

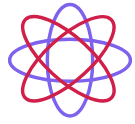
YOU ARE HERE: [HOME](#) / [ANSIBLE](#) / [HOW TO USE AWS SSM
PARAMETER STORE WITH ANSIBLE](#)

How to Use AWS SSM Parameter Store with Ansible

If you're like many frustrated Dev/Ops teams out there, you may be tired of using [Ansible's vault function](#) to encrypt secrets, and looking for other options. AWS offers a couple options for storing secrets:

- The more obviously-named Secrets Manager tool
- The less-obvious Systems Manager Parameter Store

As it turns out, Ansible has lookup plugins for both AWS tools. In this guide we'll explore both, and why (spoiler alert!) our team decided to go with Systems Manager Parameter Store in the end.



different secure variable storage systems.

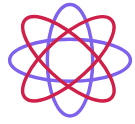
Secrets Manager

Firstly, Secrets Manager has different secret types, mostly geared toward storing, encrypting, (and regularly rotating) database credentials. Here's a quick look at the options.

The screenshot shows the 'Store a new secret' interface in AWS Secrets Manager. It features a 'Select secret type' section with five radio button options: 'Credentials for RDS database' (selected), 'Credentials for Redshift cluster', 'Credentials for DocumentDB database', 'Credentials for other database', and 'Other type of secrets (e.g. API key)'. Below this is a section to 'Specify the user name and password to be stored in this secret', containing input fields for 'User name' and 'Password', and a 'Show password' checkbox. The final section is 'Select the encryption key', which includes a dropdown menu set to 'DefaultEncryptionKey', a refresh button, and a link to 'Add new key'.

The options shown in the “Store a new secret” pane are:

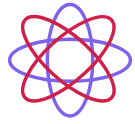
- Credentials for RDS database
- Credentials for Redshift cluster
- Credentials for DocumentDB database



database credentials. There is, however, an option to store another secret in “Key” and “Value” format. This is the option our team was planning to use for most secure variables.

The screenshot shows the AWS SSM Parameter Store console interface. The top section is titled "Select secret type" with an "Info" link. It contains five radio button options: "Credentials for RDS database", "Credentials for Redshift cluster", "Credentials for DocumentDB database", "Credentials for other database", and "Other type of secrets (e.g., API key)". The "Other type of secrets" option is selected. Below this is a section titled "Specify the key/value pairs to be stored in this secret" with an "Info" link. It has two tabs: "Secret key/value" (selected) and "Plaintext". Under the "Secret key/value" tab, there is a table with two columns: "Key" and "Value". The first row is empty, and there is a "+ Add row" link below it. At the bottom, there is a section titled "Select the encryption key" with an "Info" link. It contains a text box with the selected key "DefaultEncryptionKey", a dropdown arrow, and a refresh button. Below this is a link "Add new key" with an external link icon.

In this pane, you can add a simple key-value pair. On the next screen you can add an identifying name and any tags you wish to the key, followed by a pane where you can select automatic rotation for the key if you choose.



☒ **Disable automatic rotation**
Recommended when your applications are using this secret and have not been updated to use AWS Secrets Manager.

☐ **Enable automatic rotation**
Recommended when your applications are not using this secret yet.

Select rotation interval [Info](#)
This secret will be rotated based on the schedule you determine.

30 days ▼
Maximum 365 days

Choose an AWS Lambda function [Info](#)
Select an AWS Lambda function that has permissions to rotate this secret.

▼

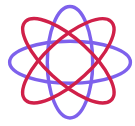
[Create function](#)

There's a lot to like about Secrets Manager, in particular the key rotation — if you haven't been obscuring your secure variables in your repos in the past, it allows for easy, hands-off rotation of these keys on a regular basis. This reduces risk in case of employee turnover or security breaches. And with encryption via KMS, you can limit access to whatever IAM users and roles actually need read/write access.

Secrets Manager stores secrets for \$0.40/secret per month, and \$0.05 per 10,000 API calls.

Systems Manager Parameter Store

By comparison, AWS Systems Manager offers a Parameter Store which is a simple key-value pair storage option. It also offers encryption via AWS KMS, which allows the same security and simplicity

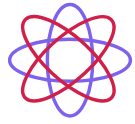


- Patch Management
- Role/Identity Association
- Scheduled commands
- Run commands on a subset of servers at once
- Organize resources into Resource Groups based on Tags
- Show compliance with Patching and Access/Permissions policies
- Store secure, encrypted variables in Parameter Store.

When it comes to storing parameters, the setup pane asks for a key name (which must be unique), and a value. You can store parameters as a basic String, a StringList, or a SecureString.

The screenshot shows the AWS Systems Manager console interface for creating a new parameter. On the left is a navigation sidebar with sections: Quick Setup, Operations Management (CloudWatch Dashboard, OpsCenter, Resource Groups, Trusted Advisor & PHD), Actions & Change (Automation, Maintenance Windows), and Instances & Nodes (Compliance, Inventory, Managed Instances, Hybrid Activations, Session Manager). The main content area is titled 'Parameter details' and includes a breadcrumb trail: 'AWS Systems Manager > Parameter Store > Create parameter'. The form contains the following fields and options:

- Name:** A text input field containing 'my_secure_variable'.
- Description- Optional:** An empty text input field.
- Tier:** A section titled 'Parameter Store offers standard and advanced parameters.' with two radio button options:
 - Standard:** Selected. Description: 'Limit of 10,000 parameters. Parameter value size up to 4 KB. Parameter policies are not available. No additional charge.'
 - Advanced:** Description: 'Can create more than 10,000 parameters. Parameter value size up to 8 KB. Parameter policies are available. Charges apply.'
- Type:** A section with three radio button options:
 - String:** Description: 'Any string value.'
 - StringList:** Description: 'Separate strings using commas.'
 - SecureString:** Selected. Description: 'Encrypt sensitive data using the KMS keys for your account.'



“Advanced” tier secrets are priced at \$0.05/advanced parameter per month.

AWS Systems Manager X

Quick Setup

Operations Management

- CloudWatch Dashboard
- OpsCenter
- Resource Groups
- Trusted Advisor & PHD

Actions & Change

- Automation
- Maintenance Windows

Instances & Nodes

- Compliance
- Inventory
- Managed Instances
- Hybrid Activations
- Session Manager
- Run Command
- State Manager

KMS key source

☒ My current account
Use the default KMS key for this account or specify a customer-managed CMK for this account. [Learn more](#)

☐ Another account
Use a KMS key from a different account. [Learn more](#)

KMS Key ID

alias/aws/ssm

Value

Maximum length 4096 characters.

Tags - Optional

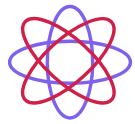
No tags associated with the resource

Add tag

Cancel Create parameter

If you choose the Advanced tier, expiration policies can be set on the parameters stored as well. Just like with Secrets Manager, additional tags can be added, and the values can be encrypted with the KMS key of your choice, making access control for your secrets more simple.

To recap, Parameter Store may offer more simplistic key-value pair storage, but is much less expensive (even at the Advanced tier). Secrets Manager offers several different storage types, most of which center around database credentials, but does offer a more



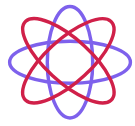
Ansible and Secret Management

With two options for secret management within AWS, it was difficult to know which to choose. We started with Secrets Manager, as Ansible offers both an [aws_secret module](#), and an [aws_secret lookup plugin](#).

aws_secret lookups

In our case, we were less interested in storing new secrets, and more interested in looking up the key and retrieving the value, for use in templates. That being the case, we chose to use the `aws_secret` lookup plugin. The example given in the documentation is:

```
- name: Create RDS instance with aws_secret
  rds:
    command: create
    instance_name: app-db
    db_engine: MySQL
    size: 10
    instance_type: db.m1.small
    username: dbadmin
    password: "{{ lookup('aws_secret', '[
```



'aws_secret' reference, and the name of the secret.

Unfortunately it was not as simple for us.

Firstly, we found that adding the region to the command was necessary, like so:

```
"{{ lookup('aws_secret', 'my_api_key',  
region='us-west-1') }}"
```

That worked well enough in our vars_files to get through the deploy, provided the server running the ansible command had the proper IAM permissions. But, to my dismay, I found that this lookup didn't return the "value" of the key-value pair, but rather a json string with BOTH the key and the value (shown below).

```
[{ 'my_api_key', 'my_api_key_value' }]
```

Unfortunately the only way I could get it to return just the "value" of the simple key-value style Secret was to add additional parsing in a script. So, as of now anyways, it looks like the Ansible aws_secret lookup plugin is limited to database secrets usage.



Systems Manager Parameter Store option instead.

As with Secrets Manager, Ansible also has

Parameter Store functionality in the form of the `aws_ssm_parameter_store_module` and the `aws_ssm_lookup plugin`. And again, since we're wanting to just read the value of secrets, we don't need to mess with the module — just the lookup plugin. Ansible provides the following examples (although there are more use cases shown in the documentation):

```
- name: lookup ssm parameter store in the
  debug: msg="{{ lookup('aws_ssm', 'Hello') }}"
```

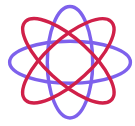
```
- name: lookup ssm parameter store in non
  debug: msg="{{ lookup('aws_ssm', 'Hello' ) }}"
```

```
- name: lookup ssm parameter store without
  debug: msg="{{ lookup('aws_ssm', 'Hello') }}"
```

```
- name: lookup ssm parameter store in non
  debug: msg="{{ lookup('aws_ssm', 'Hello' )}}
```



The examples given show easily enough how to use `aws_ssm` lookups within a playbook, but it can also be used in your `vars` files like so:



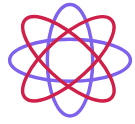
Providing your instance is setup with the proper [IAM permissions](#) to read SSM parameters and read access to the KMS key used to encrypt them (if SecureString was selected), your variables should populate into templates without having to store them in an vaulted file or vaulting/encrypting individual strings.

Automating Parameter Addition

If your project (like ours) has a lot of vars to store, you may find it very tiresome to add all the keys one by one into the Systems Manager panel in AWS. As a DevOps engineer, it made me cringe thinking of having to add the variables by hand. So, I made a script that uses the AWS CLI to upload parameters.

A couple notes:

- **This script assumes you have an AWS CLI config file setup at `~/.aws/config`, with multiple AWS account profiles.** The one referenced is called “aws-main” — replace this with your own profile, or remove the line if you only have one profile.



replace as needed.

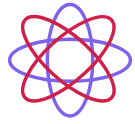
```
#!/usr/local/bin/bash -xe
declare -A vars

vars[env]=develop
vars[debug]=true
vars[key]="key_value"
#(more vars listed here...)

for i in "${!vars[@]}"
do
    aws ssm put-parameter \
    --profile "aws-main" \
    --name "dev_${i}" \
    --type "SecureString" \
    --value "${vars[$i]}" \
    --key-id "alias/dev-kms-key" \
    --tags Key=Environment,Value=Develop Ke
    --region "us-west-2"
done
```



The bash script above declares an array “vars,” of which there are keys (env, debug, key) and values (develop, true, key_value). The loop uses the key as



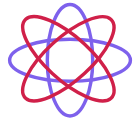
values as needed per environment, and change tags/prefixes to reflect new environments. But this script helped cut the time to add parameters in 1/4 or more! Definitely a win in my book.



Conclusions

After some trial and error, here's a recap of what we learned:

- Secrets Manager is a more robust solution that offers rotation of secrets/keys. However, it is more expensive and charges for API calls.
- If you're looking to just populate the values of secrets for your variables in Ansible, SSM Parameter Store will work better for your needs.
- Ansible's `aws_secret` lookup works best for database Secrets.



Have any success or failure stories to share with either Secrets Manager or Parameter Store? Share in the comments, or [contact me](#).

Share this:

Related

[Adding version control to an existing application](#)

March 31, 2019
In "Ansible"

[Automate Patching Using AWS Systems Manager \(SSM\)](#)

May 29, 2020
In "AWS"

[Adding Nginx HSTS Headers on AWS Load Balancer](#)

January 17, 2020
In "AWS"

Comments

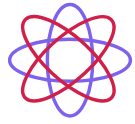


Eric says

[February 3, 2020 at 2:23 pm](#)

Still a huge gap in the AWS secret management modules as of 2.9. I still have to use the AWS cli to put secrets and parameters.

I wish they had support like Terraform



Martin Stevens says

March 20, 2020 at 5:43 am

With the secret json returned.

```
{“secret1”: “we_are_hunting_rabbits”, “secret2”:  
“i_shall_call_him_george”}
```

We should be able to:

– set_fact:

```
secret: “{{ lookup(‘aws_secret’, ‘poc-secrets’,  
region=‘eu-west-1’)}}”
```

```
no_log: true
```

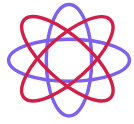
– debug:

```
msg: “{{ secret.secret1}}”
```

– debug:

```
msg: “{{ secret.secret2}}”
```

Reply



TechGirlKB

[HOME](#)[SPEAKING](#)[POSTS](#)[ABOUT JANNA HILFERTY](#)[CONTACT ME](#)

your email address will not be published. Required

fields are marked *

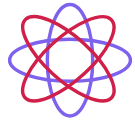
Comment

Name *

Email *

Website

☐ Notify me of follow-up comments by email.



TechGirlKB

[HOME](#)

[SPEAKING](#)

[POSTS](#)

[ABOUT JANNA HILFERTY](#)

[CONTACT ME](#)

Categories

[Ansible](#)

[AWS](#)

[Git](#)

[Linux](#)

[Optimization](#)

[Performance](#)

[PHP](#)

[Scalability](#)

[Security](#)

[Uncategorized](#)

[WordPress](#)

Copyright © 2024 · Atmosphere Pro on Genesis Framework · WordPress · [Log in](#)

Search this website