use. The value is the ARN of the template. For the list of templates provided by AWS and their ARNs, see AWS Templates You Can Use to Create Lambda Rotation Functions (p. 152).

The templates also require additional parameters provided with `--parameter-overrides`, as shown in the example that follows. The rotation template requires the parameter to send two pieces of information as Name and Value pairs to the template affecting the creation of the rotation function:

- **endpoint** – The URL of the service endpoint you want the rotation function to query. Typically, this is `https://secretsmanager.`*`region`*`.amazonaws.com`.

- **functionname** – The name of the completed Lambda rotation function created by this process.

```
$ aws serverlessrepo create-cloud-formation-change-set \
          --application-id arn:aws:serverlessrepo:us-
east-1:297356227824:applications/SecretsManagerRotationTemplate \
          --parameter-overrides '[{"Name":"endpoint","Value":"https://
secretsmanager.region.amazonaws.com"},
{"Name":"functionName","Value":"MyLambdaRotationFunction"}]' \
          --stack-name MyLambdaCreationStack
{
    "ApplicationId": "arn:aws:serverlessrepo:us-west-2:297356227824:applications/
SecretsManagerRDSMySQLRotationSingleUser",
    "ChangeSetId": "arn:aws:cloudformation:us-west-2:123456789012:changeSet/
EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE/EXAMPLE2-90ab-cdef-fedc-ba987EXAMPLE",
    "StackId": "arn:aws:cloudformation:us-west-2:123456789012:stack/aws-serverless-
repository-MyLambdaCreationStack/EXAMPLE3-90ab-cdef-fedc-ba987EXAMPLE"
}
```

The next command runs the change set you just created. The change-set-name parameter comes from the `ChangeSetId` output of the previous command. This command produces no output.

```
$ aws cloudformation execute-change-set --change-set-name arn:aws:cloudformation:us-
west-2:123456789012:changeSet/EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE/EXAMPLE2-90ab-cdef-
fedc-ba987EXAMPLE
```

The following command grants the Secrets Manager service permission to call the function. The output shows the permission added to the role trust policy.

```

$ aws lambda add-permission \
          --function-name MyLambdaRotationFunction \
          --principal secretsmanager.amazonaws.com \
          --action lambda:InvokeFunction \
          --statement-id SecretsManagerAccess
{
    "Statement": "{\"Sid\":\"SecretsManagerAccess\",\"Effect\":\"Allow
\",\"Principal\":{\"Service\":\"secretsmanager.amazonaws.com\"},
\"Action\":\"lambda:InvokeFunction\",\"Resource\":\"arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaRotationFunction\"}"
}
```

Secrets Manager only requires the following command if you run your database in a VPC. If you don't run your database on a VPC, skip this command. Look up the VPC information for your Amazon RDS DB instance by using either the Amazon RDS console, or by using the `aws rds describe-instances` CLI command. Then add the information in the following command and run it.

```
$ aws lambda update-function-configuration \
          --function-name arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaRotationFunction \
          --vpc-config SubnetIds=<COMMA SEPARATED LIST OF VPC SUBNET
 IDS>,SecurityGroupIds=<COMMA SEPARATED LIST OF SECURITY GROUP IDs>
```

If the VPC with your database instance and Lambda rotation function doesn't have an internet access, then you must configure the VPC with a private service endpoint for Secrets Manager. This enables the rotation function to access Secrets Manager at an endpoint within the VPC.

```
$ aws ec2 create-vpc-endpoint --vpc-id <VPC ID> /
```

```
                              --vpc-endpoint-type Interface /
                              --service-name com.amazonaws.<region>.secretsmanager /
                              --subnet-ids <COMMA SEPARATED LIST OF VPC SUBNET IDS> /
                              --security-group-ids <COMMA SEPARATED LIST OF SECURITY
GROUP IDs> /
                              --private-dns-enabled
```

If you created a function using a template that requires a master secret, then you must also add the following statement to the function role policy. This grants permission to the rotation function to retrieve the secret value for the master secret. For complete instructions, see Granting a Rotation Function Permission to Access a Separate Master Secret (p. 47).

```
        {
             "Action": "secretsmanager:GetSecretValue",
             "Resource":
 "arn:aws:secretsmanager:region:123456789012:secret:MyDatabaseMasterSecret",
             "Effect": "Allow"
        },
```

Finally, you can apply the rotation configuration to your secret and perform the initial rotation.

```
$ aws secretsmanager rotate-secret \
          --secret-id production/MyAwesomeAppSecret \
          --rotation-lambda-arn arn:aws:lambda:us-west-2:123456789012:function:aws-
serverless-repository-SecretsManagerRDSMySQLRo-10WGBDAXQ6ZEH \
          --rotation-rules AutomaticallyAfterDays=7
```

We recommend that if you want to create your own Lambda rotation function for a supported Amazon RDS database, you should follow the preceding steps that use the `SecretsManagerRotationTemplate` AWS CloudFormation template. Secrets Manager sets up most of the permissions and configuration settings for you.

Customizing the Lambda Rotation Function Provided by Secrets Manager

You can customize the Lambda rotation function to meet your organizational unique security requirements. Such requirements might include:

- Add additional tests on the new version of the secret. You want to ensure the permissions associate correctly with the new credentials.
- You want to use a different strategy for rotating your secrets than the Lambda function provided by Secrets Manager.

To customize the function, you must first discover which function Secrets Manager created for you. If you can't find the ARN of the function in the Secrets Manager console, you can retrieve it by using the AWS CLI or equivalent AWS SDK operations.

```
$ aws secretsmanager describe-secret --secret-id MyDatabaseSecret
```

Find the `RotationLambdaARN` value in the response.

To edit your Lambda rotation function

Follow the steps on one of the following tabs:

Using the Secrets Manager console

1. Determine the name of the Lambda rotation function for your secret:
   a. From the list of secrets, choose the name of the secret to modify the rotation.
   b. In the **Rotation configuration** section, examine the rotation ARN. The part of the ARN that follows `:function:` is the name of the function.
2. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
3. Choose the name of the Lambda rotation function you want to modify.
4. Make the required changes to the function.

Using the AWS CLI or AWS SDK operations

1. Determine the name of the Lambda rotation function for your secret. To do this, run the following command and examine the part of the `RotationLambdaARN` that follows `:function:`.

```
$ describe-secret --secret-id MySecret
{
    "ARN": "arn:aws:secretsmanager:us-west-2:123456789012:secret:MySecret-abc123",
    "Name": "MySecret",
    "RotationLambdaARN": "arn:aws:lambda:us-
west-2:123456789012:function:name_of_rotation_lambda_function",
    "LastChangedDate": 1519940941.014,
    "SecretVersionsToStages": {
        "5eae5e4a-a683-469e-96e7-af9a8180fba5": [
            "AWSCURRENT"
        ]
    }
}
```

2. Examine the `RotationLambdaARN` response value. The ARN of your Lambda rotation function and the last portion defines the name of your function.
3. Sign in to the AWS Management Console and open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
4. Choose the name of the Lambda function you identified to see the function details.
5. In the **Function code** section, you can edit the source code of the function. For more information about coding a Lambda function specifically for Secrets Manager, see Overview of the Lambda Rotation Function (p. 104). For the *AWS Lambda Developer Guide*. The provided Lambda functions support the Python 2.7 environment.

For more information about Lambda function options and coding techniques, see the AWS Lambda Developer Guide.

For more information about coding your own Secrets Manager rotation function, see Understanding and Customizing Your Lambda Rotation Function (p. 103).

Rotating Secrets for Amazon Redshift

You can configure AWS Secrets Manager to automatically rotate the secret for Amazon Redshift. Secrets Manager uses a Lambda function provided by Secrets Manager.

Secrets Manager supports Amazon Redshift Databases

Secrets Manager supports Amazon Redshift which means when you enable rotation, Secrets Manager provides a complete, ready-to-run Lambda rotation function.

When you enable rotation for a **Credentials for Redshift** as the secret type, Secrets Manager can automatically create and configure a Lambda rotation function for you. Then Secrets Manager equips your secret with the Amazon Resource Name (ARN) of the function. Secrets Manager creates the IAM role associated with the function and configures the function with all of the required permissions. Alternatively, if you already have another secret using the same rotation strategy you want to use with your new secret, you can specify the ARN of the existing function and use the rotation strategy for both secrets.

If you run your Amazon Redshift cluster in a VPC provided by Amazon VPC and the VPC doesn't have public Internet access, then Secrets Manager also configures the Lambda function to run within that VPC. Also the Lambda rotation function must be able to access a Secrets Manager service endpoint to call the required API operations. If one or more of your resources in the VPC must communicate with an internet, then you can configure the VPC with a NAT gateway to enable the Lambda rotation function to query the public Secrets Manager service endpoint. If you don't need to communicate with the internet, you can configure the VPC with a private Secrets Manager service endpoint (p. 74) accessible from within the VPC.

Otherwise, you typically only need to provide a few details that determine which template to use to construct the Lambda function:

- **Specify the secret with credentials containing permissions to rotate the secret**: Sometimes the user can change their password. In other cases, the user has very restricted permissions and can't change their password. Instead, you must use the credentials for a different administrator or "super" user to change the user's credentials.

  You must specify which secret the rotation function can use to rotate the credentials:

  - **Use this secret**: Choose this option if the credentials in the current secret have permissions to change its own password. Choosing this option causes Secrets Manager to implement a Lambda function with a rotation strategy that changes the password for a single user with each rotation. For more information about this rotation strategy, see Rotating AWS Secrets Manager Secrets for One User with a Single Password (p. 107).

    Considerations
    Secrets Manager provides this option as a "lower availability" option. Sign-in failures can occur between the moment when rotation removes the old password and the moment when the updated password becomes accessible as the new version of the secret. This window of time may be very short—on the order of a second or less. If you choose this option, make sure your client applications implement an appropriate "backoff and retry with jitter" strategy in their code. The applications should generate an error only if sign-in fails several times over a longer period of time.

  - **Use a secret that I have previously stored in AWS Secrets Manager**: Choose this option if the credentials in the current secret have more restrictive permissions and can't be used to update the credentials on the secured service. Or choose this if you require high availability for the secret. To choose this option, create a separate **master** secret and credentials with permission to create and update credentials on the secured service. Then choose that master secret from the list. Choosing this option causes Secrets Manager to implement a Lambda function. This Lambda function has a rotation strategy that clones the initial user found in the secret. Then the rotation alternates between the two users with each rotation, and updates the password for the user becoming active. For more information about this rotation strategy, see Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110).

    Considerations
    Secrets Manager provides this option as the "high availability" option because the old version of the secret continues to operate and handle service requests while preparing and testing the new version. Secrets Manager doesn't deprecate the old version until after the clients switch to the new version. No downtime occurs while changing between versions.

This option requires the Lambda function to clone the permissions of the original user and apply them to the new user. The function then alternates between the two users with each rotation.
If you need to change the permissions granted to the users, ensure you change permissions for both users.

- **You can customize the function**: You can customize the Lambda rotation function provided by Secrets Manager to meet your organizational requirements. For example, you could extend the **testSecret** phase of the function (p. 106) to test the new version with application-specific checks to ensure the new secret works as expected. For instructions, see Customizing the Lambda Rotation Function Provided by Secrets Manager (p. 82).

Topics

Enabling Rotation for an Amazon Redshift Secret

You can enable rotation for a secret with credentials for an Amazon Redshift service, by using the AWS Secrets Manager console (p. 85), the AWS CLI, or one of the AWS SDKs.

Warning
Enabling rotation causes the secret to rotate once immediately when you save the secret. Before you enable rotation, ensure you update all of your applications using this secret credentials to retrieve the secret from Secrets Manager. The original credentials might not be usable after the initial rotation. Any applications you don't update fail as soon as the old credentials become invalid.

Prerequisites: Network Requirements to Enable Rotation

To successfully enable rotation, you must have your network environment configured correctly.

- **The Lambda function must be able to communicate with the Amazon Redshift service.** If you run your Amazon Redshift cluster in a VPC, we recommend that you configure your Lambda function to run in the same VPC. This enables direct connectivity between the rotation function and your service. To configure this, on the Lambda function's details page, scroll down to the **Network** section and choose the **VPC** from the drop-down list to match the one running your instance. You must also make sure the EC2 security groups attached to your cluster enable communication between the cluster and Lambda.
- **The Lambda function must be able to communicate with the Secrets Manager service endpoint.** If your Lambda rotation function can access the internet, either because you haven't configured your the function to run in a VPC, or because the VPC has an attached NAT gateway, then you can use any of the available public endpoints for Secrets Manager. Alternatively, if you configure your Lambda function to run in a VPC without an internet access, then you can configure the VPC with a private Secrets Manager service endpoint (p. 74).

Enabling and configuring rotation for a supported Amazon Redshift cluster secret

Follow the steps under one of the following tabs:

Using the AWS Management Console

Minimum permissions
To enable and configure rotation in the console, you must have the permissions provided by the following managed policies:

- `SecretsManagerReadWrite` – Provides all of the Secrets Manager, Lambda, and AWS CloudFormation permissions.
- `IAMFullAccess` – Provides the IAM permissions required to create a role and attach a permission policy to it.

1. Sign in to the AWS Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Choose the name of the secret to enable rotation.
3. In the **Configure automatic rotation** section, choose **Enable automatic rotation**. This enables the other controls in this section.
4. For **Select rotation interval**, choose one of the predefined values—or choose **Custom**, and then type the number of days you want between rotations. If you rotate your secret to meet compliance requirements, then we recommend you set this value at least 1 day less than the compliance-mandated interval.

   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.

   Note
   If you use the Lambda function provided by Secrets Manager to alternate between two users, the console uses this template if you choose the second "master secret" option in the next step, then you should set your rotation period to one-half of your compliance-specified minimum interval. Secrets Manager retains the old credentials, if not actively used, for one additional rotation cycle, and only invalidates the old credentials after updating the user with a new password after the second rotation.
   If you modify the rotation function to immediately invalidate the old credentials after the new secret becomes active, then you can extend the rotation interval to your full compliance-mandated minimum. Leaving the old credentials active for one additional cycle with the `AWSPREVIOUS` staging label provides a "last known good" set of credentials you can use for fast recovery. If something happens to break the current credentials, you can simply move the `AWSCURRENT` staging label to the version with the `AWSPREVIOUS` label. Then your customers should be able to access the resource again.
   For more information, see Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110).
5. Choose one of the following options:
   - **You want to create a new Lambda rotation function**
     a. Choose **Create a new Lambda function to perform rotation**.
     b. For **Lambda function name**, enter the name to assign to the Lambda function that Secrets Manager creates for you.
     c. Specify the secret with credentials the rotation function can use. They must have permissions to update the user name and password on the protected service.
        - **Use this secret**: Choose this option if the credentials in this secret have permission in the service to change their password. Choosing this option causes Secrets Manager to create a Lambda function that rotates secrets with a single user after changing the password with each rotation.

          Note
          Secrets Manager provides this option as a "lower availability" option. Sign-in failures can occur between the moment when rotation removes the old password and the moment when the updated password becomes accessible as the new version of the secret. This window of time may be very short—on the order of a second or less.

If you choose this option, make sure your client applications implement an appropriate "backoff and retry with jitter" strategy in their code. The applications should generate an error only if sign-in fails several times over a longer period of time.

- **Use a secret that I have previously stored in AWS Secrets Manager**: Choose this option if the credentials in the current secret don't have permissions to update itself, or if you require high availability for the secret. To choose this option, you must create a separate "master" secret with credentials that have permissions to update this secret's credentials. Then choose the master secret from the list. Choosing this option causes Secrets Manager to create a Lambda function that rotates secrets by creating a new user and password with each rotation, and deprecating the old user.

  Note
  Secrets Manager provides this option as the "high availability" option because the old version of the secret continues to operate and handle service requests while preparing and testing the new version. Secrets Manager doesn't deprecate the old version until after the clients switch to the new version. No downtime occurs while changing between versions.
  This option requires the Lambda function to clone the permissions of the original user and apply them to the new user. The function then alternates between the two users with each rotation.
  If you need to change the permissions granted to the users, ensure you change permissions for both users.

- **You want to use a Lambda function you already created for another secret**

  a. Choose **Use an existing Lambda function to perform rotation**.

  b. Choose the Lambda function from the drop-down list.

  c. Specify the type of rotation function you chose:

     - **Single-user rotation**: A rotation function for a secret that stores credentials for a user with permissions to change their password. Secrets Manager creates this type of function when you choose the option **Use this secret** when you create a function.

     - **Multi-user rotation**: A rotation function for a secret that stores credentials for a user without permission change their password. The function requires a separate master secret that stores credentials for a user with permission to change the credentials for this secret user.

     - Secrets Manager creates this type of function when you choose the option **Use a secret that I have previously stored in AWS Secrets Manager**.

  d. If you specified the second "master secret" option, then you must also choose the secret that provides the master user credentials. The credentials in the master secret must have permission to update the credentials stored in this secret.

6. Choose **Save** to store your changes and to trigger the initial rotation of the secret.

7. If you chose to rotate your secret with a separate master secret, then you must manually grant your Lambda rotation function permission to access the master secret. Follow these instructions:

   a. When rotation configuration completes, the following message might appear at the top of the page:

      Your secret `<secret name>` has been successfully stored and secret rotation is enabled. To finish configuring rotation, you need to provide the role permissions to access the value of the secret `<Amazon Resource Name (ARN) of your master secret>`.

      If this appears, then you must manually modify the policy for the role to grant the rotation function `secretsmanager:GetSecretValue` permission to access the master secret. Secrets Manager can't do this for you for security reasons. Rotation of the secret fails until you complete the following steps if it can't access the master secret.

   b. Copy the Amazon Resource Name (ARN) from the message to your clipboard.

c. Choose the link on the word "role" in the message. This opens the IAM console to the role details page for the role attached to the Lambda rotation function that Secrets Manager created for you.

d. On the **Permissions** tab, choose **Add inline policy**, and then set the following values:

   - For **Service**, choose **Secrets Manager**.
   - For **Actions**, choose **GetSecretValue**.
   - For **Resources**, choose **Add ARN** next to the **secret** resource type entry.
   - In the **Add ARN(s)** dialog box, paste the ARN of the master secret that you copied previously.

e. Choose **Review policy**, and then choose **Create policy**.

   Note
   As an alternative to using the Visual Editor as described in the previous steps, you can paste the following statement into an existing or new policy:

```
{
    "Effect": "Allow",
    "Action": "secretsmanager:GetSecretValue",
    "Resource": "<ARN of the master secret>"
}
```

f. Return to the AWS Secrets Manager console.

If you don't already have an ARN for a Lambda function assigned to the secret, Secrets Manager creates the function, assigns all required permissions, and configures the function to work with your service. Secrets Manager counts down the number of days specified in the rotation interval. When the rotation interval reaches zero, Secrets Manager rotates the secret again and resets the interval for the next cycle. This continues until you disable rotation.

Using the AWS CLI or SDK Operations

Minimum permissions
To enable and configure rotation in the console, you must have the permissions that are provided by the following managed policies:

- `SecretsManagerReadWrite` – Provides all of the Secrets Manager, Lambda, and AWS CloudFormation permissions.
- `IAMFullAccess` – Provides the IAM permissions that are required to create a role and attach a permission policy to it.

You can use the following Secrets Manager commands to configure rotation for an existing secret for Amazon Redshift:

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

You also need to use commands from AWS CloudFormation and AWS Lambda. For more information about the following commands, see the documentation for those services.

Important
The rotation function you want to use with the secret determines the exact format of the secret value that you must use in your secret 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 AWS Templates You Can Use to Create Lambda Rotation Functions (p. 152).

To create a Lambda rotation function by using an AWS Serverless Application Repository template

The following displays an example AWS CLI session that performs the equivalent of the console-based rotation configuration described in the **Using the AWS Management Console** tab. You create the function by using an AWS CloudFormation change set. Then you configure the resulting function with the required permissions. Finally, you configure the secret with the ARN of the completed function, and rotate once to test it.

The following example uses the generic template, so the template uses the last ARN shown earlier.

The first command sets up an AWS CloudFormation change set based on the template provided by Secrets Manager.

If your service resides in a VPC provided by Amazon VPC, then you must include the fourth of the following commands to configure the function to communicate with that VPC. If no VPC is involved, then you can skip that command.

If your VPC doesn't have access to the an internet, then you must configure your VPC with a private service endpoint for Secrets Manager. The fifth of the following commands does that.

You use the `--application-id` parameter to specify which template to use. The value is the ARN of the template. For the list of templates provided by AWS and their ARNs, see AWS Templates You Can Use to Create Lambda Rotation Functions (p. 152).

The templates also require additional parameters provided with `--parameter-overrides`, as shown in the example that follows. Secrets Manager requires this parameter to sned two pieces of information as Name and Value pairs to the template affect the rotation function creation:

- **endpoint** – The URL of the service endpoint you want the rotation function to query. Typically, this is `https://secretsmanager.`*`region`*`.amazonaws.com`.
- **functionname** – The name of the completed Lambda rotation function created by this process.

```
$ aws serverlessrepo create-cloud-formation-change-set \
          --application-id arn:aws:serverlessrepo:us-
east-1:297356227824:applications/SecretsManagerRotationTemplate \
          --parameter-overrides '[{"Name":"endpoint","Value":"https://
secretsmanager.region.amazonaws.com"},
{"Name":"functionName","Value":"MyLambdaRotationFunction"}]' \
          --stack-name MyLambdaCreationStack
{
    "ApplicationId": "arn:aws:serverlessrepo:us-west-2:297356227824:applications/
SecretsManagerRDSMySQLRotationSingleUser",
    "ChangeSetId": "arn:aws:cloudformation:us-west-2:123456789012:changeSet/
EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE/EXAMPLE2-90ab-cdef-fedc-ba987EXAMPLE",
    "StackId": "arn:aws:cloudformation:us-west-2:123456789012:stack/aws-serverless-
repository-MyLambdaCreationStack/EXAMPLE3-90ab-cdef-fedc-ba987EXAMPLE"
}
```

The next command runs the change set you just created. The change-set-name parameter comes from the `ChangeSetId` output of the previous command. This command produces no output.

```
$ aws cloudformation execute-change-set --change-set-name arn:aws:cloudformation:us-
west-2:123456789012:changeSet/EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE/EXAMPLE2-90ab-cdef-
fedc-ba987EXAMPLE
```

The following command grants the Secrets Manager service permission to call the function on your behalf. The output shows the permission added to the role trust policy.

```
$ aws lambda add-permission \
          --function-name MyLambdaRotationFunction \
          --principal secretsmanager.amazonaws.com \
          --action lambda:InvokeFunction \
          --statement-id SecretsManagerAccess
{
    "Statement": "{\"Sid\":\"SecretsManagerAccess\",\"Effect\":\"Allow
\",\"Principal\":{\"Service\":\"secretsmanager.amazonaws.com\"},
\"Action\":\"lambda:InvokeFunction\",\"Resource\":\"arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaRotationFunction\"}"
}
```

Secrets Manager requires the following command if you run your service in a VPC. If it isn't, skip this command. Find the VPC information for your Amazon Redshift cluster by using either the Amazon RDS console, or by using the `aws rds describe-instances` CLI command. Then use the information in the following command and execute it.

```
$ aws lambda update-function-configuration \
          --function-name arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaRotationFunction \
          --vpc-config SubnetIds=<COMMA SEPARATED LIST OF VPC SUBNET
 IDS>,SecurityGroupIds=<COMMA SEPARATED LIST OF SECURITY GROUP IDs>
```

If the VPC with your service instance and Lambda rotation function without an internet access, then you must configure the VPC with a private service endpoint for Secrets Manager. This enables the rotation function to access Secrets Manager at an endpoint within the VPC.

```
$ aws ec2 create-vpc-endpoint --vpc-id <VPC ID> /
                              --vpc-endpoint-type Interface /
                              --service-name com.amazonaws.<region>.secretsmanager /
                              --subnet-ids <COMMA SEPARATED LIST OF VPC SUBNET IDS> /
                              --security-group-ids <COMMA SEPARATED LIST OF SECURITY
 GROUP IDs> /
                              --private-dns-enabled
```

If you created a function using a template that requires a master secret, then you must also add the following statement to the function role policy. This grants permission to the rotation function to retrieve the secret value for the master secret. For complete instructions, see Granting a Rotation Function Permission to Access a Separate Master Secret (p. 47).

```
        {
             "Action": "secretsmanager:GetSecretValue",
             "Resource":
 "arn:aws:secretsmanager:region:123456789012:secret:MyDatabaseMasterSecret",
             "Effect": "Allow"
        },
```

Finally, you can apply the rotation configuration to your secret and perform the initial rotation.

```
$ aws secretsmanager rotate-secret \
          --secret-id production/MyAwesomeAppSecret \
          --rotation-lambda-arn arn:aws:lambda:us-west-2:123456789012:function:aws-
serverless-repository-SecretsManagerRDSMySQLRo-10WGBDAXQ6ZEH \
          --rotation-rules AutomaticallyAfterDays=7
```

We recommend if you want to create your own Lambda rotation function for Amazon Redshift, you should follow the preceding steps that use the `SecretsManagerRotationTemplate` AWS

CloudFormation template. Using the template allows Secrets Manager to set up most of the permissions and configuration settings for you.

Rotating Secrets for Amazon DocumentDB

You can configure AWS Secrets Manager to automatically rotate the secret for Amazon DocumentDB. Secrets Manager uses a Lambda function that Secrets Manager provides.

Amazon DocumentDB as a supported service

Secrets Manager supports Amazon DocumentDB and provides a complete, ready-to-run Lambda rotation function designed for Amazon DocumentDB.

When you enable rotation for a secret with **Credentials for DocumentDB** as the secret type, Secrets Manager can automatically create and configure a Lambda rotation function for you. Then Secrets Manager equips your secret with the Amazon Resource Name (ARN) of the function. Secrets Manager creates the IAM role associated with the function and configures the role with all of the required permissions. Alternatively, if you already have another secret that uses the same rotation strategy you want to use with your new secret, you can specify the ARN of the existing function and use it for both secrets.

If you run your Amazon DocumentDB instance in a VPC provided by Amazon VPC and the VPC doesn't have public Internet access, then Secrets Manager also configures the Lambda function to run within that VPC. The Lambda rotation function must be able to access a Secrets Manager service endpoint to call the required API operations. If one or more of your resources in the VPC must communicate with the Internet, then you can configure the VPC with a NAT gateway to enable the Lambda rotation function to query the public Secrets Manager service endpoint. If you have no other need to communicate with the Internet, you can configure the VPC with a private Secrets Manager service endpoint (p. 74) accessible from within the VPC.

Otherwise, you typically only need to provide a few details to determine which template you use to construct the Lambda function:

- **Specify the secret with credentials and permissions to rotate the secret**:Only the *super user* can change their password. Other users have restricted permissions and cannot change their password. You must use the credentials for a different administrator or *super user* to change the user credentials.

  You must specify which secret the rotation function can use to rotate the credentials on the secured database:
  - **Use this secret**: Choose this option if the current secret has *super user* credentials. Choosing this option causes Secrets Manager to implement a Lambda function with a rotation strategy changing the password for a single user with each rotation. For more information about this rotation strategy, see Rotating AWS Secrets Manager Secrets for One User with a Single Password (p. 107).

    Considerations
    Secrets Manager provides this option as a "lower availability" option. Sign-in failures can occur between the moment when rotation removes the old password and the moment when the updated password becomes accessible as the new version of the secret. This window of time may be very short—on the order of a second or less.
    If you choose this option, make sure your client applications implement an appropriate "backoff and retry with jitter" strategy in their code. The applications should generate an error only if sign-in fails several times over a longer period of time.
  - **Use a secret that I have previously stored in AWS Secrets Manager**: Choose this if you require high availability for the secret. To choose this option, create a separate "master" secret with credentials containing permission to create and update credentials on the secured service. Then choose the master secret from the list. Choosing this option causes Secrets Manager to implement a Lambda function. This Lambda function has a rotation strategy that clones the initial user found in the

secret. Then Secrets Manager alternates between the two users with each rotation, and updates the password for the user becoming active. For more information about this rotation strategy, see Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110).

> **Note**
> Secrets Manager provides this option as the "high availability" option because the old version of the secret continues to operate and handle service requests while preparing and testing the new version. Secrets Manager doesn't deprecate the old version until the next rotation. No downtime occurs while changing between versions.
> This option requires the Lambda function to clone the permissions of the original user and apply them to the new user. The function then alternates between the two users with each rotation.
> If you need to change the permissions granted to the users, ensure you change permissions for both users.

- **You can customize the function**: You can tailor the Lambda rotation function provided by Secrets Manager to meet your organizational security requirements. For example, you could extend the **testSecret** phase of the function (p. 106) to test the new version with application-specific checks to ensure the new secret works as expected. For instructions, see Customizing the Lambda Rotation Function Provided by Secrets Manager (p. 82).

Topics

Enabling Rotation for an Amazon DocumentDB Secret

You can enable rotation for a secret with credentials for a Amazon DocumentDB database by using the AWS Secrets Manager console (p. 93), the AWS CLI, or one of the AWS SDKs.

> **Warning**
> Enabling rotation causes the secret to rotate once immediately when you save the secret. Before you enable rotation, ensure you update all of your applications using the secret credentials to retrieve the secret from Secrets Manager. The original credentials might not be usable after the initial rotation. If you don't update all of your applications, you might encounter errors as the old credentials become invalid.

Prerequisites: Network Requirements to Enable Rotation

To successfully enable rotation, configure your network environment correctly.

- **The Lambda function must be able to communicate with the database.** If you run your DocumentDB database instance in a VPC, we recommend you configure your Lambda function to run in the same VPC. This enables direct connectivity between the rotation function and your service. To configure this, on the Lambda function details page, scroll down to the **Network** section and choose the **VPC** from the drop-down list to match your instance. You must also make sure the EC2 security groups attached to your instance enable communication between the instance and Lambda.
- **The Lambda function must be able to communicate with the Secrets Manager service endpoint.** If your Lambda rotation function can access the Internet, either because you haven't configured the function to run in a VPC, or because the VPC has an attached NAT gateway, then you can use any of the available public endpoints for Secrets Manager. Alternatively, if you configure your Lambda function to run in a VPC without Internet access, then you can configure the VPC with a private Secrets Manager service endpoint (p. 74).

Enabling and configuring rotation for a Amazon DocumentDB secret

Follow the steps under one of the following tabs:

Using the AWS Management Console

Minimum permissions
To enable and configure rotation in the console, you must have the permissions provided by the following managed policies:

- `SecretsManagerReadWrite` – Provides all of the Secrets Manager, Lambda, and AWS CloudFormation permissions.
- `IAMFullAccess` – Provides the IAM permissions required to create a role and attach a permission policy to it.

1. Sign in to the AWS Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Choose the name of the secret to enable rotation.
3. In the **Configure automatic rotation** section, choose **Enable automatic rotation**. This enables the other controls in this section.
4. For **Select rotation interval**, choose one of the predefined values—or choose **Custom**, and then type the number of days you want between rotations. If you rotate your secret to meet compliance requirements, then we recommend you set this value to at least 1 day less than the compliance-mandated interval.

   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.

   Note
   If you use the Lambda function provided by Secrets Manager to alternate between two users, the console uses this template if you choose the second *master secret* option in the next step, then you should set your rotation period to one-half of your compliance-specified minimum interval. Secrets Manager retains the old credentials, if not actively used, for one additional rotation cycle. Then Secrets Manager invalidates the old credentials after updating the user with a new password after the second rotation.
   If you modify the rotation function to immediately invalidate the old credentials after the new secret becomes active, then you can extend the rotation interval to your full compliance-mandated minimum. Leaving the old credentials active for one additional cycle with the `AWSPREVIOUS` staging label provides a *last known good* set of credentials you can use for fast recovery. If something happens to break the current credentials, you can simply move the `AWSCURRENT` staging label to the version with the `AWSPREVIOUS` label. Then you can access the resource again. For more information, see Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110).
5. Choose one of the following options:
   - **You want to create a new Lambda rotation function**
     a. Choose **Create a new Lambda function to perform rotation**.
     b. For **Lambda function name**, enter the name you want to assign to the Lambda function that Secrets Manager creates for you.
     c. Specify the secret with credentials the rotation function can use. The user must have permissions to update the user name and password on the protected database.
        - **Use this secret**: Choose this option if the credentials in this secret have permission in the database to change their password. Choosing this option causes Secrets Manager to create

a Lambda function that rotates secrets with a single user whose password changes with each rotation.

> **Note**
> Secrets Manager provides this option as a lower availability option. Sign-in failures can occur between the moment when rotation removes the old password and the moment when the updated password becomes accessible as the new version of the secret. This window of time may be very short—on the order of a second or less.
> If you choose this option, make sure your client applications implement an appropriate "backoff and retry with jitter" strategy in their code. The applications should generate an error only if sign-in fails several times over a longer period of time.

- **Use a secret that I have previously stored in AWS Secrets Manager**: Choose this option if the credentials in the current secret don't allow update the permissions , or if you require high availability for the secret. To choose this option, you must create a separate "master" secret with credentials and permissions to update this secret credentials. Then choose the master secret from the list. Choosing this option causes Secrets Manager to create a Lambda function that rotates secrets by creating a new user and password with each rotation, and deprecating the old user.

> **Note**
> Secrets Manager provides this option as a high availability option because the old version of the secret continues to operate and handle service requests while preparing and testing the new version. Secrets Manager doesn't deprecate the old version until the next rotation. No downtime occurs while changing between versions.
> This option requires the Lambda function to clone the permissions of the original user and apply them to the new user. The function then alternates between the two users with each rotation.
> If you need to change the permissions granted to the users, ensure you change permissions for both users.

- **You want to use a Lambda function you already created for another secret**

  a. Choose **Use an existing Lambda function to perform rotation**.

  b. Choose the Lambda function from the drop-down list.

  c. Specify the type of rotation function :

  - **Single-user rotation**: A rotation function for a secret that stores credentials for a user with permissions to change their password. You create this type of function when you choose the option **Use this secret** when you create a function.
  - **Multi-user rotation**: A rotation function for a secret that stores credentials for a user unable to change their password. The function requires a separate master secret that stores credentials for a user with permission to change the credentials for this secret user. You create this type of function when you choose the option **Use a secret that I have previously stored in AWS Secrets Manager** .

  d. If you specified the second "master secret" option, then you must also choose the secret that can provide the master user credentials. The credentials in the master secret must have permission to update the credentials stored in this secret.

6. Choose **Save** to store your changes and to trigger the initial rotation of the secret.
7. If you chose to rotate your secret with a separate master secret, then you must manually grant your Lambda rotation function permission to access the master secret. Follow these instructions:

   a. When rotation configuration completes, the following message might appear at the top of the page:

Your secret `<secret name>` has been successfully stored and secret rotation is enabled. To finish configuring rotation, you need to provide the role permissions to access the value of the secret `<Amazon Resource Name (ARN) of your master secret>`.

If this appears, then you must manually modify the policy for the role to grant the rotation function `secretsmanager:GetSecretValue` permission to access the master secret. Secrets Manager can't do this for you for security reasons. If Secrets Manager can't access the master secret, then Rotation of the secret fails until you complete the following steps.

b. Copy the Amazon Resource Name (ARN) from the message to your clipboard.

c. Choose the link on the word "role" in the message. This opens the IAM console to the role details page for the role attached to the Lambda rotation function Secrets Manager created for you.

d. On the **Permissions** tab, choose **Add inline policy**, and then set the following values:

- For **Service**, choose **Secrets Manager**.
- For **Actions**, choose **GetSecretValue**.
- For **Resources**, choose **Add ARN** next to the **secret** resource type entry.
- In the **Add ARN(s)** dialog box, paste the ARN of the master secret you copied previously.

e. Choose **Review policy**, and then choose **Create policy**.

Note

As an alternative to using the Visual Editor as described in the previous steps, you can paste the following statement into an existing or new policy:

```
{
    "Effect": "Allow",
    "Action": "secretsmanager:GetSecretValue",
    "Resource": "<ARN of the master secret>"
}
```

f. Return to the AWS Secrets Manager console.

If you do not have an ARN for a Lambda function assigned to the secret, Secrets Manager creates the function, assigns all required permissions, and configures the function to work with your database. Secrets Manager counts down the number of days specified in the rotation interval. When the count reaches zero, Secrets Manager rotates the secret again and resets the interval for the next cycle. This continues until you disable rotation.

Using the AWS CLI or SDK Operations

Minimum permissions

To enable and configure rotation in the console, you must have the permissions provided by the following managed policies:

- `SecretsManagerReadWrite` – Provides all of the Secrets Manager, Lambda, and AWS CloudFormation permissions.
- `IAMFullAccess` – Provides the IAM permissions required to create a role and attach a permission policy to it.

You can use the following Secrets Manager commands to configure rotation for an existing secret for a supported Amazon RDS database:

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

You also need to use commands from AWS CloudFormation and AWS Lambda. For more information about the following commands , see the documentation for those services.

Important
The rotation function determines the exact format of the secret value you must use in your secret. For the details of each rotation function requirement for the secret value, see the **Expected SecretString Value** entry under the relevant rotation function at AWS Templates You Can Use to Create Lambda Rotation Functions (p. 152).

Creating a Lambda rotation function by using an AWS Serverless Application Repository template

An example follows of a AWS CLI session that performs the equivalent of the console-based rotation configuration described in the **Using the AWS Management Console** tab. You create the function by using an AWS CloudFormation change set. Then you configure the resulting function with the required permissions. Finally, you configure the secret with the ARN of the completed function, and rotate once to test it.

The following example uses the generic template, and uses the last ARN shown earlier.

The first command sets up an AWS CloudFormation change set based on the template provided by Secrets Manager.

If your database or service resides in a VPC provided by Amazon VPC, then you must include the fourth of the following commands to configure the function to communicate with that VPC. If you don't have VPC, then you can skip the command.

Also, if your VPC doesn't have access to the public internet, then you must configure your VPC with a private service endpoint for Secrets Manager. The fifth of the following commands does that.

The templates also require additional parameters provided with `--parameter-overrides`, as shown in the example that follows. This parameter requires Secrets Manager to send two pieces of information as Name and Value pairs to the template that affect the creation of the rotation function:

- **endpoint** – The URL of the service endpoint you want the rotation function to query. Typically, this is `https://secretsmanager.region.amazonaws.com`.
- **functionname** – The name of the completed Lambda rotation function created by this process.

```
$ aws serverlessrepo create-cloud-formation-change-set \
          --application-id &region-arn;serverlessrepo:us-
east-1:297356227824:applications/SecretsManagerRotationTemplate \
          --parameter-overrides '[{"Name":"endpoint","Value":"https://
secretsmanager.region.amazonaws.com"},
{"Name":"functionName","Value":"MyLambdaRotationFunction"}]' \
          --stack-name MyLambdaCreationStack
{
    "ApplicationId": "&region-arn;serverlessrepo:us-west-2:297356227824:applications/
SecretsManagerRDSMySQLRotationSingleUser",
    "ChangeSetId": "&region-arn;cloudformation:us-west-2:123456789012:changeSet/
EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE/EXAMPLE2-90ab-cdef-fedc-ba987EXAMPLE",
    "StackId": "&region-arn;cloudformation:us-west-2:123456789012:stack/aws-serverless-
repository-MyLambdaCreationStack/EXAMPLE3-90ab-cdef-fedc-ba987EXAMPLE"
}
```

The next command runs the change set you just created. The `change-set-name` parameter comes from the `ChangeSetId` output of the previous command. This command produces no output.

```
$ aws cloudformation execute-change-set --change-set-name &region-
arn;cloudformation:us-west-2:123456789012:changeSet/EXAMPLE1-90ab-cdef-fedc-
ba987EXAMPLE/EXAMPLE2-90ab-cdef-fedc-ba987EXAMPLE
```

The following command grants the Secrets Manager service permission to call the function for you. The output shows the permission added to the role trust policy.

```

$ aws lambda add-permission \
          --function-name MyLambdaRotationFunction \
          --principal secretsmanager.amazonaws.com \
          --action lambda:InvokeFunction \
          --statement-id SecretsManagerAccess
{
    "Statement": "{\"Sid\":\"SecretsManagerAccess\",\"Effect\":\"Allow
\",\"Principal\":{\"Service\":\"secretsmanager.amazonaws.com\"},\"Action
\":\"lambda:InvokeFunction\",\"Resource\":\"&region-arn;lambda:us-
west-2:123456789012:function:MyLambdaRotationFunction\"}"
}
```

If you run your database in a VPC environment, Secrets Manager requires the following command. . If not, skip this command. Find the VPC information for your Amazon RDS DB instance by using either the Amazon RDS console, or by using the `aws rds describe-instances` CLI command. Then add the information in the following command and run it.

```
$ aws lambda update-function-configuration \
          --function-name &region-arn;lambda:us-
west-2:123456789012:function:MyLambdaRotationFunction \
          --vpc-config SubnetIds=<COMMA SEPARATED LIST OF VPC SUBNET
 IDS>,SecurityGroupIds=<COMMA SEPARATED LIST OF SECURITY GROUP IDs>

```

If the VPC with your database instance and Lambda rotation function doesn't have Internet access, then you must configure the VPC with a private service endpoint for Secrets Manager. This enables the rotation function to access Secrets Manager at an endpoint within the VPC.

```
$ aws ec2 create-vpc-endpoint --vpc-id <VPC ID> /
                              --vpc-endpoint-type Interface /
                              --service-name com.amazonaws.<region>.secretsmanager /
                              --subnet-ids <COMMA SEPARATED LIST OF VPC SUBNET IDS> /
                              --security-group-ids <COMMA SEPARATED LIST OF SECURITY
 GROUP IDs> /
                              --private-dns-enabled
```

If you created a function using a template that requires a master secret, then you must also add the following statement to the function role policy. This grants permission to the rotation function to retrieve the secret value for the master secret. For complete instructions, see Granting a Rotation Function Permission to Access a Separate Master Secret (p. 47).

```
        {
             "Action": "secretsmanager:GetSecretValue",
             "Resource": "&region-
arn;secretsmanager:region:123456789012:secret:MyDatabaseMasterSecret",
             "Effect": "Allow"
        },
```

Finally, you can apply the rotation configuration to your secret and perform the initial rotation.

```
$ aws secretsmanager rotate-secret \
          --secret-id production/MyAwesomeAppSecret \
          --rotation-lambda-arn &region-arn;lambda:us-west-2:123456789012:function:aws-
serverless-repository-SecretsManagerRDSMySQLRo-10WGBDAXQ6ZEH \
          --rotation-rules AutomaticallyAfterDays=7
```

We recommend if you want to create your own Lambda rotation function, you should follow the preceding steps that use the `SecretsManagerRotationTemplate` AWS CloudFormation template. Secrets Manager sets up most of the permissions and configuration settings for you.

Rotating AWS Secrets Manager Secrets for Other Databases or Services

You can configure AWS Secrets Manager to automatically rotate the secret for a secured service or database. Secrets Manager natively knows how to rotate secrets for Amazon RDS databases (p. 75). However, Secrets Manager also enables you to rotate secrets for other databases or third-party services. Because each service or database can have a unique way of configuring its secrets, Secrets Manager uses a Lambda function that you must write to work with whatever database or service that you choose. You customize the Lambda function to implement the service-specific details of how to rotate a secret.

When you enable rotation for a secret for another database or some other type of service, you must create and configure the Lambda function and write the code.

Before you can enable rotation for other databases or services, you must first create the Lambda rotation function. AWS does provide a generic template Lambda rotation function that you can use as a starting point. The following procedure describes how to create a new Lambda function based on this template. When you then enable rotation (p. 101), you only need to add the Amazon Resource Name (ARN) of the completed function to your secret.

Topics

Rotating AWS Secrets Manager Secrets for Other Databases or Services

If you create a secret for another application besides one of the supported Amazon RDS databases, then AWS Secrets Manager doesn't create the Lambda rotation function for you. You must create and configure it, and then provide the Amazon Resource Name (ARN) of the completed function to the secret. You do this by using the Secrets Manager console, the AWS CLI, or one of the AWS SDKs.

This topic describes creating the Lambda function by using an AWS CloudFormation change set you create and run. You then attach permissions. At that point, you can edit the code to make the rotation function work the way you want it to. Finally, you associate the completed function with your secret so that Secrets Manager calls the function every time rotation triggers.

You can specify the "generic" template you must fully implement. Or you can choose one of the templates that completely implement a rotation strategy for a certain database or service, and use it as a starting point to customize the function to meet your needs.

Minimum permissions

To run the commands that enable and configure rotation, you must have the following permissions:

- `serverlessrepo:CreateCloudFormationChangeSet` – To create the AWS CloudFormation change set that configures and creates the Lambda rotation function.
- `cloudformation:ExecuteChangeSet` – To run the AWS CloudFormation change set that creates and configures the Lambda rotation function.
- `lambda:AddPermission` – Add the required permissions to the Lambda rotation function after you create. it
- `lambda:InvokeFunction` – Attach the rotation function to the secret.
- `lambda:UpdateFunctionConfiguration` – Allow the console to update the VPC configuration of the Lambda function so it can communicate with a database or service that resides in a VPC.
- `secretsmanager:RotateSecret` – Configure and trigger the initial rotation.

You can grant all of these permissions to an IAM user or role by attaching the SecretsManagerReadWrite AWS managed policy.

The following commands apply the generic `SecretsManagerRotationTemplate` to your Lambda function. This template comes from the AWS Serverless Application Repository, and used by AWS CloudFormation to automate most of the steps for you. For the complete set of templates and the ARN that you must specify, see AWS Templates You Can Use to Create Lambda Rotation Functions (p. 152).

Use the ARN of the generic template and enter it exactly as shown:

```
arn:aws:serverlessrepo:us-east-1:297356227824:applications/SecretsManagerRotationTemplate
```

If the database or service using your credentials resides in a VPC provided by Amazon VPC, then you must include the command in Step 5. This command configures the function to communicate with that VPC. If you do not have a VPC, then you can skip the command.

Creating a Lambda rotation function as a generic template for customization

1. The first command sets up an AWS CloudFormation change set based on the template provided by Secrets Manager. You provide two parameters to the template: the Secrets Manager endpoint URL and the name for the Lambda rotation function produced by the template.

```
$ aws serverlessrepo create-cloud-formation-change-set \
          --application-id arn:aws:serverlessrepo:us-east-1:297356227824:applications/
SecretsManagerRotationTemplate \
          --stack-name MyLambdaCreationStack \
          --parameter-overrides '[{"Name":"endpoint","Value":"https://
secretsmanager.region.amazonaws.com"},
{"Name":"functionName","Value":"MySecretsManagerRotationFunction"}]' --capabilities
 CAPABILITY_IAM CAPABILITY_RESOURCE_POLICY
{
    "ApplicationId": "arn:aws:serverlessrepo:us-east-1:297356227824:applications/
SecretsManagerRDSMySQLRotationSingleUser",
    "ChangeSetId": "arn:aws:cloudformation:region:123456789012:changeSet/EXAMPLE1-90ab-
cdef-fedc-ba987EXAMPLE/EXAMPLE2-90ab-cdef-fedc-ba987EXAMPLE",
    "StackId": "arn:aws:cloudformation:region:123456789012:stack/aws-serverless-
repository-MyLambdaCreationStack/EXAMPLE3-90ab-cdef-fedc-ba987EXAMPLE"
}
```

2. The next command runs the change set you just created. The `change-set-name` parameter comes from the `ChangeSetId` output of the previous command. This command produces no output:

```
$ aws cloudformation execute-change-set --change-set-name
 arn:aws:cloudformation:region:123456789012:changeSet/EXAMPLE1-90ab-cdef-fedc-
ba987EXAMPLE/EXAMPLE2-90ab-cdef-fedc-ba987EXAMPLE
```

3. Next you must locate the name of the Lambda function the previous command created for you.

```
$ aws lambda list-functions
  {
    ...
    "FunctionName": "MySecretsManagerRotationFunction",
    "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MySecretsManagerRotationFunction",
    ...
  }
```

4. The following command grants Secrets Manager permission to call the function .

```

$ aws lambda add-permission \
          --function-name MySecretsManagerRotationFunction \
          --principal secretsmanager.amazonaws.com \
          --action lambda:InvokeFunction \
          --statement-id SecretsManagerAccess
{
    "Statement": "{\"Sid\":\"SecretsManagerAccess\",\"Effect\":\"Allow\",\"Principal
\":{\"Service\":\"secretsmanager.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
\",\"Resource\":\"arn:aws:lambda:us-west-2:123456789012:function:aws-serverless-
repository-SecretsManagerRDSMySQLRo-10WGBDAXQ6ZEH\"}"
}
```

5. If you run a database in a VPC, you need the following command. If you don't have a VPC, skip this command. This command configures the Lambda rotation function to run in the VPC running your Amazon RDS DB instance Look up the VPC information for your Amazon RDS DB instance by using either the Amazon RDS console, or by using the `aws rds describe-instances` CLI command. Then add the information in the following command and execute it.

```
$ aws lambda update-function-configuration \
          --function-name arn:aws:lambda:us-
west-2:123456789012:function:MySecretsManagerRotationFunction \
          --vpc-config SubnetIds=<COMMA SEPARATED LIST OF VPC SUBNET
 IDS>,SecurityGroupIds=<COMMA SEPARATED LIST OF SECURITY GROUP IDs> \

```

6. If the VPC with your database instance and Lambda rotation function without an internet access, then you must configure the VPC with a private service endpoint for Secrets Manager. This enables the rotation function to access Secrets Manager at an endpoint within the VPC.

```
$ aws ec2 create-vpc-endpoint --vpc-id <VPC ID> \
                              --vpc-endpoint-type Interface \
                              --service-name com.amazonaws.<region>.secretsmanager \
                              --subnet-ids <COMMA SEPARATED LIST OF VPC SUBNET IDS> \
                              --security-group-ids <COMMA SEPARATED LIST OF SECURITY
 GROUP IDs> \
                              --private-dns-enabled
```

7. At this point, you can enter your code into your Lambda rotation function .

   Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
8. Customize the code to implement your chosen rotation scenario. For details, see Understanding and Customizing Your Lambda Rotation Function (p. 103).

9. Finally, you can apply the rotation configuration to your secret and perform the initial rotation. Specify the number of days between successive rotations with the `--rotation-rules` parameter, and set `AutomaticallyAfterDays` to the desired number of days.

```
$ aws secretsmanager rotate-secret \
          --secret-id production/MyAwesomeAppSecret \
          --rotation-lambda-arn arn:aws:lambda:us-
west-2:123456789012:function:MySecretsManagerRotationFunction \
          --rotation-rules AutomaticallyAfterDays=7
```

The secret rotates once immediately, and then begins to rotate as frequently as you specified.

Enabling Rotation for a Secret for Another Database or Service

To configure rotation of a secret for a database other than the supported RDS databases or some other service, you must manually perform a few extra steps. Primarily, you must create and provide the code for the Lambda rotation function.

Warning
Configuring rotation causes the secret to rotate once as soon as you store the secret. Before you do this, you must make sure that all of your applications that use the credentials stored in the secret are updated to retrieve the secret from AWS Secrets Manager. The old credentials might not be usable after the initial rotation. Any applications that you fail to update break as soon as the old credentials are no longer valid.

You must have already created your Lambda rotation function. If you haven't yet created the function, then perform the steps in Rotating AWS Secrets Manager Secrets for Other Databases or Services (p. 98). Return to this procedure when the function is created and ready to associate with your secret.

Prerequisites: Network Requirements to Enable Rotation

To successfully enable rotation, you must have your network environment configured correctly.

- **The Lambda function must be able to communicate with your database or service.** If your database or service is running on an Amazon EC2 instance in a VPC, then we recommend that you configure your Lambda function to run in the same VPC. This enables direct connectivity between the rotation function and your service. To configure this, on the Lambda function's details page, scroll down to the **Network** section and choose the **VPC** from the drop-down list to match the one the instance with your service is running in. You must also make sure that the EC2 security groups attached to your instance enable communication between the instance and Lambda.
- **The Lambda function must be able to communicate with the Secrets Manager service endpoint.** If your Lambda rotation function can access the internet, either because the function isn't configured to run in a VPC, or because the VPC has an attached NAT gateway, then you can use any of the available public endpoints for Secrets Manager. Alternatively, if your Lambda function is configured to run in a VPC that doesn't have internet access at all, then you can configure the VPC with a private Secrets Manager service endpoint (p. 74).

To enable and configure rotation for a secret for another database or service

Follow the steps under one of the following tabs:

Using the AWS Management Console

Minimum permissions
To enable and configure rotation in the console, you must have these permissions:

- `secretsmanager:ListSecrets` – To see the list of secrets in the console.
- `secretsmanager:DescribeSecrets` – To access the details page for your chosen secret.
- `secretsmanager:RotateSecret` – To configure or trigger rotation.

1. Sign in to the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Choose the name of the secret that you want to enable rotation for.
3. In the **Configure automatic rotation** section, choose **Enable automatic rotation**. This enables the other controls in this section.
4. For **Select rotation interval**, choose one of the predefined values—or choose **Custom**, and then type the number of days you want between rotations.

   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.
5. For **Choose an AWS Lambda function**, choose your rotation function from the drop-down list. If you haven't yet created the function, perform the steps in Rotating AWS Secrets Manager Secrets for Other Databases or Services (p. 98). Return and perform this step when the function is created and ready to associate with your secret.

Using the AWS CLI or AWS SDKs

Minimum permissions
To create a Lambda function by using the console, you must have these permissions:

- `lambda:CreateFunction` – To create the function in AWS Lambda.
- `lambda:InvokeFunction` – To attach the rotation function to the secret.
- `secretsmanager:DescribeSecrets` – To access the secret details page.
- `secretsmanager:RotateSecret` – To attach the rotation function to the secret or to trigger rotation.

You can use the following commands to enable and configure rotation in Secrets Manager:

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

Example

The following is an example CLI command that performs the equivalent of the console-based secret creation in the **Using the AWS Management Console** tab. It sets the rotation interval to 30 days, and specifies the Amazon Resource Name (ARN) of a second secret that has permissions to change this secret's credentials on the database.

```
$ aws secretsmanager rotate-secret --secret-id production/MyAwesomeAppSecret
 --automatically-rotate-after-days 30 --rotation-lambda-arn
 arn:aws:secretsmanager:region:accountid:secret:production/MasterSecret-AbCdEf
{
    "ARN": "arn:aws:secretsmanager:region:accountid:secret:production/
MyAwesomeAppSecret-AbCdEf",
    "Name": "production/MyAwesomeAppSecret",
    "VersionId": "EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE"
}
```

The `ClientRequestToken` parameter isn't required because we're using the AWS CLI, which automatically generates and supplies one for us. The output includes the secret version ID of the new version that's created during the initial rotation. After rotation is completed, this new version has the staging label `AWSCURRENT` attached, and the previous version has the staging label `AWSPREVIOUS`.

Understanding and Customizing Your Lambda Rotation Function

For details about Lambda rotation functions , see Overview of the Lambda Rotation Function (p. 104).

If you choose one of the supported databases (p. 4) for your secret type, AWS Secrets Manager creates and configures the Lambda rotation function for you. You can enable rotation for those databases by following the steps in Enabling Rotation for an Amazon RDS Database Secret (p. 76). However, if you want to create a custom Lambda rotation function for another service, then you must follow the steps in Enabling Rotation for a Secret for Another Database or Service (p. 101).

This section describes in detail how the Lambda function operates and how you configure the function to successfully rotate your secrets.

Important
The Lambda function and the Secrets Manager secret that invokes the function must be in the same AWS Region. If located different regions, you receive a "Lambda does not exist" error message when you try to add the Amazon Resource Name (ARN) of the function to the secret metadata.

Consider the following scenarios to consider when creating your own Lambda function for rotation. You apply a scenario based on the features supported by the authentication system protecting the secured resource, and by your security concerns.

- **You can change only the password for a single user.** A common scenario for services owned by someone other than the user accessing the service. The owner of the service allows the customer to create ***one*** user account, often with a user email address as the user name , or at least as a uniqueness key. In this scenario, the service typically allows the user to change the password as often as required. But, the scenario doesn't allow the user to create additional users, and often doesn't allow changing the user name.

  Users typically have the ability to change their password, and don't require a separate user with administrator permissions to perform the password change. However, because you change the password for the single, active user, clients with access to the service with this user might temporarily fail to sign in while processing the password change .

  The potential for downtime exists between the time you change the password and when the clients all receive notified to use the newer version of the password. This should typically be a few seconds, and you must allow for the time in your application code using the secret. Be sure to enable retries with some delay in between to be tolerant of this short-term outage during a rotation.
- **You can create two users you can alternate between.** In this scenario, you can create one user the rotation process clones to create two users with equal access to the secured resource. The rotation process alternates between the two users. Secrets Manager changes the password, and tests the

"inactive" one while your users continue to access the database or service by using the credentials in the "active" secret.

While your clients access the secured resource with one user name by querying for the version with the default staging label `AWSCURRENT` attached, the rotation function changes the password on the inactive second user. The rotation function stores the updated password in a new version of the secret with the staging label `AWSCURRENT`. After testing, you move the staging label `AWSCURRENT` to the new version pointing to the alternate user and the new password. All of the clients immediately begin accessing the secured resource with the alternate user and the updated password.

When the time arrives for the next rotation, you change the password on the original user account now idle. This creates another new version of the secret and repeats the cycle.

This scenario requires a second secret pointing to an administrator or super user with permissions to change the password on both users.

- **You can create new credentials for a single user.** Some systems enable you to create a single user with multiple sets of access credentials. Each access credential provides a complete set of credentials and operates independently of the other. You can delete and recreate the first access credential while Secrets Manager uses the second access credential. Then you can switch all of your clients over to the new first access credential. The next time you rotate, you delete and recreate the second access credential while customers to continue to use the second.

For additional details and instructions on how to configure each scenario, see the following topics:

- Overview of the Lambda Rotation Function (p. 104)
- Rotating AWS Secrets Manager Secrets for One User with a Single Password (p. 107)
- Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110)
- Rotating AWS Secrets Manager Secrets For One User Supporting Multiple Credentials (p. 113)

Overview of the Lambda Rotation Function

AWS Secrets Manager uses an AWS Lambda function to perform the actual rotation of a secret. If you use your secret for one of the supported Amazon RDS databases (p. 4), then Secrets Manager provides the Lambda function for you. And Secrets Manager automatically customizes the function to meet the requirements of the specified database. If you use your secret for another service, then you must provide the code for the Lambda function.

When a configured rotation schedule or a manual process triggers rotation , Secrets Manager calls the Lambda function several times, each time with different parameters. The Lambda function performs several tasks throughout the process of rotating a secret. The `Step` parameter in the request specifies the task performed for each request.

Secrets Manager invokes the Lambda function with the following JSON request structure of parameters:

```
{
  "Step" : "request.type",
  "SecretId" : "string",
  "ClientRequestToken" : "string"
}
```

The following describes the parameters of the request:

- **Step** – Specifies the part of the rotation function behavior to invoke. Each of the different values identifies a step of the rotation process. The following section The Steps of the Lambda Rotation Function (p. 105) explains each step in detail. The separation into independently invoked steps enables the AWS Secrets Manager team to add additional functionality to occur between steps.

- **secretId** – The ID or Amazon Resource Name (ARN) for the secret to rotate. Secrets Manager assigns an ARN to every secret when you initially create the secret. The version rotating automatically becomes the **default** version labeled `AWSCURRENT`.
- **clientRequestToken** – A string Secrets Manager provides to the Lambda function. You must pass the string to any Secrets Manager APIs you call from within the Lambda function. Secrets Manager uses this token to ensure the idempotency of requests during any required retries caused by failures of individual calls. This value is a UUID-type value to ensure uniqueness within the specified secret. This value becomes the `SecretVersionId` of the new version of the secret.

The same `secretId` and `clientTokenRequest` invoke every step. Only the `Step` parameter changes with each call. This prevents you from storing any state between steps. The parameters provide all of the necessary information—or as part of the information in the versions accessed with the `AWSPENDING` or `AWSCURRENT` labels.

For descriptions of the specific tasks to be performed in each step for the different rotation strategies, see the following topics:

- Rotating AWS Secrets Manager Secrets for One User with a Single Password (p. 107)
- Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110)
- Rotating AWS Secrets Manager Secrets For One User Supporting Multiple Credentials (p. 113)

The Steps of the Lambda Rotation Function

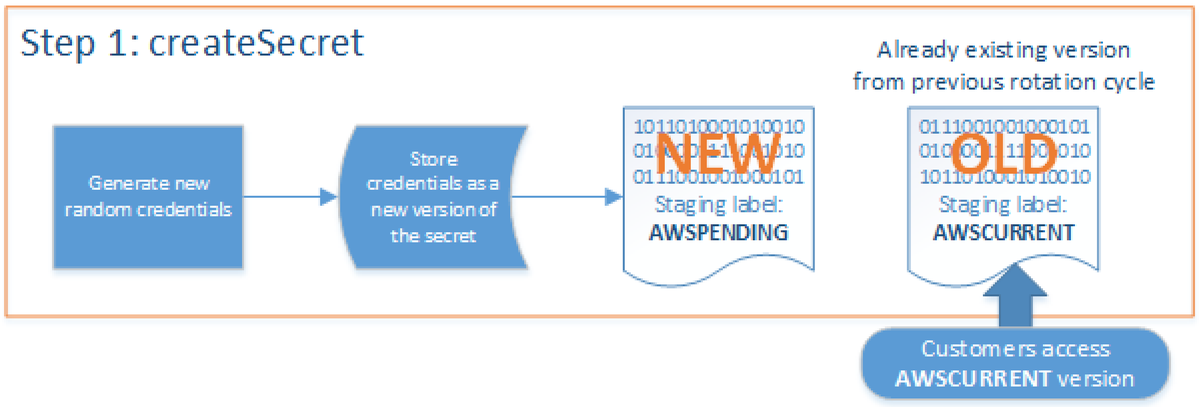
The functionality built into the Lambda rotation function breaks down into distinct steps. The `Step` parameter invokes each step by calling the function with one of the parameter values.

In this release of Secrets Manager, Secrets Manager calls the steps automatically in sequence. As soon as one step ends, Secrets Manager immediately calls the Lambda function to invoke the next step.

When you specify a secret for one of the supported Amazon RDS databases, Secrets Manager uses a standard Lambda function to rotate the secret. Secrets Manager provides the Lambda function , but you can modify it the Lambda function to meet your organizational specific rotation requirements.

The createSecret Step

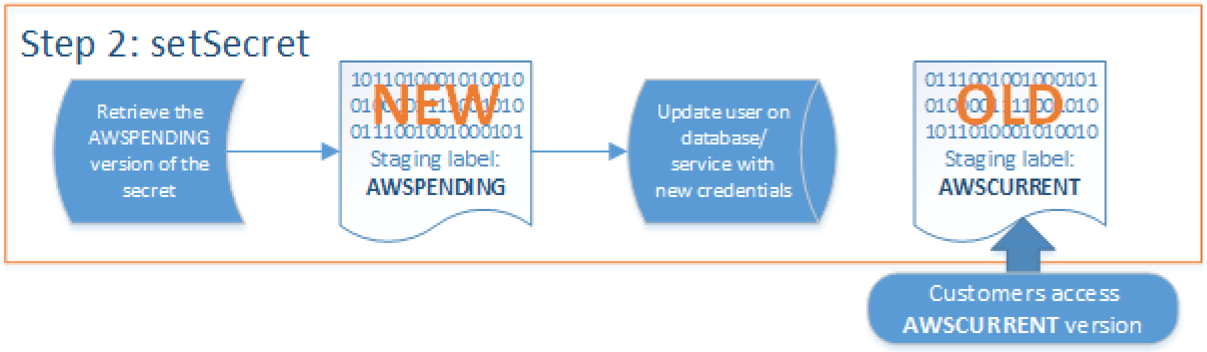
In this step, the Lambda function generates a new version of the secret. Depending on your scenario, this can be as simple as just generating a new password. Or you can generate values for a completely new set of credentials, including a user name and password appropriate for the secured resource. Secrets Manager stores these values as a new version of the secret. Secrets Manager clones the other values in the secret that don't need to change, such as the connection details, from the existing version of the secret. Secrets Manager then labels the new version of the secret with the staging label `AWSPENDING` to mark it as the **in-process** version of the secret.


Step 1: createSecret
Already existing version from previous rotation cycle
Generate new random credentials
Store credentials as a new version of the secret
NEW
Staging label: AWSPENDING
OLD
Staging label: AWSCURRENT
Customers access AWSCURRENT version

The setSecret Step

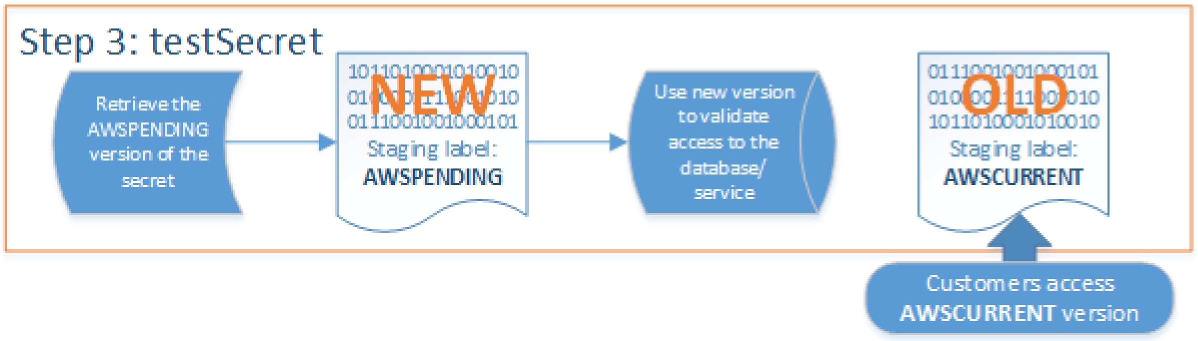
In this step, the rotation function retrieves the version of the secret labeled `AWSPENDING` from Secrets Manager, the version you just created in the previous step. The rotation function then invokes the database or service identity service to change the existing password, or to create new credentials to match the new ones in the secret. If you create a new user, then the function must clone the permissions from the previous user. Then the new user can continue to perform as needed within your custom application.

To change a password or to create new credentials in the database or service authentication system, you must allow the Lambda function to perform these tasks. Considered administrative tasks, the tasks require permissions you typically don't give users. We recommend you use a *second* set of credentials with permissions to change the password or create new users for the main secret, as dictated by your rotation strategy. We refer to these credentials as the *master secret*, and Secrets Manager stores them as a separate secret from the main secret. The rotation function stores the ARN of this master secret in the main secret for use by the rotation function. The master secret should never be accessed by your end user custom application. Only the Lambda rotation function of can access the main secret, to update or create new credentials in the database when rotation occurs.


Step 2: setSecret
Retrieve the AWSPENDING version of the secret
NEW
Staging label: AWSPENDING
Update user on database/ service with new credentials
OLD
Staging label: AWSCURRENT
Customers access AWSCURRENT version

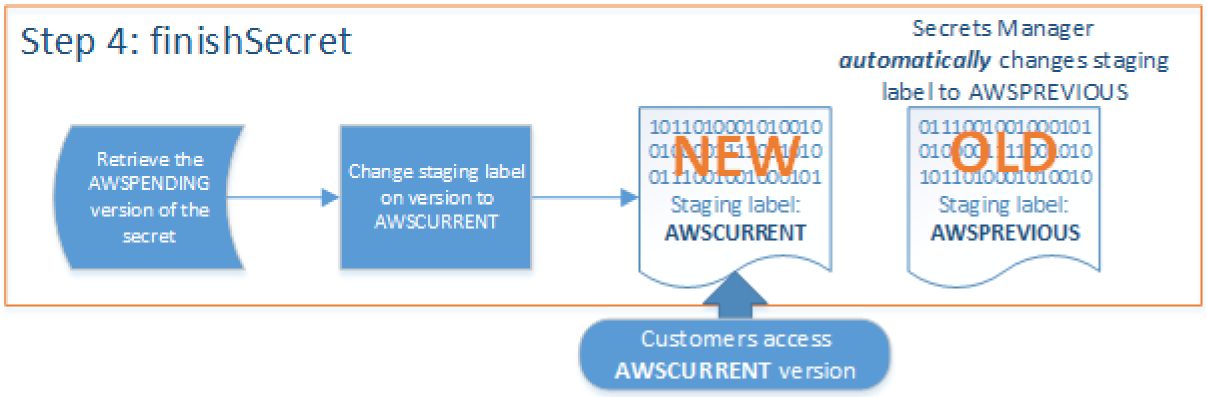
The testSecret Step

This step of the Lambda function verifies the `AWSPENDING` version of the secret by using it to access the secured resource in the same way your custom application accesses the secured resource. If the application needs read-only access to the database, then the function should verify the test reads succeed. If the application must write to the database, then the function should perform some test writes to verify that level of access.


Step 3: testSecret
Retrieve the AWSPENDING version of the secret
NEW
Staging label: AWSPENDING
Use new version to validate access to the database/ service
OLD
Staging label: AWSCURRENT
Customers access AWSCURRENT version

The finishSecret Step

This step performs any resource-specific finalization on this version of the secret. When complete, the last step requires the Lambda function to move the label `AWSCURRENT` from the current version to this new version of the secret so your clients start using it. You can also remove the `AWSPENDING` label, but technically required. At this point, the basic rotation completes. All of your clients use the new version of the secret. The old version receives the `AWSPREVIOUS` staging label, and available for recovery as the last known good version of the secret, if needed. The old version with the `AWSPREVIOUS` staging label no longer has any staging labels attached, so Secrets Manager considers the old version deprecated and subject to deletion.


Step 4: finishSecret
Secrets Manager *automatically* changes staging label to AWSPREVIOUS
Retrieve the AWSPENDING version of the secret
Change staging label on version to AWSCURRENT
NEW
Staging label: AWSCURRENT
OLD
Staging label: AWSPREVIOUS
Customers access AWSCURRENT version

Rotating AWS Secrets Manager Secrets for One User with a Single Password

You can configure AWS Secrets Manager to automatically rotate the secret for a secured resource. In this topic, we describe configuring rotation for a system that allows you to create a single user with a single password. You can change the password for the user when needed. This scenario provides simplicity, but doesn't provide the most highly available solution. Clients can continue to access the secured resource while the password changes. This can possibly result in some "access denied" situations.

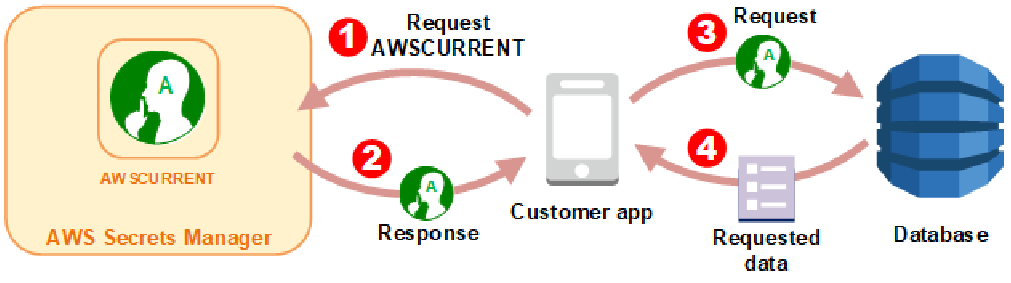
The time lag that can occur between the change of the actual password and the change in the corresponding secret that tells the client which password to use incurs a risk. This risk can increase if you host the secured resource on a "server farm" where the password change takes time to propagate to all member servers. However, with an appropriate retry strategy, this risk can be significantly mitigated.

A common scenario for services owned by someone other than the user accessing the service. The owner of the service allows the customer create ***one*** user account—often with information such as the user email address as the user name, or at least as a uniqueness key. The service typically allows the user to change the password as often as required. But, the service doesn't allow the user to create additional users or to change the user name.

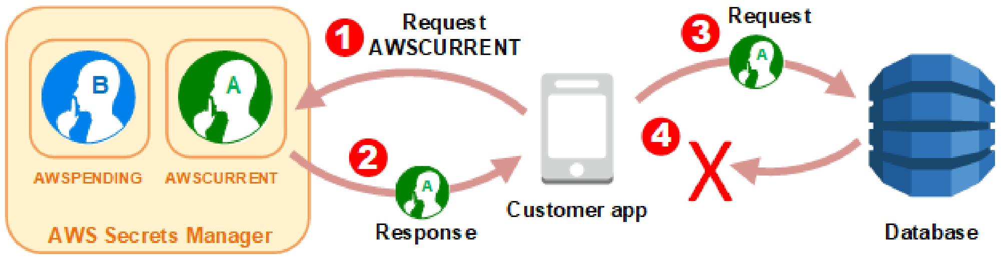
How Rotation Uses Labels to Manage a Single User with Changing Passwords

The following explains this scenario in more detail:

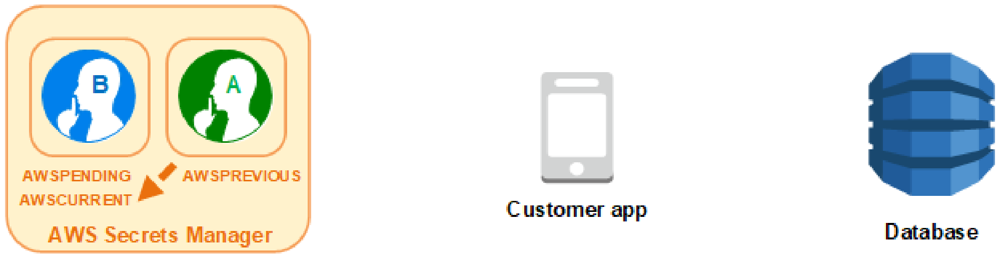
a. The scenario starts with an application accessing the secured resource, the database, by using the credentials stored in a secret with a single version "A". The A version of the secret has the staging label `AWSCURRENT` attached. The application retrieves the secret by requesting the version with the staging label `AWSCURRENT`, Steps 1 and 2 in the following graphic. The application then uses the credentials in that version of the secret to access the database, Step 3, to retrieve the data the application needs, Step 4.



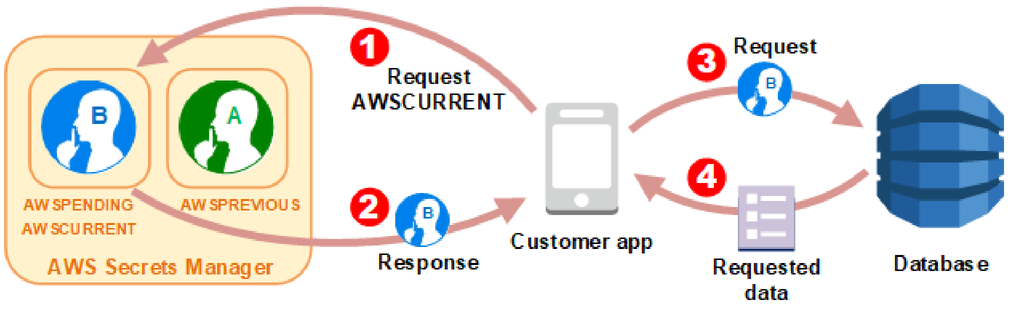
b. The secret rotation process creates a new version "B" of the secret, not a new secret but a new version of the same secret. This version B secret initially has the staging label `AWSPENDING` attached by the rotation process. Secret version B receives a new generated password. As soon as Secrets Manager successfully stores the secret, the rotation process changes the password for the user in the database authentication system. At this point, between changing the password on the database and moving the label to the new version of the secret, client sign-on failures can occur in Step 4. Because of this risk, it's vital that the rotation process immediately proceed to the next step.



c. Secret version B becomes the live password, and the rotation process moves the `AWSCURRENT` staging label from the A version to the new B version of the secret. This also automatically moves the `AWSPREVIOUS` staging label to the version previously labeled with the `AWSCURRENT` staging label. This allows the secret to act as "last known good" in case there's a need for recovery.



d. The next request from the custom application now receives the B version of the secret because B now has the `AWSCURRENT` label. At this point, the customer application depends on the new version of the secret.



Configuring Rotation to Change Passwords Only

To configure a rotation mechanism for an authentication system that allows you to have only one user, follow the steps in this procedure:

Configuring rotation for password-only rotation

1. Create the single user in the authentication system protecting the secured resource. Make note of the password.
2. Create a Secrets Manager secret to store the details of the credentials you created in the previous step:

   a. Sign in to the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.

   b. Choose **New secret**.

   c. For **Select secret type**, choose the option that best fits your service. Then configure the details for your database or service, including the user name and the initial password.

   d. For **AWS KMS Encryption Key**, choose the customer master key (CMK) to use to encrypt this secret, or leave it set to the **DefaultMasterKey** for the account. To use the default key, the credentials accessing the secret must be from the same account that owns the secret. If the user credentials reside in a different account, then you must create and specify the Amazon Resource Namer (ARN) of a custom CMK.

   e. For **Select rotation period**, choose or type the number of days between rotations.

      Note
      When you use the Secrets Manager console to configure rotation, by default, Secrets Manager automatically enables expiration and sets to the number of days in a rotation cycle + 7.

   f. For **What credentials can rotate this secret?**, choose **Use the same credentials**.

      Note
      This example scenario assumes the user can change their password, and you can't use a second user with permissions to change the password on the first user.

   g. Choose **Next step**.

   h. Type a **Secret name** and optional **Description**. You can also choose to add tags.

   i. Choose **Store secret**.
3. Examine your new secret to get the ARN of the Lambda rotation function.

   a. On the **Secrets** list page, choose the name of the secret you created in step 2.

   b. In the **Secret details / Secret rotation** section, choose the ARN of the rotation function to open it in Lambda.

   c. Customize the rotation function to meet your specific requirements. You can use the following requirements for each step as the basis for writing the function.

      - **createSecret step**:
        - Retrieve the `AWSCURRENT` version of the secret by using the `GetSecretValue` operation.
        - Extract the protected secret text from the `SecretString` field, and store it in a structure you can modify.
        - Generate a new password by using an algorithm to generate passwords with the maximum length and complexity requirements supported by the protected resource.
        - Overwrite the `password` field in the structure with the new one you generated in the previous step. Keep all other details, such as `username` and the connection details the same.

    - Store the modified copy of the secret structure by passing it as the `SecretString` parameter in a call to `PutSecretValue`. Secrets Manager labels the new version of the secret with `AWSPENDING`.
- **setSecret step**:
    - Retrieve the `AWSPENDING` version of the secret by using the `GetSecretValue` operation.
    - Issue commands to the secured resource authentication system to change the existing user password to the one stored in the new `AWSPENDING` version of the secret.
- **testSecret step**:
    - Retrieve the `AWSPENDING` version of the secret by using the `GetSecretValue` operation.
    - Issue commands to the secured resource to attempt to access it by using the credentials in the secret.
- **finishSecret step**:
    - Move the label `AWSCURRENT` to the version labeled `AWSPENDING`. This automatically also moves the staging label `AWSPREVIOUS` to the secret you just removed `AWSCURRENT` from.
    - (Optional) Remove the label `AWSPENDING` from its version of the secret.

Important
Remember that in this scenario, your users cannot continue operating with the old version while you create and verify the new version. Therefore, between the time in the `setSecret` phase when you change the password, and the time in the `finishSecret` phase when you move the `AWSCURRENT` label to the new version, your clients may use the wrong credentials and get access denied errors. Be sure to include reasonable retry functionality in your client application for this situation to help mitigate this risk.

Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users

You can configure AWS Secrets Manager to automatically rotate the secret for a secured resource.

This topic describes configuring rotation for a system that allows you to create and alternate between two users you can change the password, when needed. This enables you to remove the potential for downtime that occurs in the scenario where you're limited to only one user account (p. 107). In this case, you change more than just the password in a secret version. Each version must also capture the change in the user, and alternate between each user with each rotation cycle.

If the administrator enables you to create a third "master" user with the elevated permissions to change the password for the first two users, we recommend you do so. This provides more security than allowing the users to have permissions to change their own passwords. If you do configure the scenario this way, then you need to create a second secret used to change the password of the users alternated in the first secret. Create that "master" secret first, so that you can use it for reference when you configure the "user" secret.

Secrets Manager provides templates to implement the following scenario which deactivates the current credentials, but doesn't immediately delete them. Instead, Secrets Manager marks the version of the secret with current credentials with the staging label `AWSPREVIOUS`. This preserves those credentials for one additional rotation cycle as the "last known good" credentials. Secrets Manager keeps the credentials available for recovery if something happens to the current credentials.

Secrets Manager only removes the credentials after the second rotation cycle when the rotation function brings the user back into active service, as described later in this topic. This means if you want a given set of credentials to only be valid for a given number of days, we recommend you set the rotation period to one-half of that minus one. This allows the two rotation cycles to both complete and the credentials to become fully deprecated within the specified time frame.

For example, if you have a compliance-mandated maximum credential lifetime of 90 days, then we recommend you set your rotation interval to 44 days (90/2 - 1 = 44). At day 0, Secrets Manager creates the new credentials in a rotation and labels them as `AWSCURRENT`. On day 44, the next rotation demotes the secret with those credentials to `AWSPREVIOUS`, and clients stop actively using them. On day 88, the next rotation removes all staging labels from the version. Secrets Manager fully deprecates the version is fully deprecated at this point and recycles the user on the database with a new password, which starts the cycle over again.

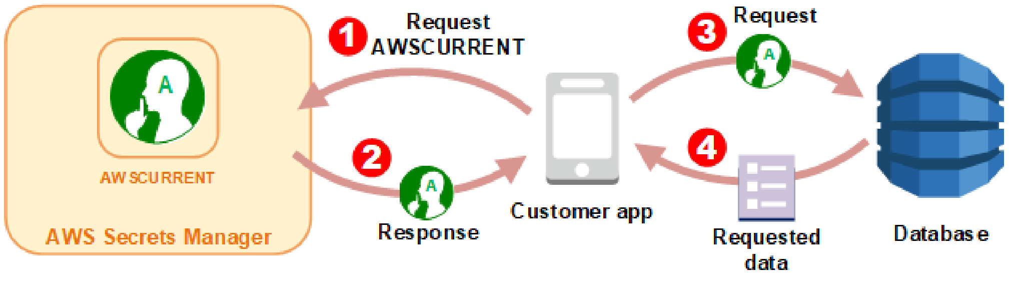
How Rotation Uses Labels to Alternate Between Two Users

Using staging labels allows Secrets Manager to switch back and forth between two users. One, labeled `AWSCURRENT`, actively used by the clients. When the Lambda function rotation triggers, the rotation process assigns a new password to the inactive second user. Secrets Manager stores those credentials in a new version of the secret. After verifying access with the credentials in the new secret version works, the rotation process moves the `AWSCURRENT` label to the new version, which makes it the active version. The next time triggers, the roles of Secrets Manager reverses the two user accounts.

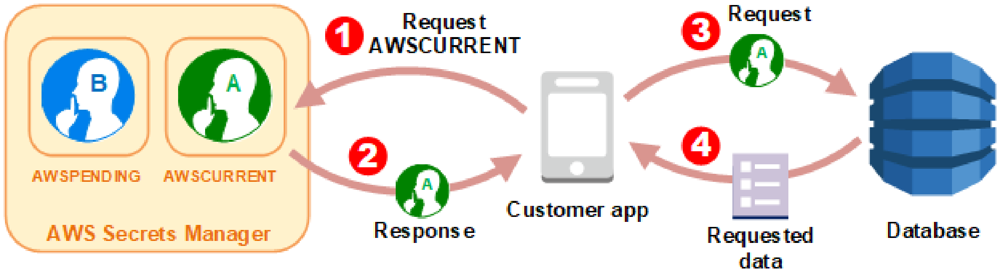
Basic Example Scenario

The following explains this scenario in more detail:

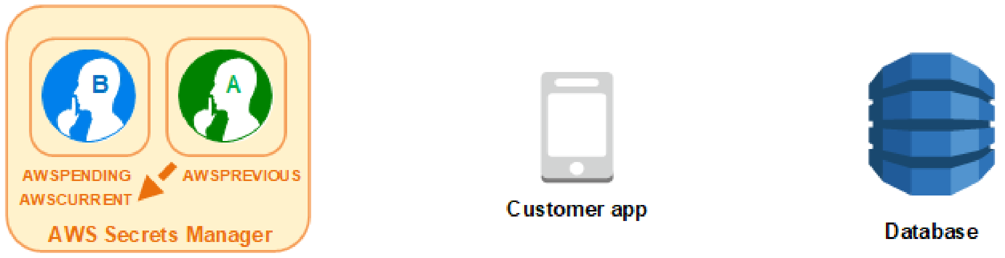
a. The scenario begins with an application accessing the database by using a secret with a single version "A". This A version of the secret has the label `AWSCURRENT` attached and contains credentials for the first of the two users. The application retrieves the secret by requesting the version labeled `AWSCURRENT` (steps 1 and 2 in the following graphic). The application then uses the credentials in that version of the secret to access the database (step 3) to retrieve the data it needs (step 4).



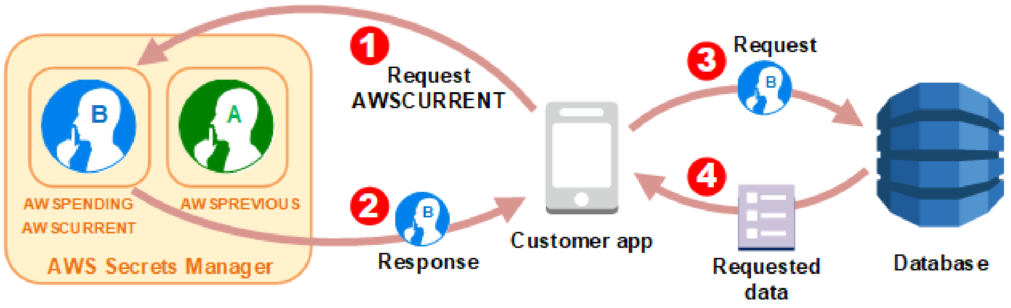
b. During the very first rotation, the process creates a new "B" version of the secret, **not a new secret**, but a **new version of the same secret**. The process clones all details of the A version, except that it switches the user name to the second user and generates a new password. Then the rotation process updates the second user with this new password. This secret version B initially has the label `AWSPENDING`, which is attached by the rotation process. Because you programmed the custom app to always request the `AWSCURRENT` label and the label hasn't changed, the application continues to retrieve and use the original A version of the secret for credentials to access the database.



c. After testing the version B secret to ensure it works, the rotation process moves the `AWSCURRENT` label from the A version, and attaches the label to the new B version of the secret. This also automatically moves the `AWSPREVIOUS` staging label to the previous version with the `AWSCURRENT` staging label. This allows it to act as "last known good" in case you need recovery. The `AWSPREVIOUS` version of the secret continues to be available for recovery for one additional rotation cycle. Secrets Manager deprecates the secret in the next cycle.



d. The next request from the custom application now receives the B version of the secret because B now has the `AWSCURRENT` label. The custom application uses the second user with the new password to access the database.



Configuring Rotation Between Two Users

If you use your secret for one of the supported Amazon RDS databases (p. 4), follow the procedures in Enabling Rotation for an Amazon RDS Database Secret (p. 76).

If instead you want to configure rotation for another service or database, create your Lambda rotation function and customize it using these instructions:

1. Create two users in your database or service. Make note of the user names and passwords you use. Make sure each user has the ability to change their password.
2. Create a secret with the credentials of the first user.

Follow the steps at Rotating AWS Secrets Manager Secrets for Other Databases or Services (p. 98) to create a generic Lambda rotation function template that you can customize. When you get to step 7, customize your function by using the following information.

Use the following logic as the basis for writing your rotation function:

- **createSecret phase**:
  - Retrieve the `AWSCURRENT` version of the secret by using the `GetSecretValue` operation.
  - Extract the protected secret text from the `SecretString` field, and place it in a file structure you can use.
  - Look at the `username` field and determine the alternate user name.
  - Determine if a user with the alternate user name exists on the database. If it doesn't, this must be the first time through the rotation process. Clone the current user and the permissions.
  - Set the `username` entry in the copy of the secret structure to the alternate user name.
  - Generate a new password with the maximum length and complexity requirements supported by the protected resource.

  - Set the `password` entry in the copy of the secret structure to the new password.
  - Store the modified copy of the structure into the secret by passing it as the `SecretString` parameter in a call to `PutSecretValue`. Secrets Manager labels the new version of the secret as `AWSPENDING`.
- **setSecret phase**:
  - Retrieve the `AWSPENDING` version of the secret by using the `GetSecretValue` operation.
  - Extract the user name and password from the `SecretString` field.
  - Issue commands to the secured resource authentication system to change the specified user password to that value.
- **testSecret phase**:
  - Retrieve the `AWSPENDING` version of the secret by using the `GetSecretValue` operation.
  - Attempt to access the secured resource to attempt access by using the credentials in the secret.
- **finishSecret phase**:
  - Move the label `AWSCURRENT` to the version with the label `AWSPENDING`.
  - (Optional) Remove the label `AWSPENDING` from its version of the secret.

Note
In this scenario, there's less chance of users getting an error during the rotation of the secret. Users continue to use the old version, while you configure the new version. Only after Secrets Manager tests the new version, you switch the label to point to the new version and the clients start using it. However, because some servers reside in server farms with propagation delays when changing passwords, you should ensure you include a reasonable delay, most likely in the `setSecret` step before you test, to ensure that the password has time to propagate to all servers. Also, be sure to include reasonable retry functionality in your client application for this situation, to help mitigate this risk.

Rotating AWS Secrets Manager Secrets For One User Supporting Multiple Credentials

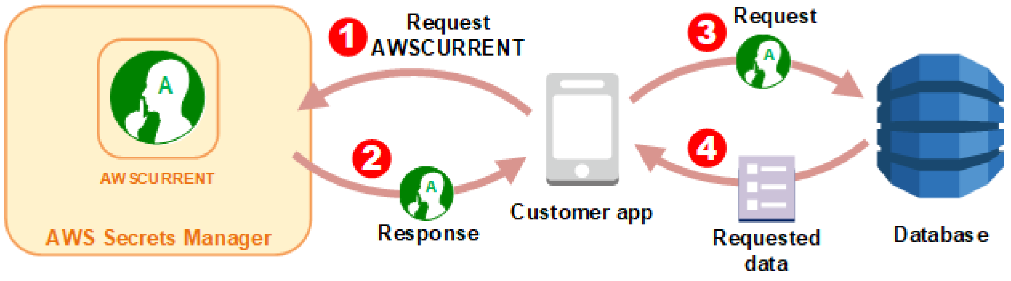
You can configure AWS Secrets Manager to automatically rotate the secret for a secured resource. This topic covers configuring rotation for an authentication system that allows you to create a single user with at least two credential sets.

As a best practice, we recommend you set up a second "master" secret to store the credentials of a different user with permissions to delete and create credentials for the main user. This enables you to limit the permissions you grant to the main user to those required by your application. By doing so, you can offload the administrative tasks to the second user, which the end users cannot access. The rotation function of the main secret accesses the second user to delete the old access key and create a new one.

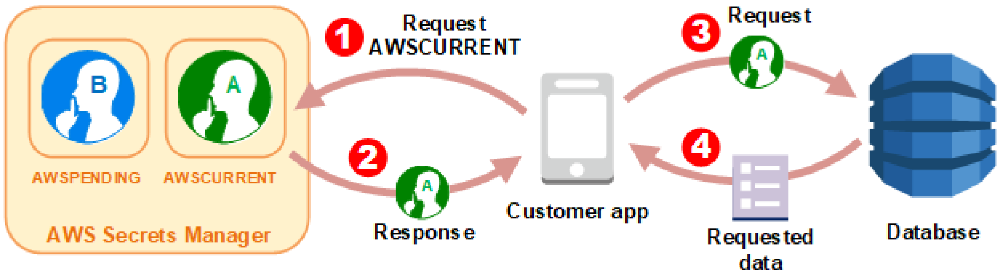
How Rotation Uses Labels to Manage a Single User with Multiple Credentials

The following example explains this scenario in more detail. It uses the example of a service that enables a user to have two separate "API keys" generated by the service and not generated by your rotation function:

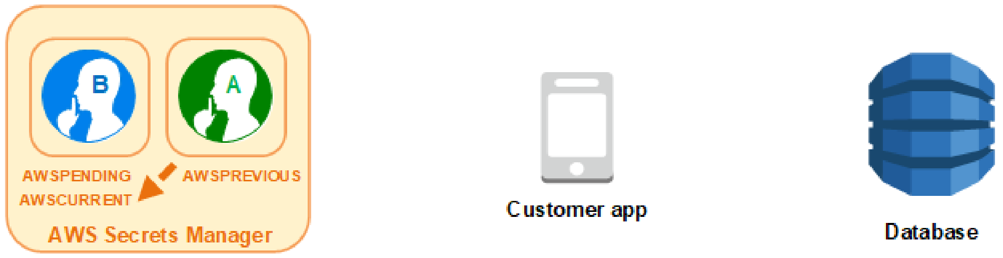
a. The scenario begins with an application accessing the secured resource,the database, by using one of the API keys stored in a secret with a single version "A". This A version of the secret has the staging label `AWSCURRENT` attached. The application retrieves the secret by requesting the version with the staging label `AWSCURRENT` , steps 1 and 2. The application then uses the API key in that version of the secret to access the database (step 3) to retrieve the data the application needs (step 4).



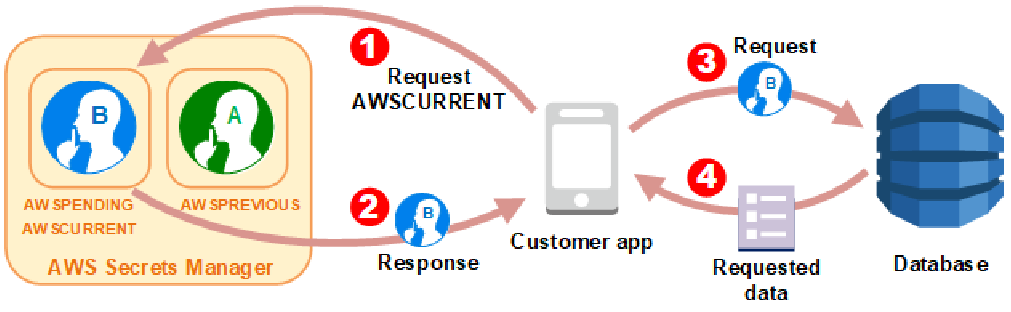
b. The secret rotation function deletes the API key not currently referenced by secret version "A", and creates a new API key for the same user. The rotation function then creates a new "B" version of the secret, not a new secret, but a **new version of the same secret**. The rotation function clones the details from the "A" version, except the function replaces the API key details with those from the newly created API key. This secret version "B" initially has the staging label `AWSPENDING` attached by the rotation process. Because the custom application always request the `AWSCURRENT` label and the label hasn't moved, the application continues to retrieve and use the original A version of the secret, for the API key to access the secured resource.



c. After testing the "B" version of the secret to ensure the secret works, the rotation process moves the `AWSCURRENT` label from the "A" version and attaches it to the new "B" version of the secret. This also automatically moves `AWSPENDING` staging label to the version with the staging label `AWSCURRENT`. This allows the scret to act as "last known good" in case there's a need for recovery.



d. The next request from the custom application now receives the "B" version of the secret because "B" now has the `AWSCURRENT` label. At this point, the customer application uses the API key in the new version of the secret. When the next rotation cycle occurs, the "B" version of the secret becomes the "A" version, and you start again in Step a.



Configuring Rotation to Alternate Credentials for a User

To configure a rotation mechanism for an authentication system that allows you to have only one user, follow the steps in this procedure:

To configure rotation for a user that has two sets of credentials

1. Create your user in the authentication system that protects the secured resource. Make note of the user name and credentials that you set. For this discussion, refer to them as **Creds1** and **Creds2**.
2. Create one Secrets Manager secret to store the details of the credentials you created in the previous step. This secret initially holds **Creds1** for the user:

   a. Sign in to the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.

   b. Choose **New secret**.

   c. For **Select secret type**, choose **Other type of secret**, including the user name and the first set of credentials. For example, enter two key-value pairs for the protected secret text. The pairs can be similar the examples:

Key	Value
UserName	*<your user name>*
APIKey	*<your API key>*
APIKeyId	*<identifies which of the two API keys is stored in this version>*

   d. For **AWS KMS Encryption Key**, choose the key to use to encrypt this secret, or leave the default set to the **DefaultMasterKey** for the account. To use the default key, the credentials that access the secret must be from the same account that owns the secret. If the user credentials reside in a different account, then you must create and specify a custom customer master key (CMK).

   e. For **Select rotation period**, choose or type the number of days between rotations.

   Note
   When you use the Secrets Manager console to configure rotation, by default, Secrets Manager automatically enables expiration and sets the number of days between rotations + 7.

   f. For **Select the Lambda rotation function**, choose **Create function**.

   g. Choose **Next step**.

   h. Type a **Secret name** and optional **Description**. Optionally, you can add tags.

   i. Choose **Store secret**.
3. Examine your new secret to locate the Amazon Resource Name (ARN) of the Lambda rotation function.

   a. On the **Secrets** list page, choose the name of the secret you created in Step 2.

   b. In the **Secret details / Secret rotation** section, choose the ARN of the rotation function to open it in Lambda.

   c. Use the following logic as the basis for writing your rotation function:

   - **createSecret step**:
     - Retrieve the `AWSCURRENT` version of the secret by using the `GetSecretValue` operation.
     - Extract the `SecretString` value from the secret and store it in a structure you can modify.
     - Determine the inactive API key - the one not referenced in the `AWSCURRENT` version of the secret.

  - Issue commands to the service to delete the inactive API key you determined in the previous step.
  - Issue commands to the service to create a new access key for the same user.
  - Overwrite the API key and the identifier in the copy of the secret structure with those from the new API key you just created. Keep all other details the same.
  - Store the modified copy of the protected secret text by passing it as the `SecretString` parameter in a call to `PutSecretValue`. Secrets Manager labels the new version of the secret with `AWSPENDING`.
- **setSecret step**:
  - The **setSecret** step in this scenario doesn't do anything. You created the API key in the `createSecret` step, because you must have the API key and the identifier to store in the secret. You don't generate your own key as you might for most other scenarios.
- **testSecret step**:
  - Retrieve the `AWSPENDING` version of the secret by using the `GetSecretValue` operation.
  - Issue commands to the secured resource to attempt to access it by using the API key in this version of the secret.
- **finishSecret step**:
  - Move the label `AWSCURRENT` to the version labeled `AWSPENDING`. This automatically also moves the staging label `AWSPREVIOUS` to the secret you just removed `AWSCURRENT` from.
  - (Optional) Remove the label `AWSPENDING` from the version of the secret.

Deleting Unused Lambda Rotation Functions

After you create a rotation function for a secret, at some point, you might decide you no longer need the secret rotation. . Deleting the secret (p. 64) might be an obvious step. However, you might also want to consider removing the Lambda rotation function that rotates the secret. If you share the rotation function among several secrets, then you don't want to delete the function until you delete the last secret rotated by the function.

If you create the rotation function as described in this guide, by using a AWS Serverless Application Repository template, then you don't simply delete the function. Secrets Manager created the function as part of an AWS CloudFormation stack. Deleting the stack deletes everything the stack created. In this case, Secrets Manager deletes both the Lambda function and the IAM role that grants permissions to the function. You must perform the following steps to delete everything cleanly.

Deleting a rotation function you created with an AWS Serverless Application Repository template

Follow the steps under one of the following tabs:

Using the AWS Management Console

Minimum permissions

To delete a Lambda rotation function that you created by using one of the AWS Serverless Application Repository templates, you must have the permissions required to navigate to your stack and delete all of the created components:

- cloudformation:ListStacks
- cloudformation:DescribeStack
- cloudformation:ListStackResources
- cloudformation:DeleteStack
- lambda:ListFunctions
- lambda:GetFunction
- lambda:DeleteFunction

- iam:ListRoles
- iam:DetachRolePolicy
- iam:DeleteRolePolicy
- iam:DeleteRole
- cloudformation:DeleteStack

You can grant all of these by attaching the following AWS managed policies:

- SecretsManagerReadWrite
- IAMFullAccess

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Navigate to the list of functions, and choose the name of the function to delete.
3. On the function details page, a banner at the top displays **This function belongs to the CloudFormation stack aws-serverless-repository-SecretsManager*<rotation_template_name><unique_guid>*. Visit the CloudFormation console to manage this stack.**.

   Choose the **CloudFormation console** link to open the AWS CloudFormation console on the **Stack Details** page.
4. If you added any inline permission policies to the IAM role instead of editing the existing inline policies, or if you attached any additional managed policies, then you must delete or detach those policies before AWS CloudFormation can delete the stack:
   a. Expand the **Resources** section for the stack, and then choose the **Physical ID** value for the row, with the **Type** set to **AWS::IAM::Role**. This opens the IAM console in a separate tab.
   b. Examine the rows with **Inline policy** as the **Policy type**. You should see the AWSLambdaBasicExecutionRole AWS managed function attached. You should also see one or two inline policies named **SecretsManager*<template name>*Policy0** and **SecretsManager*<template name>*Policy1**. If you see any policies other than those, Secrets Manager did not create them as part of the stack. The policies were added manually after the stack was created. You must manually delete or detach them. If you don't, the request to delete the stack in the following steps can fail.
   c. Return to the **Stack Details** page of the AWS CloudFormation console.
5. Choose **Other Actions**, and then choose **Delete Stack**.
6. On the **Delete Stack** confirmation dialog box, choose **Yes, Delete**.

   The **Status** changes to **DELETE_IN_PROGRESS**. If the deletion successfully completes, the **Status** eventually changes to **DELETE_COMPLETE**.
7. When you return to the list of stacks, you no longer see the deleted stack.

Using the AWS CLI or SDK Operations

Minimum permissions
To delete a Lambda rotation function you created by using one of the AWS Serverless Application Repository templates, you must have the permissions required to perform each of the AWS CLI or equivalent API operations listed in the following steps. You can grant all of these by attaching the following two AWS managed policies:

- SecretsManagerReadWrite
- IAMFullAccess

1. Open a command prompt to run the AWS CLI commands.

2. To determine which AWS CloudFormation stack contains a specific function, run the following command with the function name passed as the `--physical-resource-id` parameter. This returns a list of resources associated with the stack that owns the specified function.

```
$ aws cloudformation describe-stack-resources --physical-resource-
id MyLambdaRotationFunction
{{
    "StackResources": [
        {
            "StackName": "aws-serverless-repository-MyLambdaCreationStack",
            "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/aws-
serverless-repository-MyLambdaCreationStack/EXAMPLE2-90ab-cdef-fedc-ba987EXAMPLE",
            "LogicalResourceId": "SecretsManagerRotationTemplate",
            "PhysicalResourceId": "MySecretsManagerRotationFunction",
            "ResourceType": "AWS::Lambda::Function",
            "Timestamp": "2018-04-27T18:03:05.490Z",
            "ResourceStatus": "CREATE_COMPLETE"
        },
        {
            "StackName": "aws-serverless-repository-MyLambdaCreationStack",
            "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/aws-
serverless-repository-MyLambdaCreationStack/EXAMPLE2-90ab-cdef-fedc-ba987EXAMPLE",
            "LogicalResourceId": "SecretsManagerRotationTemplateRole",
            "PhysicalResourceId": "aws-serverless-repository-
SecretsManagerRotationTe-<random-chars>",
            "ResourceType": "AWS::IAM::Role",
            "Timestamp": "2018-04-27T18:03:00.623Z",
            "ResourceStatus": "CREATE_COMPLETE"
        }
    ]
}
```

3. Look for the `StackId` and `PhysicalResourceId` response values associated with the `"ResourceType" :"AWS::IAM::Role"`, the name of the IAM role with permissions to invoke the function. If you added any inline permission policies to the IAM role instead of editing the existing inline policies, or if you attached any additional managed policies, then you must delete or detach those policies before AWS CloudFormation can delete the stack.

   To determine if you have any embedded inline policies, run the following command.

```
$ aws iam list-role-policies --role-name <role-name-from-PhysicalResourceId-on-
previous-command>
{
    "PolicyNames": [
        "SecretsManagerRotationTemplateRolePolicy0",
        "SecretsManagerRotationTemplateRolePolicy1"
    ]
}
```

   The inline policies named as shown here are expected, and you don't need to do anything with them.

4. If you see any policies listed other than those two, you must delete them from the role before you can proceed.

```
$ aws iam delete-role-policy --role-name <role-name-from-PhysicalResourceId-on-
previous-command> /
       --policy-name <policy-name-from-previous-command>
```

5. Now check to see if you have any managed policies attached to the role:

```
$ aws iam list-attached-role-policies --role-name <role-name-from-PhysicalResourceId-on-previous-command>
```

6. If you see any policies listed at all in the previous command output, run this final preparatory command to detach them from the role:

```
$ aws iam detach-role-policy --role-name <role-name-from-PhysicalResourceId-on-previous-command> /
       --policy-arn <ARN-of-policy-discovered-in-previous-command>
```

7. Now you can delete the stack, which deletes all of the associated resources. Pass the name of the stack you retrieved in step 2 as the `--stack-name`:

```
$ aws cloudformation delete-stack --stack-name aws-serverless-repository-MyLambdaCreationStack
```

   AWS CloudFormation deletes the IAM role and the Lambda rotation function shortly after you run this command.

Using Secrets Manager with VPC Endpoints

The following sections explain these tasks:

- Connecting Secrets Manager and a VPC endpoint.
- Creating a Secrets Manager VPC endpoint.
- Creating an endpoint policy for your Secrets Manager endpoint,

For information about VPC endpoints, see the article in the VPC service documentation, VPC Endpoints.

Topics

Connecting to Secrets Manager Through a VPC Endpoint

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. One or more Each VPC endpoint is represented by one or more elastic network interfaces with private IP addresses in your VPC subnets.

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.

```
$ 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 Using VPC Endpoint Conditions in Policies with Secrets Manager Permissions (p. 166).

Regions

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

Create a Secrets Manager VPC Private Endpoint

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"
          }
      ]
  }
}
```

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
```

Using a VPC Private Endpoint in a Policy Statement

You can use IAM policies and Secrets Manager secret policies to control access to your secrets. You can also use global condition keys to restrict these policies based on the VPC endpoint or VPC in the request.

- Use the `aws:sourceVpce` condition key to grant or restrict access to a secret based on the VPC endpoint.
- Use the `aws:sourceVpc` condition key to grant or restrict access to a secret based on the VPC that hosts the private endpoint.

Note
Use caution when you create IAM and secret policies based on your VPC endpoint. If a policy statement requires the requests to originate from a particular VPC or VPC endpoint, then requests from other AWS services accessing the secret on your behalf might fail. For more information, see Using VPC Endpoint Conditions in Policies with Secrets Manager Permissions (p. 166).
Also, the `aws:sourceIP` condition key isn't effective when the request comes from an Amazon VPC endpoint. To restrict requests to a VPC endpoint, use the `aws:sourceVpce` or `aws:sourceVpc` condition keys. For more information, see VPC Endpoints - Controlling the Use of Endpoints in the *Amazon VPC User Guide*.

For example, the following sample secret policy allows a user to perform Secrets Manager operations only when the request comes through the specified VPC endpoint.

When a user sends a request to Secrets Manager, Secrets Manager compares the VPC endpoint ID in the request to the `aws:sourceVpce` condition key value in the policy. If they don't match, Secrets Manager denies the request.

To use a policy like this one, replace the placeholder AWS account ID and VPC endpoint IDs with valid values for your account.

```
{
    "Id": "example-policy-1",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable Secrets Manager permissions to principals in account
 123456789012",
            "Effect": "Allow",
            "Principal": {"AWS":["123456789012"]},
            "Action": ["secretsmanager:*"],
            "Resource": "*"
        },
        {
            "Sid": "Restrict GetSecretValue operation to only my VPC endpoint",
            "Effect": "Deny",
            "Principal": "*",
            "Action": ["secretsmanager:GetSecretValue"],
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "aws:sourceVpce": "vpce-1234a5678b9012c"
                }
            }
        }

    ]
}
```

You can also use the `aws:sourceVpc` condition key to restrict access to your secrets based on the VPC where the VPC endpoint resides.

The following sample secret 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 a policy like this one, replace the placeholder AWS account ID and VPC endpoint IDs with valid values for your account.

```
{
    "Id": "example-policy-2",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow administrative actions from ONLY vpc-12345678",
            "Effect": "Allow",
            "Principal": {"AWS": "123456789012"},
            "Action": [
                "secretsmanager:Create*",
                "secretsmanager:Put*",
                "secretsmanager:Update*",
                "secretsmanager:Delete*","secretsmanager:Restore*",
                "secretsmanager:RotateSecret","secretsmanager:CancelRotate*",
                "secretsmanager:TagResource","secretsmanager:UntagResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:sourceVpc": "vpc-12345678"
                }
            }
        },
        {
            "Sid": "Allow secret value access from ONLY vpc-2b2b2b2b",
            "Effect": "Allow",
            "Principal": {"AWS": "111122223333"},
            "Action": ["secretsmanager:GetSecretValue"],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:sourceVpc": "vpc-2b2b2b2b"
                }
            }
        },
        {
            "Sid": "Allow read actions from everywhere",
            "Effect": "Allow",
            "Principal": {"AWS": "111122223333"},
            "Action": [
                "secretsmanager:Describe*","secretsmanager:List*","kms:GetRandomPassword"
            ],
            "Resource": "*",
        }
    ]
}
```

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.

Note
AWS does not share VPC endpoints across AWS services. If you use VPC endpoints for multiple AWS services, such as Secrets Manager and 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": "Access for a specific account",
              "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-
west:123456789012:secret:a_specific_secret_name-a1b2c3"
                      ]
            }
          ]
      }
```

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. 128) 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"
     }
```

Monitoring the Use of Your AWS Secrets Manager Secrets

As a best practice, you should monitor your secrets to ensure usage of your secrets and log any changes to them are logged. 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 the activity that happens within it.

Topics

Logging 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.

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.

Retrieving 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-1\",
\"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\"}"
        }
    ]
}
```

Understanding 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-west-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-west-2",
    "requestParameters": {
      "recoveryWindowInDays": 30,
      "secretId": "MyDatabaseSecret"
    },
    "responseElements": {
      "name": "MyDatabaseSecret",
      "deletionDate": "May 3, 2018 5:51:02 PM",
      "aRN": "arn:aws:secretsmanager:us-west-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.

Monitoring 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: Configuring 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: Monitoring 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.

AWS Services Integrated with AWS Secrets Manager

AWS Secrets Manager works with other AWS services to provide additional solutions for your business challenges. This topic identifies services that either use Secrets Manager to add functionality, or services that Secrets Manager uses to perform tasks.

Topics

Automating Creation of Your Secrets with AWS CloudFormation

Secrets Manager supports AWS CloudFormation and enables you to define and reference secrets from within your stack template. Secrets Manager defines several AWS CloudFormation resource types that allow you to create a secret and associate it with the service or database with credentials stored in it. You can refer to elements in the secret from other parts of the template, such as retrieving the user name and password from the secret when you define the master user and password in 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. For more information and a comprehensive example, see Automating Secret Creation in AWS CloudFormation (p. 138).

Monitoring Your Secrets with 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 SNS notifications about a list of secrets unconfigured for rotation which enables you to drive security best practices for rotating secrets.

When you leverage the AWS Config Aggregator feature, 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.

To learn more about this feature see Monitoring Secrets Manager Secrets Using AWS Config (p. 144).

Securing Your Secrets with AWS Identity and Access Management (IAM)

Secrets Manager uses IAM to secure access to the secrets. IAM provides the following:

- **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 prove the identify of the users.
- **Authorization** – Ensures only approved individuals can perform operations on AWS resources, such as secrets. This enables you to grant write access to specific secrets to one user while granting read-only permissions on different secrets to another user.

For more information about protecting access to your secrets by using IAM, see Authentication and Access Control for AWS Secrets Manager (p. 34) in this guide. For complete information about IAM, see the IAM User Guide.

Monitoring Your Secrets with AWS CloudTrail and Amazon CloudWatch

You can use CloudTrail and CloudWatch to montor 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.

For more information, see Monitoring the Use of Your AWS Secrets Manager Secrets (p. 128) in this guide. For complete information about CloudWatch, see the Amazon CloudWatch User Guide. For complete information about CloudWatch Events, see the Amazon CloudWatch Events User Guide.

Encrypting Your Secrets with AWS KMS

Secrets Manager uses the trusted, industry-standard Advanced Encryption Standard (AES) encryption algorithm (FIPS 197) to encrypt your secrets. Secrets Manager also uses encryption keys provided by AWS Key Management Service (AWS KMS) to perform envelope encryption of the secret value. When you create a new version of a secret, Secrets Manager uses the specified AWS KMS customer master key (CMK) to generate a new data key. The data key consists of a 256-bit symmetric AES key. Secrets Manager receives both a plaintext and encrypted copy of the data key. Secrets Manager uses the plaintext data key to encrypt the secret value, and then stores the encrypted data key in the metadata of the secret version. When you later send a request to Secrets Manager to retrieve the secret value, Secrets Manager first retrieves the encrypted data key from the metadata, and then requests AWS KMS to decrypt the data key using the associated CMK. AWS KMS uses the decrypted data key to decrypt the secret value, and never stores keys in an unencrypted state, and removes the keys from memory immediately when no longer required.

For more information, see How AWS Secrets Manager Uses AWS KMS in the *AWS Key Management Service Developer Guide*.

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*.

Integrating 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

Integrating 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

Integrating 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.

Managing Amazon SageMaker Repository Credentials with Secrets Manager

Amazon SageMaker is a fully managed machine learning service. With Amazon 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, Amazon 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 Amazon SageMaker console.

You can manage your private repositories credentials using Secrets Manager.You can manage your private repositories credentials using Secrets Manager.

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

Storing 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.

Storing 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.

Automating Secret Creation in AWS CloudFormation

Note
Currently, AWS Secrets Manager does not support native rotation of secrets for Amazon RS, DocumentDB, and Redshift from CloudFormation in the eu-south-1 and af-south-1 regions.

By using an AWS CloudFormation template, you can automate creating secrets for database or service resources in your AWS cloud infrastructure.

You can use AWS CloudFormation to automate the creation of your cloud infrastructure. You create a template in either JSON or YAML to define the resources you need for your project. AWS CloudFormation then processes the template and builds the defined resources. This enables you to easily recreate 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 easily share the infrastructure as a simple text file, which enables others to replicate the resources without manual intervention.

You can use CloudFormer to capture all of the details of an existing set of resources into an AWS CloudFormation template.

Secrets Manager provides the following resource types to enable you to create secrets as part of an AWS CloudFormation template. For details about configuring each in your AWS CloudFormation template, choose the resource type name for a link to the *AWS CloudFormation User Guide*.

- **AWS::SecretsManager::Secret** – Creates a secret and stores it in Secrets Manager. You can specify a password or let Secrets Manager generate one for you. You may also create an empty secret and update it later using the parameter `SecretString.`
- **AWS::SecretsManager::ResourcePolicy** – Creates a resource-based policy and attaches it to the specified 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** – After you create a secret and then reference it to access the credentials when you create a service or database, this resource type goes back and finishes the configuration of the secret. Secrets Manager configures the secret with the details about the service or database required for rotation to work. 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.

Examples

The following example templates create a secret and an Amazon RDS MySQL DB instance using the credentials in the secret as the master user and password. The secret has a resource-based policy attached that specifies access to 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 YAML version later on this page for comments.

JSON

```
{
  "MyRDSInstanceRotationSecret": {
    "Type": "AWS::SecretsManager::Secret",
    "Properties": {
      "Description": "A Secrets Manager secret for my RDS DB instance",
      "GenerateSecretString": {
        "SecretStringTemplate": "{\"username\": \"admin\"}",
        "GenerateStringKey": "password",
        "PasswordLength": 16,
        "ExcludeCharacters": "\"@/\\"
      }
    }
  },
  "SecretRDSInstanceAttachment": {
    "Type": "AWS::SecretsManager::SecretTargetAttachment",
    "Properties": {
      "SecretId": {"Ref": "MyRDSInstanceRotationSecret"},
      "TargetId": {"Ref": "MyDBInstance"},
      "TargetType": "AWS::RDS::DBInstance"
    }
  },
  "MyDBInstance": {
    "Type": "AWS::RDS::DBInstance",
    "Properties": {
      "AllocatedStorage": 20,
      "DBInstanceClass": "db.t2.micro",
      "Engine": "mysql",
      "MasterUsername": {"Fn::Join": [
        "",
        ["{{resolve:secretsmanager:", {"Ref": "MyRDSInstanceRotationSecret"},
 "::username}}"]
      ]},
      "MasterUserPassword": {"Fn::Join": [
        "",
        ["{{resolve:secretsmanager:", {"Ref": "MyRDSInstanceRotationSecret"},
 "::password}}"]
      ]},
      "BackupRetentionPeriod": 0,
      "DBInstanceIdentifier": "rotation-instance"
    }
  },
  "MySecretRotationSchedule": {
    "Type": "AWS::SecretsManager::RotationSchedule",
    "DependsOn": "SecretRDSInstanceAttachment",
    "Properties": {
      "SecretId": {"Ref": "MyRDSInstanceRotationSecret"},
      "RotationLambdaARN": {"Fn::GetAtt": ["MyRotationLambda","Arn"]},
      "RotationRules": {
        "AutomaticallyAfterDays": 30
      }
    }
  },
  "MySecretResourcePolicy": {
    "Type": "AWS::SecretsManager::ResourcePolicy",
    "Properties": {
      "SecretId": {"Ref": "MyRDSInstanceRotationSecret"},
      "ResourcePolicy": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Deny",
            "Principal": {"AWS": {"Fn::Sub": "arn:aws:iam::${AWS::AccountId}:root"}},
            "Action": "secretsmanager:DeleteSecret",
            "Resource": "*"
          }
        ]
      }
    }
  },
  "LambdaInvokePermission": {
    "Type": "AWS::Lambda::Permission",
    "DependsOn": "MyRotationLambda",
    "Properties": {
      "FunctionName": "cfn-rotation-lambda",
      "Action": "lambda:InvokeFunction",
      "Principal": "secretsmanager.amazonaws.com"
    }
  },
  "MyLambdaExecutionRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "RoleName": "cfn-rotation-lambda-role",
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {"Service": ["lambda.amazonaws.com"]},
            "Action": ["sts:AssumeRole"]
          }
        ]
      },
      "Policies": [
        {
          "PolicyName": "AWSSecretsManagerRotationPolicy",
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
                "Action": [
                  "secretsmanager:DescribeSecret",
                  "secretsmanager:GetSecretValue",
                  "secretsmanager:PutSecretValue",
                  "secretsmanager:UpdateSecretVersionStage"
                ],
                "Resource": {"Fn::Sub": "arn:${AWS::Partition}:secretsmanager:
${AWS::Region}:${AWS::AccountId}:secret:*"},
                "Condition": {
                  "StringEquals": {
                    "secretsmanager:resource/AllowRotationLambdaArn": {"Fn::Sub": "arn:
${AWS::Partition}:lambda:${AWS::Region}:${AWS::AccountId}:function:cfn-rotation-lambda"}
                  }
                }
              },
              {
```

```
                "Effect": "Allow",
                "Action": ["secretsmanager:GetRandomPassword"],
                "Resource": "*"
              },
              {
                "Effect": "Allow",
                "Action": [
                  "logs:CreateLogGroup",
                  "logs:CreateLogStream",
                  "logs:PutLogEvents"
                ],
                "Resource": "arn:aws:logs:*:*:*"
              },
              {
                "Effect": "Allow",
                "Action": [
                  "kms:Decrypt",
                  "kms:DescribeKey",
                  "kms:GenerateDataKey"
                ],
                "Resource": "*"
              }
            ]
          }
        }
      ]
    }
  },
  "MyRotationLambda": {
    "Type": "AWS::Lambda::Function",
    "Properties": {
      "Runtime": "python2.7",
      "Role": {"Fn::GetAtt": ["MyLambdaExecutionRole","Arn"]},
      "Handler": "lambda_function.lambda_handler",
      "Description": "This is a Secrets Manager rotation function for a MySQL RDS DB
 instance",
      "FunctionName": "cfn-rotation-lambda",
      "Environment": {
        "Variables": {
          "SECRETS_MANAGER_ENDPOINT": {"Fn::Sub": "https://secretsmanager.
${AWS::Region}.amazonaws.com"}
        }
      },
      "Code": {
        "S3Bucket": "<% replace-this-with-name-of-s3-bucket-that-contains-lambda-function-
code %>",
        "S3Key": "<% replace-this-with-path-and-filename-of-zip-file-that-contains-lambda-
function-code %>",
        "S3ObjectVersion": "<% replace-this-with-lambda-zip-file-version-if-s3-bucket-
versioning-is-enabled %>"
      }
    }
  }
}
```

YAML

```
---
Description: "This is an example template to demonstrate CloudFormation resources for
 Secrets Manager"
Resources:

  #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 a 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 use a ref to the Secret resource logical id in order to construct the dynamic
 reference, since the Secret name is being
  #generated by CloudFormation
  MyDBInstance:
    Type: AWS::RDS::DBInstance
    Properties:
      AllocatedStorage: 20
      DBInstanceClass: db.t2.micro
      Engine: mysql
      MasterUsername: !Join ['', ['{{resolve:secretsmanager:', !Ref
 MyRDSInstanceRotationSecret, ':SecretString:username}}' ]]
      MasterUserPassword: !Join ['', ['{{resolve:secretsmanager:', !Ref
 MyRDSInstanceRotationSecret, ':SecretString:password}}' ]]
      BackupRetentionPeriod: 0
      DBInstanceIdentifier: 'rotation-instance'

  #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
  #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
      RotationLambdaARN: !GetAtt MyRotationLambda.Arn
      RotationRules:
        AutomaticallyAfterDays: 30

  #This is ResourcePolicy resource which can be used to attach a resource policy to the
 referenced secret.
  #The resource policy in this example denies the DeleteSecret action to all principals
 within the current account
  MySecretResourcePolicy:
    Type: AWS::SecretsManager::ResourcePolicy
    Properties:
      SecretId: !Ref MyRDSInstanceRotationSecret
      ResourcePolicy:
        Version: '2012-10-17'
        Statement:
          - Effect: Deny
            Principal:
              AWS: !Sub 'arn:aws:iam::${AWS::AccountId}:root'
            Action: secretsmanager:DeleteSecret
            Resource: "*"

  #This is a lambda Function resource. We will use this lambda to rotate secrets
  #For details about rotation lambdas, see https://docs.aws.amazon.com/secretsmanager/
latest/userguide/rotating-secrets.html
  #The below example assumes that the lambda code has been uploaded to a S3 bucket, and
 that it will rotate a mysql database password
  MyRotationLambda:
    Type: AWS::Lambda::Function
    Properties:
      Runtime: python2.7
      Role: !GetAtt MyLambdaExecutionRole.Arn
      Handler: lambda_function.lambda_handler
      Description: 'This is a lambda to rotate MySql user passwd'
      FunctionName: 'cfn-rotation-lambda'
      Environment:
        Variables:
          SECRETS_MANAGER_ENDPOINT: !Sub 'https://secretsmanager.
${AWS::Region}.amazonaws.com'
      Code:
        S3Bucket: <% name-of-s3-bucket-that-contains-lambda-function-code %>
        S3Key: <% path-and-filename-of-zip-file-that-contains-lambda-function-code %>
        S3ObjectVersion: <% lambda-zip-file-version-if-s3-bucket-versioning-is-enabled %>

  #This is a lambda Permission resource which grants Secrets Manager permission to invoke
 the rotation lambda function
  LambdaInvokePermission:
    Type: AWS::Lambda::Permission
    DependsOn: MyRotationLambda
    Properties:
      FunctionName: 'cfn-rotation-lambda'
      Action: 'lambda:InvokeFunction'
      Principal: secretsmanager.amazonaws.com

  #This is the IAM Role resource for the rotation lambda, it grants permissions to the
 lambda to get and update the secret as part of the
  #rotation process. This includes required KMS permissions. It also includes permissions
 needed for logging to CloudWatch.
  MyLambdaExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: 'cfn-rotation-lambda-role'
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          -
            Effect: "Allow"
            Principal:
              Service:
                - "lambda.amazonaws.com"
            Action:
              - "sts:AssumeRole"
      Policies:
        -
          PolicyName: "AWSSecretsManagerRotationPolicy"
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              -
                Effect: "Allow"
                Action:
                  - "secretsmanager:DescribeSecret"
                  - "secretsmanager:GetSecretValue"
                  - "secretsmanager:PutSecretValue"
                  - "secretsmanager:UpdateSecretVersionStage"
                Resource: !Sub 'arn:${AWS::Partition}:secretsmanager:${AWS::Region}:
${AWS::AccountId}:secret:*'
                Condition:
                  StringEquals:
                    secretsmanager:resource/AllowRotationLambdaArn: !Sub 'arn:
${AWS::Partition}:lambda:${AWS::Region}:${AWS::AccountId}:function:cfn-rotation-lambda'
              -
                Effect: "Allow"
                Action:
                  - "secretsmanager:GetRandomPassword"
                Resource: "*"
              -
                Effect: "Allow"
                Action:
                  - "logs:CreateLogGroup"
                  - "logs:CreateLogStream"
                  - "logs:PutLogEvents"
                Resource: "arn:aws:logs:*:*:*"
              -
                Effect: "Allow"
                Action:
                  - "kms:Decrypt"
                  - "kms:DescribeKey"
                  - "kms:GenerateDataKey"
                Resource: "*"
```

Monitoring 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 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.

Topics

Benefits of Tracking Secrets with AWS Config

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.

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 (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.

Implementing Secrets Management 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.

Configuring 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 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`.

Viewing Secrets Manager Rules in AWS Config

To view your Secrets Manager rules in AWS Config Developer Guideg;, 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.

Configuring 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. You can use CloudFormation StackSets to enable AWS Config and provision rules across all accounts and regions here.
> 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 master 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 master 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

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

Data Protection in AWS Secrets Manager

AWS Secrets Manager conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM), so that each user receives only the permissions necessary to fulfill their job duties.

Encryption at Rest

Secrets Manager uses encryption via AWS Key Management Service (KMS) to protect the confidentiality of data at rest. AWS KMS provides a key storage and encryption service used by many AWS services. AWS Secrets Manager associates every secret with an AWS KMS customer master key (CMK). The associated CMK can either be the default Secrets Manager CMK for the account, or you can create your own CMK. For additional information, on creating CMKs, see How Secrets Manager Uses AWS KMS.

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 AWS 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 (CLI) or any of the AWS SDKs to make calls to AWS, those tools automatically use the specified access key to sign the requests for you. When you configure the CLI or any of the AWS SDKs, you specify the access key to use.

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 specified CMK. 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 he data key in unencrypted form, and removes the key from memory as soon as possible.

For additional information on the encryption and decryption process, as well as a complete explanation of how Secrets Manager uses AWS KMS, see How AWS Secrets Manager Uses AWS KMS.

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.

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, highthroughput, 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 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.

AWS Secrets Manager Reference

Use the topics in this section to find detailed reference information for various aspects of AWS Secrets Manager.

Topics

Quotas for AWS Secrets Manager

This section specifies quotas for AWS Secrets Manager.

Restrictions on Names

AWS Secrets Manager has the following restrictions on names you create, including the names of secrets:

- Secret names must use Unicode characters.
- Secrets names must not exceed 512 characters in length.

Maximum Quotas

The following are the default maximum quotas for entities in AWS Secrets Manager:

Entity	**Value**
Secrets in an AWS account	40,000
Versions of a secret	~100
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

These parameters have the following rate quotas for AWS Secrets Manager:

Request type	Number of operations permitted per second
Overall API quota for the account	2,000
DescribeSecret GetSecretValue	2,000
PutResourcePolicy GetResourcePolicy DeleteResourcePolicy CreateSecret UpdateSecret UpdateSecretVersionStage PutSecretValue DeleteSecret RestoreSecret RotateSecret CancelRotateSecret AssociateSecretLabels DisassociateSecretLabels GetRandomPassword	40
TagResource UntagResource ListSecrets ListSecretVersionIds	20

AWS Templates You Can Use to Create Lambda Rotation Functions

This section identifies the AWS managed templates you can use to create a Lambda rotation function for your AWS Secrets Manager secret. These templates associate with the AWS Serverless Application Repository, which uses AWS CloudFormation to create 'stacks' of preconfigured resources. In this case, the templates create a stack that consists of the Lambda function and an IAM role that Secrets Manager can assume to invoke the function when rotation occurs.

To create a Lambda rotation function with any of the following templates, you can copy and paste the ARN of the specified template into the CLI commands described in the topic Rotating AWS Secrets Manager Secrets for Other Databases or Services (p. 98).

Each of the following templates creates a Lambda rotation function for a different combination of database and rotation strategy. The first bullet under each shows the database or service supported by the function. The second bullet describes the rotation strategy implemented by the function. The third bullet specifies the JSON structure the rotation function expects to find in the `SecretString` value of the rotated secret.

RDS databases

- RDS MariaDB Single User (p. 153)
- RDS MariaDB Master User (p. 154)
- RDS MySQL Single User (p. 154)
- RDS MySQL Multiple Users (p. 155)
- RDS Oracle Single User (p. 155)
- RDS Oracle Master User (p. 156)
- RDS PostgreSQL Single User (p. 156)
- RDS PostgreSQL Master User (p. 156)
- RDS Microsoft SQLServer Single User (p. 157)
- RDS Microsoft SQLServer Master User (p. 157)

Other databases and services

- MongoDB Single User (p. 158)
- MongoDB Master User (p. 158)
- Amazon Redshift Single User (p. 159)
- Amazon Redshift Master User (p. 159)
- Generic Rotation Function Template (p. 160)

Templates for Amazon RDS Databases

RDS MariaDB Single User

```
arn:aws:serverlessrepo:us-east-1:1234456789012:applications/
SecretsManagerRDSMariaDBRotationSingleUser
```

- **Name:** SecretsManagerRDSMariaDBRotationSingleUser
- **Supported database/service:** MariaDB database hosted on an Amazon Relational Database Service (Amazon RDS) database instance.
- **Rotation strategy:** This changes the password for a user with credentials stored in the rotated secret. For more information about this strategy, see Rotating AWS Secrets Manager Secrets for One User with a Single Password (p. 107).
- **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**

RDS MariaDB Master User

```
arn:aws:serverlessrepo:us-east-1:123456789012:applications/
SecretsManagerRDSMariaDBRotationMultiUser
```

- **Name:** SecretsManagerRDSMariaDBRotationMultiUser
- **Supported database/service:** MariaDB database hosted on an Amazon RDS database instance.
- **Rotation strategy:** Two users alternate during rotation by using the credentials of a separate master user, stored in a separate secret. Secrets Manager changes the password of the inactive user before the user becomes the active user. For more information about this strategy, see Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110).
- **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 master secret used to create 2nd user and
 change passwords>"
}
```

- **Source code**

RDS MySQL Single User

```
arn:aws:serverlessrepo:us-east-1:123456789012:applications/
SecretsManagerRDSMySQLRotationSingleUser
```

- **Name:** SecretsManagerRDSMySQLRotationSingleUser
- **Supported database/service:** MySQL database hosted on an Amazon Relational Database Service (Amazon RDS) database instance.
- **Rotation strategy:** This changes the password for a user with credentials stored in the rotated secret. For more information about this strategy, see Rotating AWS Secrets Manager Secrets for One User with a Single Password (p. 107).
- **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**

RDS MySQL Multiple Users

```
arn:aws:serverlessrepo:us-east-1:123456789012:applications/
SecretsManagerRDSMySQLRotationMultiUser
```

- **Name:** SecretsManagerRDSMySQLRotationMultiUser
- **Supported database/service:** MySQL database hosted on an Amazon RDS database instance.
- **Rotation strategy:** Two users alternate during rotation by using the credentials of a separate master user, stored in a separate secret. Secrets Manager changes the password of the inactive user before the user becomes the active user. For more information about this strategy, see Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110).
- **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 master secret used to create 2nd user and
 change passwords>"
}
```

- **Source code**

RDS Oracle Single User

```
arn:aws:serverlessrepo:us-east-1:123456789012:applications/
SecretsManagerRDSOracleRotationSingleUser
```

- **Name:** SecretsManagerRDSOracleRotationSingleUser
- **Supported database/service:** Oracle database hosted on an Amazon Relational Database Service (Amazon RDS) database instance.
- **Rotation strategy:** This changes the password for a user with credentials stored in the rotated secret. For more information about this strategy, see Rotating AWS Secrets Manager Secrets for One User with a Single Password (p. 107).
- **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**

RDS Oracle Master User

```
arn:aws:serverlessrepo:us-east-1:123456789012:applications/
SecretsManagerRDSOracleRotationMultiUser
```

- **Name:** SecretsManagerRDSOracleRotationMultiUser
- **Supported database/service:** Oracle database hosted on an Amazon RDS database instance.
- **Rotation strategy:** Two users alternate during rotation by using the credentials of a separate master user, stored in a separate secret. Secrets Manager changes the password of the inactive user before the user becomes the active user. For more information about this strategy, see Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110).
- **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 master secret used to create 2nd user and
 change passwords>"
}
```

- **Source code**

RDS PostgreSQL Single User

```
arn:aws:serverlessrepo:us-east-1:123456789012:applications/
SecretsManagerRDSPostgreSQLRotationSingleUser
```

- **Name:** SecretsManagerRDSPostgreSQLRotationSingleUser
- **Supported database/service:** PostgreSQL database hosted on an Amazon RDS database instance.
- **Rotation strategy:** This changes the password for a user with credentials stored in the rotated secret. For more information about this strategy, see Rotating AWS Secrets Manager Secrets for One User with a Single Password (p. 107).
- **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**

RDS PostgreSQL Master User

```
arn:aws:serverlessrepo:us-east-1:123456789012:applications/
SecretsManagerRDSPostgreSQLRotationMultiUser
```

- **Name:** SecretsManagerRDSPostgreSQLRotationMultiUser
- **Supported database/service:** PostgreSQL database hosted on an Amazon RDS database instance.
- **Rotation strategy:** Two users alternate during rotation by using the credentials of a separate master user, stored in a separate secret. Secrets Manager changes the password of the inactive user before the user becomes the active user. For more information about this strategy, see Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110).
- **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 master secret used to create 2nd user and
 change passwords>"
}
```

- **Source code**

RDS Microsoft SQLServer Single User

```
arn:aws:serverlessrepo:us-east-1:123456789012:applications/
SecretsManagerRDSSQLServerRotationSingleUser
```

- **Name:** SecretsManagerRDSSQLServerRotationSingleUser
- **Supported database/service:** Microsoft SQLServer database hosted on an Amazon RDS database instance.
- **Rotation strategy:** This changes the password for a user with credentials stored in the rotated secret. For more information about this strategy, see Rotating AWS Secrets Manager Secrets for One User with a Single Password (p. 107).
- **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**

RDS Microsoft SQLServer Master User

```
arn:aws:serverlessrepo:us-east-1:123456789012:applications/
SecretsManagerRDSSQLServerRotationMultiUser
```

- **Name:** SecretsManagerRDSSQLServerRotationMultiUser
- **Supported database/service:** Microsoft SQLServer database hosted on an Amazon RDS database instance.

- **Rotation strategy:** Two users alternate during rotation by using the credentials of a separate master user, stored in a separate secret. Secrets Manager changes the password of the inactive user before the user becomes the active user. For more information about this strategy, see Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110).
- **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 master secret used to create 2nd user and
 change passwords>"
}
```

- **Source code**

Templates for Other Databases

MongoDB Single User

```
arn:aws:serverlessrepo:us-east-1:123456789012:applications/
SecretsManagerMongoDBRotationSingleUser
```

- **Name:** SecretsManagerMongoDBRotationSingleUser
- **Supported database/service:** MongoDB database version 3.2 or 3.4.
- **Rotation strategy:** This changes the password for a user with credentials stored in the rotated secret. For more information about this strategy, see Rotating AWS Secrets Manager Secrets for One User with a Single Password (p. 107).
- **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

MongoDB Master User

```
arn:aws:serverlessrepo:us-east-1:123456789012:applications/
SecretsManagerMongoDBRotationMultiUser
```

- **Name:** SecretsManagerMongoDBRotationMultiUser
- **Supported database or service:** MongoDB database version 3.2 or 3.4.
- **Rotation strategy:** Two users alternate during rotation by using the credentials of a separate master user, and stored in a separate secret.Secrets Manager changes the password of the inactive user before

the user becomes the active user. For more information about this strategy, see Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110).

- **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 master secret used to create 2nd user and
 change passwords>"
}
```

- Source code

Amazon Redshift Single User

```
arn:aws:serverlessrepo:us-east-1:123456789012:applications/
SecretsManagerRedShiftRotationSingleUser
```

- **Name:** SecretsManagerRedShiftRotationSingleUser
- **Supported database/service:** Amazon Redshift
- **Rotation strategy:** This changes the password for a user with credentials stored in the rotated secret. For more information about this strategy, see Rotating AWS Secrets Manager Secrets for One User with a Single Password (p. 107).
- **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 Master User

```
arn:aws:serverlessrepo:us-east-1:123456789012:applications/
SecretsManagerRedShiftRotationMultiUser
```

- **Name:** SecretsManagerRedShiftRotationMultiUser
- **Supported database/service:** Amazon Redshift
- **Rotation strategy:** Two users alternate during rotation by using the credentials of a separate master user, stored in a separate secret. Secrets Manager changes the password of the inactive user before becomeing the active user. For more information about this strategy, see Rotating AWS Secrets Manager Secrets by Alternating Between Two Existing Users (p. 110).
- **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 ?????>",
  "masterarn": "<required: the ARN of the master secret used to create 2nd user and
 change passwords>"
}
```

- **Source code**

Templates for Other Services

Generic Rotation Function Template

```
arn:aws:serverlessrepo:us-east-1:297356227824:applications/SecretsManagerRotationTemplate
```

- **Name:** SecretsManagerRotationTemplate
- **Supported database/service:** None. You supply the code to interact with whatever service you want.
- **Rotation strategy:** None. You supply the code to implement whatever rotation strategy you want. For more information about customizing your own function, see Understanding and Customizing Your Lambda Rotation Function (p. 103).
- **Expected `SecretString` structure:** You define this as part of the code that you write.
- **Source code:** **Source code**

AWS Managed Policies Available for Use with AWS Secrets Manager

This section identifies the AWS managed policies you can use to help manage access to your secrets. You can't modify or delete an AWS managed policy, but you can attach or detach them to entities in your account as needed.

Policy Name	Description	ARN
SecretsManagerReadWrite	Provides access to most Secrets Manager operations. The policy doesn't enable configuring rotation because rotation requires IAM permissions to create roles. For someone who must configure Lambda rotation functions and enable rotation, you should also assign the IAMFullAccess managed policy.	arn:aws:iam::aws:policy/SecretsManagerReadWrite

Actions, Resources, and Context Keys You Can Use in an IAM Policy or Secret Policy for AWS Secrets Manager

Actions You Can Reference in an IAM Policy or Secret Policy

The following table lists the permissions you can specify in an IAM policy or secret policy to control access to your secrets. Each permission on an Action can be associated with a Resource that specifies what the action can work on.

You can restrict use of some actions to only secrets with Amazon Resource Names (ARNs) matching the `Resource` element in the policy. See the section Resources You Can Reference in an IAM Policy or Secret Policy (p. 163) later in this topic.

In addition to the AWS Secrets Manager-specific context keys shown in the last column, you also use the AWS Global Condition Context Keys.

If you see an expand arrow (↗) in the upper-right corner of the table, you can open the table in a new window. To close the window, choose the close button (**X**) in the lower-right corner.

Permission for "Action" element	**API operation enabled by this action**	**Resource ARNs used as a "Resource" with this action**	**Secrets Manager Context keys used with this action**
secretsmanager:CancelRotateSecret	CancelRotateSecret	Secret (p. 163)	secretsmanager:SecretId (p. 165) secretsmanager:AllowRotationLamb secretsmanager:ResourceTag/ *<tagname>* (p. 165)
secretsmanager:CreateSecret	CreateSecret		secretsmanager:Name (p. 165) secretsmanager:Description (p. 164) secretsmanager:KmsKeyId (p. 164)
secretsmanager:DeleteResourcePolicy	DeleteResourcePolicy	Secret (p. 163)	secretsmanager:SecretId (p. 165) secretsmanager:AllowRotationLamb secretsmanager:ResourceTag/ *<tagname>* (p. 165)
secretsmanager:DeleteSecret	DeleteSecret	Secret (p. 163)	secretsmanager:SecretId (p. 165) secretsmanager:AllowRotationLamb secretsmanager:ResourceTag/ *<tagname>* (p. 165)
secretsmanager:DescribeSecret	DescribeSecret	Secret (p. 163)	secretsmanager:SecretId (p. 165) secretsmanager:AllowRotationLamb

Permission for "Action" element	API operation enabled by this action	Resource ARNs used as a "Resource" with this action	Secrets Manager Context keys used with this action
			secretsmanager:RecoveryWindowInD secretsmanager:ForceDeleteWithoutF secretsmanager:ResourceTag/ *<tagname>* (p. 165)
secretsmanager:GetRandomPassword	GetRandomPassword		
secretsmanager:GetResourcePolicy	GetResourcePolicy	Secret (p. 163)	secretsmanager:SecretId (p. 165) secretsmanager:AllowRotationLambd secretsmanager:ResourceTag/ *<tagname>* (p. 165)
secretsmanager:GetSecretValue	GetSecretValue	Secret (p. 163)	secretsmanager:SecretId (p. 165) secretsmanager:VersionId (p. 165) secretsmanager:VersionStage (p. 165 secretsmanager:AllowRotationLambd secretsmanager:ResourceTag/ *<tagname>* (p. 165)
secretsmanager:ListSecrets	ListSecrets		
secretsmanager:ListSecretVersionIds	ListSecretVersionIds	Secret (p. 163)	secretsmanager:SecretId (p. 165) secretsmanager:AllowRotationLambd secretsmanager:ResourceTag/ *<tagname>* (p. 165)
secretsmanager:PutResourcePolicy	PutResourcePolicy	Secret (p. 163)	secretsmanager:SecretId (p. 165) secretsmanager:AllowRotationLambd secretsmanager:ResourceTag/ *<tagname>* (p. 165)
secretsmanager:PutSecretValue	PutSecretValue	Secret (p. 163)	secretsmanager:SecretId (p. 165)
secretsmanager:RestoreSecret	RestoreSecret	Secret (p. 163)	secretsmanager:SecretId (p. 165) secretsmanager:AllowRotationLambd secretsmanager:ResourceTag/ *<tagname>* (p. 165)

Permission for "Action" element	API operation enabled by this action	Resource ARNs used as a "Resource" with this action	Secrets Manager Context keys used with this action
secretsmanager:RotateSecret	RotateSecret	Secret (p. 163)	secretsmanager:SecretId (p. 165) secretsmanager:RotationLambdaArn secretsmanager:AllowRotationLambd secretsmanager:ResourceTag/ *<tagname>* (p. 165)
secretsmanager:TagResource	TagResource	Secret (p. 163)	secretsmanager:SecretId (p. 165) secretsmanager:AllowRotationLambd secretsmanager:ResourceTag/ *<tagname>* (p. 165)
secretsmanager:UntagResource	UntagResource	Secret (p. 163)	secretsmanager:SecretId (p. 165) secretsmanager:AllowRotationLambd secretsmanager:ResourceTag/ *<tagname>* (p. 165)
secretsmanager:UpdateSecret	UpdateSecret	Secret (p. 163)	secretsmanager:SecretId (p. 165) secretsmanager:Description (p. 164) secretsmanager:KmsKeyId (p. 164) secretsmanager:AllowRotationLambd secretsmanager:ResourceTag/ *<tagname>* (p. 165)
secretsmanager:UpdateSecretVersionStage	UpdateSecretVersionStage	Secret (p. 163)	secretsmanager:SecretId (p. 165) secretsmanager:VersionStage (p. 165 secretsmanager:AllowRotationLambd secretsmanager:ResourceTag/ *<tagname>* (p. 165)

Resources You Can Reference in an IAM Policy or Secret Policy

The following table shows the ARN formats supported in IAM policies for AWS Secrets Manager. You can view the IDs for each entity on the **Secret details** page for each secret in the Secrets Manager console.

If you see an expand arrow (↗) in the upper-right corner of the table, you can open the table in a new window. To close the window, choose the close button (**X**) in the lower-right corner.

Resource Type	ARN Format
Secret	arn:aws:secretsmanager:*<Region>*:*<AccountId>*:secret:*OptionalPath/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 individuals with permissions to the original secret don't automatically get access to the new secret because Secrets Manager generates six new random characters.

Context Keys That You Can Reference in an IAM Policy or Secret Policy

Context keys in AWS Secrets Manager generally correspond to the request parameters of an API call. This enables you to allow or block any request based on the parameter value.

Each context key can be compared using a condition operator to a specified value. The context keys you use depend on the action selected. See the "Context keys" column in the Actions (p. 161) section at the beginning of this topic.

For example, you can allow someone to retrieve *only* the `AWSCURRENT` version of a secret value by using a `Condition` element similar to the following:

```
   "Condition": {"ForAnyValue:StringEquals" : {"secretsmanager:VersionStage" :
"AWSCURRENT"}}
```

The following table shows the Secrets Manager-specific context keys you specify in the `Condition` element of an IAM permissions policy to more granularly control access to an action. In addition to the keys below, you can also use the AWS Global Condition Context Keys.

Context keys for "Condition" element	Description	
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.	
secretsmanager:Description	Filters the request based on the Description parameter in the request.	
secretsmanager:ForceDeleteWithoutRecovery	Filters the request based if the delete specifies no recovery window. This enables you to effectively disable this feature.	
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.	

Context keys for "Condition" element	Description	
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.	
secretsmanager:RecoveryWindowInDays	Filters the request based on the recovery window specified. Enables you to enforce that recovery windows occur an approved number of days.	
secretsmanager:ResourceTag/ *<tagname>*	Filters the request based on a tag attached to the secret. Replace *<tagname>* with the actual tag name. You can then use condition operators to ensure the presence of the tag, and tcontains the requested value.	
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.	
secretsmanager:SecretId	Filters the request based on the unique identifier for the secret provided in the SecretId parameter. The value can be either the friendly name or the ARN of the secret. This enables you to limit which secrets can be accessed by a request.	
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.	
secretsmanager:VersionStage	Filters the request based on the staging labels identified in the VersionStage parameter of a request. This enables you to restrict access to only the secret versions with a staging label matching one of the values in this string array parameter. Because the key usesa multi-valued string array, you must use one of the set operators to compare strings with this value.	

AWS Global Condition Keys

AWS provides global condition keys, a set of predefined condition keys for all AWS services that use IAM for access control. For example, you can use the `aws:PrincipalType` condition key to allow access only when the principal in the request contains the type you specify.

Secrets Manager supports all global condition keys, including the `aws:TagKeys` and `aws:RequestTag` condition keys that control access based on the resource tag in the request. Some, not all AWS services support these condition keys.

Topics

Using the IP Address Condition in Policies with Secrets Manager Permissions

You can use Secrets Manager to protect your credentials for a database or service. However, use caution when you specify the IP address condition operators or the `aws:SourceIp` condition key in the same policy statement that allows or denies access to Secrets Manager. For example, the policy in AWS: Denies Access to AWS Based on the Source IP restricts AWS actions to requests from the specified IP range.

If you attach a similar policy to a secret enabling the secret for access from your corporate network IP address range, 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 becomes less effective when the request comes from an Amazon VPC endpoint. To restrict requests to a specific VPC endpoint, including a Secrets Manager VPC endpoint, use the `aws:sourceVpce` or `aws:sourceVpc` condition keys. For more information, see VPC Endpoints - Controlling the Use of Endpoints in the *Amazon VPC User Guide*.

Using VPC Endpoint Conditions in Policies with Secrets Manager Permissions

Secrets Manager supports Amazon VPC endpoints (p. 120) provided by AWS PrivateLink. You can use the following global condition keys in IAM policies or resource-based policies (p. 48) to allow or deny access to requests from a particular VPC or VPC endpoint.

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

If you use these condition keys in a secret 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.

Troubleshooting AWS Secrets Manager

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

Topics

Troubleshooting 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 Developer Guide*
- Ensuring Consistency When Using Amazon S3 and Amazon Elastic MapReduce for ETL Workflows in the AWS Big Data Blog
- EC2 Eventual Consistency in the *Amazon EC2 API Reference*
-

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

Verify you are using a symmetric customer master key (CMK) instead of an asymmetric CMK. Secrets Manager uses a symmetric customer master key (CMK) associated with a secret to generate a data key for each secret value. Secrets Manager also uses the CMK 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 CMK at this time.

Troubleshooting 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. Choose the **Monitoring** tab.
4. In the **Invocation errors** section, choose **Jump to Logs**.

   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. 171).

```
{
    "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 AWS Templates You Can Use to Create Lambda Rotation Functions (p. 152).

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. 169).
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. 169). 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": "<arn_of_master_secret>"
        }
    ]
}
```

Calling the API by Sending HTTP 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.*

Signing 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.

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
Changed the Rotate Secret tutorial to include a link to Amazon RDS. (p. 175)	To keep the tutorial steps up to date in the guide, a link to the Amazon RDS documentation replaced the steps to set up a test database.	May 12, 2020
Added FedRAMP compliance for Secrets Manager. (p. 175)	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. 175)	Added documentation for using AWS Config with Secrets Manager.	April 16, 2020
Replaced CloudFormation templates with shorter and easier to use templates. (p. 175)	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 Redshift and DocumentDB	Added Redshift and 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 is provided for the initial release of AWS Secrets Manager.	April 4, 2018

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.