Administrator Guide

AWS Service Catalog

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Service Catalog: Administrator Guide

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

What Is Service Catalog? 1
Video: Introduction to AWS Service Catalog 2
Overview 2
Users 2
Products 2
HashiCorp Terraform Open Source and Terraform Cloud support 3
Provisioned Products 3
Portfolios 3
Versioning 4
Permissions 4
Constraints 4
Initial Administrator Workflow 5
Initial End User Workflow 5
Quotas 5
AWS Organizations 6
Constraint quotas 6
Portfolio quotas 6
Product quotas 6
Provisioned product quotas 6
Regional quotas 6
Service action quotas 7
TagOptions quotas 7
Setting Up 8
..... 8
Sign up for an AWS account 8
Create a user with administrative access 8
Grant permissions to administrators 10
Grant permissions to end users 13
Install and configure the Terraform provisioning engine 13
Queue determination 14
Adding Confused Deputy to your Terraform provisioning engine 14
Getting Started 19
Getting Started Library 19
Prerequisites 20

Learn More ... 20
Getting started with an AWS CloudFormation product ... 20
Step 1: Download the template ... 21
Step 2: Create a key pair ... 25
Step 3: Create a portfolio ... 26
Step 4: Create a new product in the portfolio ... 27
Step 5: Add a template constraint ... 28
Step 6: Add a launch constraint ... 29
Step 7: Grant end users access to the portfolio ... 31
Step 8: Test the end user experience ... 32
Getting started with a Terraform product ... 33
Updating to External product type ... 35
Prerequisite: Configure your Terraform provisioning engine ... 35
Step 1: Terraform configuration file download ... 37
Step 2: Create a Terraform product ... 38
Step 3: Create a portfolio ... 39
Step 4: Add product to portfolio ... 39
Step 5: Create launch roles ... 40
Step 6: Add a launch constraint ... 44
Step 7: Grant end user access ... 45
Step 8: Share portfolio with end user ... 46
Step 9: Test the end user experience ... 46
Step 10: Monitoring Terraform provisioning operations ... 47
Security ... 49
Data Protection ... 50
Protecting Data with Encryption ... 51
Identity and Access Management ... 51
Audience ... 51
Identity-based policy examples for AWS Service Catalog ... 52
AWS managed policies ... 57
Using service-linked roles ... 67
Troubleshooting AWS Service Catalog identity and access ... 72
Controlling Access ... 74
Logging and Monitoring ... 74
Compliance Validation ... 74
Resilience ... 75

Infrastructure Security ... 76
Security Best Practices ... 76
Managing Catalogs ... 78
Managing Portfolios ... 78
Creating, Viewing, and Deleting Portfolios ... 79
Viewing Portfolio Details ... 79
Creating and Deleting Portfolios ... 79
Adding products ... 80
Adding Constraints ... 83
Granting Access to Users ... 84
Sharing a Portfolio ... 85
Sharing and Importing Portfolios ... 92
Managing Products ... 96
Viewing the Products Page ... 97
Creating Products ... 97
Adding products to portfolios ... 100
Updating products ... 101
Syncing products to template files from external repositories ... 102
Deleting products ... 110
Managing Versions ... 118
Using Constraints ... 120
Launch Constraints ... 120
Notification Constraints ... 126
Tag Update Constraints ... 127
Stack Set Constraints ... 127
Template Constraints ... 128
Using Service Actions ... 133
Prerequisites ... 133
Step 1: Configure end user permissions ... 134
Step 2: Create a service action ... 135
Step 3: Associate the service action with a product version ... 136
Step 4: Test the end user experience ... 136
Step 5: Managing service actions with AWS CloudFormation ... 136
Step 6: Troubleshooting ... 137
Adding AWS Marketplace Products to Your Portfolio ... 139
Managing AWS Marketplace Products Using AWS Service Catalog ... 139

Managing and Adding AWS Marketplace Products Manually ... 140
Using AWS CloudFormation StackSets ... 145
Stack sets vs. stack instances ... 145
Stack set constraints ... 145
Managing Budgets ... 145
Prerequisites ... 146
Creating a budget ... 148
Associating a Budget ... 148
Viewing a Budget ... 149
Disassociating a Budget ... 149
Managing Provisioned Products ... 151
Managing provisioned products as the administrator ... 151
Changing Provisioned Product Owner ... 152
See Also ... 152
Updating templates for provisioned products ... 153
Tutorial: Identifying User Resource Allocation ... 154
Managing Terraform Open Source product status errors ... 157
Status error examples ... 158
Managing the Terraform Open Source product state file ... 159
Managing Tags ... 160
AutoTags ... 160
TagOption Library ... 161
Launching a Product with TagOptions ... 162
Managing TagOptions ... 166
Using TagOptions with AWS Organizations tag policies ... 168
External Engines ... 172
Considerations ... 173
Parameter Parsing ... 173
Provisioning ... 176
Updating ... 180
Terminating ... 183
Tagging ... 184
Monitoring ... 186
Monitoring Tools ... 186
Automated Tools ... 186
CloudWatch Metrics ... 187

Enabling CloudWatch Metrics 187
Available Metrics and Dimensions 187
Viewing AWS Service Catalog Metrics 189
CloudTrail logs 190
AWS Service Catalog information in CloudTrail 190
Understanding AWS Service Catalog log file entries 191
Console branding 193
AWS Region support for console branding 193
Document History 196
Earlier Updates 197

What Is Service Catalog?

Service Catalog enables organizations to create and manage catalogs of IT services that are approved for AWS. These IT services can include everything from virtual machine images, servers, software, databases, and more to complete multi-tier application architectures.

Service Catalog allows organizations to centrally manage commonly deployed IT services, and helps organizations achieve consistent governance and meet compliance requirements. End users can quickly deploy only the approved IT services they need, following the constraints set by your organization.

Service Catalog provides the following benefits:

- **Standardization**

  Administer and manage approved assets by restricting where the product can be launched, the type of instance that can be used, and many other configuration options. The result is a standardized landscape for product provisioning for your entire organization.
- **Self-service discovery and launch**

  Users browse listings of products (services or applications) that they have access to, locate the product that they want to use, and launch it all on their own as a provisioned product.
- **Fine-grain access control**

  Administrators assemble portfolios of products from their catalog, add constraints and resource tags to be used at provisioning, and then grant access to the portfolio through AWS Identity and Access Management (IAM) users and groups.
- **Extensibility and version control**

  Administrators can add a product to any number of portfolios and restrict it without creating another copy. Updating the product to a new version propagates the update to all products in every portfolio that references it.

For more information, see the Service Catalog detail page.

The Service Catalog API provides programmatic control over all end-user actions as an alternative to using the AWS Management Console. For more information, see Service Catalog Developer Guide.

Video: Introduction to AWS Service Catalog

This video (7:27) describes how to create, organize, and govern a curated catalog of AWS products, and share products with permissions level. As a result, end users can quickly provision approved IT resources without direct access to the underlying AWS services.

Introduction to AWS Service Catalog

Overview of Service Catalog

As you get started with Service Catalog, you'll benefit from understanding its components and the initial workflows for administrators and end users.

Users

Service Catalog supports the following types of users:

- **Catalog administrators (administrators)** – Manage a catalog of products (applications and services), organizing them into portfolios and granting access to end users. Catalog administrators prepare AWS CloudFormation templates, configure constraints, and manage IAM roles for products to provide for advanced resource management.
- **End users** – Receive AWS credentials from their IT department or manager and use the AWS Management Console to launch products to which they have been granted access. Sometimes referred to as simply *users*, end users may be granted different permissions depending on your operational requirements. For example, a user may have the maximum permission level (to launch and manage all of the resources required by the products they use) or only permission to use particular service features.

Products

A *product* is an IT service that you want to make available for deployment on AWS. A product consists of one or more AWS resources, such as EC2 instances, storage volumes, databases, monitoring configurations, and networking components, or packaged AWS Marketplace products. A product can be a single compute instance running AWS Linux, a fully configured multi-tier web application running in its own environment, or anything in between.

You create a product by importing an AWS CloudFormation template. AWS CloudFormation templates define the AWS resources required for the product, the relationships between resources,

and the parameters that end users can plug in when they launch the product to configure security groups, create key pairs, and perform other customizations.

HashiCorp Terraform Open Source and Terraform Cloud support

AWS Service Catalog enables quick, self-service provisioning with governance for your HashiCorp Terraform Open Source and Terraform Cloud configurations within AWS. You can use Service Catalog as a single tool to organize, govern, and distribute your Terraform configurations at scale within AWS. You can access Service Catalog key features, including cataloging of standardized and pre-approved Terraform templates, access control, least-privilege provisioning, versioning, tagging, and sharing to thousands of AWS accounts. Your end users see a simple list of products and versions they have access to, and can then deploy those products in a single action.

To learn more and to complete a Terraform product tutorial, review Getting started with a Terraform product.

Provisioned Products

AWS CloudFormation stacks make it easier to manage the life cycle of your product by enabling you to provision, tag, update, and terminate your product instance as a single unit. An AWS CloudFormation stack includes an AWS CloudFormation template, written in either JSON or YAML format, and its associated collection of resources. A *provisioned product* is a stack. When an end user launches a product, the instance of the product that is provisioned by Service Catalog is a stack with the resources necessary to run the product. For more information, see AWS CloudFormation User Guide.

Portfolios

A *portfolio* is a collection of *products* that contains configuration information. Portfolios help manage who can use specific products and how they can use them. With Service Catalog, you can create a customized portfolio for each type of user in your organization and selectively grant access to the appropriate portfolio. When you add a new *version* of a product to a portfolio, that version is automatically available to all current users.

You also can share your portfolios with other AWS accounts and allow the administrator of those accounts to distribute your portfolios with additional *constraints*, such as limiting which EC2 instances a user can create. Through the use of portfolios, permissions, sharing, and constraints, you can ensure that users are launching products that are configured properly for the organization's needs and standards.

Versioning

Service Catalog allows you to manage multiple versions of the products in your catalog. This approach allows you to add new versions of templates and associated resources based on software updates or configuration changes.

When you create a new version of a product, the update is automatically distributed to all users who have access to the product, allowing the user to select which version of the product to use. Users can update running instances of the product to the new version quickly and easily.

Permissions

Granting a user access to a portfolio enables that user to browse the portfolio and launch the products in it. You apply AWS Identity and Access Management (IAM) permissions to control who can view and modify your catalog. IAM permissions can be assigned to IAM users, groups, and roles.

When a user launches a product that has an IAM role assigned to it, Service Catalog uses the role to launch the product's cloud resources using AWS CloudFormation. By assigning an IAM role to each product, you can avoid giving users permissions to perform unapproved operations and enable them to provision resources using the catalog.

Constraints

Constraints control the ways that you can deploy specific AWS resources for a product. You can use them to apply limits to products for governance or cost control. There are different types of AWS Service Catalog constraints: *launch constraints*, *notification constraints*, and *template constraints*.

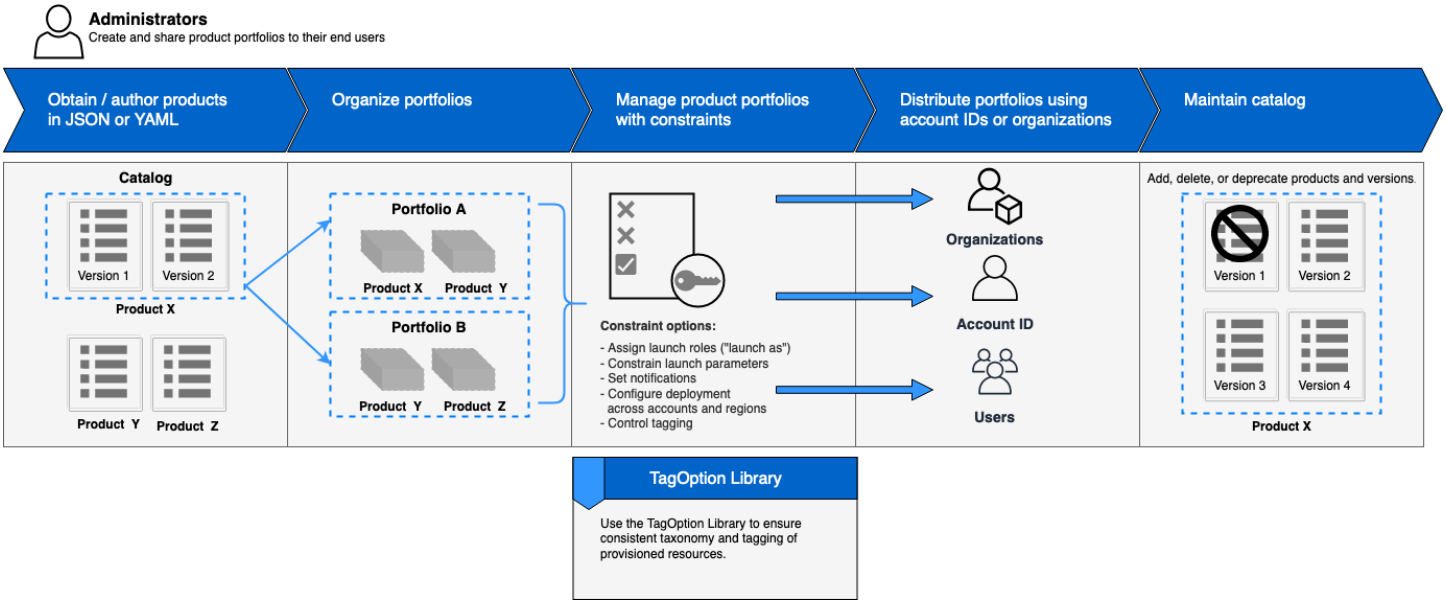
With launch constraints, you specify a role for a product in a portfolio. Use this role to provision the resources at launch, so you can restrict user permissions without impacting users' ability to provision products from the catalog.

Notification constraints enable you to get notifications about stack events using an Amazon SNS topic.

Template constraints restrict the configuration parameters that are available for the user when launching the product (for example, EC2 instance types or IP address ranges). With template constraints, you reuse generic AWS CloudFormation templates for products and apply restrictions to the templates on a per-product or per-portfolio basis.

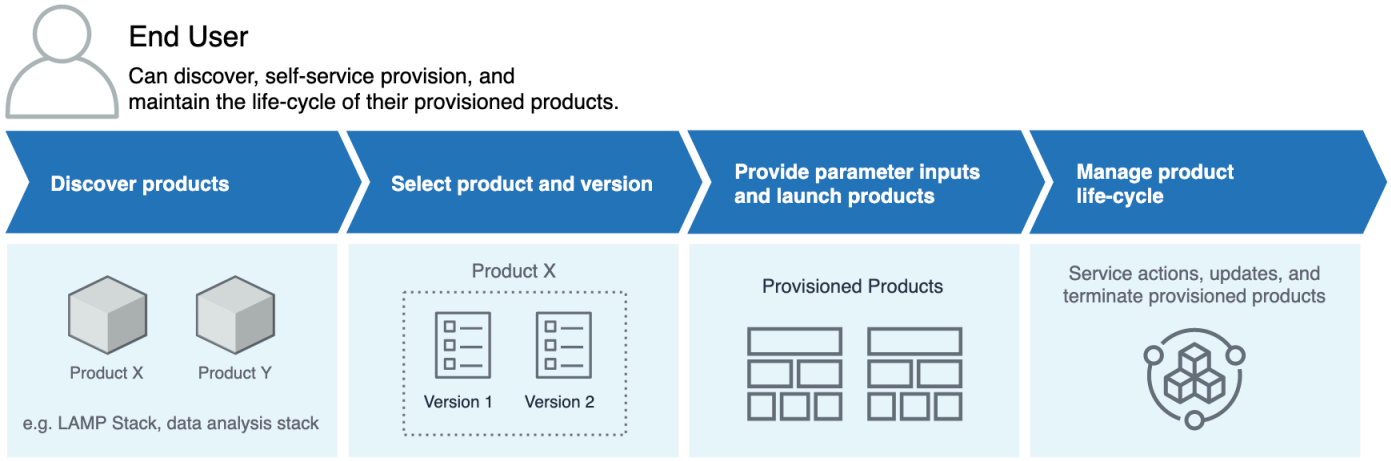
Initial Administrator Workflow

This diagram shows the initial workflow for an administrator to create a catalog.



Initial End User Workflow

This diagram shows the initial workflow for an end user.



AWS Service Catalog default service quotas

Your AWS account has the following default quotas for AWS Organizations, constraint, portfolio, product, provisioned product, regional, service action, and TagOptions.

You can use Service Quotas to manage your quotas or to request a quota increase. For more information about Service Quotas, see What Is Service Quotas? in the *Service Quotas User Guide*. To learn how to request a quota increase, see Requesting a Quota Increase.

AWS Organizations

- AWS Service Catalog delegated administrators per organization: 50

Constraint quotas

- Constraints per product per portfolio: 100

Portfolio quotas

- Users, groups, and roles per portfolio: 100
- Products per portfolio: 150
- Tags per portfolio: 20
- Shared accounts per portfolio: 5000
- Tag values per tag key: 25

Product quotas

- Users, groups, and roles per product: 200
- Product versions per product: 100
- Tags per product: 20
- Tag values per tag key: 25

Provisioned product quotas

- Tags per provisioned product: 50

Regional quotas

- Portfolios: 100

- Products: 350

Service action quotas

- Service actions per region: 200
- Service action associations per product version: 25

TagOptions quotas

- TagOptions per resource: 25
- Values per TagOption: 25

Setting Up AWS Service Catalog

Before you get started with AWS Service Catalog, complete the following tasks.

Topics

- Sign up for an AWS account
- Create a user with administrative access

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Create a role for a third-party identity provider (federation) in the *IAM User Guide*.

- IAM users:
  - Create a role that your user can assume. Follow the instructions in Create a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

Grant permissions to AWS Service Catalog administrators

As a catalog administrator, you require access to the AWS Service Catalog administrator console view and IAM permissions that allow you to perform tasks such as the following:

- Creating and managing portfolios
- Creating and managing products
- Adding template constraints to control the options that are available to end users when launching a product
- Adding launch constraints to define the IAM roles that AWS Service Catalog assumes when end users launch products
- Granting end users access to your products

You, or an administrator who manages your IAM permissions, must attach policies to your IAM user, group, or role that are required to complete this tutorial.

To grant permissions to a catalog administrator

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

2. In the navigation pane, choose **Access management**, and then choose **Users**. If you already created an IAM user that you would like to use as the catalog administrator, choose the user name, and then choose **Add permissions**. Otherwise, create a user as follows:

   a. Choose **Add user**.

   b. For **User name**, type **`ServiceCatalogAdmin`**.

   c. Select **Programmatic access** and **AWS Management Console access**.

   d. Choose **Next: Permissions**.

3. Choose **Attach existing policies directly**.

4. Choose **Create policy**, and then do the following:

   a. Choose the **JSON** tab.

   b. Copy the following example policy, and paste it in **Policy Document**:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateKeyPair",
                "iam:AddRoleToInstanceProfile",
                "iam:AddUserToGroup",
                "iam:AttachGroupPolicy",
                "iam:CreateAccessKey",
                "iam:CreateGroup",
                "iam:CreateInstanceProfile",
                "iam:CreateLoginProfile",
                "iam:CreateRole",
                "iam:CreateUser",
                "iam:Get*",
                "iam:List*",
                "iam:PutRolePolicy",
                "iam:UpdateAssumeRolePolicy"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
```

```
}
```

c. Choose **Next: Tags**.

d. (Optional) Choose **Add tag** to associate a key-value pair with the resource. You can add a maximum of 50 tags.

> **Note**
>
> Tags are key-value pairs that you can add to resources. This helps identify, organize, and search for resources. For more information, see Tagging AWS resources in the *AWS General Reference Reference Guide*.

e. Choose **Next: Review**.

f. For **Policy Name**, type **`ServiceCatalogAdmin-AdditionalPermissions`**.

> **Important**
>
> You must grant administrators Amazon S3 permissions to access templates that AWS Service Catalog stores in Amazon S3. For more information, see User Policy Examples in the *Amazon Simple Storage Service User Guide*.

g. Choose **Create Policy**.

5. Return to the browser window with the permissions page and choose **Refresh**.
6. In the search field, type **`ServiceCatalog`** to filter the policy list.
7. Select the checkboxes for the **`AWSServiceCatalogAdminFullAccess`** and **`ServiceCatalogAdmin-AdditionalPermissions`** policies, and then choose **Next: Review**.
8. If you are updating a user, choose **Add permissions**.

   If you are creating a user, choose **Create user**. You can download or copy the credentials and then choose **Close**.
9. To sign in as the catalog administrator, use your account-specific URL. To find this URL, choose **Dashboard** in the navigation pane and choose **Copy Link**. Paste the link in your browser, and use the name and password of the IAM user you created or updated in this procedure.

Grant permissions to AWS Service Catalog end users

Before the end user can use AWS Service Catalog, you must grant access to the AWS Service Catalog end user console view. To grant access, you attach policies to the IAM user, group, or role that is used by the end user. In the following procedure, we attach the **`AWSServiceCatalogEndUserFullAccess`** policy to an IAM group.

To grant permissions to an end user group

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **User groups**.
3. Choose **Create group** and do the following:
    a. For **User group name**, type **`Endusers`**.
    b. In the search field, type **`AWSServiceCatalog`** to filter the policy list.
    c. Select the checkbox for the **`AWSServiceCatalogEndUserFullAccess`** policy. You also have the option to choose **`AWSServiceCatalogEndUserReadOnlyAccess`** instead.
    d. Choose **Create Group**.
4. In the navigation pane, choose **Users**.
5. Choose **Add users** and do the following:
    a. For **User name**, type a name for the user.
    b. Select **Password - AWS Management Console access**.
    c. Choose **Next: Permissions**.
    d. Choose **Add user to group**.
    e. Select the checkbox for the **Endusers** group and choose **Next: Tags** and then **Next: Review**.
    f. On the **Review** page, choose **Create user**. Download or copy the credentials and then choose **Close**.

Install and configure the Terraform provisioning engine

To successfully use Terraform products with AWS Service Catalog, you must install and configure a Terraform provisioning engine in the same account where you will be administering Terraform products. To get started, you can use the Terraform provisioning engine provided by AWS, which

installs and configures the code and infrastructure required for the Terraform provisioning engine to work with AWS Service Catalog. This one-time setup takes approximately 30 minutes. AWS Service Catalog provides a GitHub repository with instructions on installing and configuring the Terraform provisioning engine.

Queue determination

When you call a provisioning operation, AWS Service Catalog prepares a payload message to send to the relevant queue in the provisioning engine. In order to build the ARN for the queue, AWS Service Catalog makes a the following assumptions:

- The provisioning engine is located in the account of the product owner
- The provisioning engine is located in the same region in which the call to AWS Service Catalog was made
- The provisioning engine queues follows the documented naming schema detailed below

For example, if ProvisionProduct is called in `us-east-1` from account 1111111111 using a product created by account 0000000000000, AWS Service Catalog assumes the correct SQS ARN is `arn:aws:sqs:us-east-1:0000000000000:ServiceCatalogTerraformOSProvisionOperationQueue`.

The same logic applies for the Lambda function called by `DescribeProvisioningParameters`.

Adding Confused Deputy to your Terraform provisioning engine

Confused Deputy context keys on the endpoints to restrict access for `lambda:Invoke` operations

The parameter parser Lambda function created by AWS Service Catalog-provided engines has an access policy that grants cross-account `lambda:Invoke` permission only to the AWS Service Catalog service principal:

```
{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
```

```
            "Service": "servicecatalog.amazonaws.com"
          },
          "Action": "lambda:InvokeFunction",
          "Resource": "arn:aws:lambda:us-
east-1:account_id:function:ServiceCatalogTerraformOSParameterParser"
    }
  ]
}
```

This should be the only permission necessary in order for the integration with AWS Service Catalog to function properly. However, you can constrain this further using the `aws:SourceAccount` Confused Deputy context key. When AWS Service Catalog sends messages to these queues, AWS Service Catalog populates the key with the provisioning account's ID. This is helpful when you intend to distribute products via portfolio sharing and want to ensure that only specific accounts are using your engine.

For example, you can restrict your engine to only allow requests that originate from 000000000000 and 111111111111 using the condition shown below:

```
{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "servicecatalog.amazonaws.com"
          },
          "Action": "lambda:InvokeFunction",
          "Resource": "arn:aws:lambda:us-
east-1:account_id:function:ServiceCatalogTerraformOSParameterParser",
          "Condition": {
            "StringLike": {
              "aws:SourceAccount": ["000000000000", "111111111111"]
        }
      }
    }
  ]
}
```

Confused Deputy context keys on the endpoints to restrict access for `sqs:SendMessage` operations

The provisioning operation intake Amazon SQS queues created by AWS Service Catalog-provided engines have an access policy that grants cross-account `sqs:SendMessage` (and associated KMS) permissions only to the AWS Service Catalog service principal:

```
{
      "Version": "2008-10-17",
      "Statement": [
        {
          "Sid": "Enable AWS Service Catalog to send messages to the queue",
          "Effect": "Allow",
          "Principal": {
            "Service": "servicecatalog.amazonaws.com"
          },
          "Action": "sqs:SendMessage",
          "Resource": [
            "arn:aws:sqs:us-
east-1:account_id:ServiceCatalogTerraformOSProvisionOperationQueue"
          ]
        },
        {
          "Sid": "Enable AWS Service Catalog encryption/decryption permissions when
 sending message to queue",
          "Effect": "Allow",
          "Principal": {
            "Service": "servicecatalog.amazonaws.com"
          },
          "Action": [
            "kms:DescribeKey",
            "kms:Decrypt",
            "kms:ReEncrypt",
            "kms:GenerateDataKey"
          ],
          "Resource": "arn:aws:kms:us-east-1:account_id:key/key_id"
    }
  ]
}
```

This should be the only permission necessary in order for the integration with AWS Service Catalog to function properly. However, you can constrain this further using the `aws:SourceAccount`

Confused Deputy context key. When AWS Service Catalog sends messages to these queues, AWS Service Catalog populates the keys with the provisioning account's ID. This is helpful when you intend to distribute products via portfolio sharing and want to ensure that only specific accounts are using your engine.

For example, you can restrict your engine to only allow requests that originate from 000000000000 and 111111111111 using the condition shown below:

```
{
      "Version": "2008-10-17",
      "Statement": [
      {
        "Sid": "Enable AWS Service Catalog to send messages to the queue",
        "Effect": "Allow",
        "Principal": {
          "Service": "servicecatalog.amazonaws.com"
        },
        "Action": "sqs:SendMessage",
        "Resource": [
          "arn:aws:sqs:us-
east-1:account_id:ServiceCatalogTerraformOSProvisionOperationQueue"
        ],
        "Condition": {
          "StringLike": {
            "aws:SourceAccount": ["000000000000", "111111111111"]
          }
        }
      },
      {
        "Sid": "Enable AWS Service Catalog encryption/decryption permissions when
 sending message to queue",
        "Effect": "Allow",
        "Principal": {
          "Service": "servicecatalog.amazonaws.com"
        },
        "Action": [
          "kms:DescribeKey",
          "kms:Decrypt",
          "kms:ReEncrypt",
          "kms:GenerateDataKey"
        ],
        "Resource": "arn:aws:kms:us-east-1:account_id:key/key_id"
    }
```

```
    ]
}
```

Getting Started

You can get started with AWS Service Catalog by using one of the well-architected product templates in the Getting Started Library or by following the steps in one of the getting started tutorials.

In the tutorial, you perform tasks as the catalog administrator and end user. As the catalog administrator, you create a portfolio and then a product. As the end user, you verify that you can access the end user console and launch the product. The product is one of the following:

- A cloud development environment that runs on Amazon Linux and is based on an AWS CloudFormation template that defines the AWS resources the product can use.
- An open source environment that runs on a Terraform provisioning engine and is based on a tar.gz configuration file that defines the AWS resources the product can use.

> **Note**
>
> Before you begin, make sure that you complete the action items in Setting Up AWS Service Catalog.

Topics

- Getting Started Library
- Getting started with an AWS CloudFormation product
- Getting started with a Terraform product

Getting Started Library

AWS Service Catalog provides a Getting Started Library of well-architected product templates so you can get started quickly. You can copy any of the products in our Getting Started Library portfolios to your own account, then customize them to suit your needs.

Topics

- Prerequisites
- Learn More

Prerequisites

Before you use the templates in our Getting Started Library, make sure you have the following:

- The required permissions to use AWS CloudFormation templates. For more information, see Controlling Access with AWS Identity and Access Management.
- The required administrator permissions to manage AWS Service Catalog. For more information, see the section called "Identity and Access Management".

Learn More

For more information about the well-architected framework, see AWS Well-Architected.

Getting started with an AWS CloudFormation product

You can get started with AWS Service Catalog by using one of the well-architected product templates in the Getting Started Library or by following the steps in the getting started tutorial.

In the tutorial, you perform tasks as the catalog administrator and end user. As the catalog administrator, you create a porfolio and then a product. As the end user, you verify that you can access the end user console and launch the product. The product is a cloud development environment that runs on Amazon Linux and is based on an AWS CloudFormation template that defines the AWS resources the product can use.

> **Note**
>
> Before you begin, make sure that you complete the action items in Setting Up AWS Service Catalog.

Topics

- Step 1: Download the AWS CloudFormation template
- Step 2: Create a key pair
- Step 3: Create a portfolio
- Step 4: Create a new product in the portfolio
- Step 5: Add a template constraint to limit instance size
- Step 6: Add a launch constraint to assign an IAM role

- Step 7: Grant end users access to the portfolio
- Step 8: Test the end user experience

Step 1: Download the AWS CloudFormation template

You can use AWS CloudFormation templates to configure and provision portfolios and products. These templates are text files that can be formatted in JSON or YAML and describe the resources that you want to provision. For more information, see Template Formats in the *AWS CloudFormation User Guide*. You can use the AWS CloudFormation editor or a text editor of your choice to create and save templates. In this tutorial, we provide a simple template, so you can get started. The template launches a single Linux instance that's configured for SSH access.

> **Note**
>
> Using AWS CloudFormation templates requires special permissions. Before you begin, make sure that you have the correct permissions. For more information, see the prerequisites in Getting Started Library.

Template Download

The sample template provided for this tutorial, `development-environment.template`, is available at https://awsdocs.s3.amazonaws.com/servicecatalog/development-environment.template.

Template Overview

The text of the sample template follows:

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",

  "Description" : "AWS Service Catalog sample template. Creates an Amazon EC2 instance
                   running the Amazon Linux AMI. The AMI is chosen based on the
 region
                   in which the stack is run. This example creates an EC2 security
                   group for the instance to give you SSH access. **WARNING** This
                   template creates an Amazon EC2 instance. You will be billed for
 the
                   AWS resources used if you create a stack from this template.",
```

```
 "Parameters" : {
   "KeyName": {
     "Description" : "Name of an existing EC2 key pair for SSH access to the EC2
instance.",
     "Type": "AWS::EC2::KeyPair::KeyName"
   },

   "InstanceType" : {
     "Description" : "EC2 instance type.",
     "Type" : "String",
     "Default" : "t2.micro",
     "AllowedValues" : [ "t2.micro", "t2.small", "t2.medium", "m3.medium",
"m3.large",
       "m3.xlarge", "m3.2xlarge" ]
   },

   "SSHLocation" : {
     "Description" : "The IP address range that can SSH to the EC2 instance.",
     "Type": "String",
     "MinLength": "9",
     "MaxLength": "18",
     "Default": "0.0.0.0/0",
     "AllowedPattern": "(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})/(\\d{1,2})",
     "ConstraintDescription": "Must be a valid IP CIDR range of the form x.x.x.x/x."
  }
 },

 "Metadata" : {
   "AWS::CloudFormation::Interface" : {
     "ParameterGroups" : [{
       "Label" : {"default": "Instance configuration"},
       "Parameters" : ["InstanceType"]
     },{
       "Label" : {"default": "Security configuration"},
       "Parameters" : ["KeyName", "SSHLocation"]
     }],
     "ParameterLabels" : {
       "InstanceType": {"default": "Server size:"},
       "KeyName": {"default": "Key pair:"},
       "SSHLocation": {"default": "CIDR range:"}
     }
   }
 },
```

```
  "Mappings" : {
    "AWSRegionArch2AMI" : {
      "us-east-1"      : { "HVM64" : "ami-08842d60" },
      "us-west-2"      : { "HVM64" : "ami-8786c6b7" },
      "us-west-1"      : { "HVM64" : "ami-cfa8a18a" },
      "eu-west-1"      : { "HVM64" : "ami-748e2903" },
      "ap-southeast-1" : { "HVM64" : "ami-d6e1c584" },
      "ap-northeast-1" : { "HVM64" : "ami-35072834" },
      "ap-southeast-2" : { "HVM64" : "ami-fd4724c7" },
      "sa-east-1"      : { "HVM64" : "ami-956cc688" },
      "cn-north-1"     : { "HVM64" : "ami-ac57c595" },
      "eu-central-1"   : { "HVM64" : "ami-b43503a9" }
    }

  },

  "Resources" : {
    "EC2Instance" : {
      "Type" : "AWS::EC2::Instance",
      "Properties" : {
        "InstanceType" : { "Ref" : "InstanceType" },
        "SecurityGroups" : [ { "Ref" : "InstanceSecurityGroup" } ],
        "KeyName" : { "Ref" : "KeyName" },
        "ImageId" : { "Fn::FindInMap" : [ "AWSRegionArch2AMI", { "Ref" :
"AWS::Region" }, "HVM64" ] }
      }
    },

    "InstanceSecurityGroup" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "GroupDescription" : "Enable SSH access via port 22",
        "SecurityGroupIngress" : [ {
          "IpProtocol" : "tcp",
          "FromPort" : "22",
          "ToPort" : "22",
          "CidrIp" : { "Ref" : "SSHLocation"}
        } ]
      }
    }
  },

  "Outputs" : {
```

```
    "PublicDNSName" : {
      "Description" : "Public DNS name of the new EC2 instance",
      "Value" : { "Fn::GetAtt" : [ "EC2Instance", "PublicDnsName" ] }
    },
    "PublicIPAddress" : {
      "Description" : "Public IP address of the new EC2 instance",
      "Value" : { "Fn::GetAtt" : [ "EC2Instance", "PublicIp" ] }
    }
  }
}
```

Template Resources

The template declares resources to be created when the product is launched. It consists of the following sections:

- **AWSTemplateFormatVersion** (optional) – The version of the AWS Template Format used to create this template. The latest template format version is 2010-09-09 and is currently the only valid value.
- **Description** (optional) – A description of the template.
- **Parameters** (optional) – The parameters that your user must specify to launch the product. For each parameter, the template includes a description and constraints that must be met by the value typed. For more information about constraints, see Using AWS Service Catalog Constraints.

  The `KeyName` parameter allows you to specify an Amazon Elastic Compute Cloud (Amazon EC2) key pair name that end users must provide when they use AWS Service Catalog to launch your product. You will create the key pair in the next step.
- **Metadata** (optional) – Objects that provide additional information about the template. The AWS::CloudFormation::Interface key defines how the end user console view displays parameters. The `ParameterGroups` property defines how parameters are grouped and headings for those groups. The `ParameterLabels` property defines friendly parameter names. When a user is specifying parameters to launch a product that is based on this template, the end user console view displays the parameter labeled `Server size:` under the heading `Instance configuration`, and it displays the parameters labeled `Key pair:` and `CIDR range:` under the heading `Security configuration`.
- **Mappings** (optional) – A mapping of keys and associated values that you can use to specify conditional parameter values, similar to a lookup table. You can match a key to a corresponding value by using the Fn::FindInMap intrinsic function in the Resources and Outputs sections.

The template above includes a list of AWS Regions and the Amazon Machine Image (AMI) that corresponds to each. AWS Service Catalog uses this mapping to determine which AMI to use based on the AWS Region that the user selects in the AWS Management Console.

- **Resources** (required) – Stack resources and their properties. You can refer to resources in the **Resources** and **Outputs** sections of the template. In the template above, we specify an EC2 instance running Amazon Linux and a security group that allows SSH access to the instance. The **Properties** section of the EC2 instance resource uses the information that the user types to configure the instance type and a key name for SSH access.

  AWS CloudFormation uses the current AWS Region to select the AMI ID from the mappings defined earlier and assigns a security group to it. The security group is configured to allow inbound access on port 22 from the CIDR IP address range that the user specifies.

- **Outputs** (optional) – Text that tells the user when the product launch is complete. The provided template gets the public DNS name of the launched instance and displays it to the user. The user needs the DNS name to connect to the instance using SSH.

  For more information about the Template anatomy page, see Template reference in the *AWS CloudFormation User Guide*.

Step 2: Create a key pair

To enable your end users to launch the product that is based on the sample template for this tutorial, you must create an Amazon EC2 key pair. A key pair is a combination of a public key that is used to encrypt data and a private key that is used to decrypt data. For more information about key pairs, ensure you are signed into the AWS console and then review Amazon EC2 Key Pairs in the *Amazon EC2 User Guide*.

The AWS CloudFormation template for this tutorial, `development-environment.template`, includes the `KeyName` parameter:

```
. . .
  "Parameters" : {
    "KeyName": {
      "Description" : "Name of an existing EC2 key pair for SSH access to the EC2
 instance.",
      "Type": "AWS::EC2::KeyPair::KeyName"
    },
```

```
. . .
```

End users must specify the name of a key pair when they use AWS Service Catalog to launch the product that is based on the template.

If you already have a key pair in your account that you would prefer to use, you can skip ahead to Step 3: Create a portfolio. Otherwise, complete the following steps.

To create a key pair

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane, under **Network & Security**, choose **Key Pairs**.
3. On the **Key Pairs** page, choose **Create Key Pair**.
4. For **Key pair name**, type a name that is easy for you to remember, and then choose **Create**.
5. When the console prompts you to save the private key file, save it in a safe place.

> **Important**
>
> This is the only chance for you to save the private key file.

Step 3: Create a portfolio

To provide users with products, begin by creating a portfolio for those products.

To create a portfolio

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. In the left navigation panel, choose **Portfolios**, and then choose **Create portfolio**.
3. Type the following values:
   - **Portfolio name** – **`Engineering Tools`**
   - **Portfolio description** – **`Sample portfolio that contains a single product.`**
   - **Owner** – **`IT (it@example.com)`**
4. Choose **Create**.

Step 4: Create a new product in the portfolio

After you have created a portfolio, you are ready to create a product within the portfolio. For this tutorial, you will create a product called **Linux Desktop**, a cloud development environment that runs on Amazon Linux, inside of the **Engineering Tool** portfolio.

To create a product within a portfolio

1. If you've just completed the previous step, the **Portfolios** page is already displayed. Otherwise, open https://console.aws.amazon.com/servicecatalog/.
2. Choose and open the **Engineering Tool** portfolio you created in Step 2.
3. Choose **Upload new product.**
4. On the **Create product** page in the Product details section, enter the following:
   - **Product name** – `Linux Desktop`
   - **Product description** – `Cloud development environment configured for engineering staff. Runs AWS Linux.`
   - **Owner** – `IT`
   - **Distributor** – *(blank)*
5. On the **Version details** page, choose **Use a CloudFormation template**. Then choose **Specify an Amazon S3 template URL** and enter the following:
   - **Select template** – `https://awsdocs.s3.amazonaws.com/servicecatalog/development-environment.template`
   - **Version title** – `v1.0`
   - **Description** – `Base Version`
6. In the **Support details** section, enter the following:
   - **Email contact** – `ITSupport@example.com`
   - **Support link** – `https://wiki.example.com/IT/support`
   - **Support description** – `Contact the IT department for issues deploying or connecting to this product.`
7. Choose **Create product**.

Step 5: Add a template constraint to limit instance size

Constraints add another layer of control over products at the portfolio level. Constraints can control the launch context of a product (launch constraints), or add rules to the AWS CloudFormation template (template constraints). For more information, see Using AWS Service Catalog Constraints.

Add a template constraint to the Linux Desktop product that prevents users from selecting large instance types at launch time. The development-environment template allows the user to select from six instance types; this constraint limits valid instance types to the two smallest types, `t2.micro` and `t2.small`. For more information, see T2 Instances in the *Amazon EC2 User Guide*.

To add a template constraint to the Linux Desktop product

1. On the **Portfolio details** page, choose **Constraints**, then choose **Create constraint**.
2. In the **Create constraint** page, for **Product,** choose **Linux Desktop**. Then, for **Constraint type**, choose **Template**.
3. In the T**emplate constraint** section, choose **Text editor**.
4. Paste the following into the text editor:

```
{
  "Rules": {
    "Rule1": {
      "Assertions": [
        {
          "Assert" : {"Fn::Contains": [["t2.micro", "t2.small"], {"Ref":
 "InstanceType"}]},
          "AssertDescription": "Instance type should be t2.micro or t2.small"
        }
      ]
    }
  }
}
```

5. For **Constraint description**, enter **`Small instance sizes`**.
6. Choose **Create**.

Step 6: Add a launch constraint to assign an IAM role

A launch constraint designates an IAM role that AWS Service Catalog assumes when an end user launches a product.

For this step, you add a launch constraint to the Linux Desktop product, so AWS Service Catalog can use the IAM resources that make up the product's AWS CloudFormation template.

The IAM role that you assign to a product as a launch constraint must have the following permissions

1. AWS CloudFormation
2. Services in the AWS CloudFormation template for the product
3. Read access to the AWS CloudFormation template in a service-owned Amazon S3 bucket.

This launch constraint enables the end user to launch the product and, after launch, manage it as a provisioned product. For more information, see AWS Service Catalog Launch Constraints.

Without a launch constraint, you need to grant additional IAM permissions to your end users before they can use the Linux Desktop product. For example, the `ServiceCatalogEndUserAccess` policy grants the minimum IAM permissions required to access the AWS Service Catalog end user console view.

Using a launch constraint allows you follow the IAM best practice of keeping end user IAM permissions to a minimum. For more information, see Grant least privilege in the *IAM User Guide*.

To add a launch constraint

1. Follow the instructions to Create new policies on the JSON tab in the *IAM User guide*.
2. Paste the following JSON policy document:

   - `cloudformation`– Allows AWS Service Catalog full permissions to create, read, update, delete, list, and tag AWS CloudFormation stacks.
   - `ec2`— Allows AWS Service Catalog full permissions to list, read, write, provision, and tag Amazon Elastic Compute Cloud (Amazon EC2) resources that are part of the AWS Service Catalog product. Depending on the AWS resource that you want to deploy, this permission might change.

- `ec2`– Creates a new managed policy for you AWS account and attaches the specified managed policy to the specified IAM role.
- `s3`— Allows access to Amazon S3 buckets owned by AWS Service Catalog. To deploy the product, AWS Service Catalog requires access to provisioning artifacts.
- `servicecatalog`— Allows AWS Service Catalog permissions to list, read, write, tag, and launch resources on behalf of the end-user.
- `sns`— Allows AWS Service Catalog permissions to list, read, write, and tag Amazon SNS topics for the launch constraint.

> **Note**
>
> Depending on the underlying resources that you want to deploy, you might need to modify the example JSON policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStackEvents",
                "cloudformation:DescribeStacks",
                "cloudformation:GetTemplateSummary",
                "cloudformation:SetStackPolicy",
                "cloudformation:ValidateTemplate",
                "cloudformation:UpdateStack",
                "ec2:*",
                "servicecatalog:*",
                "sns:*"
            ],
            "Resource": "*"
        },
        {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject"
```

```
            ],
            "Resource":"*",
            "Condition":{
               "StringEquals":{
                  "s3:ExistingObjectTag/servicecatalog:provisioning":"true"
               }
            }
         }
      ]
}
```

3. Choose **Next**, **Tags**.
4. Choose **Next, Review**.
5. In the **Review policy** page, for the **Name**, enter **`linuxDesktopPolicy`**.
6. Choose **Create policy**.
7. In the navigation pane, choose **Roles**. Then choose **Create role** and do the following:
   a. For **Select trusted entity**, choose **AWS service** and then under **Use case for other AWS services** choose **Service Catalog**. Select the Service Catalog use case and then choose **Next**.
   b. Search for the **linuxDesktopPolicy** policy and then select the checkbox.
   c. Choose **Next**.
   d. For **Role name**, type **`linuxDesktopLaunchRole`**.
   e. Choose **Create role**.
8. Open the AWS Service Catalog console at https://console.aws.amazon.com/servicecatalog.
9. Choose the **Engineering Tools** portfolio.
10. On the **Portfolio details** page, choose the **Constraints** tab, and then choose **Create constraint**.
11. For **Product**, choose **Linux Desktop**, and for **Constraint type**, choose **Launch**.
12. Choose **Select IAM role**. Next choose **linuxDesktopLaunchRole**, and then choose **Create**.

Step 7: Grant end users access to the portfolio

Now that you have created a portfolio and added a product, you are ready to grant access to end users.

Prerequisites

If you haven't created an IAM group for the endusers, see Grant permissions to AWS Service Catalog end users.

To provide access to the portfolio

1. On the portfolio details page, choose the **Access** tab.
2. Choose **Grant access**.
3. On the **Groups** tab, select the checkbox for the IAM group for the end users.
4. Choose **Add Access**.

Step 8: Test the end user experience

To verify the end user can successfully access the end user console view and launch your product, sign in to AWS as the end user and perform those tasks.

To verify that the end user can access the end user console

1. Follow the instructions to Sign in as an IAM user in the *IAM User guide*.
2. In the menu bar, choose the AWS Region in which you created the `Engineering Tools` portfolio. For this tutorial, choose **us-east-1 region**.
3. Open the AWS Service Catalog console at https://console.aws.amazon.com/servicecatalog/ to see:

   - **Products** – The products that the user can use.
   - **Provisioned products** – The provisioned products that the user has launched.

To verify the end user can launch the Linux Desktop product

Note that for this tutorial, choose **us-east-1 region**.

1. In the **Products** section of the console, choose **Linux Desktop**.
2. Choose **Launch product** to start the wizard that configures your product.
3. On the **Launch: Linux Desktop** page, enter **`Linux-Desktop`** for the provisioned product name.
4. On the **Parameters** page, enter the following and choose **Next**:

   - **Server size** – Choose **`t2.micro`**.

- **Key pair** – Select the key pair that you created in Step 2: Create a key pair.
- **CIDR range** – Enter a valid CIDR range for the IP address to connect to the instance. You can use the default value (0.0.0.0/0) to allow access from any IP address, then your IP address, followed by **/32** to restrict access to your IP address only, or something in between.

5. Choose **Launch product** to launch the stack. The console displays the stack details page for the Linux-Desktop stack. The initial status of the product is **Under change**. It takes several minutes for AWS Service Catalog to launch the product. To see the current status, refresh your browser. After the product launches, the status is A**vailable**.

Getting started with a Terraform product

AWS Service Catalog enables quick, self-service provisioning with governance for your HashiCorp Terraform configurations within AWS. You can use AWS Service Catalog as a single tool to organize, govern, and distribute your Terraform configurations at scale within AWS. AWS Service Catalog supports Terraform across several key features, including cataloging of standardized and pre-approved Terraform templates, access control, versioning, tagging, and sharing to other AWS accounts. In AWS Service Catalog, your end users see a simple list of products and versions they have access to, and can then deploy those products in a single action.

> **Note**
>
> To continue support of HashiCorp technologies, as a result of the recent licensing changes to Terraform, AWS Service Catalog changed any previous references of *Terraform Open Source* to *External*. The External product type includes support for the Terraform Community Edition, previously known as Terraform Open Source. For more information and instructions about migrating your existing Terraform Open Source products and provisioned products to the External product type, review Updating existing Terraform Open Source products and provisioned products to the External product type.

The steps in the following tutorial will help you get started with a Terraform product in AWS Service Catalog.

As the catalog administrator, you work in a central administrator account (hub account). Both Terrafrm Community Edition and Terraform Cloud products require a Terraform provisioning engine, which you can learn more about in Provisioning engine for Terraform Community Edition (External product type) and Provisioning engine for Terraform Cloud .

During the tutorial, you perform the following tasks in the administrator account:

- Create a Terraform product using either the *Terraform Cloud* or *External* product type. Service Catalog uses the External product type to support Terraform Community Edition products.
- Associate the product with a portfolio
- Create a launch constraint to allow your end users to provision the product
- Tag the product
- Share the portfolio and the Terraform product with the end user account (spoke account)

In the tutorial, you share a portfolio using the organization sharing option from the admin hub account, which is also the management account of the Organization. For more information on organization sharing, see Sharing a Portfolio.

The AWS resource contained in the Terraform product you create in the tutorial is a simple Amazon S3 bucket.

> **Note**
>
> Before you begin, make sure that you complete the action items in Setting Up AWS Service Catalog.

Topics

- Updating existing Terraform Open Source products and provisioned products to the External product type
- Prerequisite: Configure your Terraform provisioning engine
- Step 1: Terraform configuration file download
- Step 2: Create a Terraform product
- Step 3: Create a AWS Service Catalog portfolio
- Step 4: Add product to portfolio
- Step 5: Create launch roles
- Step 6: Add a Launch constraint to your Terraform product
- Step 7: Grant end user access
- Step 8: Share portfolio with end user
- Step 9: Test the end user experience

- Step 10: Monitoring Terraform provisioning operations

Updating existing Terraform Open Source products and provisioned products to the External product type

To continue support of HashiCorp technologies, as a result of the recent licensing changes to Terraform, AWS Service Catalog changed any previous references of *Terraform Open Source* to *External*. The External product type includes support for Terraform Community Edition, previously known as Terraform Open Source. AWS Service Catalog no longer supports Terraform Open Source as a valid product type for any *new* products or provisioned products. You can only update or terminate existing Terraform Open Source resources, including product versions and provisioned products.

If you have not already done so, you must transition all existing Terraform Open Source products and provisioned products to External products, by following the instructions in this section.

1. Update your existing Terraform Reference Engine for AWS Service Catalog to include support for both **External** and **Terraform Open Source** product types. For instructions about updating your Terraform Reference Engine, review our GitHub Repository.
2. Recreate any existing Terraform Open Source products using the new External product type.
3. Delete any existing products that use the Terraform Open Source product type.
4. Reprovision remaining resources to use the new External product type.
5. Terminate any existing provisioned products that use the Terraform Open Source product type.

After transitioning your existing products, use the External product type for any new products that use a tar.gz configuration file.

AWS Service Catalog will support customers through this change as needed. If these changes require extensive effort for your account, or impact critical product workloads, contact your account representitive to request assistance.

Prerequisite: Configure your Terraform provisioning engine

As a prerequisite to creating Terraform products in AWS Service Catalog, you must install and configure a provisioning engine in your Service Catalog administrator account (hub account). The provisioning engine is required for both Terraform Community Edition products (using the External product type) and Terraform Cloud products (using the Terraform Cloud product type).

> **Note**
>
> Engine configuration is a one-time setup that takes approximately 30 minutes.

Provisioning engine for Terraform Community Edition (External product type)

AWS Service Catalog uses the *External* product type to support Terraform Community Edition products. The *External* product type also supports other provisioning tools, including Pulumi, Ansible, Chef, and more based on the configuration of the provisioning engine.

For AWS Service Catalog products that use the External product type with HashiCorp's Terraform Community Edition, you must install and configure a Terraform provisioning engine in your AWS Service Catalog administrator account (hub account). AWS manages this engine and its resources.

AWS Service Catalog provides a GitHub repository with instructions on installing and configuring the AWS-provided Terraform provisioning engine. The repo includes the following information:

- Required installation tools
- Building the code
- Deploying to an AWS account
- Additional information about provisioning workflows, quality assurance, and limitations

Provisioning engine for Terraform Cloud

For AWS Service Catalog products that use the Terraform Cloud product type with HashiCorp's Terraform Cloud, you must install and configure a Terraform provisioning engine in your AWS Service Catalog administrator account (hub account). HashiCorp manages this engine in a remote environment.

HashiCorp provides a GitHub repository with instructions on configuring the Terraform Cloud engine for AWS Service Catalog. The repo includes the following information:

- Required installation tools
- Building the code
- Deploying to an AWS account
- Additional information about provisioning workflows, quality assurance, and limitations

Step 1: Terraform configuration file download

You can use a Terraform configuration file to create and provision HashiCorp Terraform products. These configurations are plain text files and describe the resources that you want to provision. You can use the text editor of your choice to create, update, and save configurations. For product creation, you must upload Terraform configurations as a **tar.gz file**. In this tutorial, AWS Service Catalog provides a simple configuration file so you can get started. The configuration creates an Amazon S3 bucket.

Configuration file download

AWS Service Catalog provides a sample `simple-s3-bucket.tar.gz` configuration file for you to use in this tutorial.

Configuration file overview

The text of the sample configuration file follows:

```
variable "bucket_name" {
  type = string
}
provider "aws" {
}
resource "aws_s3_bucket" "bucket" {
  bucket = var.bucket_name
}
output regional_domain_name {
  value = aws_s3_bucket.bucket.bucket_regional_domain_name
}
```

Configuration Resources

The configuration file declares the resources to be created when AWS Service Catalog provisions the product. It consists of the following sections:

- **Variable** (optional) – The value definitions that an administrator user (hub account administrator) can assign to customize the configuration. Variables provide a consistent interface to change how a given configuration behaves. The label after the variable keyword is a name for the variable, which must be unique among all variables in the same module. This name is used

to assign an outside value to the variable, and to reference the variable's value from within the module.

- **Provider** (optional) – The cloud service provider for resource provisioning, which is `AWS`. AWS Service Catalog only supports `AWS` as the provider. As a result, the Terraform provisioning engine overrides any other listed provider to `AWS`.
- **Resource** (required) – The AWS infrastructure resource for provisioning. For this tutorial, the Terraform configuration file specifies Amazon S3.
- **Output** (optional) – The returned information or value, similar to returned values in a programming language. You can use outputs data to configure infrastructure workflow with automation tools.

Step 2: Create a Terraform product

After installing the Terraform provisioning engine, you are ready to create a HashiCorp Terraform product in AWS Service Catalog. In this tutorial, you create a Terraform product containing a simple Amazon S3 bucket.

To create a Terraform product

1. Open the AWS Service Catalog console at https://console.aws.amazon.com/servicecatalog/ and sign in as an admin user.
2. Navigate to the **Administration** section, and then choose **Product list**.
3. Choose **Create product**.
4. On the **Create product** page in the Product details section, choose the **External** or **Terraform Cloud** product type. Service Catalog uses the *External* product type to support Terraform Community Edition products.
5. Enter the following product details:
   - **Product name** – **`Simple S3 bucket`**
   - **Product description** – Terraform product containing an Amazon S3 bucket.
   - **Owner** – **`IT`**
   - **Distributor** – *(blank)*
6. On the **Version details** pane, choose **upload a template file** and then choose **Choose file**. Select the file you downloaded in Step 1: Terraform configuration file download.
7. Enter the following:

- **Version name** – **`v1.0`**
- **Version description** – **`Base Version`**

8. In the **Support details** section, enter the following and then choose **Create product**.

- **Email contact** – **`ITSupport@example.com`**
- **Support link** – **`https://wiki.example.com/IT/support`**
- **Support description** – **`Contact the IT department for issues deploying or connecting to this product.`**

9. Choose **Create product**.

After successfully creating the product, AWS Service Catalog displays a confirmation banner on the product page.

Step 3: Create a AWS Service Catalog portfolio

You can create a portfolio in your AWS Service Catalog administrator account (hub account) for easy product organization and distribution to end user accounts (spoke accounts).

To create a portfolio

1. Open the AWS Service Catalog console at https://console.aws.amazon.com/servicecatalog/ and sign in as an administrator.
2. In the left navigation panel, choose **Portfolios**, and then choose **Create portfolio**.
3. Enter the following values:

- **Portfolio name** – **`S3 bucket`**
- **Portfolio description** – **`Sample portfolio for Terraform configurations.`**
- **Owner** – **`IT (it@example.com)`**

4. Choose **Create**.

Step 4: Add product to portfolio

After creating a portfolio, you can add the HashiCorp Terraform product you created in Step 2.

To add a product to a portfolio

1. Navigate to the **Products list** page.
2. Select the Simple S3 bucket Terraform product you created in Step 2, and then choose **Actions**. From the drop down menu, choose **Add product to portfolio**. AWS Service Catalog displays the **Add Simple S3 bucket to portfolio** pane.
3. Select the S3 bucket portfolio, and then turn off **Create launch constraint**. You will create the launch constraint later in the tutorial.
4. Choose **Add product to portfolio**.

After successfully adding the product to the portfolio, AWS Service Catalog displays a confirmation banner on the Product list page.

Step 5: Create launch roles

In this step, you will create an IAM role (launch role) specifying the permissions that the Terraform provisioning engine and AWS Service Catalog can assume when an end user launches a HashiCorp Terraform product.

The IAM role (launch role) that you later assign to your simple Amazon S3 bucket Terraform product as a launch constraint must have the following permissions:

- Access to the underlying AWS resources for your Terraform product. In this tutorial, this includes access to the `s3:CreateBucket*`, `s3:DeleteBucket*`, `s3:Get*`, `s3:List*`, and `s3:PutBucketTagging` Amazon S3 operations.
- Read access to the Amazon S3 template in a AWS Service Catalog-owned Amazon S3 bucket
- Access to the `CreateGroup`, `ListGroupResources`, `DeleteGroup`, and `Tag` resource group operations. These operations enable AWS Service Catalog to manage resource groups and tags

To create a launch role in the AWS Service Catalog administrator account

1. While logged in to the AWS Service Catalog administrator account, follow the instructions to Create new policies on the JSON tab in the *IAM User guide*.
2. Create a **policy** for your simple Amazon S3 bucket Terraform product. This policy must be created before you create the launch role, and consists of the following permissions:

- `s3`— Allows AWS Service Catalog full permissions to list, read, write, provision, and tag the Amazon S3 product.
- `s3`— Allows access to Amazon S3 buckets owned by AWS Service Catalog. To deploy the product, AWS Service Catalog requires access to provisioning artifacts.
- `resourcegroups`— Allows AWS Service Catalog to create, list, delete, and tag AWS Resource Groups.
- `tag`— Allows AWS Service Catalog tagging permissions.

> **Note**
>
> Depending on the underlying resources that you want to deploy, you may need to modify the example JSON policy.

Paste the following JSON policy document:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/servicecatalog:provisioning": "true"
                }
            }
        },
        {
            "Action": [
                "s3:CreateBucket*",
                "s3:DeleteBucket*",
                "s3:Get*",
                "s3:List*",
                "s3:PutBucketTagging"
            ],
            "Resource": "arn:aws:s3:::*",
```

```
            "Effect": "Allow"
        },
        {
            "Action": [
                "resource-groups:CreateGroup",
                "resource-groups:ListGroupResources",
                "resource-groups:DeleteGroup",
                "resource-groups:Tag"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "tag:GetResources",
                "tag:GetTagKeys",
                "tag:GetTagValues",
                "tag:TagResources",
                "tag:UntagResources"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

3. a. Choose **Next**, **Tags**.

   b. Choose **Next, Review**.

   c. In the **Review policy** page, for the **Name**, enter **`S3ResourceCreationAndArtifactAccessPolicy`**.

   d. Choose **Create policy**.

4. In the navigation pane, choose **Roles**, and then choose **Create role**.

5. For **Select trusted entity**, choose **Custom trust policy** and then enter the following JSON policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GivePermissionsToServiceCatalog",
            "Effect": "Allow",
```

```
            "Principal": {
                "Service": "servicecatalog.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        },
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::account_id:root"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringLike": {
                    "aws:PrincipalArn": [
                        "arn:aws:iam::accounti_id:role/TerraformEngine/
TerraformExecutionRole*",
                        "arn:aws:iam::accounti_id:role/TerraformEngine/
ServiceCatalogExternalParameterParserRole*",
                        "arn:aws:iam::accounti_id:role/TerraformEngine/
ServiceCatalogTerraformOSParameterParserRole*"
                    ]
                }
            }
        }
    ]
}
```

6. Choose **Next**.
7. In the **Policies** list, select the `S3ResourceCreationAndArtifactAccessPolicy` you just created.
8. Choose **Next**.
9. For **Role name**, enter **`SCLaunch-S3product`**.

> **Important**
>
> Launch role names **must** begin with "SCLaunch" followed by the desired role name.

10. Choose **Create role**.

> **Important**
>
> After creating the launch role in your AWS Service Catalog administrator account, you must also create an identical launch role in the AWS Service Catalog end user account. The role in the end user account must have the same name and include the same policy as the role in the administrator account.

To create a launch role in the AWS Service Catalog end user account

1. Log in as the administrator to the end user account, and then follow the instructions to Create new policies on the JSON tab in the *IAM User guide*.
2. Repeat steps 2-10 from *To create a launch role in the AWS Service Catalog administrator account* above.

> **Note**
>
> When creating a launch role in the AWS Service Catalog end user account, ensure you use the same administrator **`AccountId`** in the custom trust policy.

Now that you have created a launch role in both the administrator and end user accounts, you can add a launch constraint to the product.

Step 6: Add a Launch constraint to your Terraform product

> **Important**
>
> You must create a launch constraint for HashiCorp Terraform products. Without a launch constraint, end users cannot provision the product.

After creating a launch role in your administrator account, you are ready to associate the launch role to a launch constraint on your External or Terraform Cloud product.

This launch constraint enables the end user to launch the product and, after launch, manage it as a provisioned product. For more information, see AWS Service Catalog Launch Constraints.

Using a launch constraint allows you follow the IAM best practice of keeping end user IAM permissions to a minimum. For more information, see Grant least privilege in the *IAM User Guide*.

To assign a launch constraint to the product

1. Open the AWS Service Catalog console at https://console.aws.amazon.com/servicecatalog.
2. In the left navigation console, choose **Portfolio**.
3. Choose the **S3 bucket** portfolio.
4. On the **Portfolio details** page, choose the **Constraints** tab, and then choose **Create constraint**.
5. For **Product**, choose **Simple S3 bucket**. AWS Service Catalog automatically selects the **Launch** constraint type.
6. Choose **Enter role name**, and then choose **SCLaunch-S3product**.
7. Choose **Create**.

> **Note**
>
> The given role name must exist in the account that created the launch constraint and the account of the user who launches a product with this launch constraint.

Step 7: Grant end user access

After applying the launch constraint to your HashiCorp Terraform product, you are ready to grant access to end users in the spoke account.

In this tutorial, you grant access to end users using Principal Name sharing. Principal Names are names for groups, roles, and users that administrators can specify in a portfolio, and then share with the portfolio. When you share the portfolio, AWS Service Catalog verifies if those Principal Names already exist. If they do exist, AWS Service Catalog automatically associates the matching IAM principals with the shared portfolio to grant access to end users. Review Sharing a Portfolio for more information.

Prerequisites

If you haven't created an IAM group for the end users, see Grant permissions to AWS Service Catalog end users.

To provide access to the portfolio

1. Navigate to the **Portfolio** page and choose the **S3 bucket** portfolio.
2. Choose the **Access** tab, and then choose **Grant access**.
3. In the **Access type** pane, choose **Principal name**.
4. In the **Principal name** pane, select the **Principal name** type, and then enter the principal **Name** of the desired end user in the spoke account.
5. Choose **Grant access**.

Step 8: Share portfolio with end user

The AWS Service Catalog administrator can distribute portfolios with end user accounts using either account-to-account sharing or AWS Organizations sharing. In this tutorial, you are sharing your portfolio with the organization from the administrator account (hub account), which is also the management account of the Organization.

To share the portfolio from the admin hub account

1. Open the AWS Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. On the **Portfolios** page, select the S3 bucket portfolio. In the **Actions** menu, choose **Share**.
3. Choose **AWS Organizations**, and then filter into your organizational structure.
4. In the **AWS Organization** pane, choose the end user account (spoke account).

   You can also select a **Root node** to share the portfolio with the entire organization, a **parent Organizational Unit (OU)**, or a **child OU** within your organization based on your organization structure. For more information, review Sharing a Portfolio.
5. In the **Share settings** pane, choose **Principal sharing**.
6. Choose **Share**.

After successfully sharing the portfolio with end users, the next step is to verify the end user experience and provision the Terraform product.

Step 9: Test the end user experience

To verify end users can successfully access the end user console view and launch your **`Simple S3 bucket`** product, sign in to AWS as the end user and perform the tasks below.

To verify that the end user can access the end user console

- Open the AWS Service Catalog console at https://console.aws.amazon.com/servicecatalog/ to see:
  - **Products** – The products that the user can use.
  - **Provisioned products** – The provisioned products that the user has launched.

To verify the end user can launch the Terraform product

1. In the **Products** section of the console, choose **Simple S3 bucket**.
2. Choose **Launch product** to start the wizard that configures your product.
3. On the **Launch Simple S3 bucket** page, enter **`Amazon S3 product`** for the provisioned product name.
4. On the **Parameters** page, enter the following and choose **Next**:
   - **bucket_name** – Provide a unique name for the Amazon S3 bucket. For example, **`terraform-s3-product`**.
5. Choose **Launch product**. The console displays the stack details page for the Amazon S3 product launch. The initial status of the product is **Under change**. It takes several minutes for AWS Service Catalog to launch the product. To see the current status, refresh your browser. After a successful product launch, the status is **Available**.

   AWS Service Catalog creates a new Amazon S3 bucket named **`terraform-s3-product`**.

Step 10: Monitoring Terraform provisioning operations

If you want to monitor provisioning operations, you can review Amazon CloudWatch logs and AWS Step Functions for any provisioning workflow.

There are two state machines for the provisioning workflow:

- `ManageProvisionedProductStateMachine` — AWS Service Catalog invokes this state machine when provisioning a new Terraform product and when updating an existing Terraform provisioned product.
- `TerminateProvisionedProductStateMachine` — AWS Service Catalog invokes this state machine when terminating an existing Terraform provisioned product.

To execute the monitoring state machine

1. Open the AWS management console and log in as an administrator in the admin hub account where the Terraform provisioning engine is installed.
2. Open **AWS Step Functions**.
3. In the left navigation panel, choose **State machines**.
4. Choose **ManageProvisionedProductStateMachine**.
5. In the **Executions** list, enter the provisioned product ID to locate your execution.

   > **Note**
   >
   > AWS Service Catalog creates the provisioned product ID when you provision the product. The provisioned product ID is formatted as follows: **`pp-1111pwtn[ID number]`**.

6. Choose the **execution ID**.

On the resulting **Execution details** page, you can view all of the steps in the provisioning workflow. You can also review any failed steps to identify the cause of the failure.

Security in AWS Service Catalog

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. 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 that 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 Service Catalog, see AWS Services in Scope by Compliance Program
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS Service Catalog. The following topics show you how to configure AWS Service Catalog to meet your security and compliance objectives. You also will be introduced to other AWS services that help you to monitor and secure your AWS Service Catalog resources.

Topics

- Data Protection in AWS Service Catalog
- Identity and Access Management in AWS Service Catalog
- Logging and Monitoring in AWS Service Catalog
- Compliance Validation for AWS Service Catalog
- Resilience in AWS Service Catalog
- Infrastructure Security in AWS Service Catalog
- Security Best Practices for AWS Service Catalog

Data Protection in AWS Service Catalog

The AWS shared responsibility model applies to data protection in AWS Service Catalog. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AWS Service Catalog or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Protecting Data with Encryption

Encryption at rest

AWS Service Catalog uses Amazon S3 buckets and Amazon DynamoDB databases that are encrypted at rest using Amazon-managed keys. To learn more, refer to information about encryption at rest provided by Amazon S3 and Amazon DynamoDB.

Encryption in transit

AWS Service Catalog uses Transport Layer Security (TLS) and client-side encryption of information in transit between the caller and AWS.

You can privately access AWS Service Catalog APIs from your Amazon Virtual Private Cloud (Amazon VPC) by creating VPC endpoints. With VPC endpoints, the routing between the VPC and AWS Service Catalog is handled by the AWS network without the need for an internet gateway, NAT gateway, or VPN connection.

The latest generation of VPC endpoints used by AWS Service Catalog is powered by AWS PrivateLink, an AWS technology enabling the private connectivity between AWS services using Elastic Network Interfaces with private IPs in your VPCs.

Identity and Access Management in AWS Service Catalog

Access to AWS Service Catalog requires credentials. Those credentials must have permission to access AWS resources, such as a AWS Service Catalog portfolio or product. AWS Service Catalog integrates with AWS Identity and Access Management (IAM) to enable you to grant AWS Service Catalog administrators the permissions they need to create and manage products, and to grant AWS Service Catalog end users the permissions they need to launch products and manage provisioned products. These policies are either created and managed by AWS or individually by administrators and end users. To control access, you attach these policies to users, groups, and roles that you use with AWS Service Catalog.

Audience

The permissions you have *with* AWS Identity and Access Management (IAM) can depend on the role you play in AWS Service Catalog.

The permissions you have *through* AWS Identity and Access Management (IAM) can also depend on the role you play in AWS Service Catalog.

Administrator - As a AWS Service Catalog administrator, you need full access to the administrator console and IAM permissions that allow you to perform tasks such as creating and managing portfolios and products, managing constraints, and granting access to end users.

End user - Before your end users can use your products, you need to grant them permissions that give them access to the AWS Service Catalog end user console. They can also have permissions to launch products and manage provisioned products.

IAM administrator - If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS Service Catalog. To view example AWS Service Catalog identity-based policies that you can use in IAM, see the section called “AWS managed policies”.

Identity-based policy examples for AWS Service Catalog

Topics

- Console access for end users
- Product access for end users
- Example policies for managing provisioned products

Console access for end users

The **`AWSServiceCatalogEndUserFullAccess`** and **`AWSServiceCatalogEndUserReadOnlyAccess`** policies grant access to the AWS Service Catalog end user console view. When a user who has either of these policies chooses AWS Service Catalog in the AWS Management Console, the end user console view displays the products they have permission to launch.

Before end users can successfully launch a product from AWS Service Catalog to which you give access, you must provide them additional IAM permissions to allow them to use each of the underlying AWS resources in a product's AWS CloudFormation template. For example, if a product template includes Amazon Relational Database Service (Amazon RDS), you must grant the users Amazon RDS permissions to launch the product.

To learn about how to enable end users to launch products while enforcing least-access permissions to AWS resources, see the section called “Using Constraints”.

If you apply the **`AWSServiceCatalogEndUserReadOnlyAccess`** policy, your users have access to the end user console, but they won't have the permissions that they need to launch products and manage provisioned products. You can grant these permissions directly to an end user

using IAM, but if you want to limit the access that end users have to AWS resources, you should attach the policy to a launch role. You then use AWS Service Catalog to apply the launch role to a launch constraint for the product. For more information about applying a launch role, launch role limitations, and a sample launch role, see AWS Service Catalog Launch Constraints.

> **Note**
>
> If you grant users IAM permissions for AWS Service Catalog administrators, the administrator console view displays instead. Don't grant end users these permissions unless you want them to have access to the administrator console view.

Product access for end users

Before end users can use a product to which you give access, you must provide them additional IAM permissions to allow them to use each of the underlying AWS resources in a product's AWS CloudFormation template. For example, if a product template includes Amazon Relational Database Service (Amazon RDS), you must grant the users Amazon RDS permissions to launch the product.

If you apply the **`AWSServiceCatalogEndUserReadOnlyAccess`** policy, your users have access to the end user console view, but they won't have the permissions that they need to launch products and manage provisioned products. You can grant these permissions directly to an end user in IAM, but if you want to limit the access that end users have to AWS resources, you should attach the policy to a launch role. You then use AWS Service Catalog to apply the launch role to a launch constraint for the product. For more information about applying a launch role, launch role limitations, and a sample launch role, see AWS Service Catalog Launch Constraints.

Example policies for managing provisioned products

You can create custom policies to help meet the security requirements of your organization. The following examples describe how to customize the access level for each action with support for user, role, and account levels. You can grant users access to view, update, terminate, and manage provisioned products created only by that user or created by others also under their role or the account to which they are logged in. This access is hierarchical — granting account level access also grants role level access and user level access, while adding role level access also grants user level access but not account level access. You can specify these in the policy JSON using a `Condition` block as `accountLevel`, `roleLevel`, or `userLevel`.

These examples also apply to access levels for AWS Service Catalog API write operations: `UpdateProvisionedProduct` and `TerminateProvisionedProduct`, and read operations: `DescribeRecord`, `ScanProvisionedProducts`, and `ListRecordHistory`. The `ScanProvisionedProducts` and `ListRecordHistory` API operations use `AccessLevelFilterKey` as input, and that key's values correspond to the `Condition` block levels discussed here (`accountLevel` is equivalent to an `AccessLevelFilterKey` value of "Account", `roleLevel` to "Role", and `userLevel` to "User"). For more information, see the Service Catalog Developer Guide.

Examples

- Full admin access to provisioned products
- End-user access to provisioned products
- Partial admin access to provisioned products

Full admin access to provisioned products

The following policy allows full read and write access to provisioned products and records within the catalog at the account level.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "servicecatalog:*"
         ],
         "Resource":"*",
         "Condition": {
            "StringEquals": {
               "servicecatalog:accountLevel": "self"
            }
         }
      }
   ]
}
```

This policy is functionally equivalent to the following policy:

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

Not specifying a `Condition` block in any policy for AWS Service Catalog is treated as the same as specifying `"servicecatalog:accountLevel"` access. Note that `accountLevel` access includes `roleLevel` and `userLevel` access.

End-user access to provisioned products

The following policy restricts access to read and write operations to only the provisioned products or associated records that the current user created.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "servicecatalog:DescribeProduct",
                "servicecatalog:DescribeProductView",
                "servicecatalog:DescribeProvisioningParameters",
                "servicecatalog:DescribeRecord",
                "servicecatalog:ListLaunchPaths",
                "servicecatalog:ListRecordHistory",
                "servicecatalog:ProvisionProduct",
                "servicecatalog:ScanProvisionedProducts",
                "servicecatalog:SearchProducts",
                "servicecatalog:TerminateProvisionedProduct",
                "servicecatalog:UpdateProvisionedProduct"
            ],
            "Resource": "*",
            "Condition": {
```

```
                "StringEquals": {
                    "servicecatalog:userLevel": "self"
                }
            }
        }
    ]
}
```

Partial admin access to provisioned products

The two policies below, if both applied to the same user, allow what might be called a type of "partial admin access" by providing full read-only access and limited write access. This means the user can see any provisioned product or associated record within the catalog's account but cannot perform any actions on any provisioned products or records that aren't owned by that user.

The first policy allows the user access to write operations on the provisioned products that the current user created, but no provisioned products created by others. The second policy adds full access to read operations on provisioned products created by all (user, role, or account).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "servicecatalog:DescribeProduct",
                "servicecatalog:DescribeProductView",
                "servicecatalog:DescribeProvisioningParameters",
                "servicecatalog:ListLaunchPaths",
                "servicecatalog:ProvisionProduct",
                "servicecatalog:SearchProducts",
                "servicecatalog:TerminateProvisionedProduct",
                "servicecatalog:UpdateProvisionedProduct"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "servicecatalog:userLevel": "self"
                }
            }
        }
    ]
```

```
}
```

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "servicecatalog:DescribeRecord",
                "servicecatalog:ListRecordHistory",
                "servicecatalog:ScanProvisionedProducts"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "servicecatalog:accountLevel": "self"
                }
            }
        }
    ]
}
```

AWS managed policies for AWS Service Catalog AppRegistry

AWS managed policy: AWSServiceCatalogAdminFullAccess

You can attach AWSServiceCatalogAdminFullAccess to your IAM entities. AppRegistry also attaches this policy to a service role that allows AppRegistry to perform actions on your behalf.

This policy grants *administrative* permissions that allow full access to the administrator console view and grants permission to create and manage products and portfolios.

Permissions details

This policy includes the following permissions.

- servicecatalog – Allows principals full permissions to the administrator console view and the ability to create and manage portfolios and products, manage constraints, grant access to end users, and perform other administrative tasks within AWS Service Catalog.
- cloudformation– Allows AWS Service Catalog full permissions to list, read, write, and tag AWS CloudFormation stacks.

- `config`– Allows AWS Service Catalog limited permissions to portfolios, products, and provisioned products via AWS Config.
- `iam`– Allows principals full permissions to view and create service users, gropus, or roles that are required for creating and managing products and portfolios.
- `ssm` – Allows AWS Service Catalog to use AWS Systems Manager to list and read Systems Manager documents in the current AWS account and AWS Region.

View the policy: AWSServiceCatalogAdminFullAccess.

AWS managed policy: AWSServiceCatalogAdminReadOnlyAccess

You can attach `AWSServiceCatalogAdminReadOnlyAccess` to your IAM entities. AppRegistry also attaches this policy to a service role that allows AppRegistry to perform actions on your behalf.

This policy grants *`read-only`* permissions that allow full access to the administrator console view. This policy does not grant access to create or manage products and portfolios.

Permissions details

This policy includes the following permissions.

- `servicecatalog` – Allows principals read-only permissions to the administrator console view.
- `cloudformation`– Allows AWS Service Catalog limited permissions to list and read AWS CloudFormation stacks.
- `config`– Allows AWS Service Catalog limited permissions to portfolios, products, and provisioned products via AWS Config.
- `iam`– Allows principals limited permissions to view service users, groups, or roles that are required for creating and managing products and portfolios.
- `ssm` – Allows AWS Service Catalog to use AWS Systems Manager to list and read Systems Manager documents in the current AWS account and AWS Region.

View the policy: AWSServiceCatalogAdminReadOnlyAccess.

AWS managed policy: AWSServiceCatalogEndUserFullAccess

You can attach `AWSServiceCatalogEndUserFullAccess` to your IAM entities. AppRegistry also attaches this policy to a service role that allows AppRegistry to perform actions on your behalf.

This policy grants *contributor* permissions that allow full access to the end user console view and grants permission to launch products and manage provisioned products.

Permissions details

This policy includes the following permissions.

- servicecatalog – Allows principals full permissions to the end user console view and the ability to launch products and manage provisioned products.
- cloudformation– Allows AWS Service Catalog full permissions to list, read, write, and tag AWS CloudFormation stacks.
- config– Allows AWS Service Catalog limited permissions to list and read details about portfolios, products, and provisioned products via AWS Config.
- ssm – Allows AWS Service Catalog to use AWS Systems Manager to read Systems Manager documents in the current AWS account and AWS Region.

View the policy: AWSServiceCatalogEndUserFullAccess.

AWS managed policy: AWSServiceCatalogEndUserReadOnlyAccess

You can attach AWSServiceCatalogEndUserReadOnlyAccess to your IAM entities. AppRegistry also attaches this policy to a service role that allows AppRegistry to perform actions on your behalf.

This policy grants *read-only* permissions that allow read-only access to the end user console view. This policy does not grant permission to launch products or manage provisioned products.

Permissions details

This policy includes the following permissions.

- servicecatalog – Allows principals read-only permissions to the end user console view.
- cloudformation– Allows AWS Service Catalog limited permissions to list and read AWS CloudFormation stacks.
- config– Allows AWS Service Catalog limited permissions to list and read details about portfolios, products, and provisioned products via AWS Config.
- ssm – Allows AWS Service Catalog to use AWS Systems Manager to read Systems Manager documents in the current AWS account and AWS Region.

View the policy: AWSServiceCatalogEndUserReadOnlyAccess.

AWS managed policy: AWSServiceCatalogSyncServiceRolePolicy

AWS Service Catalog attaches this policy to the AWSServiceRoleForServiceCatalogSync service-linked role (SLR), allowing AWS Service Catalog to sync templates in an external repository to AWS Service Catalog products.

This policy grants permissions that allows limited access to AWS Service Catalog actions (for example, API calls), and to other AWS service actions that AWS Service Catalog depends on.

Permissions details

This policy includes the following permissions.

- servicecatalog – Allows the AWS Service Catalog artifact sync role limited access to AWS Service Catalog public APIs.
- codeconnections– Allows the AWS Service Catalog artifact sync role limited access to CodeConnections public APIs.
- cloudformation– Allows the AWS Service Catalog artifact sync role limited access to AWS CloudFormation public APIs.

View the policy: AWSServiceCatalogSyncServiceRolePolicy.

Service-linked role details

AWS Service Catalog uses the permission details above for the AWSServiceRoleForServiceCatalogSync service-linked role that is created when a user creates or updates a AWS Service Catalog product that uses CodeConnections. You can modify this policy using the AWS CLI, AWS API, or through the AWS Service Catalog console. For more information on how to create, edit, and delete service-linked roles, refer to Using service-linked roles (SLRs) for AWS Service Catalog.

The permissions included in the AWSServiceRoleForServiceCatalogSync service-linked role allow AWS Service Catalog to perform the following actions on behalf of the customer.

- servicecatalog:ListProvisioningArtifacts — Allows the AWS Service Catalog artifact sync role to list the provisioning artifacts for a given AWS Service Catalog product that is synced to a template file in a repository.

- `servicecatalog:DescribeProductAsAdmin` — Allows the AWS Service Catalog artifact sync role to use the `DescribeProductAsAdmin` API to get details for a AWS Service Catalog product and its associated provisioned artifacts that are synced to a template file in a repository. The artifact sync role uses the output from this call to verify the product's service quota limit for provisioning artifacts.
- `servicecatalog:DeleteProvisioningArtifact` — Allows the AWS Service Catalog artifact sync role to delete a provisioned artifact.
- `servicecatalog:ListServiceActionsForProvisioningArtifact` — Allows the AWS Service Catalog artifact sync role to determine if Service Actions are associated with a provisioning artifact and ensure that the provisioning artifact is not deleted if a Service Action is associated.
- `servicecatalog:DescribeProvisioningArtifact` — Allows the AWS Service Catalog artifact sync role to retrieve details from the `DescribeProvisioningArtifact` API, including the commit ID, which is provided in the `SourceRevisionInfo` output.
- `servicecatalog:CreateProvisioningArtifact` — Allows the AWS Service Catalog artifact sync role to create a new provisioned artifact if a change is detected (for example, a git-push is committed) to the source template file in the external repository.
- `servicecatalog:UpdateProvisioningArtifact` — Allows the AWS Service Catalog artifact sync role to update the provisioned artifact for a connected or synced product.
- `codeconnections:UseConnection` — Allows the AWS Service Catalog artifact sync role to use the existing connection to update and sync a product.
- `cloudformation:ValidateTemplate` — Allows the AWS Service Catalog artifact sync role limited access to AWS CloudFormation to validate the template format for the template that is being used in external repository and verify if AWS CloudFormation can support the template.

AWS managed policy: `AWSServiceCatalogOrgsDataSyncServiceRolePolicy`

AWS Service Catalog attaches this policy to the `AWSServiceRoleForServiceCatalogOrgsDataSync` service-linked role (SLR), allowing AWS Service Catalog to sync with AWS Organizations.

This policy grants permissions that allows limited access to AWS Service Catalog actions (for example, API calls), and to other AWS service actions that AWS Service Catalog depends on.

Permissions details

This policy includes the following permissions.

- `organizations`— Allows the AWS Service Catalog data sync role limited access to AWS Organizations public APIs.

View the policy: AWSServiceCatalogOrgsDataSyncServiceRolePolicy.

Service-linked role details

AWS Service Catalog uses the permission details above for the `AWSServiceRoleForServiceCatalogOrgsDataSync` service-linked role that is created when a user enables AWS Organizations shared portfolio access or creates a portfolio share. You can modify this policy using the AWS CLI, AWS API, or through the AWS Service Catalog console. For more information on how to create, edit, and delete service-linked roles, refer to Using service-linked roles (SLRs) for AWS Service Catalog.

The permissions included in the `AWSServiceRoleForServiceCatalogOrgsDataSync` service-linked role allow AWS Service Catalog to perform the following actions on behalf of the customer.

- `organizations:DescribeAccount` — Allows the AWS Service Catalog Organizations Data Sync role to retrieve AWS Organizations-related information about the specified account.
- `organizations:DescribeOrganization` — Allows the AWS Service Catalog Organizations Data Sync role to retrieve information about the organization that the user's account belongs to.
- `organizations:ListAccounts` — Allows the AWS Service Catalog Organizations Data Sync role to list the accounts in the user's organization.
- `organizations:ListChildren` — Allows the AWS Service Catalog Organizations Data Sync role to list all of the organizational units (UOs) or accounts that are contained in the specified parent OU or root.
- `organizations:ListParents` — Allows the AWS Service Catalog Organizations Data Sync role to list the root or OUs that serve as the immediate parent of the specified child OU or account.
- `organizations:ListAWSServiceAccessForOrganization` — Allows the AWS Service Catalog Organizations Data Sync role to retrieve a list of the AWS services that the user enabled to integrate with their organization.

Deprecated policies

The following managed policies are deprecated:

- **ServiceCatalogAdminFullAccess** — Use **AWSServiceCatalogAdminFullAccess** instead.
- **ServiceCatalogAdminReadOnlyAccess** — Use **AWSServiceCatalogAdminReadOnlyAccess** instead.
- **ServiceCatalogEndUserFullAccess** — Use **AWSServiceCatalogEndUserFullAccess** instead.
- **ServiceCatalogEndUserAccess** — Use **AWSServiceCatalogEndUserReadOnlyAccess** instead.

Use the following procedure to ensure that your administrators and end users are granted permissions using the current policies.

To migrate from the deprecated policies to the current policies, see Adding and removing IAM identity permissions in *AWS Identity and Access Management User Guide*.

AppRegistry updates to AWS managed policies

View details about updates to AWS managed policies for AppRegistry since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the AppRegistry Document history page.

Change	Description	Date
AWSServiceCatalogSyncServiceRolePolicy – Update managed policy	AWS Service Catalog updated the `AWSServiceCatalogSyncServiceRolePolicy` policy to change `codestar-connections` to `codeconnections` .	May 7, 2024
AWSServiceCatalogAdminFullAccess – Update managed policy	AWS Service Catalog updated the `AWSServiceCatalogAdminFullAccess` policy to include permissions required for the AWS Service Catalog administrator to	April 14, 2023

Change	Description	Date
	create the AWSServiceRoleForServiceCatalogOrgsDataSync service-linked role (SLR) in their account.	
AWSServiceCatalogOrgsDataSyncServiceRolePolicy – New managed policy	AWS Service Catalog added the AWSServiceCatalogOrgsDataSyncServiceRolePolicy, which is attached to the AWSServiceRoleForServiceCatalogOrgsDataSync service-linked role (SLR), allowing AWS Service Catalog to sync with AWS Organizations. This policy allows limited access to AWS Service Catalog actions (for example, API calls), and to other AWS service actions that AWS Service Catalog depends on.	April 14, 2023
AWSServiceCatalogAdminFullAccess – Update managed policy	AWS Service Catalog updated the AWSServiceCatalogAdminFullAccess policy to include all permissions for the AWS Service Catalog Administrator and create compatibility with AppRegistry.	January 12, 2023

Change	Description	Date
AWSServiceCatalogSyncServiceRolePolicy – New managed policy	AWS Service Catalog added the `AWSServiceCatalogSyncServiceRolePolicy` policy, which is attached to the `AWSServiceRoleForServiceCatalogSync` service-linked role (SLR). This policy allows AWS Service Catalog to sync templates in an external repository to AWS Service Catalog products.	November 18, 2022
AWSServiceRoleForServiceCatalogSync – New service-linked role	AWS Service Catalog added the `AWSServiceRoleForServiceCatalogSync` service-linked role (SLR). This role is required for AWS Service Catalog to use CodeConnections and to create, update, and describe AWS Service Catalog Provisioning Artifacts for a product.	November 18, 2022

Change	Description	Date
AWSServiceCatalogAdminFullAccess – Updated managed policy	AWS Service Catalog updated the `AWSServiceCatalogAdminFullAccess` policy to include all of the required permissions for a AWS Service Catalog Administrator. The policy identifies the specific actions administrator can take on all AWS Service Catalog resources, such as create, describe, delete, and more. Additionally, the policy was changed to support a recently launched feature, Attribute Based Access Control (ABAC) for AWS Service Catalog. ABAC allows you to use the `AWSServiceCatalogAdminFullAccess` policy as a template to allow or deny actions on AWS Service Catalog resources based on tags. For more information about ABAC, see What is ABAC for AWS in *AWS Identity and Access Management*.	September 30, 2022
AppRegistry started tracking changes	AppRegistry started tracking changes for its AWS managed policies.	September 15, 2022

Using service-linked roles for AWS Service Catalog

AWS Service Catalog uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to AWS Service Catalog. Service-linked roles are predefined by AWS Service Catalog and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up AWS Service Catalog easier because you don't have to manually add the necessary permissions. AWS Service Catalog defines the permissions of its service-linked roles, and unless defined otherwise, only AWS Service Catalog can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your AWS Service Catalog resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for `AWSServiceRoleForServiceCatalogSync`

AWS Service Catalog can use the service-linked role named **`AWSServiceRoleForServiceCatalogSync`** – This service-linked role is required for AWS Service Catalog to use CodeConnections and to create, update, and describe AWS Service Catalog Provisioning Artifacts for a product.

The `AWSServiceRoleForServiceCatalogSync` service-linked role trusts the following services to assume the role:

- `sync.servicecatalog.amazonaws.com`

The role permissions policy named **AWSServiceCatalogSyncServiceRolePolicy** allows AWS Service Catalog to complete the following actions on the specified resources:

- Action: `Connection` on `CodeConnections`
- Action: `Create, Update, and Describe` on `ProvisioningArtifact` for a AWS Service Catalog product

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating the `AWSServiceRoleForServiceCatalogSync` service-linked role

You do not need to manually create the `AWSServiceRoleForServiceCatalogSync` service-linked role. AWS Service Catalog creates the service-linked role for you automatically when you establish CodeConnections in the AWS Management Console, the AWS CLI, or the AWS API.

> **Important**
>
> This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role. Also, if you were using the AWS Service Catalog service before November 18, 2022, when it began supporting service-linked roles, then AWS Service Catalog created the `AWSServiceRoleForServiceCatalogSync` role in your account. To learn more, see A new role appeared in my IAM account.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you establish CodeConnections, AWS Service Catalog creates the service-linked role for you again.

You can also use the IAM console to create a service-linked role with the **synced AWS Service Catalog Products** use case. In the AWS CLI or the AWS API, create a service-linked role with the `sync.servicecatalog.amazonaws.com` service name. For more information, see Creating a service-linked role in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

Service-linked role permissions for `AWSServiceRoleForServiceCatalogOrgsDataSync`

AWS Service Catalog can use the service-linked role named **`AWSServiceRoleForServiceCatalogOrgsDataSync`** – This service-linked role is required for AWS Service Catalog organizations to stay in sync with AWS Organizations.

The `AWSServiceRoleForServiceCatalogOrgsDataSync` service-linked role trusts the following services to assume the role:

- orgsdatasync.servicecatalog.amazonaws.com

The AWSServiceRoleForServiceCatalogOrgsDataSync service-linked role requires that you use the following trust policy in addition to the AWSServiceCatalogOrgsDataSyncServiceRolePolicy managed policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "orgsdatasync.servicecatalog.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

The role permissions policy named **AWSServiceCatalogOrgsDataSyncServiceRolePolicy** allows AWS Service Catalog to complete the following actions on the specified resources:

- Action: DescribeAccount, DescribeOrganization, and ListAWSServiceAccessForOrganization on Organizations accounts
- Action: ListAccounts, ListChildren, and ListParent on Organizations accounts

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating the AWSServiceRoleForServiceCatalogOrgsDataSync service-linked role

You do not need to manually create the AWSServiceRoleForServiceCatalogOrgsDataSyncservice-linked role. AWS Service Catalog considers your action of enabling Sharing with AWS Organizations or Sharing a Portfolio as permission for AWS Service Catalog to create a SLR in the background on your behalf.

AWS Service Catalog creates the service-linked role for you automatically when you request `EnableAWSOrganizationsAccess` or `CreatePortfolioShare` in the AWS Management Console, the AWS CLI, or the AWS API.

> **Important**
>
> This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role. To learn more, see A new role appeared in my IAM account.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you request `EnableAWSOrganizationsAccess` or `CreatePortfolioShare`, AWS Service Catalog creates the service-linked role for you again.

Editing a service-linked role for AWS Service Catalog

AWS Service Catalog does not allow you to edit the `AWSServiceRoleForServiceCatalogSync` or `AWSServiceRoleForServiceCatalogOrgsDataSync` service-linked roles. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for AWS Service Catalog

You can use the IAM console, the AWS CLI, or the AWS API to manually delete the `AWSServiceRoleForServiceCatalogSync` or `AWSServiceRoleForServiceCatalogOrgsDataSync` SLR. To do this, you must first manually remove all resources that are using the service-linked role (for example, any AWS Service Catalog products that are synced to an external repository), and then the service-linked role can be manually deleted.

Supported regions for AWS Service Catalog service-linked roles

AWS Service Catalog supports using service-linked roles in all of the regions where the service is available. For more information, see AWS regions and endpoints.

Region name	Region identity	Support in AWS Service Catalog
US East (N. Virginia)	us-east-1	Yes
US East (Ohio)	us-east-2	Yes
US West (N. California)	us-west-1	Yes
US West (Oregon)	us-west-2	Yes
Africa (Cape Town)	af-south-1	Yes
Asia Pacific (Hong Kong)	ap-east-1	Yes
Asia Pacific (Jakarta)	ap-southeast-3	Yes
Asia Pacific (Mumbai)	ap-south-1	Yes
Asia Pacific (Osaka)	ap-northeast-3	Yes
Asia Pacific (Seoul)	ap-northeast-2	Yes
Asia Pacific (Singapore)	ap-southeast-1	Yes
Asia Pacific (Sydney)	ap-southeast-2	Yes
Asia Pacific (Tokyo)	ap-northeast-1	Yes
Canada (Central)	ca-central-1	Yes
Europe (Frankfurt)	eu-central-1	Yes
Europe (Ireland)	eu-west-1	Yes
Europe (London)	eu-west-2	Yes
Europe (Milan)	eu-south-1	Yes
Europe (Paris)	eu-west-3	Yes
Europe (Stockholm)	eu-north-1	Yes

Region name	Region identity	Support in AWS Service Catalog
Middle East (Bahrain)	me-south-1	Yes
South America (São Paulo)	sa-east-1	Yes
AWS GovCloud (US-East)	us-gov-east-1	No
AWS GovCloud (US-West)	us-gov-west-1	No

Troubleshooting AWS Service Catalog identity and access

Use the following information to help you diagnose and fix common issues you might encounter when working with AWS Service Catalog and IAM.

Topics

- I am not authorized to perform an action in AWS Service Catalog
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my AWS Service Catalog resources

I am not authorized to perform an action in AWS Service Catalog

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your sign-in credentials. The following example error occurs when the mateojackson user tries to use the console to view details about a fictional my-example-widget resource but does not have the fictional `aws:GetWidget` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `my-example-widget` resource using the `aws:GetWidget` action.

I am not authorized to perform `iam:PassRole`

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to AWS Service Catalog.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when a user named marymajor tries to use the console to perform an action in AWS Service Catalog. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the iam:PassRole action.

I want to allow people outside of my AWS account to access my AWS Service Catalog resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS Service Catalog supports these features, see AWS Identity and Access Management in AWS Service Catalog in the *AWS Service Catalog Administrator Guide*.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.

- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Controlling Access

a AWS Service Catalog portfolio gives your administrators a level of access control for your groups of end users. When you add users to a portfolio, they can browse and launch any of the products in the portfolio. For more information, see the section called “Managing Portfolios”.

Constraints

Constraints control which rules are applied to your end users when launching a product from a specific portfolio. You use them to apply limits to products for governance or cost control. For more information about constraints, see the section called “Using Constraints”.

AWS Service Catalog launch constraints give you more control over permissions needed by an end user. When your administrator creates a launch constraint for a product in a portfolio, the launch constraint associates a role ARN that is used when your end users launch the product from that portfolio. Using this pattern, you can control access to AWS resource creation. For more information, see the section called “Launch Constraints”.

Logging and Monitoring in AWS Service Catalog

AWS Service Catalog integrates with AWS CloudTrail, a service that captures all of the AWS Service Catalog API calls and delivers the log files to an Amazon S3 bucket that you specify. For more information, see Logging AWS Service Catalog API Calls with CloudTrail.

You can also use notification constraints to set up Amazon SNS notifications about stack events. For more information, see the section called “Notification Constraints”.

Compliance Validation for AWS Service Catalog

Third-party auditors assess the security and compliance of AWS Service Catalog as part of multiple AWS compliance programs, including the following:

- System and Organization Controls (SOC)
- Payment Card Industry Data Security Standard (PCI DSS)
- Federal Risk and Authorization Management Program (FedRAMP)
- Health Insurance Portability and Accountability Act (HIPAA)

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 AWS Service Catalog depends on the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides these 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 could apply to your industry and location.
- AWS Config – This AWS 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.

Resilience in AWS Service Catalog

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are 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.

In addition to the AWS global infrastructure, AWS Service Catalog offers AWS Service Catalog self-service actions. With self-service actions, customers can reduce administrative maintenance and end-user training while adhering to compliance and security measures. With self-service actions, as the administrator, you can enable end users to perform operational tasks such as backup and restore, troubleshoot issues, run approved commands, and request permissions in AWS Service Catalog. To learn more, see the section called “Using Service Actions”.

Infrastructure Security in AWS Service Catalog

As a managed service, AWS Service Catalog is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access AWS Service Catalog through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). 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 that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

With AWS Service Catalog, you can control the Regions in which data is stored. Portfolios and products are only available in the Regions in which you have made them available. You can use the `CopyProduct` API to copy a product to another Region.

Security Best Practices for AWS Service Catalog

AWS Service Catalog provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don’t represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

You can define rules that limit the parameter values that a user enters when launching a product. These rules are called template constraints because they constrain how the AWS CloudFormation template for the product is deployed. You use a simple editor to create template constraints, and you apply them to individual products.

AWS Service Catalog applies constraints when provisioning a new product or updating a product that is already in use. It always applies the most restrictive constraint among all constraints applied to the portfolio and the product. For example, consider a scenario where the product allows all Amazon EC2 instances to be launched and the portfolio has two constraints: one that allows all non-GPU type EC2 instances to be launched and one that allows only t1.micro and m1.small EC2 instances to be launched. For this example, AWS Service Catalog applies the second, more restrictive constraint (t1.micro and m1.small).

You can limit the access end users have to AWS resources when you attach an IAM policy to a launch role. You then use AWS Service Catalog to create a launch constraint to use the role when launching the product.

To learn more about managed policies for AWS Service Catalog, see AWS Managed Policies for AWS Service Catalog.

Managing Catalogs

AWS Service Catalog provides an interface for managing portfolios, products, and constraints from an administrator console.

> **Note**
>
> To perform any of the tasks in this section, you must have administrator permissions for AWS Service Catalog. For more information, see Identity and Access Management in AWS Service Catalog.

Tasks

- Managing Portfolios
- Managing Products
- Using AWS Service Catalog Constraints
- AWS Service Catalog Service Actions
- Adding AWS Marketplace Products to Your Portfolio
- Using AWS CloudFormation StackSets
- Managing Budgets

Managing Portfolios

You create, view, and update portfolios on the **Portfolios** page in the AWS Service Catalog administrator console.

Tasks

- Creating, Viewing, and Deleting Portfolios
- Viewing Portfolio Details
- Creating and Deleting Portfolios
- Adding products
- Adding Constraints
- Granting Access to Users

- Sharing a Portfolio
- Sharing and Importing Portfolios

Creating, Viewing, and Deleting Portfolios

The **Portfolios** page displays a list of the portfolios that you have created in the current region. Use this page to create new portfolios, view a portfolio's details, or delete portfolios from your account.

To view the Portfolios page

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. Select a different region as necessary.
3. If you are new to AWS Service Catalog, you see the AWS Service Catalog start page. Choose **Get started** to create a portfolio. Follow the instructions to create your first portfolio, and then proceed to the **Portfolios** page.

While using AWS Service Catalog, you can return to the **Portfolios** page at any time; choose **Service Catalog** in the navigation bar and then choose **Portfolios**.

Viewing Portfolio Details

In the AWS Service Catalog administrator console, the **Portfolio details** page lists the settings for a portfolio. Use this page to manage the products in the portfolio, grant users access to products, and apply TagOptions and constraints.

To view the Portfolio details page

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. Choose the portfolio that you want to manage.

Creating and Deleting Portfolios

Use the **Portfolios** page to create and delete portfolios.

To create a new portfolio

1. In the left navigation menu, choose **Portfolios**.

2. Choose **Create portfolio**.
3. On the **Create portfolio** page, enter the requested information.
4. Choose **Create**. AWS Service Catalog creates the portfolio and displays the portfolio details.

To delete a portfolio

> **Note**
>
> You can only delete *local* portfolios. You can remove *imported* (shared) portfolios, but you cannot delete imported portfolios.

Before you can delete a portfolio, you must remove all its products, constraints, groups, roles, users, shares, and TagOptions. To do so, open a portfolio to display **Portfolio details**. Then choose a tab to remove them.

> **Note**
>
> To avoid errors, remove the constraints from the portfolio *before* you remove any products.

1. In the left navigation menu, choose **Portfolios**.
2. Select the portfolio you want to delete.
3. Choose **Delete**. You can only delete *local* portfolios. If you are attempting to delete an *imported* (shared) portfolio, the **Actions** menu is not available.
4. In the confirmation window, choose **Delete**.

Adding products

You can add products to a portfolio by uploading a new product directly to an existing portfolio or by associating an existing product from your catalog to the portfolio.

> **Note**
>
> When you create a AWS Service Catalog product, you can upload an AWS CloudFormation template or Terraform configuration file. The AWS CloudFormation template is stored in

an Amazon Simple Storage Service (Amazon S3) bucket, and the bucket name begins with "***cf-templates-***." You also must have permission to retrieve objects from additional buckets when provisioning a product. For more information, see Creating products.

Adding a new product

You add new products directly from the **Portfolio details** page. When you create a product from this page, AWS Service Catalog adds it to the currently selected portfolio.

To add a new product

1. Navigate to the **Portfolios** page, and then choose the name of the portfolio to which you want to add the product.
2. On the **Portfolio details** page, expand the **Products** section, and then choose **Upload new product**.
3. For **Enter product details**, enter the following:

   - **Product name** – The name of the product.
   - **Product description** (optional) – The product description. This description is shown in the product listing to help you choose the correct product.
   - **Description** – The full description. This description is shown in the product listing to help you choose the correct product.
   - **Owner or Distributor** – The name or email address of the owner. The contact information for the distributor is optional.
   - **Vendor** (optional) – The name of the application's publisher. This field allows you to sort the products list to make it easier to find products.
4. On the **Version details** page, enter the following:

   - **Choose template** – For AWS CloudFormation products, choose your own template file, an AWS CloudFormation template from a local drive or a URL that points to a template stored in Amazon S3, an existing AWS CloudFormation Stack ARN template, or a template file stored in an external repository.

     For Teraform products, choose your own template file, a tar.gz configuration file from a local drive or a URL that points to a template stored in Amazon S3, or a tar.gz configuration file stored in an external repository.

- **Version name** (optional) – The name of the product version (e.g., "v1", "v2beta"). No spaces are allowed.
- **Description** (optional) – A description of the product version including how this version differs from the previous version.

5. For **Enter support details**, enter the following:

- **Email contact** (optional) – The email address for reporting issues with the product.
- **Support link** (optional) – An URL to a site where users can find support information or file tickets. The URL must begin with `http://` or `https://`. Administrators are responsible for maintaining the accuracy and access of support information.
- **Support description** (optional) – A description of how you should use the **Email contact** and **Support link**.

6. Choose **Create product.**

Adding an existing product

You can add existing products to a portfolio from three places: **Portfolios** list, **Portfolio details** page, or the **Product list** page.

To add an existing product to a portfolio

1. Navigate to the **Portfolios** page.
2. Choose a portfolio. Then choose **Actions** - **Add product to portfolio**.
3. Choose a product, and then choose **Add product to portfolio**.

Removing a product from a portfolio

When you no longer want to use a product, remove it from a portfolio. The product is still available in your catalog from the **Products** page, and you can still add it to other portfolios. You can remove multiple products from a portfolio at one time.

To remove a product from a portfolio

1. Navigate to the **Portfolios** page, and then choose the portfolio that contains the product. The **Portfolio details** page opens.
2. Expand the **Products** section.

3. Choose one or more products, and then choose **Remove**.
4. Confirm your choice.

Adding Constraints

You should add constraints to control how users engage with products. For more information about the types of constraints that AWS Service Catalog supports, see Using AWS Service Catalog Constraints.

You add constraints to products after they have been placed in a portfolio.

To add a constraint to a product

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. Choose **Portfolios** and select a portfolio.
3. In the portfolio details page, expand the **Create constraint** section and choose **Add constraints**.
4. For **Product**, select the product to which to apply the constraint.
5. For **Constraint type**, choose one of the following options:

   Launch – Allows you to assign an IAM role to the product that is used to provision the AWS resources. For more information, see AWS Service Catalog Launch Constraints.

   Notification – Allows you to stream product notifications to an Amazon SNS topic. For more information, see AWS Service Catalog Notification Constraints.

   Template – Allows you to limit the options that are available to end users when they launch the product. A Template consists of a JSON–formatted text file that contains one or more rules. Rules are added to the AWS CloudFormation template used by the product. For more information, see Template Constraint Rules.

   Stack Set – Allows you to configure product deployment across accounts and regions using AWS CloudFormation StackSets. For more information, see AWS Service Catalog Stack Set Constraints.

   Tag Update – Allows you to update tags after the product has been provisioned. For more information, see AWS Service Catalog Tag Update Constraints.
6. Choose **Continue** and enter the required information.

To edit a constraint

1. Sign in to the AWS Management Console and open the AWS Service Catalog administrator console at https://console.aws.amazon.com/catalog/.
2. Choose **Portfolios** and select a portfolio.
3. In the **Portfolio details** page, expand the **Create constraint** section and select the constraint to edit.
4. Choose **Edit constraints**.
5. Edit the constraint as needed, and choose **Save**.

Granting Access to Users

Give users access to portfolios through groups or roles. The best way to provide portfolio access for many users is to put the users in an IAM group and grant access to that group. That way you can simply add and remove users from the group to manage portfolio access. For more information, see IAM users and groups in the *IAM User Guide*.

In addition to access to a portfolio, users must also have access to the AWS Service Catalog end user console. You grant access to the console by applying permissions in IAM. For more information, see Identity and Access Management in AWS Service Catalog.

If you want to share a portfolio and its Principals with other accounts, you can associate Principal Names (groups, roles or users) with the Portfolio. Principal Names are shared with the Portfolio and used in recipient accounts to grant access to end users.

To grant portfolio access to users or groups

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. From the navigation pane, choose **Administration**, and then choose **Portfolios**.
3. Choose a portfolio that you want to grant groups, roles, or users access to. AWS Service Catalog directs to the **Portfolio details** page.
4. On the **Portfolio details** page, choose the **Access** tab.
5. Under **Portfolio access**, choose **Grant access**.
6. For **Type**, choose **Principal Name**, and then select the **group/**, **role/**, or **user/**, Type. You can add up to 9 principal names.
7. Choose **Grant Access** to associate the principal to the current portfolio.

To remove access to a portfolio

1. On the **Portfolio details** page, choose a group, role, or user name.
2. Choose **Remove access**.

Sharing a Portfolio

To enable a AWS Service Catalog administrator for another AWS account to distribute your products to end users, share your AWS Service Catalog portfolio with them using either account-to-account sharing or AWS Organizations.

When you share a portfolio using account-to-account sharing or Organizations, you are sharing a *reference* of that portfolio. The products and constraints in the imported portfolio stay in sync with changes that you make to the *shared portfolio*, the original portfolio that you shared.

The recipient cannot change the products or constraints, but can add AWS Identity and Access Management access for end users.

> **Note**
>
> You cannot share a shared resource. This includes portfolios that contain a shared product.

Account-to-account sharing

To complete these steps, you must obtain the account ID of the target AWS account. You can find the ID on the **My Account** page in the AWS Management Console of the target account.

To share a portfolio with an AWS account

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. In the left navigation menu, choose **Portfolios** and then select the portfolio you want to share. In the **Actions** menu, select **Share**.
3. In **Enter account ID** enter the account ID of the AWS account that you are sharing with. (Optional) Select TagOption Sharing. Then, choose **Share**.
4. Send the URL to the AWS Service Catalog administrator of the target account. The URL opens the **Import Portfolio** page with the ARN of the shared portfolio automatically provided.

Importing a Portfolio

If a AWS Service Catalog administrator for another AWS account shares a portfolio with you, import that portfolio into your account so that you can distribute its products to your end users.

You do not need to import a portfolio if the portfolio was shared through AWS Organizations.

To import the portfolio, you must get the portfolio ID from the administrator.

To view all imported portfolios, open the AWS Service Catalog console at https://console.aws.amazon.com/servicecatalog/. On the **Portfolios** page, select the **Imported** tab. Review the **Imported Portfolios** table.

Sharing with AWS Organizations

You can share AWS Service Catalog portfolios using AWS Organizations.

First, you must decide if you're sharing from the management account or a delegated administrator account. If you don't want to share from your management account, register a delegated admin account that you can use for sharing. For more information, see Register a delegated administrator in the *AWS CloudFormation User Guide*.

Next, you must decide who to share to. You can share to the following entities:

- An organization account.
- An organizational unit (OU).
- The organization itself. (This shares with every account in the organization.)

Sharing from a management account

You can share a portfolio with an organization when you use your organizational structure or input the ID of an organizational node.

To share a portfolio with an organization by using the organizational structure

1. Open the AWS Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. On the **Portfolios** page, select the portfolio that you want to share. In the **Actions** menu, select **Share**.
3. Select **AWS Organizations** and filter into your organizational structure.

You can select the Root node to share the portfolio with your entire organization, a parent Organizational Unit (OU), a child OU, or an AWS account within your organization.

Sharing to a parent OU shares the portfolio to all accounts and child OU's within that parent OU.

You can select **View AWS accounts only** to see a list of all of the AWS accounts in your organization.

To share a portfolio with an organization by entering the ID of the organizational node

1. Open the AWS Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. On the **Portfolios** page, select the portfolio that you want to share. In the **Actions** menu, select **Share**.
3. Select **Organization Node**.

   Select whether you want to share with your entire organization, an AWS account within your organization, or an OU.

   Input the ID of the organizational node you selected, which you can find within the AWS Organizations console at https://console.aws.amazon.com/organizations/.

Sharing from a delegated administrator account

The management account of an organization can register and de-register other accounts as delegated administrators for the organization.

A delegated administrator can share AWS Service Catalog resources in their organization the same way a management account can. They are authorized to create, delete, and share portfolios.

To register or de-register a delegated administrator, you must use the API or CLI from the management account. For more information, see RegisterDelegatedAdministrator and DeregisterDelegatedAdministrator in the *AWS Organizations API Reference*.

> **Note**
>
> Before you can designate a delegate , the administrator must call `EnableAWSOrganizationsAccess`.

The procedure for sharing a portfolio from a delegated administrator account is the same as sharing from a management account, as seen above in the section called “Sharing from a management account”.

If a member is de-registered as a delegated administrator, the following occurs:

- Portfolio shares that were created from that account are removed.
- They can no longer create new portfolio shares.

> **Note**
>
> If the portfolio and shares created by a delegated administrator do not get removed after the delegated administrator is de-registered, register and de-register the delegated administrator again. This action removes the portfolio and shares created by that account.

Moving accounts within your organization

If you move an account within your organization, the AWS Service Catalog portfolios shared with the account might change.

Accounts only have access to portfolios shared with their destination organization or organizational unit.

Sharing TagOptions when sharing portfolios

As an administrator, you can create a share to include TagOptions. TagOptions are key-value pairs that enables administrators to:

- Define and enforce the taxonomy for tags.
- Define tag options and associate them to products and portfolios.
- Share tag options associated with portfolios and products with other accounts.

When you add or remove tag options in the main account, the change automatically appears in recipient accounts. In recipient accounts, when an end user provisions a product with TagOptions, they must choose values for tags that become tags on the provisioned product.

In recipient accounts administrators can associate additional local TagOptions to their imported portfolio to enforce tagging rules that are specific to that account.

> **Note**
>
> To share a portfolio, you need the the consumer's AWS account ID. Find the AWS account ID in **My Account** in the console.

> **Note**
>
> If a TagOption has a single value, AWS automatically enforces that value during the provisioning process.

To share TagOptions when sharing portfolios

1. In the left navigation menu, choose **Portfolios**.
2. In **Local portfolios**, choose and open a portfolio.
3. Choose **Share** from the list above and then choose the **Share** button.
4. Choose to share with another AWS account or organization.
5. Enter the 12 digit account ID number, select **Enable**, and then choose **Share**.

   The account you shared displays in the **Accounts shared with** section. It indicates whether TagOptions were enabled.

You can also update a portfolio share to include TagOptions. All TagOptions that belong to the portfolio and product now share to this account.

To update a portfolio share to include TagOptions

1. In the left navigation menu, choose **Portfolios**.
2. In **Local portfolio**, choose and open a portfolio.
3. Choose **Share** from the list above.
4. In **Accounts shared with**, choose an account ID and then choose **Actions**.
5. Select **Update unshare** or **Unshare**.

   When you select **Update unshare**, choose **Enable** to initiate sharing TagOptions. The account you shared displays in the **Accounts shared with** section.

When you select **Unshare**, confirm you no longer want to share the account.

Sharing Principal Names when sharing portfolios

As an administrator, you can create a Portfolio share that includes Principal Names. Principal Names are names for groups, roles and users that administrators can specify in a Portfolio, and then share with the portfolio. When you share the portfolio, AWS Service Catalog verifies if those Principal Names already exist. If they do exist, AWS Service Catalog automatically associates the matching IAM Principals with the shared Portfolio to grant access to users.

> **Note**
>
> When you associate a principal with portfolio, a potential privilege escalation path may occur when that portfolio is then shared with other accounts. For a user in a recipient account who is *not* a AWS Service Catalog Admin, but still has the ability to create Principals (Users/Roles), that user could create an IAM Principal that matches a principal name association for the portfolio. Although this user may not know which principal names are associated through AWS Service Catalog, they may be able to guess the user. If this potential escalation path is a concern, then AWS Service Catalog recommends using `PrincipalType` as `IAM`. With this configuration, the `PrincipalARN` must already exist in the recipient account before it can be associated.

When you add or remove Principal Names in the main account, AWS Service Catalog automatically applies those changes in the recipient account. Users in recipient account can then perform tasks based on their role:

- **End users** can provision, update, and terminate the portfolio's product.
- **Administrators** can associate additional IAM Principals to their imported portfolio to grant access to end users specific to that account.

> **Note**
>
> Principal Name Sharing is only available for AWS Organizations.

To share Principal Names when sharing portfolios

1. In the left navigation menu, choose **Portfolios**.
2. In **Local portfolios**, choose the portfolio you want to share.
3. In the **Actions** menu, choose **Share**.
4. Select an organization in AWS Organizations.
5. Select the entire **organization root**, an **organization unit (OU)**, or an **organization member**.
6. In **Share** settings, enable the **Principal sharing** option.

You can also update a portfolio share to include Principal Name sharing. This shares all Principal Names that belong to that portfolio with the recipient account.

To update a portfolio share to enable or disable Principal Names

1. In the left navigation menu, choose **Portfolios**.
2. In **Local portfolio**, choose the portfolio you want to update.
3. Choose the **Share** tab.
4. Select the share you want to update, and then chose **Share**.
5. Choose **Update share**, and then choose **Enable** to initiate Principal sharing. AWS Service Catalog then shares Principal Names in recipient accounts.

Disable Principal sharing if you want to stop sharing the Principal Names with recipient accounts.

Using wildcards when sharing Principal Names

AWS Service Catalog supports granting portfolio access to IAM principals (user, group or role) names with wildcards, such as '*' or '?'. Using wildcard patterns enables you to cover multiple IAM principal names at one time. The ARN path and principal name allow unlimited wildcard characters.

Examples of an **acceptable** wildcard ARN:

- `arn:aws:iam:::role/ResourceName_*`
- `arn:aws:iam:::role/*/ResourceName_?`

Examples of an **unacceptable** wildcard ARN:

- **arn:aws:iam:::*/ResourceName**

In the IAM Principal ARN format (**arn:partition:iam:::resource-type/resource-path/resource-name**), valid values include **user/**, **group/**, or **role/**. The "?" and "*" are allowed only after the resource-type in the resource-id segment. You can use special characters anywhere within the resource-id.

The "*" character also matches the "/" character, allowing paths to be formed *within* the resource-id. For example:

arn:aws:iam:::role/*/ResourceName_? matches both **arn:aws:iam:::role/pathA/pathB/ResourceName_1** and **arn:aws:iam:::role/pathA/ResourceName_1**.

Sharing and Importing Portfolios

To make your AWS Service Catalog products available to users who are not in your AWS accounts, such as users who belong to other organizations or to other AWS accounts in your organization, you share your portfolios with them. You can share in several ways, including account-to-account sharing, organizational sharing, and deploying catalogs using stack sets.

Before you share your products and portfolios to other accounts, you must decide whether you want to share a reference of the catalog or to deploy a copy of the catalog into each recipient account. Note that if you deploy a copy, you must redeploy if there are updates you want to propagate to the recipient accounts.

You can use stack sets to deploy your catalog to many accounts at the same time. If you want to share a reference (an imported version of your portfolio that stays in sync with the original), you can use account-to-account sharing or you can share using AWS Organizations.

To use stack sets to deploy a copy of your catalog, see How to set up a multi-region, multi-account catalog of company standard AWS Service Catalog products.

When you share a portfolio using account-to-account sharing or AWS Organizations, you allow a AWS Service Catalog administrator of another AWS account to import your portfolio into their account and distribute the products to end users in that account.

This *imported portfolio* isn't an independent copy. The products and constraints in the imported portfolio stay in sync with changes that you make to the *shared portfolio*, the original portfolio that you shared. The *recipient administrator*, the administrator with whom you share a portfolio, cannot

change the products or constraints, but can add AWS Identity and Access Management (IAM) access for end users. For more information, see Granting Access to Users.

The recipient administrator can distribute the products to end users who belong to their AWS account in the following ways:

- By adding users, groups, and roles to the imported portfolio.
- By adding products from the imported portfolio to a **local portfolio**, a separate portfolio that the recipient administrator creates and that belongs to their AWS account. The recipient administrator then adds users, groups, and roles to that local portfolio. Any constraints originally applied to products in the shared portfolio are also present in the local portfolio. The local portfolio recipient administrator can add additional constraints, but cannot remove the constraints that were originally imported from the shared portfolio.

When you add products or constraints to the shared portfolio or remove products or constraints from it, the change propagates to all imported instances of the portfolio. For example, if you remove a product from the shared portfolio, that product is also removed from the imported portfolio. It is also removed from all local portfolios that the imported product was added to. If an end user launched a product before you removed it, the end user's provisioned product continues to run, but the product becomes unavailable for future launches.

If you apply a launch constraint to a product in a shared portfolio, it propagates to all imported instances of the product. To override this launch constraint, the recipient administrator adds the product to a local portfolio and then applies a different launch constraint to it. The launch constraint that is in effect sets a launch role for the product.

A *launch role* is an IAM role that AWS Service Catalog uses to provision AWS resources (such as Amazon EC2 instances or Amazon RDS databases) when an end user launches the product. As an administrator, you can choose to designate a specific launch role ARN or a local role name. If you use the role ARN, the role will be used even if the end user belongs to a different AWS account than the one that owns the launch role. If you use a local role name, the IAM role with that name in the end user's account is used.

For more information about launch constraints and launch roles, see AWS Service Catalog Launch Constraints. The AWS account that owns the launch role provisions the AWS resources, and this account incurs the usage charges for those resources. For more information, see AWS Service Catalog Pricing.

This video shows you how to share portfolios across accounts in AWS Service Catalog.

Share (https://www.youtube.com/embed/BVSohYOppjk%22%3EShare) Portfolios Across Accounts in AWS Service Catalog.

> **Note**
>
> You cannot re-share products from a portfolio that has been imported or shared.

> **Note**
>
> Portfolio imports must occur in the same region between the management and dependent accounts.

Relationship Between Shared and Imported Portfolios

This table summarizes the relationship between an imported portfolio and a shared portfolio, and the actions that an administrator who imports a portfolio can and can't take with that portfolio and the products in it.

Element of Shared Portfolio	Relationship to Imported Portfolio	Recipient Administrator Can	Recipient Administrator Cannot
Products and product versions	Inherited. If the portfolio creator adds products to or removes products from the shared portfolio, the change propagates to the imported portfolio.	Add imported products to local portfolios. Products stay in sync with shared portfolio.	Upload or add products to the imported portfolio or remove products from the imported portfolio.
Launch constraints	Inherited. If the portfolio creator adds launch	In a local portfolio, the administrator can apply launch constrain	Add launch constraints to or remove launch constraints

Element of Shared Portfolio	Relationship to Imported Portfolio	Recipient Administrator Can	Recipient Administrator Cannot
	constraints to or removes launch constraints from a *shared product*, the change propagates to all imported instances of the product. If the recipient administrator adds an imported product to their *local* portfolio, that imported launch constraint is not carried over to the shared portfolio.	ts that affect the *local* launch of the product.	from the imported portfolio.

Element of Shared Portfolio	Relationship to Imported Portfolio	Recipient Administrator Can	Recipient Administrator Cannot
Template constraints	Inherited. If the portfolio creator adds a template constraint to or removes a template constraints from a shared product, the change propagates to all imported instances of the product. If the recipient administrator adds an imported product to a local portfolio, the imported template constraints are not carried over to the local portfolio.	In a local portfolio, the administrator can add template constraints that constrain the local product.	Remove the imported template constraints.
Users, groups, and roles	Not inherited.	Add users, groups, and roles that are in administrator's AWS account.	Not applicable.

Managing Products

You can create products, update products by creating a new version based on an updated template, and group products together into portfolios to distribute them to users.

New versions of products are propagated to all users who have access to the product through a portfolio. When you distribute an update, end users can update existing provisioned products.

Tasks

- Viewing the Products Page
- Creating Products
- Adding products to portfolios
- Updating products
- Syncing products to template files from GitHub, GitHub Enterprise, or Bitbucket
- Deleting products
- Managing Versions

Viewing the Products Page

You manage products from the **Products list** page in the AWS Service Catalog administrator console.

To view the Products list page

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. Choose **Product list**.

Creating Products

You create products from the **Products** page in the AWS Service Catalog administrator console.

> **Note**
>
> Creating Terraform products require additional configuration, including a Terraform provisioning engine and launch role. For more information, review Getting started with a Terraform product.

To create a new AWS Service Catalog product

1. Navigate to the **Products list** page.
2. Choose **Create product**, and the choose **Create product**.
3. **Product details** – Enables you to choose the type of product you want to create. AWS Service Catalog supports AWS CloudFormation, Terraform Cloud, and External (supports Terraform

Community Edition) product types. Product details also contains the metadata that appears when you search for and view products in a list or detail page. Enter the following:

- **Product name** – The name of the product.
- **Product description** – The description shows in the product listing to help you choose the correct product.
- **Owner** – The person or organization that publishes this product. The owner could be the name of your IT organization, or administrator.
- **Distributor** (optional) – The name of the application's publisher. This field allows you to sort the products list to make it easier to find products.

4. **Version details** enables you to add your template file and build your product. Enter the following:

   - **Choose method** – There are four ways to add a template file.
     - **Use a local template file** - Upload an AWS CloudFormation template or a Terraform tar.gz configuration file from a local drive.
     - **Use an Amazon S3 URL** - Specify a URL that points to an AWS CloudFormation template or a Terraform tar.gz configuration file stored in Amazon S3. If you specify an Amazon S3 URL, it must begin with `https://`.
     - **Use an external repository** - Specify your GitHub, GitHub Enterprise, or Bitbucket code repository. AWS Service Catalog allows you to sync products to template files. For Terraform products, the template file format is required to be a single file archived in Tar and compressed in Gzip.
     - **Use an existing CloudFormation stack** - Enter the ARN for an existing CloudFormation stack. This method does not support Terraform Cloud or External products.
   - **Version name** (optional) – The name of the product version (e.g., "v1", "v2beta"). No spaces are allowed.
   - **Description** (optional) – A description of the product version, including how this version differs from the other versions.
   - **Guidance** – Managed in the versions tab on a **Product details** page. When a product version is created—during the create product workflow—guidance for that version is set to default. To learn more about guidance, see Managing Versions.

5. **Support details** identifies the organization within your company, and provides a point of contact for support. Enter the following:

- **Email contact** (optional) – The email address for reporting issues with the product.
- **Support link** (optional) – An URL to a site where users can find support information or file tickets. The URL must begin with `http://` or `https://`. Administrators are responsible for maintaining the accuracy and access of support information.
- **Support description** (optional) – A description of how you should use the **Email contact** and **Support** link.

6. **Manage tags** (optional) – In addition to using tags to categorize your resources, you can also use them to authenticate your permissions to create this resource.
7. **Create product** – When you have completed the form, select **Create product**. After a few seconds, the product appears on the **Products list** page. You might need to refresh your browser to see the product.

You can also use CodePipeline to create and configure a pipeline to deploy your product template to AWS Service Catalog and deliver changes you have made in your source repository. For more information, see Tutorial: Create a Pipeline That Deploys to AWS Service Catalog.

You can define parameter properties in your AWS CloudFormation or Terraform template and enforce those rules during provisioning. These properties can define the minimum and maximum length, minimum and maximum values, allowed values, and a regular expression for the value. AWS Service Catalog issues a warning during provisioning if the value provided does not adhere to the parameter property. To learn more about parameter properties, see Parameters in the *AWS CloudFormation User Guide*.

Troubleshooting

You must have permission to retrieve objects from Amazon S3 buckets. Otherwise, you might encounter the following error when launching or updating a product.

```
Error: failed to process product version s3 access denied exception
```

If you encounter this message, ensure have permission to retrieve objects from the following buckets:

- The bucket where the provisioning artifact template is stored.
- The bucket that begins with "***cf-templates-****" and where AWS Service Catalog stores the provisioning artifact template.

- The internal bucket that begins with "***sc-****" and where AWS Service Catalog stores metadata. You won't be able to see this bucket from your account.

The following example policy shows the minimum permissions that are required to retrieve objects from the previously mentioned buckets.

```
{
          "Sid": "VisualEditor1",
          "Effect": "Allow",
          "Action": "s3:GetObject*",
          "Resource": [
              "arn:aws:s3:::YOUR_TEMPLATE_BUCKET",
              "arn:aws:s3:::YOUR_TEMPLATE_BUCKET/*",
              "arn:aws:s3:::cf-templates-*",
              "arn:aws:s3:::cf-templates-*/*",
              "arn:aws:s3:::sc-*",
              "arn:aws:s3:::sc-*/*"
          ]
      }
```

Adding products to portfolios

You can add products to any number of portfolios. When a product is updated, all of the portfolios (including shared portfolios) that contain the product automatically receive the new version.

To add a product from your catalog to a portfolio

1. Navigate to the **Products list** page.
2. Select a product, and then choose **Actions**. From the dropdown menu, choose **Add product to portfolio**. You're directed to the **Add *name-of-product* to portfolio** page.
3. Choose a portfolio, and then choose **Add product to portfolio**.

When adding a Terraform product to a portfolio, the product requires a launch constraint. You must select an IAM role from your account, enter an IAM role ARN, or enter a role name. If you specify a role name and if an account uses the launch constraint, the account uses that name for the IAM role. This allows launch-role constraints to be account-agnostic, ensuring you can create fewer resources per shared account. For details and instructions, review Step 6: Add a Launch constraint to your Terraform product

A portfolio can contain numerous products that are mix of AWS CloudFormation and Terraform product types.

Updating products

When you update a product's template, you create a new version of the product. New product versions are automatically available to all users who have access to a portfolio containing the product.

> **Note**
>
> When updating an existing product, you cannot change the product type (AWS CloudFormation or Teraform). For example, if you update a AWS CloudFormation product, you cannot replace the existing AWS CloudFormation template with a Terraform tar.gz configuration file. You must update the existing AWS CloudFormation template file with a new AWS CloudFormation template file.

End users who are currently running a provisioned product of the previous product version can update their provisioned product to the new version. When a new version of a product is available, users can use the **Update provisioned product** command on the **Provisioned product list** or **Provisioned product details** pages.

Before you create a new version of a product, AWS Service Catalog recommends that you test your product updates in AWS CloudFormation or in the Terraform engine to ensure that they function properly.

To create a new product version

1. Navigate to the **Product list** page.
2. Choose the product product that you would like to update. You're directed to the *Product details* page.
3. On the *Product details* page, expand the **Versions** tab, and then choose **Create new version**.
4. Under **Version details**, perform the following:

   - **Choose template** – There are four ways to add a template file.

     Use a local template file - Upload an AWS CloudFormation template or a Terraform tar.gz configuration file from a local drive.

Use an Amazon S3 URL - Specify a URL that points to an AWS CloudFormation template or a Terraform tar.gz configuration file stored in Amazon S3. If you specify an Amazon S3 URL, it must begin with https://.

Use an external repository - Specify your GitHub, GitHub Enterprise, or Bitbucket code repository. AWS Service Catalog allows you to sync products to template files. For Terraform products, the template file format is required to be a single file archived in Tar and compressed in Gzip.

Use an existing CloudFormation stack - Enter the ARN for an existing CloudFormation stack. This method does not support Terraform Cloud or External products.

- **Version title** – The name of the product version (e.g., "v1", "v2beta"). No spaces are allowed.
- **Description** (optional) – A description of the product version, including how this version differs from the previous version.

5. Choose **Create product version**.

You can also use CodePipeline to create and configure a pipeline to deploy your product template to AWS Service Catalog, and deliver your changes in your source repository. For more information, see Tutorial: Create a Pipeline That Deploys to AWS Service Catalog.

Syncing products to template files from GitHub, GitHub Enterprise, or Bitbucket

AWS Service Catalog allows you to sync products to template files that are managed through external repository provider. AWS Service Catalog refers to products with this type of template connection as *Git-synced* products. Repository options include GitHub, GitHub Enterprise, or Bitbucket. After you authorize your AWS account with an external repository account, you can create new AWS Service Catalog products or update existing products to sync to a template file in the repository. When changes are made to the template file and committed in the repository (for example, using git-push), AWS Service Catalog automatically detects the changes and creates a new product version (artifact).

Topics

- Required permissions to sync products to external template files
- Create an account connection
- Viewing Git-synced product connections

- Updating Git-synced product connections
- Deleting Git-synced product connections
- Syncing Terraform products to template files from GitHub, GitHub Enterprise, or Bitbucket
- AWS Region support for Git-synced products

Required permissions to sync products to external template files

You can use the following AWS Identity and Access Management (IAM) policy as a template to enable AWS Service Catalog administrators to sync products to template files from an external repository. This policy includes required permissions from both CodeConnections and AWS Service Catalog. AWS Service Catalog recommends that you copy the template policy below, and also use the AWS Service Catalog `AWSServiceCatalogAdminFullAccess` managed policy when enabling repository-synced products.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CodeStarAccess",
            "Effect": "Allow",
            "Action": [
            "codestar-connections:UseConnection",
            "codestar-connections:PassConnection",
            "codestar-connections:CreateConnection",
            "codestar-connections:DeleteConnection",
            "codestar-connections:GetConnection",
            "codestar-connections:ListConnections",
            "codestar-connections:ListInstallationTargets",
            "codestar-connections:GetInstallationUrl",
            "codestar-connections:StartOAuthHandshake",
            "codestar-connections:UpdateConnectionInstallation",
            "codestar-connections:GetIndividualAccessToken"
            ],
            "Resource": "arn:aws:codestar-connections:*:*:connection/*"
        },
        {
        "Sid": "CreateSLR",
        "Effect": "Allow",
        "Action": "iam:CreateServiceLinkedRole",
```

```
        "Resource": "arn:aws:iam::*:role/aws-service-role/
sync.servicecatalog.amazonaws.com/AWSServiceRoleForServiceCatalogArtifactSync",
        "Condition": {
        "StringLike": {
        "iam:AWSServiceName": "sync.servicecatalog.amazonaws.com"
                }
            }
        }
    ]
}
```

Create an account connection

Before syncing a template file to a AWS Service Catalog product, you must create and authorize a one-time, account-to-account connection. You use this connection to specify the details of the repository containing the desired template file. You can create a connection using the AWS Service Catalog console, CodeConnections console, AWS Command Line Interface (CLI), or CodeConnections APIs.

After establishing a connection, you can use the AWS Service Catalog console, AWS Service Catalog API, or CLI to create a synced AWS Service Catalog product. AWS Service Catalog administrators can create new or update existing AWS Service Catalog products based on a template file in a repository and branch. If a change is committed in the repository, AWS Service Catalog automatically detects the change and creates a new product version. Previous product versions are maintained up to the prescribed version limit and assigned a **deprecated** status.

Additionally, AWS Service Catalog automatically creates a service-linked role (SLR) after the connection is created. This SLR allows AWS Service Catalog to detect any template file changes that are committed to the repository. The SLR also allows AWS Service Catalog to automatically create new product versions for synced products. For more information about SLR permissions and functionality, refer to Service-linked roles for AWS Service Catalog.

To create a new Git-synced product

1. In the left navigation panel, choose **Product list**, and then choose **Create product**.
2. Enter the **Product details**.
3. In Version details, choose **Specify your code repository using an AWS CodeStar provider**, and then choose the **Create a new AWS CodeStar connection** link.

4. After you create the connection, refresh the connections list, and then select the new connection. Specify the repository details, including the **repository**, **branch**, and **template file path**.

   For infomration about using a Terraform configuration file, see Syncing Terraform products to template files from GitHub, GitHub Enterprise, or Bitbucket .

   a. (Optional when creating a new AWS Service Catalog product resource) In the **Support Details** section, add metadata for the product.

   b. (Optional when creating a new AWS Service Catalog product resource) In the **Tags** section, choose **Add new tag** and enter the **Key** and **Value** pairs.

5. Choose **Create new product**.

To create multiple Git-synced products

1. In the AWS Service Catalog console left navigation panel, choose **Product list**, and then choose **Create multiple git-managed products**.
2. Enter the **Common product details**.
3. In External repository details, select an **AWS CodeStar connection**, and then specify the **repository** and **branch**.
4. In the Add products pane, enter the **Template file path** and **Product name**. Choose **Add new item** and continue adding products as desired.
5. After adding all desired products, choose **Bulk create products**.

To connect an existing AWS Service Catalog product to an external repository

1. In the AWS Service Catalog console left navigation panel, choose **Product list**, and then choose **Connect products to an external repository**.
2. On the Select products page, select the products you want to connect to an external repository, and then choose **Next**.
3. On the Specify source details page, select an existing AWS CodeStar connection, and then specify the **repository**,the **branch**, and the **template file path**.
4. Choose **Next**.
5. On the Review and submit page, verify the connection details, and then choose **Connect products to an external repository**.

Viewing Git-synced product connections

You can use the AWS Service Catalog console, API, or AWS CLI to view repository connection details. For AWS Service Catalog products that are linked to a template file, you can retrieve information about the repository connection and the last time the template was synced with the product from the **Last Sync Status**.

> **Note**
>
> You can view repository information and the **Last Sync Status** at the product level. Users must have IAM permissions in the CodeConnections APIs to view repository details. Refer to Required permissions to sync AWS Service Catalog products to template files for more information about the required policy for these IAM permissions.

To view connection and repository details using AWS Management Console

1. In the left navigation panel, choose **Product list**.
2. Select the product from the list.
3. On the **Product** page, navigate to the **Product source details** section.
4. To view the source revision ID for a product version, choose the **Last version created** link. The **Version details** section display the source revision ID.

To view connection and repository details using AWS CLI

From the AWS CLI, run the following commands:

```
$ aws servicecatalog describe-product-as-admin

$ aws servicecatalog describe-provisioning-artifact

$ aws servicecatalog search-product-as-admin

$ aws servicecatalog list-provisioning-artifacts
```

Updating Git-synced product connections

You can update existing account connections and Git-synced products using the AWS Service Catalog console, AWS Service Catalog API, or AWS CLI.

To learn how to connect an existing AWS Service Catalog product to a template file, refer to Creating new Git-synced product connections.

To update existing products to Git-synced products

1. In the left navigation panel, choose **Product list**, and then choose one of the following options:
   - To update a **single product**, select the product, navigate to the **Product source details** section, and then choose **Edit details**.
   - To update **multiple products**, choose **Connect products to an external repository**, select up to ten products, and then choose **Next**.
2. In the **Product source details** section, perform the following updates:
   - Specify the connection.
   - Specify the repository.
   - Specify the branch.
   - Name the template file.
3. Choose **Save changes**.

> **Note**
>
> For products not yet connected to an external repository, you can use the **Connect to an external repository** option displayed in the alert at the top of the product info page after selecting the product.

You can also use the AWS Service Catalog console or the AWS CLI to

- Connect an existing AWS Service Catalog product to a template file in an external repository
- Update product metadata, including the product name, description, and tags.
- Reconfigure (update the sync to use a different repository source) a connection for a previously connected AWS Service Catalog product.

To update connection and repository details using AWS Service Catalog console

1. In the AWS Service Catalog console left navigation panel, choose **Product list**, and then select a product that is currently connected to an external repository.
2. In the **Product source details** section, choose **Edit product source**.
3. In the **Product source details** section, specify the new desired repository.
4. Choose **Save changes**.

To update connection and repository details using AWS CLI

From the AWS CLI run the `$ aws servicecatalog update-product` and `$ aws servicecatalog update-provisioning-artifact` commands.

Deleting Git-synced product connections

You can delete a connection between a AWS Service Catalog product and a template file using the AWS Service Catalog console, CodeConnections API, or AWS CLI. When you disconnect a product from a template file, the synced AWS Service Catalog product switches to a regularly managed product. After disconnecting the product, if the template file is changed and committed in the previously connected repository, the changes are *not* reflected. To re-connect a AWS Service Catalog product to a template file in an external repository, refer to Updating connections and synced AWS Service Catalog products.

To disconnect a Git-synced product using the AWS Service Catalog console

1. In the AWS Management Console, choose **Product list** from the left navigation panel.
2. Select a product from the list.
3. On the **Product** page, navigate to the **Product source details** section.
4. Choose **Disconnect**.
5. Confirm the action, and then choose **Disconnect**.

To disconnect a Git-synced product using AWS CLI

From the AWS CLI, run the `$ aws servicecatalog update-product` command. In the `ConnectionParameters` input, remove the specified connection.

To delete a connection using the CodeConnections API or AWS CLI

In the CodeConnections API or AWS CLI, run the `$ aws codestar-connections delete-connection` command.

Syncing Terraform products to template files from GitHub, GitHub Enterprise, or Bitbucket

When creating a Git-synced product using a Terraform configuration file, the file path only accepts the tar.gz format. Terraform folder formats are not accepted in the file path.

AWS Region support for Git-synced products

AWS Service Catalog supports Git-synced produtcs in AWS Regions as indicated in the table below.

AWS Region name	AWS Region identity	Support for Git-synced products
US East (N. Virginia)	us-east-1	Yes
US East (Ohio)	us-east-2	Yes
US West (N. California)	us-west-1	Yes
US West (Oregon)	us-west-2	Yes
Africa (Cape Town)	af-south-1	No
Asia Pacific (Hong Kong)	ap-east-1	No
Asia Pacific (Jakarta)	ap-southeast-3	No
Asia Pacific (Mumbai)	ap-south-1	Yes
Asia Pacific (Osaka)	ap-northeast-3	No
Asia Pacific (Seoul)	ap-northeast-2	Yes
Asia Pacific (Singapore)	ap-southeast-1	Yes
Asia Pacific (Sydney)	ap-southeast-2	Yes
Asia Pacific (Tokyo)	ap-northeast-1	Yes

AWS Region name	AWS Region identity	Support for Git-synced products
Canada (Central)	ca-central-1	Yes
Europe (Frankfurt)	eu-central-1	Yes
Europe (Ireland)	eu-west-1	Yes
Europe (London)	eu-west-2	Yes
Europe (Milan)	eu-south-1	No
Europe (Paris)	eu-west-3	Yes
Europe (Stockholm)	eu-north-1	Yes
Middle East (Bahrain)	me-south-1	No
South America (São Paulo)	sa-east-1	Yes
AWS GovCloud (US-East)	us-gov-east-1	No
AWS GovCloud (US-West)	us-gov-west-1	No

Deleting products

When you delete a product, AWS Service Catalog removes all product versions from every portfolio containing the product.

AWS Service Catalog allows you to delete a product using the AWS Service Catalog console or AWS CLI. To successfully delete a product, you must disassociate all resources associated with the product first. Examples of product resource associations include portfolio associations, budgets, TagOptions, and Service Actions.

> **Important**
>
> You cannot recover a product after it is deleted.

To delete a product using the AWS Service Catalog console

1. Navigate to the **Portfolios** page and select the portfolio containing the product you want to delete.
2. Select the product that you want to delete, and then choose **Delete** on the upper right of the product pane.
3. For products *without associated resources*, confirm the product you want to delete by entering **delete** in the text box, and then choose **Delete**.

   For products *with associated resources*, continue to step 4.
4. In the **Delete product** window, review the **Associations** table, which displays all of the product's associated resources. AWS Service Catalog attempts to disassociate these resources when you delete the product.
5. Confirm you want to delete the product and remove all of its associated resources by entering **delete** in the text box.
6. Choose **Disassociate and delete**.

If AWS Service Catalog is unable to disassociate all of the product's resources, the product is not deleted. The **Delete product** window displays the number of failed disassociations and a description for each failure. For more information about resolving failed resource disassociations when deleting a product, see *Resolving failed resource disassociations when deleting a product* below.

Topics

- Deleting products using the AWS CLI
- Resolving failed resource disassociations when deleting a product

Deleting products using the AWS CLI

AWS Service Catalog allows you to use the AWS Command Line Interface (AWS CLI) to delete products from your portfolio. The AWS CLI is an open source tool that enables you to interact with AWS services using commands in your command-line shell. The AWS Service Catalog force-delete function requires an AWS CLI alias, which is a shortcut you can create in the AWS CLI to shorten commands or scripts that you frequently use.

Prerequisites

- Install and configure the AWS CLI. For more information, see Installing or updating the latest version of the AWS CLI and Configuration basics. Use a minimum AWS CLI version of 1.11.24 or 2.0.0.
- The delete product CLI alias requires a bash-compatible terminal and the JQ command-line JSON processor. For more information about installing the Command-line JSON processor, see Download jq.
- Create a AWS CLI Alias to batch `Disassociation` API calls, enabling you to delete a product in a single command.

To successfully delete a product, you must disassociate all resources associated with the product first. Examples of product resource associations include portfolio associations, budgets, Tag Options, and Service Actions. When using the CLI to delete a product, the CLI `force-delete-product` alias enables you to call the `Disassociate` API to disassociate any resources that would prevent the `DeleteProduct` API. This avoids a seperate call for individual disassociations.

> **Note**
>
> The file paths shown in the procedures below may vary depending on which operating system you use to perform these actions.

Creating an AWS CLI alias to delete AWS Service Catalog products

When using the AWS CLI to delete a AWS Service Catalog product, the CLI `force-delete-product` alias enables you to call the `Disassociate` API to disassociate any resources that would prevent the `DeleteProduct` call.

Create an `alias` file in your AWS CLI configuration folder

1. In the AWS CLI console, navigate to the configuraiton folder. By default, the configuration folder path is `~/.aws/` on Linux and macOS, or `%USERPROFILE%\.aws\` on Windows.
2. Create a sub-folder named `cli` using file navigation or by entering the following command in your preferred terminal:

```
$ mkdir -p ~/.aws/cli
```

The resulting `cli` folder default path is `~/.aws/cli/` on Linux and MacOS, or `%USERPROFILE%\.aws\cli` on Windows.

3. In the new `cli` folder, create a text file named `alias` with no file extension. You can create the `alias` file using file navigation or by entering the following command in your preferred terminal:

```
$ touch ~/.aws/cli/alias
```

4. Enter `[toplevel]` on the first line.
5. Save the file.

Next, you can add the force-delete-product alias to your `alias` file by manually pasting the alias script into the file, or by using a command in the terminal window.

Manually add the force-delete-product alias to your `alias` file

1. In the AWS CLI console, navigate to your AWS CLI configuration folder and open the `alias` file.
2. Enter the following code alias into the file, below the `[toplevel]` line:

```
 [command servicecatalog]
force-delete-product =
  !f() {
    if [ "$#" -ne 1 ]; then
        echo "Illegal number of parameters"
        exit 1
    fi

    if [[ "$1" != prod-* ]]; then
       echo "Please provide a valid product id."
       exit 1
    fi

    productId=$1
```

```
                describeProductAsAdminResponse=$(aws servicecatalog describe-
product-as-admin --id $productId)
                listPortfoliosForProductResponse=$(aws servicecatalog list-
portfolios-for-product --product-id $productId)

                tagOptions=$(echo "$describeProductAsAdminResponse" | jq -r
 '.TagOptions[].Id')
                budgetName=$(echo "$describeProductAsAdminResponse" | jq -r
 '.Budgets[].BudgetName')
                portfolios=$(echo "$listPortfoliosForProductResponse" | jq -r
 '.PortfolioDetails[].Id')
                provisioningArtifacts=$(echo "$describeProductAsAdminResponse" | jq
 -r '.ProvisioningArtifactSummaries[].Id')
                provisioningArtifactServiceActionAssociations=()

                for provisioningArtifactId in $provisioningArtifacts; do
                  listServiceActionsForProvisioningArtifactResponse=$(aws
 servicecatalog list-service-actions-for-provisioning-artifact --product-id
 $productId --provisioning-artifact-id $provisioningArtifactId)
                  serviceActions=$(echo
 "$listServiceActionsForProvisioningArtifactResponse" | jq -r
 '[.ServiceActionSummaries[].Id] | join(",")')
                  if [[ -n "$serviceActions" ]]; then
                    provisioningArtifactServiceActionAssociations
+=("${provisioningArtifactId}:${serviceActions}")
                  fi
                done

                echo "Before deleting a product, the following associated resources
 must be disassociated. These resources will not be deleted. This action may take
 some time, depending on the number of resources being disassociated."

                echo "Portfolios:"
                for portfolioId in $portfolios; do
                  echo "\t${portfolioId}"
                done

                echo "Budgets:"
                if [[ -n "$budgetName" ]]; then
                  echo "\t${budgetName}"
                fi

                echo "Tag Options:"
                for tagOptionId in $tagOptions; do
```

```
                  echo "\t${tagOptionId}"
                done

                echo "Service Actions on Provisioning Artifact:"
                for association in
 "${provisioningArtifactServiceActionAssociations[@]}"; do
                  echo "\t${association}"
                done

                read -p "Are you sure you want to delete ${productId}? y,n "
                if [[ ! $REPLY =~ ^[Yy]$ ]]; then
                   exit
                fi

                for portfolioId in $portfolios; do
                  echo "Disassociating ${portfolioId}"
                  aws servicecatalog disassociate-product-from-portfolio --product-
id $productId --portfolio-id $portfolioId
                done

                if [[ -n "$budgetName" ]]; then
                  echo "Disassociating ${budgetName}"
                  aws servicecatalog disassociate-budget-from-resource --budget-
name "$budgetName" --resource-id $productId
                fi

                for tagOptionId in $tagOptions; do
                  echo "Disassociating ${tagOptionId}"
                  aws servicecatalog disassociate-tag-option-from-resource --tag-
option-id $tagOptionId --resource-id $productId
                done

                for association in
 "${provisioningArtifactServiceActionAssociations[@]}"; do
                  associationPair=(${association//:/ })
                  provisioningArtifactId=${associationPair[0]}
                  serviceActionsList=${associationPair[1]}
                  serviceActionIds=${serviceActionsList//,/ }
                  for serviceActionId in $serviceActionIds; do
                    echo "Disassociating ${serviceActionId} from
 ${provisioningArtifactId}"
                    aws servicecatalog disassociate-service-action-from-
provisioning-artifact --product-id $productId --provisioning-artifact-id
 $provisioningArtifactId --service-action-id $serviceActionId
```

```
    done
  done

  echo "Deleting product ${productId}"
  aws servicecatalog delete-product --id $productId

}; f
```

3. Save the file.

Use the terminal window to add the force-delete-product alias to your `alias` file

1. Open your terminal window and run the following command

   ```
   $ cat >> ~/.aws/cli/alias
   ```

2. Paste the alias script to the terminal window, and then press *CTRL+D* to exit the `cat` command.

Call the force-delete-product alias

1. In your terminal window, run the following command to call the delete product alias

   ```
   $ aws servicecatalog force-delete-product {product-id}
   ```

   The example below shows the `force-delete-product` alias command and its resulting response

   ```
   $ aws servicecatalog force-delete-product prod-123
   ```

   ```
   Before deleting a product, the following associated resources must
be disassociated. These resources will not be deleted. This action may take some
time, depending on the number of resources being disassociated.
   Portfolios:
     port-123
   Budgets:
       budgetName
   ```

```
            Tag Options:
                tag-123
            Service Actions on Provisioning Artifact:
                pa-123:act-123
            Are you sure you want to delete prod-123? y,n
```

2. Enter y to confirm you want to delete the product.

After successfully deleting the product, the terminal window displays the following results

```
        Disassociating port-123
        Disassociating budgetName
        Disassociating tag-123
        Disassociating act-123 from pa-123
        Deleting product prod-123
```

Additional resources

For more information about AWS CLI, using aliases, and deleting AWS Service Catalog products, review the following resources:

- Creating and using AWS CLI aliases in the *AWS Command Line Interface (CLI)* user guide.
- AWS CLI alias repository git repository.
- Deleting AWS Service Catalog products.
- AWS re:Invent 2016: The Effective AWS CLI User on *YouTube*.

Resolving failed resource disassociations when deleting a product

If your prior attempt to delete a product failed due to resource disassociation exceptions, review the list of exceptions and their resolutions below.

> **Note**
>
> If you closed the **Deleting products** window prior to receiving the failed resource disassociation message, you can follow steps one through three in the proceeding *Delete a product* section to open the window again.

To resolve a failed resource disassociation

In the **Delete product** window, review the Associations table **Status** column. Identify the failed resource disassociation exception and the suggested resolutions:

Status exception type	Cause	Resolution
Product prod-****	AWS Service Catalog could not delete the product because the product still has associated TagOptions, budgets, at least one `ProvisioningArtifact` with associated actions, the product is still assigned to a Portfolio, the product has users, or the product has constraints.	Attempt to delete the product again.
User: `username` is not authorized to perform:	The user attempting to delete the product does not have the necessary permissions to disassociate the product's resources.	AWS Service Catalog recommends contacting your account administrator for more information about disassociating product resources you do not currently have permissions to disassociate.

Managing Versions

You assign product versions when you create a product, and you can update product versions any time.

Versions have an AWS CloudFormation template, a title, a description, a status, and guidance.

Version Status

A version can have one of three statuses:

- **Active** - An active version appears in the version list and allows users to launch it.
- **Inactive** - An inactive version is hidden from the version list. Existing provisioned products launched from this version will not be affected.
- **Deleted** - A deleted version is removed from the version list. Deleting a version cannot be undone.

Version Guidance

You can set version guidance to provide information to end users about the product version. Version guidance only affects active product versions.

There are two options for version guidance:

- **None** - By default, product versions do not have any guidance. End users can use that version to update and launch provisioned products.
- **Deprecated** - Users cannot launch new provisioned products using a deprecated product version. If a p provisioned product launched previously uses a now deprecated version, users can only update that provisioned product using the existing version or a new version.

Updating Versions

You assign product versions when creating a product, and you can also update a version any time. For more information about creating a product, see Creating Products.

To update a product version

1. In the AWS Service Catalog console, choose **Products**.
2. From the product list, choose the product you want to update the version of.
3. On the **Product details** page, choose the **Versions** tab, then choose the version you want to update.
4. On the **Version details** page, edit the product version, then choose **Save changes**.

Using AWS Service Catalog Constraints

You apply constraints to control the rules that are applied to a product in a specific portfolio when the end users launches it. When the end users launches the product, they will see the rules you have applied using constraints. You can apply constraints to a product once it is put into a portfolio. Constraints are active as soon as you create them, and they're applied to all current versions of a product that have not been launched.

Constraints

- AWS Service Catalog Launch Constraints
- AWS Service Catalog Notification Constraints
- AWS Service Catalog Tag Update Constraints
- AWS Service Catalog Stack Set Constraints
- AWS Service Catalog Template Constraints

AWS Service Catalog Launch Constraints

A launch constraint specifies the AWS Identity and Access Management (IAM) role that AWS Service Catalog assumes when an end user launches, updates, or terminates a product. An IAM role is a collection of permissions that a user or AWS service can assume temporarily to use AWS services. For an introductory example, see:

- AWS CloudFormation product type: Step 6: Add a launch constraint to assign an IAM role
- Terraform Open Source or Terraform Cloud product type: Step 5: Create launch roles

Launch constraints apply to products in the portfolio (product-portfolio association). Launch constraints do not apply at the portfolio level or to a product across all portfolios. To associate a launch constraint with all products in a portfolio, you must apply the launch constraint to each product individually.

Without a launch constraint, end users must launch and manage products using their own IAM credentials. To do so, they must have permissions for AWS CloudFormation, AWS services that the products use, and AWS Service Catalog. By using a launch role, you can instead limit the end users' permissions to the minimum they require for that product. For more information about end user permissions, see Identity and Access Management in AWS Service Catalog.

To create and assign IAM roles, you must have the following IAM administrative permissions:

- `iam:CreateRole`
- `iam:PutRolePolicy`
- `iam:PassRole`
- `iam:Get*`
- `iam:List*`

Configuring a Launch Role

The IAM role that you assign to a product as a launch constraint must have permissions to use the following:

For Cloudformation products

- The `arn:aws:iam::aws:policy/AWSCloudFormationFullAccess` AWS CloudFormation managed policy
- Services in the AWS CloudFormation template for the product
- Read access to the AWS CloudFormation template in a service-owned Amazon S3 bucket.

For Terraform products

- Services in the Amazon S3 template for the product
- Read access to the Amazon S3 template in a service-owned Amazon S3 bucket.
- `resource-groups:Tag`for tagging in an Amazon EC2 instance (assumed by the Terraform provisioning engine when performing provisioning operations)
- `resource-groups:CreateGroup` for resource group tagging (assumed by AWS Service Catalog to create resource groups and assign tags)

The IAM role's trust policy must allow AWS Service Catalog to assume the role. In the procedure below, the trust policy will be set automatically when you select AWS Service Catalog as the role type. If you are not using the console, see the section *Creating trust policies for AWS services that assume roles* in How to use trust policies with IAM roles.

Note

The `servicecatalog:ProvisionProduct`, `servicecatalog:TerminateProvisionedProduct`, and `servicecatalog:UpdateProvisionedProduct` permissions cannot be assigned in a launch role. You must use IAM roles, as shown in the inline policy steps in the section Grant Permissions to AWS Service Catalog End Users.

Note

To view provisioned Cloudformation products and resources in the AWS Service Catalog console, end users need AWS CloudFormation read access. Viewing provisioned products and resources in the console does **not** use the launch role.

To create a launch role

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

   Terraform products require additional launch role configurations. For more information, review Step 5: Create launch roles in *Getting Started with a Terraform Open Source product*.
2. Choose **Roles**.
3. Choose **Create New Role**.
4. Enter a role name and choose **Next Step**.
5. Under **AWS Service Roles** next to **AWS Service Catalog**, choose **Select**.
6. On the **Attach Policy** page, Choose **Next Step**.
7. To create the role, choose **Create Role**.

To attach a policy to the new role

1. Choose the role that you created to view the role details page.
2. Choose the **Permissions** tab, and expand the **Inline Policies** section. Then, choose **click here**.
3. Choose **Custom Policy**, and then choose **Select**.
4. Enter a name for the policy, and then paste the following into the **Policy Document** editor:

```
          "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject"
         ],
         "Resource":"*",
         "Condition":{
            "StringEquals":{
               "s3:ExistingObjectTag/servicecatalog:provisioning":"true"
            }
         }
   ]
}
```

> **Note**
>
> When you configure a launch role for a launch constraint, you must use this string: `"s3:ExistingObjectTag/servicecatalog:provisioning":"true"`.

5. Add a line to the policy for each additional service the product uses. For example, to add permission for Amazon Relational Database Service (Amazon RDS), enter a comma at the end of the last line in the `Action` list, and then add the following line:

```
"rds:*"
```

6. Choose **Apply Policy**.

Applying a Launch Constraint

After you configure the launch role, assign the role to the product as a launch constraint. This action tells AWS Service Catalog to assume the role when an end user launches the product.

To assign the role to a product

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. Choose the portfolio that contains the product.

3. Choose the **Constraints** tab and choose **Create constraint**.
4. Choose the product from **Product** and choose **Launch** under **Constraint type**. Choose **Continue**.
5. In the **Launch constraint** section, you can select an IAM role from your account and enter an IAM role ARN, or enter the role name.

   If you specify the role name and if an account uses the launch constraint, the account uses that name for the IAM role. This approach allows launch-role constraints to be account-agnostic so you can create fewer resources per shared account.

   > **Note**
   >
   > The given role name must exist in the account that created the launch constraint and the account of the user who launches a product with this launch constraint.

6. After specifying the IAM role, choose **Create**.

Adding Confused Deputy to Launch Constraint

AWS Service Catalog supports Confused Deputy protection for the APIs that run with an Assume Role request. When you add a launch constraint, you can restrict the launch role access by using `sourceAccount` and `sourceArn` conditions in the launch role trust policy. It ensures that the launch role is called by a trusted source.

In the following example, the AWS Service Catalog end-user belongs to account 111111111111. When the AWS Service Catalog administrator creates a `LaunchConstraint` for a product, the end-user can specify the following conditions in the launch role trust policy to restrict the assume role to account 111111111111.

```
"Condition":{
   "ArnLike":{
      "aws:SourceArn":"arn:aws:servicecatalog:us-east-1:111111111111:*"
   },
   "StringEquals":{
      "aws:SourceAccount":"111111111111"
   }

}
```

A user who provisions a product with the `LaunchConstraint` must have the same `AccountId` (111111111111). If not, the operation fails with an `AccessDenied` error, preventing launch role misuse.

The following AWS Service Catalog APIs are secured for Confused Deputy protection:

- `LaunchConstraint`
- `ProvisionProduct`
- `UpdateProvisionedProduct`
- `TerminateProvisionedProduct`
- `ExecuteProvisionedProductServiceAction`
- `CreateProvisionedProductPlan`
- `ExecuteProvisionedProductPlan`

The `sourceArn` protection for AWS Service Catalog only supports templated ARNs, such as "`arn:<aws-partition>:servicecatalog:<region>:<accountId>:`" It does not support specific resource ARNs.

Verifying the Launch Constraint

To verify AWS Service Catalog uses the role to launch the product and successfully provisions the product, launch the product from the AWS Service Catalog console. To test a constraint prior to releasing it to users, create a test portfolio that contains the same products and test the constraints with that portfolio.

To launch the product

1. In the menu for the AWS Service Catalog console, choose **Service Catalog**, **End user**.
2. Choose the product to open the **Product details** page. In the **Launch options** table, verify the Amazon Resource Name (ARN) of the role appears.
3. Choose **Launch product**.
4. Proceed through the launch steps, filling in any required information.
5. Verify that the product starts successfully.

AWS Service Catalog Notification Constraints

> **Note**
>
> AWS Service Catalog does not support notification constraints for Terraform Open Source or Terraform Cloud products.

A notification constraint specifies an Amazon SNS topic to receive notifications about stack events.

Use the following procedure to create an SNS topic and subscribe to it.

To create an SNS topic and a subscription

1. Open the Amazon SNS console at https://console.aws.amazon.com/sns/v3/home.
2. Choose **Create topic**.
3. Type a topic name and then choose **Create topic**.
4. Choose **Create subscription**.
5. For **Protocol**, select **Email**. For **Endpoint**, type an email address that you can use to receive notifications. Choose **Create subscription**.
6. You'll receive a confirmation email with the subject line `AWS Notification - Subscription Confirmation`. Open the email and follow the directions to complete your subscription.

Use the following procedure to apply a notification constraint using the SNS topic that you created using the previous procedure.

To apply a notification constraint to a product

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. Choose the portfolio that contains the product.
3. Expand **Constraints** and choose **Add constraints**.
4. Choose the product from **Product** and set **Constraint type** to **Notification**. Choose **Continue**.
5. Choose **Choose a topic from your account** and select the SNS topic that you created from **Topic Name**.
6. Choose **Submit**.

AWS Service Catalog Tag Update Constraints

> **Note**
>
> AWS Service Catalog does not support tag update constraints for Terraform Open Source products.

With tag update constraints, AWS Service Catalog administrators can allow or disallow end users to update tags on resources associated with a provisioned product. If tag updating is allowed, then new tags associated with the product or portfolio will be applied to provisioned resources during a provisioned product update.

To enable tag updates to a product

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. Choose the portfolio that contains the product you want to update.
3. Choose the **Constraints** tab and choose **Add constraints**.
4. Under **Constraint type**, choose **Tag Update**.
5. Choose the product from **Product**, then choose **Continue**.
6. On the **Tag Updates page**, select **Enable Tag Updates**.
7. Choose **Submit**.

AWS Service Catalog Stack Set Constraints

> **Note**
>
> - AWS Service Catalog does not support stack set constraints for Terraform Open Source products.
> - AutoTags are not currently supported with AWS CloudFormation StackSets.

A stack set constraint allows you to configure product deployment options using AWS CloudFormation StackSets. You can specify multiple accounts and regions for the product launch.

End users can manage those accounts and determine where products deploy and the order of deployment.

To apply a stack set constraint to a product

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. Choose the portfolio with the product you want.
3. Choose the **Constraints** tab and then choose **Create constraints**.
4. In **Product**, choose the product. In **Constraint type**, choose **Stack Set**.
5. Configure the accounts, regions, and permissions for your stack set constraints.
   - In **Account settings**, identify the accounts where you want to create products.
   - In **Region settings**, choose the geographic regions to deploy products and the order you want those products to be deployed in those regions.
   - In **Permissions**, choose an IAM StackSet Administrator Role to manage your target accounts. If you don't choose a role, StackSets uses the default ARN. Learn more about setting up stack set permissions.
6. Choose **Create**.

AWS Service Catalog Template Constraints

> **Note**
>
> AWS Service Catalog does not support template constraints for Terraform Open Source or Terraform Cloud products.

To limit the options that are available to end users when they launch a product, you apply template constraints. Apply template constraints to ensure that the end users can use products without breaching the compliance requirements of your organization. You apply template constraints to a product in a AWS Service Catalog portfolio. A portfolio must contain one or more products before you can define template constraints.

A template constraint consists of one or more rules that narrow the allowable values for parameters that are defined in the product's underlying AWS CloudFormation template. The parameters in an AWS CloudFormation template define the set of values that users can specify

when creating a stack. For example, a parameter might define the various instance types that users can choose from when launching a stack that includes EC2 instances.

If the set of parameter values in a template is too broad for the target audience of your portfolio, you can define template constraints to limit the values that users can choose when launching a product. For example, if the template parameters include EC2 instance types that are too large for users who should use only small instance types (such as `t2.micro` or `t2.small`), then you can add a template constraint to limit the instance types that end users can choose. For more information about AWS CloudFormation template parameters, see Parameters in the *AWS CloudFormation User Guide*.

Template constraints are bound within a portfolio. If you apply template constraints to a product in one portfolio, and if you then include the product in another portfolio, the constraints will not apply to the product in the second portfolio.

If you apply a template constraint to a product that has already been shared with users, the constraint is active immediately for all subsequent product launches and for all versions of the product in the portfolio.

You define template constraint rules by using a rule editor or by writing the rules as JSON text in the AWS Service Catalog administrator console. For more information about rules, including syntax and examples, see Template Constraint Rules.

To test a constraint prior to releasing it to users, create a test portfolio that contains the same products and test the constraints with that portfolio.

To apply template constraints to a product

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. On the **Portfolios** page, choose the portfolio that contains the product to which you want to apply a template constraint.
3. Expand the **Constraints** section and choose **Add constraints**.
4. In the **Select product and type** window, for **Product** choose the product for which you want to define the template constraints. Then, for **Constraint type**, choose **Template**. Choose **Continue**.
5. On the **Template constraint builder** page, edit the constraint rules by using the JSON editor or the rule builder interface.

- To edit the JSON code for the rule, choose the **Constraint Text Editor** tab. Several samples are provided on this tab to help you get started.

  To build the rules by using a rule builder interface, choose the **Rule Builder** tab. On this tab, you can choose any parameter that is specified in the template for the product, and you can specify the allowable values for that parameter. Depending on the type of parameter, you specify the allowable values by choosing items in a checklist, by specifying a number, or by specifying a set of values in a comma-separated list.

  When you have finished building a rule, choose **Add rule**. The rule appears in the table on the **Rule Builder** tab. To review and edit the JSON output, choose the **Constraint Text Editor** tab.

6. When you are done editing the rules for your constraint, choose **Submit**. To see the constraint, go to the portfolio details page and expand **Constraints**.

Template Constraint Rules

The rules that define template constraints in a AWS Service Catalog portfolio describe when end users can use the template and which values they can specify for parameters that are declared in the AWS CloudFormation template used to create the product they are attempting to use. Rules are useful for preventing end users from inadvertently specifying an incorrect value. For example, you can add a rule to verify whether end users specified a valid subnet in a given VPC or used `m1.small` instance types for test environments. AWS CloudFormation uses rules to validate parameter values before it creates the resources for the product.

Each rule consists of two properties: a rule condition (optional) and assertions (required). The rule condition determines when a rule takes effect. The assertions describe what values users can specify for a particular parameter. If you don't define a rule condition, the rule's assertions always take effect. To define a rule condition and assertions, you use *rule-specific intrinsic functions*, which are functions that can only be used in the `Rules` section of a template. You can nest functions, but the final result of a rule condition or assertion must be either true or false.

As an example, assume that you declared a VPC and a subnet parameter in the `Parameters` section. You can create a rule that validates that a given subnet is in a particular VPC. So when a user specifies a VPC, AWS CloudFormation evaluates the assertion to check whether the subnet parameter value is in that VPC before creating or updating the stack. If the parameter value is

invalid, AWS CloudFormation immediately fail to create or update the stack. If users don't specify a VPC, AWS CloudFormation doesn't check the subnet parameter value.

Syntax

The `Rules` section of a template consists of the key name `Rules`, followed by a single colon. Braces enclose all rule declarations. If you declare multiple rules, they are delimited by commas. For each rule, you declare a logical name in quotation marks followed by a colon and braces that enclose the rule condition and assertions.

A rule can include a `RuleCondition` property and must include an `Assertions` property. For each rule, you can define only one rule condition; you can define one or more asserts within the `Assertions` property. You define a rule condition and assertions by using rule-specific intrinsic functions, as shown in the following pseudo template:

```
"Rules":{
   "Rule01":{
      "RuleCondition":{
         "Rule-specific intrinsic function"
      },
      "Assertions":[
         {
            "Assert":{
               "Rule-specific intrinsic function"
            },
            "AssertDescription":"Information about this assert"
         },
         {
            "Assert":{
               "Rule-specific intrinsic function"
            },
            "AssertDescription":"Information about this assert"
         }
      ]
   },
   "Rule02":{
      "Assertions":[
         {
            "Assert":{
               "Rule-specific intrinsic function"
            },
            "AssertDescription":"Information about this assert"
```

```
        }
      ]
   }
}
```

The pseudo template shows a `Rules` section containing two rules named `Rule01` and `Rule02`. `Rule01` includes a rule condition and two assertions. If the function in the rule condition evaluates to true, both functions in each assert are evaluated and applied. If the rule condition is false, the rule doesn't take effect. `Rule02` always takes effect because it doesn't have a rule condition, which means the one assert is always evaluated and applied.

For information on rule-specific intrinsic functions to define rule conditions and assertions, see AWS Rule Functions in the *AWS CloudFormation User Guide*.

Example: Conditionally Verify a Parameter Value

The following two rules check the value of the `InstanceType` parameter. Depending on the value of the Environment parameter (`test` or `prod`), the user must specify `m1.small` or `m1.large` for the `InstanceType` parameter. The `InstanceType` and `Environment` parameters must be declared in the `Parameters` section of the same template.

```
"Rules" : {
  "testInstanceType" : {
    "RuleCondition" : {"Fn::Equals":[{"Ref":"Environment"}, "test"]},
    "Assertions" : [
      {
        "Assert" :  { "Fn::Contains" : [ ["m1.small"], {"Ref" : "InstanceType"} ] },
        "AssertDescription" : "For the test environment, the instance type must be
 m1.small"
      }
    ]
  },
  "prodInstanceType" : {
    "RuleCondition" : {"Fn::Equals":[{"Ref":"Environment"}, "prod"]},
    "Assertions" : [
      {
        "Assert" :  { "Fn::Contains" : [ ["m1.large"], {"Ref" : "InstanceType"} ] },
        "AssertDescription" : "For the prod environment, the instance type must be
 m1.large"
      }
    ]
  }
```

```
}
```

AWS Service Catalog Service Actions

> **Note**
>
> AWS Service Catalog does not support service actions for Terraform Open Source or Terraform Cloud products.

AWS Service Catalog enables you to reduce administrative maintenance and end user training while adhering to compliance and security measures. With service actions, as the administrator you can enable end users to perform operational tasks, troubleshoot issues, run approved commands, or request permissions in AWS Service Catalog. You use AWS Systems Manager documents to define service actions. The AWS Systems Manager documents provide access to pre-defined actions that implement AWS best practices, such as Amazon EC2 stop and reboot, and you can define custom actions too.

In this tutorial, you provide end users with the ability to restart an Amazon EC2 instance. You add the necessary permissions, define the service action, associate the service action with a product, and test the end user experience using the action with a provisioned product.

Prerequisites

This tutorial assumes that you have full AWS administrator permissions, you are already familiar with AWS Service Catalog, and that you already have a base set of products, portfolios, and users. If you are not familiar with AWS Service Catalog, complete the Setting Up and Getting Started tasks before using this tutorial.

Topics

- Step 1: Configure end user permissions
- Step 2: Create a service action
- Step 3: Associate the service action with a product version
- Step 4: Test the end user experience
- Step 5: Managing service actions with AWS CloudFormation
- Step 6: Troubleshooting

Step 1: Configure end user permissions

End users must have the necessary permissions to view and perform specific service actions. In this example, the end user needs permission to access the AWS Service Catalog service actions feature and to perform an Amazon EC2 restart.

To update permissions

1. Open the AWS Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.
2. From the menu, locate user groups.
3. Choose the groups that end users will use to access AWS Service Catalog resources. In this example, we select the end user group. In your own implementation, choose the group that is used by the relevant end users.
4. On the **Permissions** tab of your group's detail page, you either create a new policy or edit an existing policy. In this example, we add permissions to the existing policy by selecting the custom policy created for the group's AWS Service Catalog Provision and Terminate permissions.
5. On the **Policy** page, choose **Edit Policy** to add the necessary permissions. You can use either the visual editor or the JSON editor to edit the policy. In this example, we use the JSON editor to add the permissions. For this tutorial, add the following permissions to the policy:

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "Stmt1536341175150",
   "Action": [
    "servicecatalog:ListServiceActionsForProvisioningArtifact",
    "servicecatalog:ExecuteprovisionedProductServiceAction",
    "ssm:DescribeDocument",
    "ssm:GetAutomationExecution",
    "ssm:StartAutomationExecution",
    "ssm:StopAutomationExecution",
    "cloudformation:ListStackResources",
    "ec2:DescribeInstanceStatus",
    "ec2:StartInstances",
    "ec2:StopInstances"
```

```
    ],
    "Effect": "Allow",
    "Resource": "*"
   }
  ]
}
```

6. After you edit the policy, review and approve the change to the policy. Users in the end user group now have the necessary permissions to perform the Amazon EC2 restart action in AWS Service Catalog.

Step 2: Create a service action

Next, you create a service action to restart Amazon EC2 instances.

1. Open the AWS Service Catalog console at https://console.aws.amazon.com/sc/.
2. From the menu, choose **Service actions**.
3. On the **Service actions** page, choose **Create action**.
4. On the **Create action** page, choose an AWS Systems Manager document to define the service action. The Amazon EC2 Instance Restart action is defined by an AWS Systems Manager document, so we keep the default option on the drop-down menu, **Amazon documents**.
5. Search for and choose the **AWS-RestartEC2Instance** action.
6. Provide a name and description for the action that make sense for your environment and team. The end user will see this description, so choose something that helps them understand what the action does.
7. Under **Parameter and target configuration**, choose the SSM document parameter that will be the target of the action (for example, the **Instance ID**), and choose the target of the parameter. Choose **Add parameter** to add additional parameters.
8. Under **Permissions**, choose a role. We are using default permissions for this example. Other permission configurations are possible and are defined on this page.
9. After you have reviewed the configuration, choose **Create action**.
10. On the next page, a confirmation appears when the action has been created and is ready to use.

Step 3: Associate the service action with a product version

After you define an action, you must associate a product with that action.

1. On the **Service actions** page, choose **AWS-RestartEC2instance**, and then choose **Associate action**.
2. On the **Associate action** page, choose the product that you want your end users to take the service action on. In this example, we choose **Linux Desktop**.
3. Select a product version. Note that you can use the topmost check box to select all versions.
4. Choose **Associate action**.
5. On the next page, a confirmation message appears.

You have now created the service action in AWS Service Catalog. The next step of this tutorial is to use the service action as an end user.

Step 4: Test the end user experience

End users can perform service actions on provisioned products. For the purposes of this tutorial, the end user must have at least one provisioned product. The provisioned product should be launched from the product version that you associated with the service action in the previous step.

To access the service action as an end user

1. Log in to the AWS Service Catalog console as an end user.
2. On the AWS Service Catalog dashboard, in the navigation pane, choose **Provisioned products list**. The list shows the products that are provisioned for the end user's account.
3. On the **Provisioned products list** page, choose the instance that is provisioned.
4. On the **Provisioned product details** page, choose **Actions** in the upper right side, and then choose the **AWS-RestartEC2instance** action.
5. Confirm that you want to execute the custom action. You receive confirmation that the action has been sent.

Step 5: Managing service actions with AWS CloudFormation

You can create service actions and their associations with AWS CloudFormation resources. For more information, see the following in the *AWS CloudFormation User Guide*:

- AWS::ServiceCatalog::CloudFormationProduct ProvisioningArtifactProperties
- AWS::ServiceCatalog::ServiceActionAssociation

> **Note**
>
> If you manage service action associations with AWS CloudFormation resources, don't add or remove service actions through the AWS Command Line Interface or AWS Management Console. When you perform a stack update, any changes to sevice actions that are made outside of AWS CloudFormation are replaced.

Step 6: Troubleshooting

If your service action execution fails, you can find the error message in the **Outputs** section of the service action execution event on the **Provisioned product** page. Below you can see explanations for common error messages you may find.

> **Note**
>
> The exact text of the error message is subject to change, so you should avoid using these in any kind of automated process.

Internal failure

AWS Service Catalog experienced an internal error. Try again later. If the issue persists, contact customer support.

An error occurred (ThrottlingException) when calling the StartAutomationExecution operation

The service action execution was throttled by the backend service, such as SSM.

Access denied while assuming the role

AWS Service Catalog was unable to assume the role specified in the service action definition. Make sure that the *servicecatalog.amazonaws.com* principal, or a regional principal such as *servicecatalog.us-east-1.amazonaws.com*, is allowlisted in the role's trust policy.

An error occurred (AccessDeniedException) when calling the StartAutomationExecution operation: User is not authorized to perform: ssm:StartAutomationExecution on the resource.

The role specified in the service action definition does not have permissions to invoke ssm:StartAutomationExecution. Make sure the role has the appropriate SSM permissions.

Cannot find any resources with type *TargetType* in provisioned product

The provisioned product does not contain any resources that match the target type specified in the SSM document, such as AWS::EC2::Instance. Check your provisioned product for these resources or confirm the document is correct.

Document with that name does not exist

The document specified in the service action definition does not exist.

Failed to describe SSM Automation document

AWS Service Catalog encountered an unknown exception from SSM when trying to describe the specified document.

Failed to retrieve credentials for role

AWS Service Catalog encountered an unknown error when assuming the specified role.

Parameter has value "*InvalidValue*" not found in *{ValidValue1}, {ValidValue2}*

The parameter value passed to SSM is not in the allowed values list for the document. Confirm the parameters provided are valid, and try again.

Parameter type error. The value supplied for *ParameterName* is not a valid string.

The value of the parameter passed to SSM is not valid for the type on the document.

Parameter is not defined in service action definition

A parameter was passed to AWS Service Catalog that is not defined in the service action definition. You can only use parameters defined in the service action definition.

Step fails when it is executing/canceling action. *Error message.* Please refer to Automation Service Troubleshooting Guide for more diagnosis details.

A step in the SSM automation document failed. See the error in the message to troubleshoot further.

The following values for the parameter are not allowed because they are not in the provisioned product: *InvalidResourceId*

The user requested action on a resource that is not in the provisioned product.

TargetType not defined for SSM Automation document

Service actions require SSM automation documents to have a TargetType defined. Check your SSM automation document.

Adding AWS Marketplace Products to Your Portfolio

You can add AWS Marketplace products to your portfolios to make those products available to your AWS Service Catalog end users.

AWS Marketplace is an online store in which you can find, subscribe to, and immediately start using a large selection of software and services. The types of products in AWS Marketplace include databases, application servers, testing tools, monitoring tools, content management tools, and business intelligence software. AWS Marketplace is available at https://aws.amazon.com/marketplace. Note that you can't add software as a service (SaaS) products from AWS Marketplace to AWS Service Catalog.

You distribute an AWS Marketplace product to AWS Service Catalog end users by copying the product with the AWS CloudFormation template to AWS Service Catalog, and then adding the product to a portfolio.

> **Note**
>
> AWS Service Catalog does not support distributing AWS Marketplace products to AWS Service Catalog end users using a Terraform Open Source or Terraform Cloud product template.

AWS Marketplace supports AWS Service Catalog directly or subscribe and add products using the manual option. We recommend adding products using the functionality specifically designed for AWS Service Catalog.

Managing AWS Marketplace Products Using AWS Service Catalog

You can add your subscribed AWS Marketplace products directly to AWS Service Catalog using a custom interface. In AWS Marketplace, choose **Service Catalog**. For more information, see Copying Products to AWS Service Catalog in the *AWS Marketplace Help and FAQ*.

Managing and Adding AWS Marketplace Products Manually

Complete the following steps to subscribe to an AWS Marketplace product, define that product in an AWS CloudFormation template, and add the template to a AWS Service Catalog portfolio.

To subscribe to an AWS Marketplace product

1. Go to AWS Marketplace at https://aws.amazon.com/marketplace.
2. Browse the products or search to find the product that you want to add to your AWS Service Catalog portfolio. Choose the product to view the product details page.
3. Choose **Continue** to view the fulfillment page, and then choose the **Manual Launch** tab.

   The information on the fulfillment page includes the supported Amazon Elastic Compute Cloud (Amazon EC2) instance types, the supported AWS Regions, and the Amazon Machine Image (AMI) ID that the product uses for each AWS region. Note that some choices will affect cost. You will use this information to customize the AWS CloudFormation template in later steps.
4. Choose **Accept Terms** to subscribe to the product.

   After you subscribe to a product, you can access the information on the product fulfillment page in AWS Marketplace at any time by choosing **Your Software**, and then choosing the product.

To define your AWS Marketplace product in an AWS CloudFormation template

To complete the following steps, you will use one of the AWS CloudFormation sample templates as a starting point, and you will customize the template so that it represents your AWS Marketplace product. To access the sample templates, see Sample Templates in the *AWS CloudFormation User Guide*.

1. On the Sample Templates page in the *AWS CloudFormation User Guide*, choose an AWS Region for your product. The AWS Region must be supported by your AWS Marketplace product. You can view the supported regions on the product fulfillment page in AWS Marketplace.
2. To view a list of service sample templates that are appropriate for the Region, choose the **Services** link.
3. You can use any of the samples that are appropriate for your needs as a starting point. The steps in this procedure use the **Amazon EC2 instance in a security group** template. To view

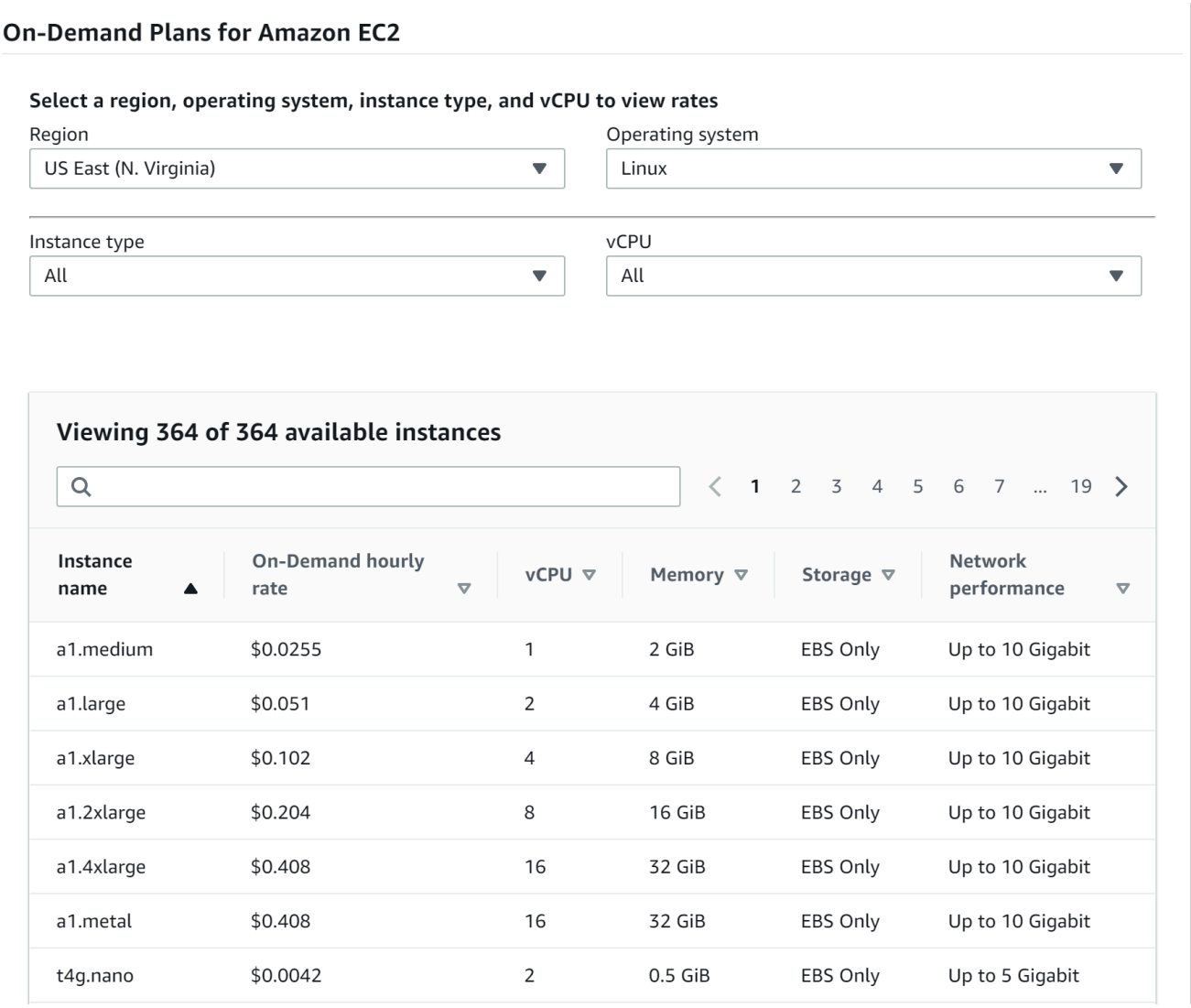
the sample template, choose **View** , and then save a copy of the template locally so that you can edit it. Your local file must have the `.template` extension.

4. Open your template file in a text editor.
5. Customize the description at the top of the template. Your description might look like the following example:

   ```
   "Description": "Launches a LAMP stack from AWS Marketplace",
   ```

6. Customize the `InstanceType` parameter so that it includes only EC2 instance types that are supported by your product. If your template includes unsupported EC2 instance types, the product will fail to launch for your end users.

   a. On the product fulfillment page in AWS Marketplace, view the supported EC2 instance types in the **Pricing Details** section.



On-Demand Plans for Amazon EC2

Select a region, operating system, instance type, and vCPU to view rates

Region: US East (N. Virginia)

Operating system: Linux

Instance type: All

vCPU: All

Viewing 364 of 364 available instances

1 2 3 4 5 6 7 ... 19

Instance name	On-Demand hourly rate	vCPU	Memory	Storage	Network performance
a1.medium	$0.0255	1	2 GiB	EBS Only	Up to 10 Gigabit
a1.large	$0.051	2	4 GiB	EBS Only	Up to 10 Gigabit
a1.xlarge	$0.102	4	8 GiB	EBS Only	Up to 10 Gigabit
a1.2xlarge	$0.204	8	16 GiB	EBS Only	Up to 10 Gigabit
a1.4xlarge	$0.408	16	32 GiB	EBS Only	Up to 10 Gigabit
a1.metal	$0.408	16	32 GiB	EBS Only	Up to 10 Gigabit
t4g.nano	$0.0042	2	0.5 GiB	EBS Only	Up to 5 Gigabit

b. In your template, change the default instance type to a supported EC2 instance type of your choice.

c. Edit the `AllowedValues` list so that it includes only EC2 instance types that are supported by your product.

d. Remove any EC2 instance types that you do not want your end users to use when they launch the product from the `AllowedValues`list .

When you are done editing the `InstanceType` parameter, it might look similar to the following example:

```
    "InstanceType" : {
      "Description" : "EC2 instance type",
      "Type" : "String",
      "Default" : "m1.small",
      "AllowedValues" : [ "t1.micro", "m1.small", "m1.medium", "m1.large",
"m1.xlarge", "m2.xlarge", "m2.2xlarge", "m2.4xlarge", "c1.medium", "c1.xlarge",
"c3.large", "c3.large", "c3.xlarge", "c3.xlarge", "c3.4xlarge", "c3.8xlarge" ],
      "ConstraintDescription" : "Must be a valid EC2 instance type."
    },
```

7. In the `Mappings` section of your template, edit the `AWSInstanceType2Arch` mappings so that only supported EC2 instance types and architectures are included.

a. Edit the list of mappings by removing all EC2 instance types that are not included in the `AllowedValues` list for the `InstanceType` parameter.

b. Edit the `Arch` value for each EC2 instance type to be the architecture type that is supported by your product. Valid values are `PV64`, `HVM64`, and `HVMG2`. To learn which architecture your product supports, refer to the product details page in AWS Marketplace. To learn which architectures are supported by EC2 instance families, see Amazon Linux AMI Instance Type Matrix.

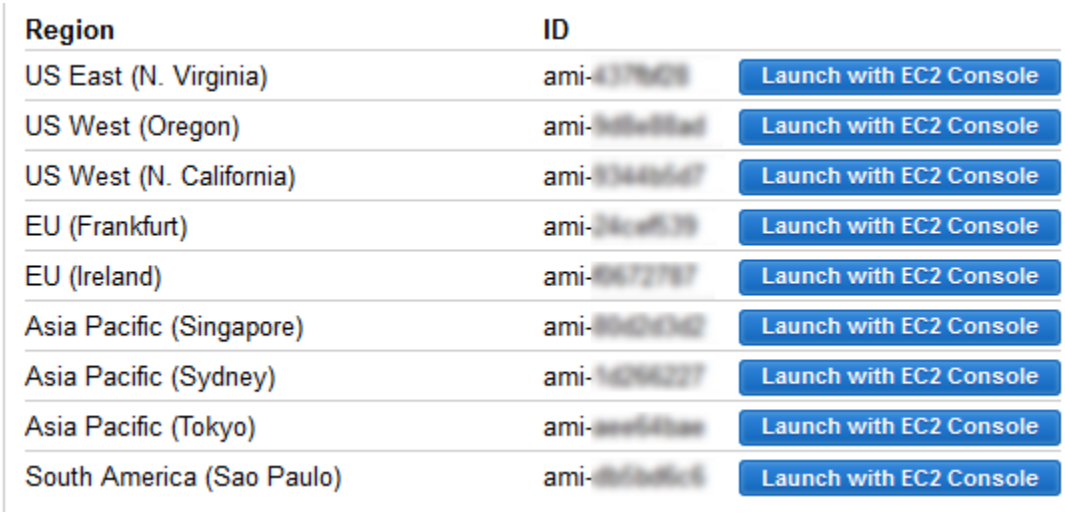
When you have finished editing the `AWSInstanceType2Arch` mappings, it might look similar to the following example:

```
    "AWSInstanceType2Arch" : {
      "t1.micro"    : { "Arch" : "PV64"  },
      "m1.small"    : { "Arch" : "PV64"  },
      "m1.medium"   : { "Arch" : "PV64"  },
```

```
      "m1.large"    : { "Arch" : "PV64"  },
      "m1.xlarge"   : { "Arch" : "PV64"  },
      "m2.xlarge"   : { "Arch" : "PV64"  },
      "m2.2xlarge"  : { "Arch" : "PV64"  },
      "m2.4xlarge"  : { "Arch" : "PV64"  },
      "c1.medium"   : { "Arch" : "PV64"  },
      "c1.xlarge"   : { "Arch" : "PV64"  },
      "c3.large"    : { "Arch" : "PV64"  },
      "c3.xlarge"   : { "Arch" : "PV64"  },
      "c3.2xlarge"  : { "Arch" : "PV64"  },
      "c3.4xlarge"  : { "Arch" : "PV64"  },
      "c3.8xlarge"  : { "Arch" : "PV64"  }
    }
,
```

8. In the `Mappings` section of your template, edit the `AWSRegionArch2AMI` mappings to associate each AWS Region with the corresponding architecture and AMI ID for your product.

   a. On the product fulfillment page in AWS Marketplace, view the AMI ID that your product uses for each AWS Region, as in the following example:



Region	ID	
US East (N. Virginia)	ami-	Launch with EC2 Console
US West (Oregon)	ami-	Launch with EC2 Console
US West (N. California)	ami-	Launch with EC2 Console
EU (Frankfurt)	ami-	Launch with EC2 Console
EU (Ireland)	ami-	Launch with EC2 Console
Asia Pacific (Singapore)	ami-	Launch with EC2 Console
Asia Pacific (Sydney)	ami-	Launch with EC2 Console
Asia Pacific (Tokyo)	ami-	Launch with EC2 Console
South America (Sao Paulo)	ami-	Launch with EC2 Console

   b. In your template, remove the mappings for any AWS Regions that you do not support.

   c. Edit the mapping for each region to remove the unsupported architectures (`PV64`, `HVM64`, or `HVMG2`) and their associated AMI IDs.

   d. For each remaining AWS Region and architecture mapping, specify the corresponding AMI ID from the product details page in AWS Marketplace.

When you have finished editing the `AWSRegionArch2AMI` mappings, your code might look similar to the following example:

```
"AWSRegionArch2AMI" : {
  "us-east-1"        : {"PV64" : "ami-nnnnnnnn"},
  "us-west-2"        : {"PV64" : "ami-nnnnnnnn"},
  "us-west-1"        : {"PV64" : "ami-nnnnnnnn"},
  "eu-west-1"        : {"PV64" : "ami-nnnnnnnn"},
  "eu-central-1"     : {"PV64" : "ami-nnnnnnnn"},
  "ap-northeast-1"   : {"PV64" : "ami-nnnnnnnn"},
  "ap-southeast-1"   : {"PV64" : "ami-nnnnnnnn"},
  "ap-southeast-2"   : {"PV64" : "ami-nnnnnnnn"},
  "sa-east-1"        : {"PV64" : "ami-nnnnnnnn"}
}
```

You can now use the template to add the product to a AWS Service Catalog portfolio. If you want to make additional changes, see Working with AWS CloudFormation Templates to learn more about templates.

To add your AWS Marketplace product to a AWS Service Catalog portfolio

1. Sign in to the AWS Management Console and navigate to the AWS Service Catalog administrator console at https://console.aws.amazon.com/servicecatalog/.
2. On the **Portfolios** page, choose the portfolio to which you want to add your AWS Marketplace product.
3. On the portfolio details page, choose **Upload new product**.
4. Type the requested product and support details.
5. On the **Version details** page, choose **Upload a template file**, choose **Browse**, and then choose your template file.
6. Type a version title and description.
7. Choose **Next**.
8. On the **Review** page, verify that the summary is accurate, and then choose **Confirm and upload**. The product is added your portfolio. It is now available to end users who have access to the portfolio.

Using AWS CloudFormation StackSets

> **Note**
>
> AutoTags are not currently supported with AWS CloudFormation StackSets.

You can use AWS CloudFormation StackSets to launch AWS Service Catalog products across multiple AWS Regions and accounts. You can specify the order in which products deploy sequentially within AWS Regions. Across accounts, products are deployed in parallel. When launching, users can specify failure tolerance and the maximum number of accounts in which to deploy in parallel. For more information, see Working with AWS CloudFormation StackSets.

Stack sets vs. stack instances

A *stack set* lets you create stacks in AWS accounts across AWS Regions by using a single AWS CloudFormation template.

A *stack instance* refers to a stack in a target account within an AWS Region and is associated with only one stack set.

For more information, see StackSets Concepts.

Stack set constraints

In AWS Service Catalog, you can use stack set constraints to configure product deployment options.

AWS Service Catalog supports stack set constraints on products in two AWS GovCloud (US) Regions: AWS GovCloud (US-West) and AWS GovCloud (US-East).

For more information, see AWS Service Catalog Stack Set Constraints.

Managing Budgets

You can use AWS Budgets to track your service costs and usage within AWS Service Catalog. You can associate budgets with AWS Service Catalog products and portfolios.

> **Note**
>
> AWS Service Catalog does not support budgets for Terraform Open Source products.

AWS Budgets gives you the ability to set custom budgets that alert you when your costs or usage exceed (or are forecasted to exceed) your budgeted amount. Information about AWS Budgets is available at https://aws.amazon.com/aws-cost-management/aws-budgets.

Tasks

- Prerequisites
- Creating a budget
- Associating a Budget
- Viewing a Budget
- Disassociating a Budget

Prerequisites

Before using AWS Budgets, you need to activate cost allocation tags in the AWS Billing and Cost Management console. For more information, see Activating User-Defined Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

> **Note**
>
> Tags take up to 24 hours to activate.

You also need to enable user access to the AWS Billing and Cost Management console for any users or groups who will be using the Budgets feature. You can do this by creating a new policy for your users.

To allow users to create budgets, you must also allow users to view billing information. If you want to use Amazon SNS notifications, you can give users the ability to create Amazon SNS notifications, as shown in the policy example below.

To create the budgets policy

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. In the content pane, choose **Create policy**.
4. Choose the **JSON** tab and copy the text from the following JSON policy document. Paste this text into the **JSON** text box.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1435216493000",
            "Effect": "Allow",
            "Action": [
                "aws-portal:ViewBilling",
                "aws-portal:ModifyBilling",
                "budgets:ViewBudget",
                "budgets:ModifyBudget"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "Stmt1435216552000",
            "Effect": "Allow",
            "Action": [
                "sns:*"
            ],
            "Resource": [
                "arn:aws:sns:us-east-1"
            ]
        }
    ]
}
```

5. When you are finished, choose **Review policy**. The Policy Validator reports any syntax errors.
6. On the **Review** page, give your policy a name. Review the policy **Summary** to see the permissions granted by your policy, and then choose **Create policy** to save your work.

   The new policy appears in the list of managed policies and is ready to attach to your users and groups. For more information, see Create and Attach Customer Managed Policy in the *AWS Identity and Access Management User Guide*.

Creating a budget

In the AWS Service Catalog administrator console, the **Product list** and **Portfolios** pages list information about existing products and portfolios and allow you to take actions on them. To create a budget, first decide which product or portfolio that you want to associate the budget to.

To create a budget

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. Choose **Product list** or **Portfolios**.
3. Select the product or portfolio that you want to add a budget to.
4. Open the **Actions** menu, and then choose **Create budget**.
5. On the **Budget creation** page, associate one tag type to your budget.

   There are two types of tags: AutoTags and TagOptions. AutoTags identify the portfolio, product, and user that launched a product. AWS Service Catalog applies these tags automatically to provisioned resources. A TagOption is an administrator-defined key-value pair that's managed in AWS Service Catalog.

   In order for spending that occurs on a portfolio or product to reflect on the associated budget, they must have the same tag. Note that a tag key being used for the first time can take 24 hours to activate. For more information, see the section called “Prerequisites”.
6. Choose **Create in AWS Budgets**. You're directed to the **Set your budget** page. Continue setting up your budget by following the steps in Creating a Budget.

> **Note**
>
> After you create a budget, you must associate it to the product or portfolio.

Associating a Budget

Each portfolio or product can have one budget associated to it. Each budget can be associated to multiple portfolios and products.

When you associate a budget to a portfolio or product, you're able to view information about the budget from that portfolio or product's details page. In order for spending that occurs on

the portfolio or product to be reflected on the budget, you must associate the same tags on the budget and portfolio or product.

> **Note**
>
> If you delete a budget from AWS Budgets, existing associations with AWS Service Catalog products and portfolios still exist. AWS Service Catalog won't be able to display any information about the deleted budget.

To associate a budget

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. Choose **Product list** or **Portfolios**.
3. Select the product or portfolio that you want to associate a budget to.
4. Open the **Actions** menu, and then choose **Associate budget**.
5. On the **Budget association** page, select an existing budget, and then choose **Continue**.
6. The **products** or **portfolios** table now includes data for the budget you just added.

Viewing a Budget

If a budget is associated to a product, you can view information about the budget on the **Product details** and **Product list** pages. If a budget is associated to a portfolio, you can view information about the budget on the **Portfolios** and **Portfolio details** pages.

The **Portfolios** and **Product list** pages display budget information for existing resources. You can see columns displaying **Current vs. budget** and **Forecast vs. budget**.

When you choose on a product or portfolio, you're directed to a details page. The **Portfolio details** and **Product details** pages have sections with detailed information about the associated budgets. You can see the budgeted amount, current spend, and forecasted spend. You also have the option to view budget details and edit the budget.

Disassociating a Budget

You can disassociate a budget from a portfolio or product.

> **Note**
>
> If you delete a budget from AWS Budgets, existing associations with AWS Service Catalog products and portfolios still exist. AWS Service Catalog won't be able to display any information about the deleted budget.

To disassociate a budget

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. Choose **Product list** or **Portfolios**.
3. Select the product or portfolio that you want to disassociate a budget from.
4. Choose **Actions**. From the dropdown, choose **Disassociate budget**. A confirmation alert appears.
5. After you confirm that you want to dissacciate the budget from the product or portfolio, choose **Confirm**.

Managing Provisioned Products

AWS Service Catalog provides an interface for managing provisioned products. You can view, update, and terminate all provisioned products for your catalog based on access level. Refer to the following sections for example procedures.

Topics

- Managing provisioned products as the administrator
- Changing Provisioned Product Owner
- Updating templates for provisioned products
- Tutorial: Identifying User Resource Allocation
- Managing Terraform Open Source product status errors
- Managing the Terraform Open Source product state file

Managing provisioned products as the administrator

To manage all of the provisioned products for an account, you must have `AWSServiceCatalogAdminFullAccess` or an equivalent IAM permission to access provisioned-product write operations. For more information, see Identity and Access Management in AWS Service Catalog.

> **Tip**
>
> For static provisioned-product chaining, you must reference provisioned-product outputs in a product-artifact template before the provisioned product is provisioned. For more information, including an example, see the following:
>
> - AWS::ServiceCatalog::CloudFormationProvisionedProduct in the *AWS CloudFormation User Guide*.
> - DescribeProvisioningParameters (ProvisioningArtifactOutputKeys) in the *AWS Service Catalog Developer Guide*.

To view and manage all provisioned products

1. Open the AWS Service Catalog console at https://console.aws.amazon.com/servicecatalog/.

If you are already logged in to the AWS Service Catalog console, choose **Service Catalog**, then **End user**.

2. If necessary, scroll down to the **Provisioned products** section.
3. In the **Provisioned products** section, choose the **View:** list and select the level of access you want to see: **User**, **Role**, or **Account**. This action displays all the provisioned products in the catalog.
4. Choose a provisioned product to view, update, or terminate. For more information about the information provided in this view, see Viewing Provisioned Product Information.

Changing Provisioned Product Owner

You can change the owner of a provisioned product anytime. You need to know the ARN of the user or role you want to set as the new owner.

By default, this feature is available to administrators using the `AWSServiceCatalogAdminFullAccess` managed policy. You can enable it for end users by granting them the `servicecatalog:UpdateProvisionedProductProperties` permission in AWS Identity and Access Management (IAM).

To change the owner of a provisioned product

1. In the AWS Service Catalog console, choose **Provisioned products list**.
2. Locate the provisioned product you want to update, then choose the three dots beside it and choose **Change provisioned product owner.** You can also find the **Change owner** option on the provisioned product's detail page, in the **Actions** menu.
3. In the dialog box, enter the ARN of the user or role you want to set as the new owner. An ARN begins with `arn:` and includes other information separated by colons or slashes, for example, `arn:aws:iam::123456789012:user/NewOwner`.
4. Choose **Submit**. You will see a success message when the owner has been updated.

See Also

- UpdateProvisionedProductProperties

Updating templates for provisioned products

You can change the current template of a provisioned product to a different template. For example if you have an EC2 product in Service Catalog, you can update that EC2 product to retain the same provisioned product ID, but change the template to a S3 bucket.

> **Note**
>
> Updating templates is not supported for provisioned Terraform Open Source or Terraform Cloud products. If you want to use a different template for an existing Terraform product, you must delete the product and then create a new product using the desired template.

To update a template for a provisioned product

1. In the left navigation menu, choose **Provisioned products**.
2. In **Provisioned products**, choose a provisioned product and select **Actions**, **Update**.

   Note that you can also select **Actions**, **Update** in the **Provisioned product details** page.
3. (Optional) In **Product details**, choose **Change product**.

   In **Change product**, note this warning:

   Changing the product will update this provisioned product to a different product template. This may terminate resources and create new resources.

   You can update a provisioned product to a different version within the same product.
4. (Optional) In **Products**, choose the product you want to update with a different template. Then choose **Change**.

   In **Product details**, note this warning:

   [Product name] will be updated from [current template name] to [new template name]. However, the name of your provisioned product, [Provisioned Product name], will not change.

   You can update a provisioned product to a different version within the same product.
5. In **Product versions**, choose the version of the product you want.
6. In **Parameters**, choose the appropriate parameters.
7. Choose **Update**.

In **Provisioned product details**, you can see the details of the update. The provisioned product name does not change, but the provisioned product now has a different template.

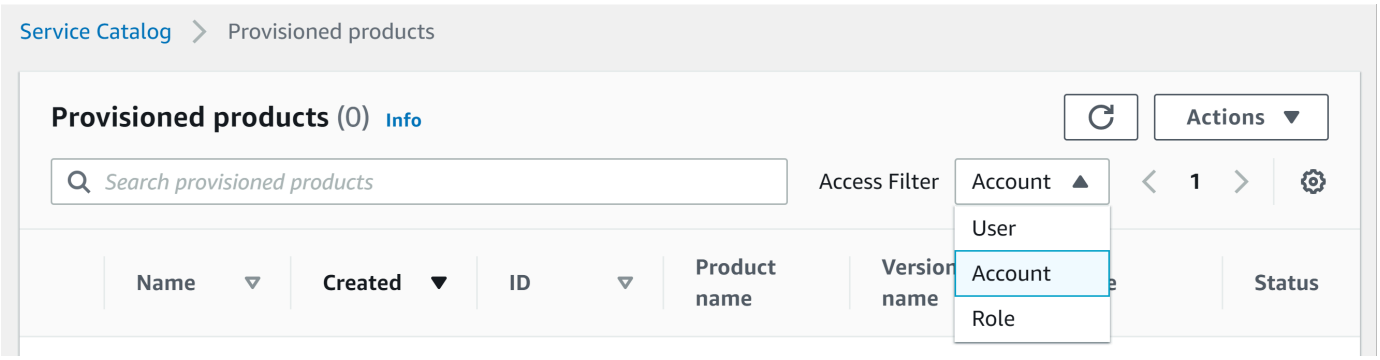
Tutorial: Identifying User Resource Allocation

You can identify the user who provisioned a product and resources associated with the product using the AWS Service Catalog console. This tutorial helps translate this example to your own specific provisioned products.

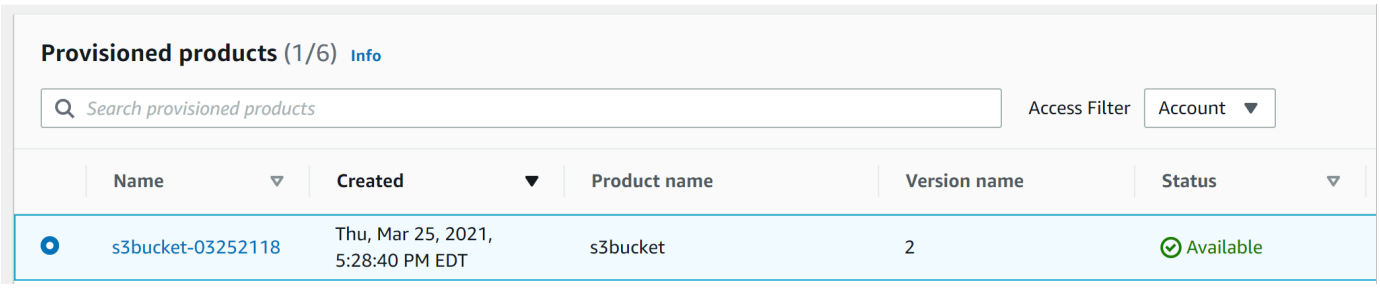
To manage all provisioned products for the account, you need `AWSServiceCatalogAdminFullAccess` or equivalent access to the provisioned product write operations. For more information, see Identity and Access Management in the *AWS Service Catalog Administrator Guide.*

To identify the user who provisioned a product and the associated resources

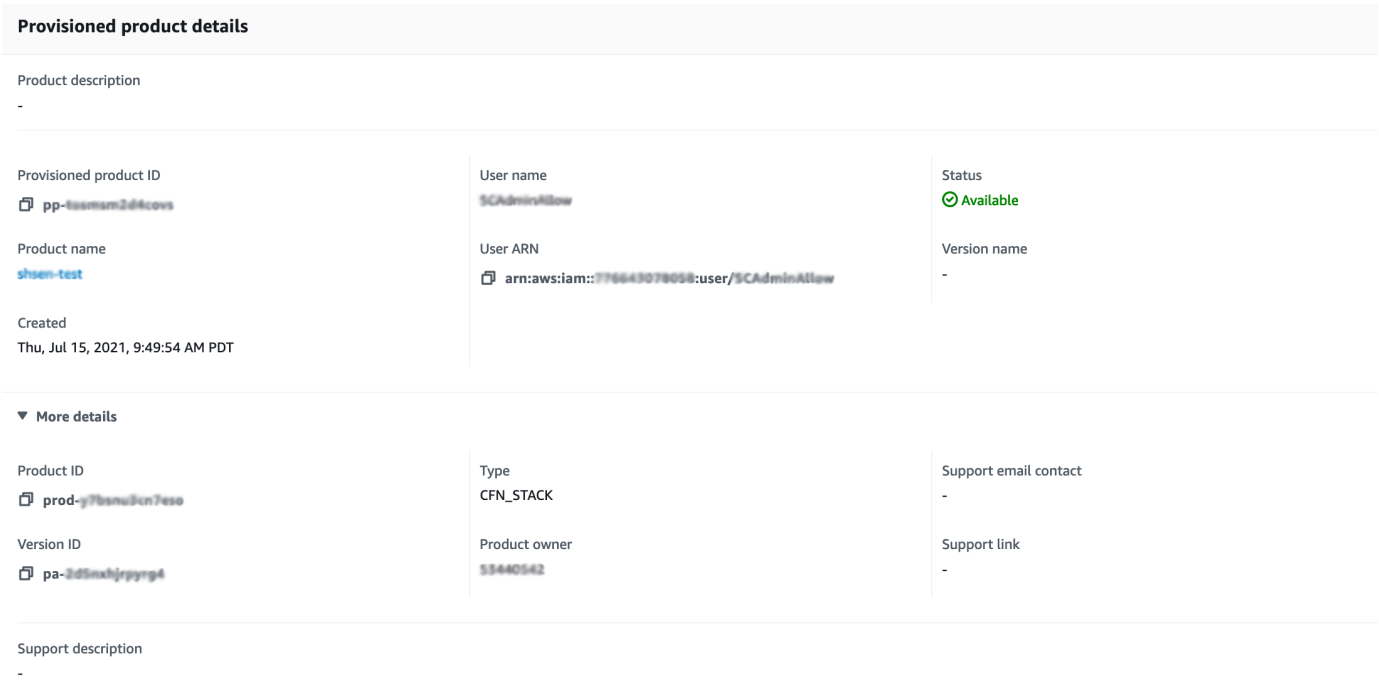
1. Open https://console.aws.amazon.com/servicecatalog.
2. In the left navigation menu, choose **Provisioned product**.
3. In the **Access Filter** dropdown menu, choose **Account**.



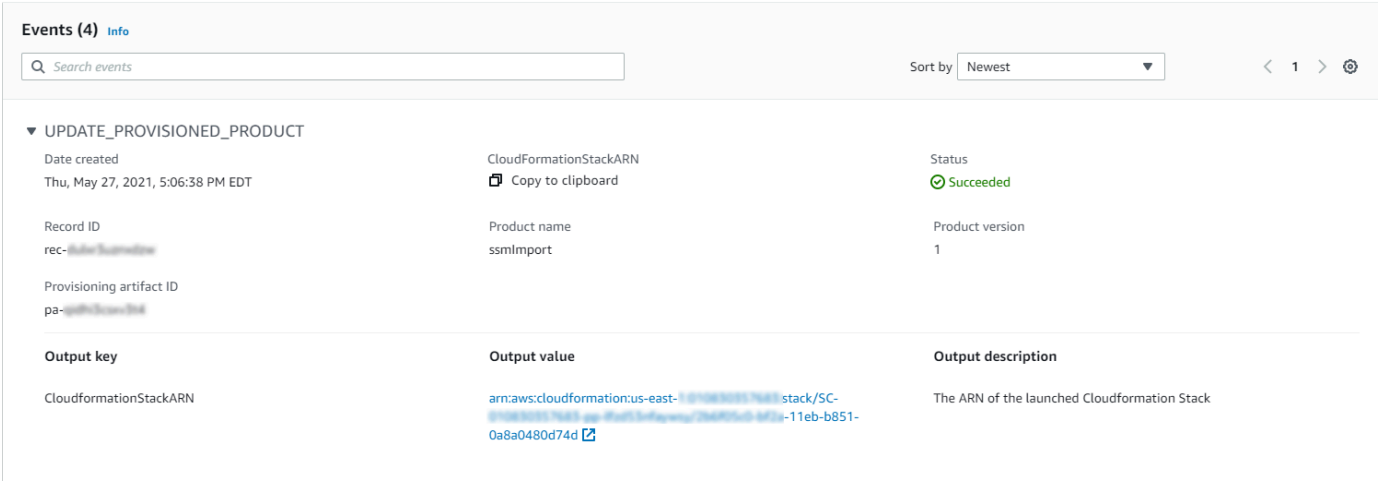
4. In the **Account** view, choose and open a provisioned product to display its details.



You can see the details of the provisioned product.



5. Scroll down to expand the **Events** section. Note the `Provisioned product ID` and `CloudformationStackARN` values.



6. Use the provisioned product ID to identify the AWS CloudTrail record that corresponds to this launch and identify the requesting user (typically, you enter an email address during federation). In this example, it is "steve".

```
{
  "eventVersion":"1.03","userIdentity":
  {
    "type":"AssumedRole",
    "principalId":"[id]:steve",
```

```
    "arn":"arn:aws:sts::[account number]:assumed-role/SC-usertest/steve",
    "accountId":[account number],
    "accessKeyId":[access key],
    "sessionContext":
    {
      "attributes":
      {
        "mfaAuthenticated":[boolean],
        "creationDate":[timestamp]
      },
      "sessionIssuer":
      {
        "type":"Role",
        "principalId":"AROAJEXAMPLELH3QXY",
        "arn":"arn:aws:iam::[account number]:role/[name]",
        "accountId":[account number],
        "userName":[username]
      }
    }
  },
  "eventTime":"2016-08-17T19:20:58Z","eventSource":"servicecatalog.amazonaws.com",
  "eventName":"ProvisionProduct",
  "awsRegion":"us-west-2",
  "sourceIPAddress":[ip address],
  "userAgent":"Coral/Netty",
  "requestParameters":
  {
    "provisioningArtifactId":[id],
    "productId":[id],
    "provisioningParameters":[Shows all the parameters that the end user entered],
    "provisionToken":[token],
    "pathId":[id],
    "provisionedProductName":[name],
    "tags":[],
    "notificationArns":[]
  },
  "responseElements":
  {
    "recordDetail":
    {
      "provisioningArtifactId":[id],
      "status":"IN_PROGRESS",
      "recordId":[id],
      "createdTime":"Aug 17, 2016 7:20:58 PM",
```

```
      "recordTags":[],
      "recordType":"PROVISION_PRODUCT",
      "provisionedProductType":"CFN_STACK",
      "pathId":[id],
      "productId":[id],
      "provisionedProductName":"testSCproduct",
      "recordErrors":[],
      "provisionedProductId":[id]
    }
  },
  "requestID":[id],
  "eventID":[id],
  "eventType":"AwsApiCall",
  "recipientAccountId":[account number]
}
```

7. Use the `CloudformationStackARN` value to identify AWS CloudFormation events to find information about the created resources. You can also use the AWS CloudFormation API to obtain this information. For more information, see AWS CloudFormation API Reference.

You can perform steps 1 through 4 using the AWS Service Catalog API or the AWS CLI. For more information, see AWS Service Catalog Developer Guide. and AWS Service Catalog Command Line Reference.

Managing Terraform Open Source product status errors

Terraform Open Source `ProvisionProduct` failures are routed to the `TAINTED` state, allowing each provisioned product to proceed to `UpdateProvisionedProduct`. When this occurs:

- `UpdateProvisionedProduct` does **not** make an attempt to update or correct tags, or to create or modify a resource group.
- `UpdateProvisionedProduct` does **not** consider failures from previous provisioning operations when deciding if the provisioned product should be set to `AVAILABLE` or `TAINTED`.

AWS Service Catalog only applies Tags during `ProvisionProduct`. Any failed tagging that results from a failure of the `ProvisionProduct` operation are **not** automatically resolved.

Status error examples

Example 1: AWS Service Catalog does not create a resource group during `ProvisionProduct`

In the scenario below, you have a provisioned product in the `AVAILABLE` state even if there is not a supporting resource group, and without any tags applied to the resources.

1. Your action initiates `ProvisionProduct`.
2. The Terraform provisioning engine responds to `ProvisionProduct` with a workflow failure and does not provide a `ResourceIdentifier`.
3. The `ProvisionProduct` workflow does not create a resource group, and then sets the provisioned product state to `ERROR`.
4. You then initiate the `UpdateProvisionedproduct` operation.
5. The Terraform provisioning engine responds indicating "success."
6. As a result, the `UpdateprovisionedProduct` workflow sets the provisioned product state to `AVAILABLE`, but does **not** create a resource group, or attempt to apply any Tags.

Example 2: AWS Service Catalog creates new resources during `UpdateProvisionedProduct`

In the scenario below, you have a provisioned product in the `AVAILABLE` state even if new resources do **not** have any tags applied.

1. Your action initiates `ProvisionProduct`.
2. The Terraform provisioning engine responds indicating "success" and provides a `ResourceIdentifier`.
3. The `ProvisionProduct` workflow creates a resource group and applies tags to all of the identified resources.
4. You initiate `UpdateProvisionedProduct` on a new artifact that creates new resources.
5. The Terraform provisioning engine responds indicating "success."
6. The `UpdateProvisionedProduct` workflow sets the provisioned product state to `AVAILABLE` but does **not** attempt to apply any additional tags to the new resources.

Status error solution

AWS Service Catalog ensures that a resource group is created for all provisioned products set to `TAINTED` from `ProvisionProduct`. If the Terraform provisioning engine does not return

a `ResourceIdentifier`, or if AWS Service Catalog fails to create a resource group, then the provisioned product is set to the `ERROR` state, forcing you to terminate.

Managing the Terraform Open Source product state file

Every Terraform Open Source provisioned product has a **single-state file**. There is a 1:1 relationship between the provisioned product and its state file. The files are stored in an Amazon S3 bucket named `sc-terraform-engine-state-${AWS::AccountId}-${AWS::Region}`. The state file is saved under the `AccountID` or `ProvisionedProductID` object key.

State file access is limited to the `GetStateFile` AWS Lambda and Amazon EC2 launch templates. AWS Service Catalog administrators do **not** have direct access to the state files in Amazon S3. Administrators must access the files using Amazon EC2. By default, AWS Service Catalog administrators can see the list of state files, but cannot read or write the file contents. Only the Terraform provisioning engine can read or write the file contents.

Managing Tags in AWS Service Catalog

AWS Service Catalog provides tags so you can categorize your resources. There are two types of tags: AutoTags and TagOptions.

AutoTags are tags that identify information about the origin of a provisioned resource in AWS Service Catalog and are automatically applied by AWS Service Catalog to provisioned resources.

TagOptions are key-value pairs managed in AWS Service Catalog that serve as templates for creating AWS tags.

Topics

- AWS Service Catalog AutoTags
- AWS Service Catalog TagOption Library

AWS Service Catalog AutoTags

> **Note**
>
> AWS Service Catalog does not support AutoTags for Terraform Open Source products.

AutoTags are tags that identify information about the origin of a provisioned resource in AWS Service Catalog and are automatically applied by AWS Service Catalog to provisioned resources.

AutoTags include tags for the unique identifiers for portfolio, product, user, product version, and provisioned product. This provides a set of tags that reflect the AWS Service Catalog structure that customers have configured in the catalog. AutoTags do not count against the customer's 50-tag limit.

> **Note**
>
> AWS Service Catalog does not support AutoTags for Terraform Open Source products.

AWS Service Catalog AutoTags can help provide consistent tagging for your resources, which is useful when setting budgets for a portfolio, product, or user. You can also use the AutoTags to

identify resources for post-launch operations such as setting AWS Config rules. AutoTags for your provisioned resources can be viewed in the Tags section of the downstream services used for provisioning, such as AWS CloudFormation, Amazon EC2, and Amazon S3.

> **Note**
>
> AWS Service Catalog does not update AutoTags after you apply AutoTags to provisioned resources. If you update the provisioned product to a different product, provisioned artifact, or new launch path, the existing AutoTags still show the original values.

AutoTag details

- **aws:servicecatalog:portfolioArn** - The ARN of the portfolio from which the provisioned product was launched.
- **aws:servicecatalog:productArn** - The ARN of the product from which the provisioned product was launched.
- **aws:servicecatalog:provisioningPrincipalArn** - The ARN of the provisioning principal (user) who created the provisioned product.
- **aws:servicecatalog:provisionedProductArn** - The provisioned product ARN.
- **aws:servicecatalog:provisioningArtifactIdentifier** - The ID of the original provisioning artifact (product version).

AWS Service Catalog TagOption Library

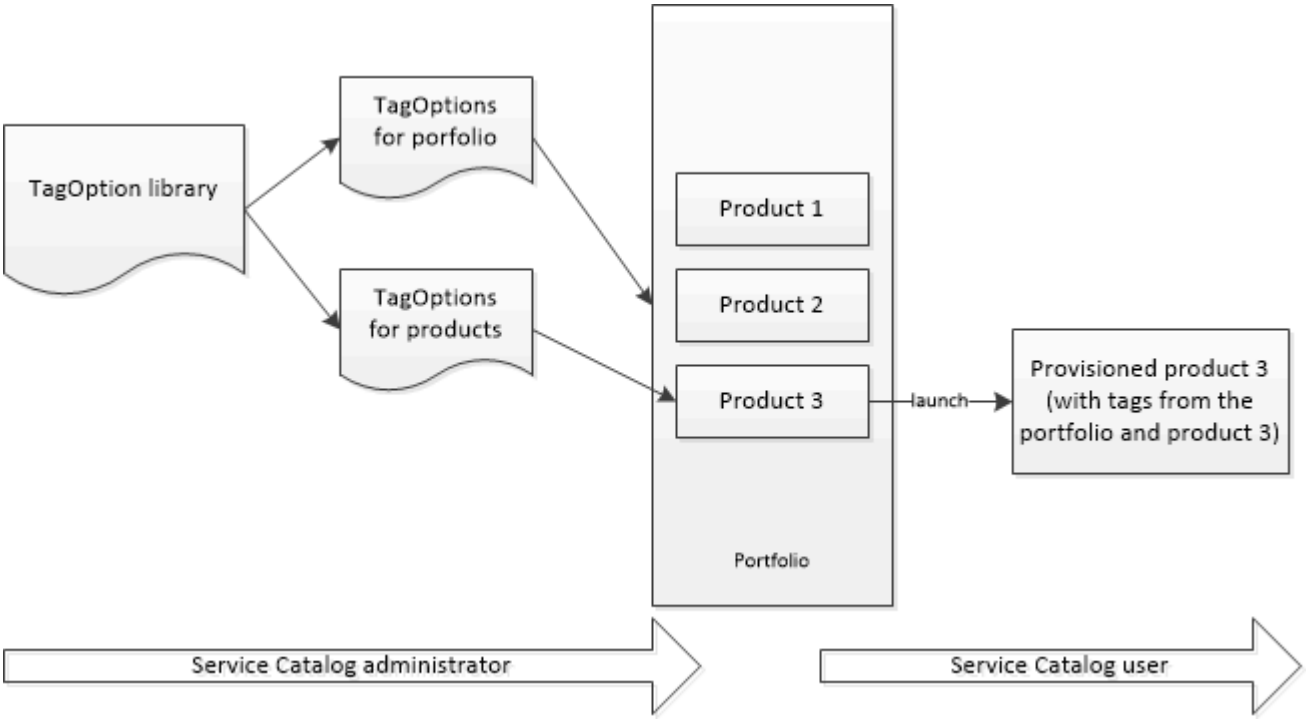
To allow administrators to easily manage tags on provisioned products, AWS Service Catalog provides a TagOption library. A TagOption is a key-value pair managed in AWS Service Catalog. It is not an AWS tag, but serves as a template for creating an AWS tag based on the TagOption.

AWS Service Catalog does not support TagOptions for Terraform Open Source or Terraform Cloud products.

The TagOption library makes it easier to enforce the following:

- A consistent taxonomy
- Proper tagging of AWS Service Catalog resources
- Defined, user-selectable options for allowed tags

Administrators can associate TagOptions with portfolios and products. During a product launch (provisioning), AWS Service Catalog aggregates the associated portfolio and product TagOptions, and applies them to the provisioned product, as shown in the following diagram.



With the TagOption library, you can deactivate TagOptions and retain their associations to portfolios or products, and reactivate them when you need them. This approach not only helps maintain library integrity, it also allows you to manage TagOptions that might be used intermittently, or only under special circumstances.

You manage TagOptions with the AWS Service Catalog console or the TagOption library API. For more information, see Service Catalog API Reference.

Contents

- Launching a Product with TagOptions
- Managing TagOptions
- Using TagOptions with AWS Organizations tag policies

Launching a Product with TagOptions

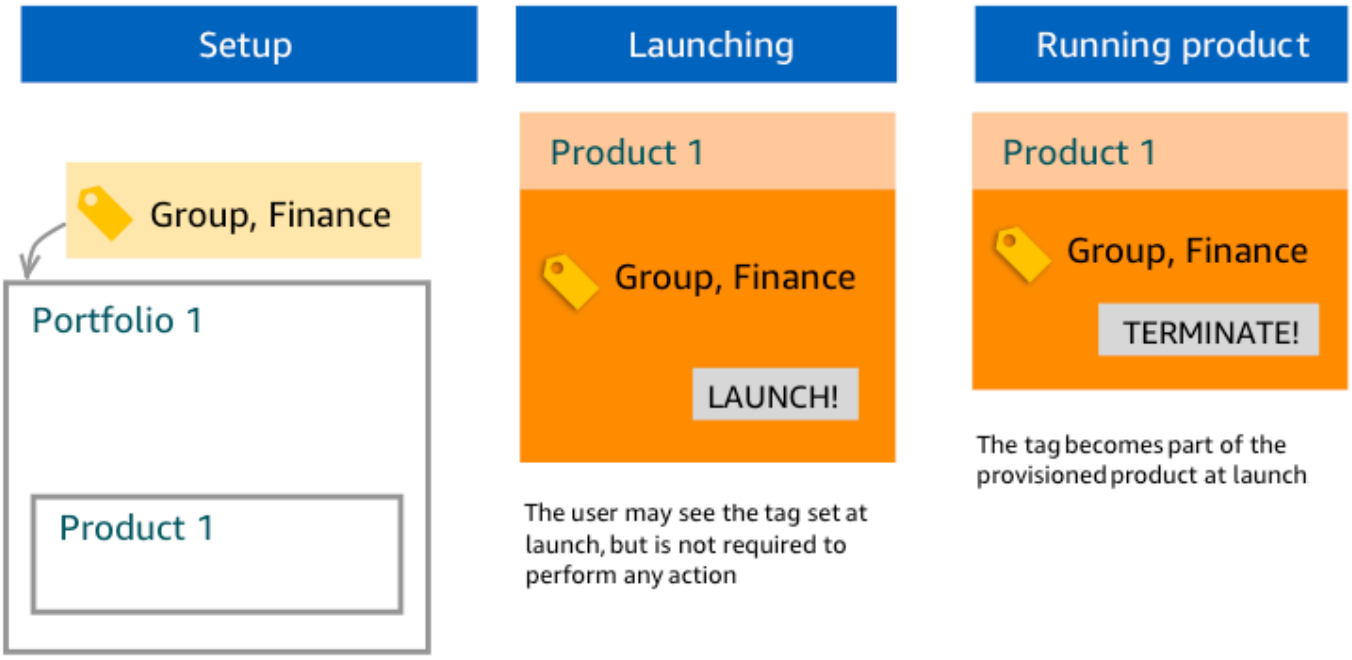
When a user launches a product that has TagOptions, AWS Service Catalog performs the following actions on your behalf:

- Collects all TagOptions for the product and the launching portfolio.
- Ensures that only TagOptions with unique keys are used in a tag on the provisioned product. Users get a multiple-choice value lists for a key. After the user chooses a value, it becomes a tag on the provisioned product.
- Allows users to add non-conflicting tags to the product during provisioning.

The following use cases demonstrate how TagOptions work during launch.

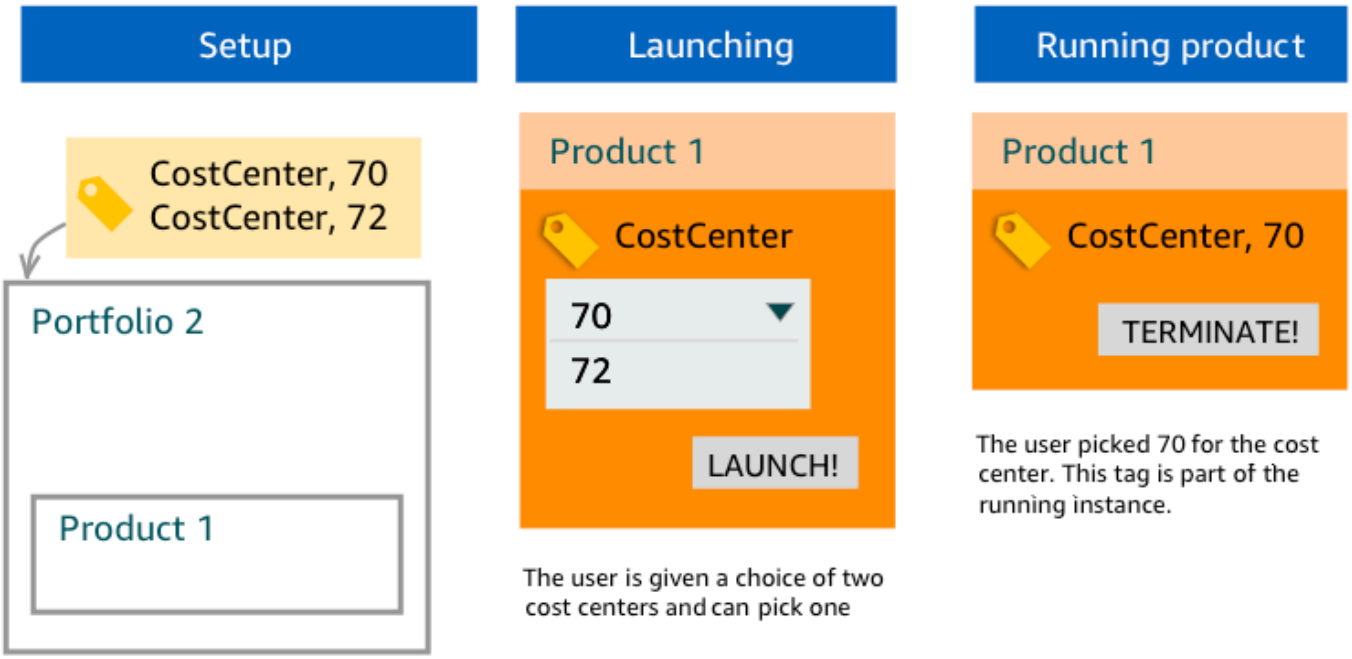
Example 1: A Unique TagOption Key

An administrator creates **TagOption[Group=Finance]** and associates it with **Portfolio1**, which has **Product1** with no TagOptions. When a user launches the provisioned product, the single TagOption becomes **Tag[Group=Finance]**, as follows:



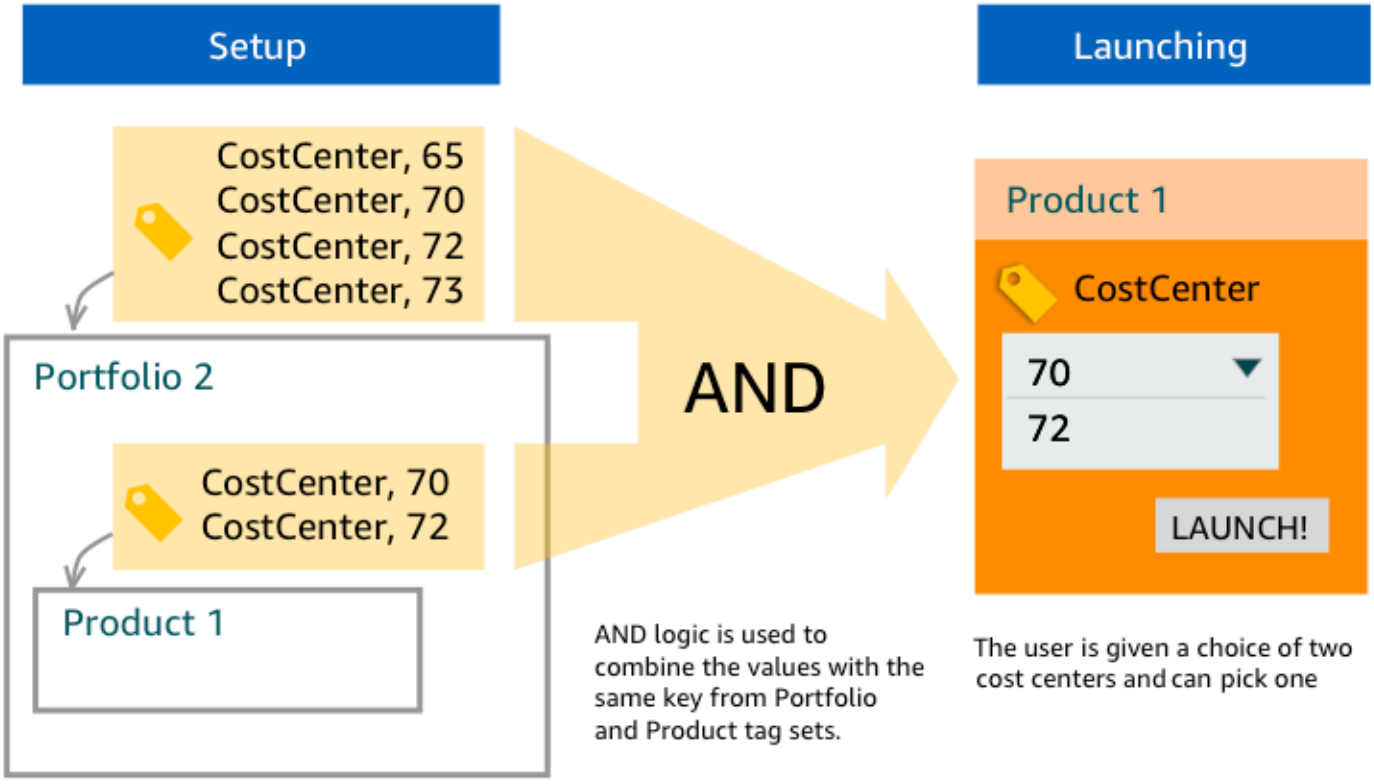
Example 2: A Set of TagOptions with the Same Key on a Portfolio

An administrator has placed two TagOptions with the same key on a portfolio, and there are no TagOptions with the same key on any products within that portfolio. During launch, the user must select one of the two values associated with the key. The provisioned product is then tagged with the key and the user-selected value.



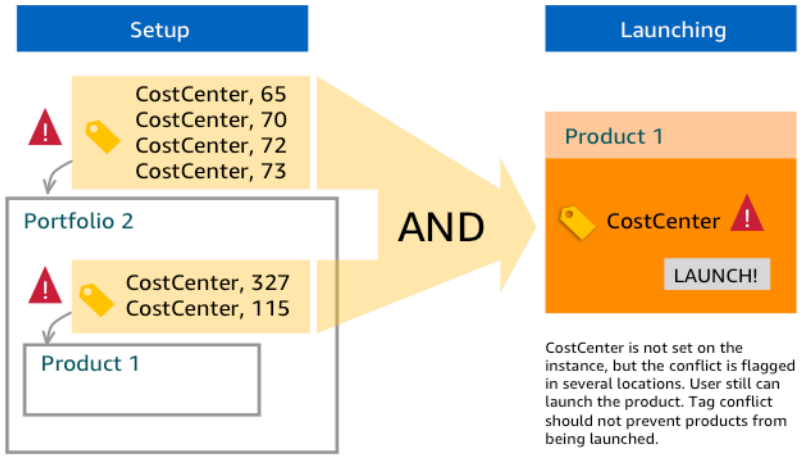
Example 3: A Set of TagOptions with the Same Key on Both the Portfolio and a Product in that Portfolio

An administrator has placed several TagOptions with the same key on a portfolio, and there are also several TagOptions with the same key on the product within that portfolio. AWS Service Catalog creates a set of values from the aggregation (logical AND operation) of the TagOptions. When the user launches the product, he or she sees and selects from this set of values. The provisioned product is tagged with the key and the user-selected value.



Example 4: Multiple TagOptions with the Same Key and Conflicting Values

An administrator has placed several TagOptions with the same key on a portfolio, and there are also several TagOptions with the same key on the product in that portfolio. AWS Service Catalog creates a set of values from the aggregation (logical AND operation) of the TagOptions. If the aggregation doesn't find values for the key, AWS Service Catalog creates a tag with the same key and a value of `sc-tagconflict-`*`portfolioid`*`-`*`productid`*, where *`portfolioid`* and *`productid`* are the ARNs of the portfolio and product. This ensures that the provisioned product is tagged with the correct key and with a value that the administrator can find and correct.



Managing TagOptions

As an administrator, you can perform the following actions to manage TagOptions in the TagOptions library:

- Create and delete
- Activate or deactivate
- Associate or disassociate
- Edit

To create TagOptions in the console

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. In the left navigation menu, choose **TagOptions library**.
3. In **Create new TagOption**, enter a key and value, and then choose **Add**.

   After the new TagOption has been created, it's grouped by key-value pair and sorted alphabetically in the **TagOptions** list.

   To create a TagOption using the AWS Service Catalog API, see CreateTagOption.

To delete TagOptions in the console

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. In the left navigation menu, choose **TagOptions library** and then choose **Actions**.

3. Select **Delete** and confirm the deletion.

To activate or deactivate one or more TagOptions in the console

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. In the left navigation menu, choose **TagOptions library** and then choose **Actions**.
3. To activate, choose the inactive TagOption you want. Then choose **Actions** and select **Activate** from the dropdown menu, and confirm your selection.

   To deactivate, choose the active TagOption you want. Then choose **Actions** and select **Deactivate** from the dropdown menu, and confirm your selection.

To associate or disassociate one or more TagOptions with a portfolio in the console

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. In the left navigation menu, choose **Portfolios**, and then open the portfolio you want to associate or disassociate.
3. Choose the **TagOptions** tab and select one or more TagOptions to associate or disassociate with the portfolio.
4. Choose **Actions**. Then select **Associate** or **Disassociate** and confirm your selection.

To associate or disassociate one or more TagOptions with a product in the console

1. Open the AWS Service Catalog console at: https://console.aws.amazon.com/servicecatalog/.
2. In the left navigation menu, under **Administration**, choose **Products**. Then open the product you want to associate or disassociate.
3. Choose the **TagOptions** tab and select one or more TagOptions to associate or disassociate with the portfolio.
4. Choose **Actions**. Then select **Associate** or **Disassociate** and confirm your selection.

> **Note**
>
> To associate TagOptions with a portfolio or product using the AWS Service Catalog API, see AssociateTagOptionWithResource.

To remove (disassociate) TagOptions using the AWS Service Catalog API, see DisassociateTagOptionFromResource.

To edit values for TagOptions in the console

1. Open the Service Catalog console at https://console.aws.amazon.com/servicecatalog/.
2. In the left navigation menu, choose **TagOptions library**.
3. Choose a TagOption and open the value. (The value is hyperlinked.) Then choose **Edit**.
4. In the **Value** field, edit the value and choose **Save changes**.

Using TagOptions with AWS Organizations tag policies

This topic provides a brief overview of tag policies for AWS Organizations and TagOptions for AWS Service Catalog. It also suggests how to prevent tagging conflicts when using both features simultaneously.

TagOptions for AWS Service Catalog apply to provisioned products (CloudFormation stacks), while tag policies for AWS Organizations apply to AWS accounts and organizational units (OU) or an organizational root. For example, if you attach a tag policy to an OU, the same tag policy applies to all accounts in that OU. If you use both tagging features simultaneously, you should configure them so they won't conflict.

Tag policies

Tag policies allow you to define rules on how to use tags on AWS resources in your accounts in AWS Organizations. You can use tag policies to create and maintain a consistent approach for tagging AWS resources at the account level.

Tag policies provide an easy way to ensure users apply consistent tags, audit tagged resources, and maintain proper resource categorization. You can also define how tag keys should be capitalized, and the values you want to permit. For example, you can require that all EC2 instances in an account must have a tag key set as **`CostCenter`** and values for that tag to be **`Data Insights`** or **`Marketing`**.

Tag policies enable you to select options to enforce tagging rules, prevent noncompliant operations for tags, and specify the resource types to which enforcement applies. If you don't

choose an enforcement option, tag polices let you create or mutate the noncompliant tags, but reports them as noncompliant in the AWS Organizations console.

For more information on how to set up account level tagging enforcement, see Tag policies in *AWS Organizations*.

TagOptions

TagOptions are a tagging feature that AWS Service Catalog applies to provisioned products at the CloudFormation stack level if they're applied to an associated product. AWS Service Catalog provides a TagOptions library where you can define the key-value pairs to associate with your AWS Service Catalog products. When you launch a AWS Service Catalog product, you must choose TagOption values for the existing TagOption keys associated to that portfolio or product to launch that product. Because you set TagOptions at the portfolio or product levels, you can enforce a consistent taxonomy for tagging with portfolios shared across accounts and regions.

For more information on how to set up TagOptions in AWS Service Catalog, see AWS Service Catalog TagOption Library.

Avoiding conflicts between AWS Organizations tag policies and AWS Service Catalog TagOptions

If you configure AWS Organizations tag policies for accounts in your organization, we recommend the following:

- Share the requirements for conformant tags with administrators who also manage TagOptions for AWS Service Catalog portfolios and products.
- Share the requirements for conformant tags with end users who might launch products in AWS Service Catalog and append optional end user tags to their product launches.

Suppose you want to launch a product in AWS Service Catalog that uses the TagOption key `city`, and you have a tag policy that requires tag keys with `city` to have tag values of **U.S cities**, such as **`Atlanta`**, **`San Francisco`**, or **`Austin`**. AWS Service Catalog does not allow you to launch a product without having selected TagOption values for the required TagOption keys for a product.

In this case, if you have TagOption values for the TagOption key `city` that include **South American cities**, such as **`Rio de Janeiro`** or **`Buenos Aires`**, AWS Service Catalog will not launch the product. Instead, you must select a TagOption value that includes a U.S. city during launch to comply with the tag policy.

The following table provides scenarios that describe how to resolve the tagging conflict issues you might encounter when using tag policies and TagOptions at the same time.

Scenario	Reason	Solution
Product fails to launch because of noncompliant tags if tag enforcement is checked in the tag policy.	Specifying TagOptions with keys and values that you have not added to the allowed list of compliant tags in your tag policy. Adding optional custom tags that are not conformant with your tag policy.	If you configure a specific capitalization schema in your tag policy tag key capitaliz ation enforcement, ensure that your TagOptions tag keys and optional custom tag keys are consistent with what you've specified in your tag policy. Note when the tag key capitalization enforceme nt box is unchecked in your tag policy, it results in all lowercase tag keys being compliant, and ensures your TagOptions tag keys and optional custom tag keys are consistent (such as all lowercase) with what you've required in your tag policy.
Product fails to launch due to nonconformant tag key capitalization.	Specifying capitalization in the TagOptions keys that is inconsistent with your tag policy capitalization enforcement rules.	Correctly configure your tag policies. If you don’t specify tag key capitaliz ation compliance, the default tag key capitalization is all lowercase. In addition, if you don’t specify tag key capitalization

Scenario	Reason	Solution
		compliance in your tag policy, make sure your TagOption s tag keys in AWS Service Catalog are all lowercase to comply to enforcement rules. If you use a tag policy that doesn't have capitalization compliance enabled, that tag policy only considers all lower case tag keys to be compliant.
Product fails to launch because of incompatible tag values.	Selecting a TagOptions tag value for a product launch that is not in your tag policy Tag Value Compliance allowed list.	Associate TagOptions to your products and portfolios that are consistent with what you've required in the list tag policy Tag Value Compliance allowed tag values.

External Engines for AWS Service Catalog

In AWS Service Catalog, *external engines* are represented through an EXTERNAL product type. The EXTERNAL product type allows for the integration of third-party provisioning engines, such as Terraform. You can use external engines to extend the capabilities of Service Catalog beyond the native AWS CloudFormation templates, enabling the use of other instructure as code (IaC) tools.

The EXTERNAL product type allows you to manage and deploy resources using the familiar interface of Service Catalog while leveraging the specific features and syntax of your chosen IaC tool.

To enable EXTERNAL product types in Service Catalog, you must define a set of standard resources in your account. These resources are known as the *engine*. Service Catalog delegate tasks to the engine at specific points in the artifact-parsing and provisioning operations.

A *provisioning artifact* represents the specific version of a product within Service Catalog, allowing you to manage and deploy consistent resources.

When you call AWS Service Catalog's DescribeProvisioningArtifact or DescribeProvisioningParameters operations for a provisioning artifact for an EXTERNAL product type, Service Catalog invokes an AWS Lambda function in the engine. This is required to extract the list of parameters from the provided provisioning artifact and return them to AWS Service Catalog. These parameters will be used later as part of the provisioning process.

When you provision an EXTERNAL provisioning artifact by calling ProvisionProduct, Service Catalog first performs some actions internally, then send a message to an Amazon SQS queue in the engine. Next, the engine assumes the provided *launch role* (the IAM role that you assign to a product as a launch constraint), provisions the resources based on the provided provisioning artifact, and invokes the NotifyProvisionProductEngineWorkflowResult API to report success or failure.

Calls to UpdateProvisionedProduct and TerminateProvisionedProduct are handled similarly, with each having a distinct queue and Notify APIs:

- NotifyProvisionProductEngineWorkflowResult
- NotifyUpdateProvisionedProductEngineWorkflowResult
- NotifyTerminateProvisionedProductEngineWorkflowResult.

Topics

- Considerations
- Parameter Parsing
- Provisioning
- Updating
- Terminating
- Tagging

Considerations

Limit of one external engine per hub account

You can only use one EXTERNAL provisioning engine per Service Catalog hub account. The Service Catalog *hub-and-spoke* model allows the hub account to create baseline products and share the portfolio while the spoke accounts import portfolios and leverage the products.

This limit is because EXTERNAL can only be routed to one engine in an account. If an administrator wants to have multiple external engines, the administrator must set up the external engines (along with the portfolios and products) in different hub accounts.

External engines only support launch roles with launch contraints

EXTERNAL provisioning artifacts only support provisioning with launch roles which are specified using *launch constraints*. A launch constraint specifies the IAM role that Service Catalog assumes when an end user launches, updates, or terminates a product. For more information on launch constraints, see AWS Service Catalog Launch Constraints.

Parameter Parsing

EXTERNAL provisioning artifacts can be of any format. This means that when creating an EXTERNAL product type, the engine is required to extract the list of parameters from the provided provisioning artifact and return them to Service Catalog. This is done by creating a Lambda function in your account that can accept the following request format, process the provisioning artifact, and return the following response format.

Important

The Lambda function must be named `ServiceCatalogExternalParameterParser`.

Request syntax:

```
{
    "artifact": {
        "path": "string",
        "type": "string"
    },
    "launchRoleArn": "string"
}
```

Field	Type	Required	Description
artifact	object	Yes	Details for the artifact to be parsed.
artifact / path	string	Yes	Location from where the parser downloads the artifact. For example, for `AWS_S3`, this is the Amazon S3 URI.
artifact / type	string	Yes	Type of artifact. Allowed value: `AWS_S3`.
launchRole	string	No	The Amazon Resource Name (ARN) of the launch role to assume when downloading the artifact. If no launch role is provided, the

Field	Type	Required	Description
			Lambda's execution role is used.

Response syntax:

```
{
    "parameters": [
        {
            "key": "string"
            "defaultValue": "string",
            "type": "string",
            "description": "string",
            "isNoEcho": boolean
        },
    ]
}
```

Field	Type	Required	Description
parameters	list	Yes	The list of parameters that Service Catalog asks the end user to provide when provisioning a product or updating a provisioned product. If no parameters are defined in the artifact, an empty list is returned.
key	string	Yes	The parameter key.
defaultValue	string	No	The default value of the parameter if the

Field	Type	Required	Description
			end user does not provide a value.
type	string	Yes	The expected type of the parameter value for the engine. For example, a string, boolean, or map. The allowed values are specific to each engine. Service Catalog passes each parameter value to the engine as a string.
description	string	No	Description for the parameter. It is recommended that this is user-friendly.
isNoEcho	boolean	no	Determines if the parameter value is not echoed in logs. Default value is false (parameter values are echoed).

Provisioning

For the ProvisionProduct operation, Service Catalog delegates the actual provisioning of resources to the engine. The engine is responsible for interfacing with your IaC solution of choice (such as Terraform) to provision resources as defined in the artifact. The engine is also responsible for notifying Service Catalog of the result.

Service Catalog sends all Provision requests to an Amazon SQS queue in your account named `ServiceCatalogExternalProvisionOperationQueue`.

Request syntax:

```
{
    "token": "string",
    "operation": "string",
    "provisionedProductId": "string",
    "provisionedProductName": "string",
    "productId": "string",
    "provisioningArtifactId": "string",
    "recordId": "string",
    "launchRoleArn": "string",
    "artifact": {
        "path": "string",
        "type": "string"
    },
    "identity": {
        "principal": "string",
        "awsAccountId": "string",
        "organizationId": "string"
    },
    "parameters": [
        {
            "key": "string",
            "value": "string"
        }
    ],
    "tags": [
        {
            "key": "string",
            "value": "string"
        }
    ]
}
```

Field	Type	Required	Description
token	string	Yes	The token that identifies this operation. The token

Field	Type	Required	Description
			must be returned to Service Catalog to notify of execution results.
operation	string	Yes	This field must be `PROVISION_PRODUCT` for this operation.
provisionedProductId	string	Yes	ID of the provisioned product.
provisionedProductName	string	Yes	Name of the provisioned product.
productId	string	Yes	ID of the product.
provisioningArtifactId	string	Yes	ID of the provisioning artifact.
recordId	string	Yes	ID of the Service Catalog record for this operation.
launchRoleArn	string	Yes	Amazon Resource Name (ARN) for the IAM role to use for provisioning resources.
artifact	object	Yes	Details for the artifact that defines how the resources are provisioned.

Field	Type	Required	Description
artifact / path	string	Yes	Location from where the engine downloads the artifact. For example, for `AWS_S3`, this is the Amazon S3 URI.
artifact / type	string	Yes	Type of artifact. Allowed value: `AWS_S3`.
identity	string	No	The field is currently not used.
parameters	list	Yes	List of parameter key-value pairs the user entered to Service Catalog as inputs for this operation.
tags	list	Yes	List of key-value-pairs the user entered to Service Catalog as tags to apply to the provisioned resources .

Workflow Result Notification:

Invoke the NotifyProvisionProductEngineWorkflowResult API with the response object specified on the API details page.

Updating

For the UpdateProvisionedProduct operation, Service Catalog delegates the actual updating of resources to the engine. The engine is responsible for interfacing with your IaC solution of choice (such as Terraform) to updating resources as defined in the artifact. The engine is also responsible for notifying Service Catalog of the result.

Service Catalog sends all Update requests to an Amazon SQS queue in your account named `ServiceCatalogExternalUpdateOperationQueue`.

Request syntax:

```
{
    "token": "string",
    "operation": "string",
    "provisionedProductId": "string",
    "provisionedProductName": "string",
    "productId": "string",
    "provisioningArtifactId": "string",
    "recordId": "string",
    "launchRoleArn": "string",
    "artifact": {
        "path": "string",
        "type": "string"
    },
    "identity": {
        "principal": "string",
        "awsAccountId": "string",
        "organizationId": "string"
    },
    "parameters": [
        {
            "key": "string",
            "value": "string"
        }
    ],
    "tags": [
        {
            "key": "string",
            "value": "string"
        }
    ]
```

```
}
```

Field	Type	Required	Description
token	string	Yes	The token that identifies this operation. The token must be returned to Service Catalog to notify of execution results.
operation	string	Yes	This field must be `UPDATE_PROVISION_PRODUCT` for this operation.
provisionedProductId	string	Yes	ID of the provisioned product.
provisionedProductName	string	Yes	Name of the provisioned product.
productId	string	Yes	ID of the product.
provisioningArtifactId	string	Yes	ID of the provisioning artifact.
recordId	string	Yes	ID of the Service Catalog record for this operation.
launchRoleArn	string	Yes	Amazon Resource Name (ARN) for the IAM role to use for provisioning resources.

Field	Type	Required	Description
artifact	object	Yes	Details for the artifact that defines how the resources are provisioned.
artifact / path	string	Yes	Location from where the engine downloads the artifact. For example, for `AWS_S3`, this is the Amazon S3 URI.
artifact / type	string	Yes	Type of artifact. Allowed value: `AWS_S3`.
identity	string	No	The field is currently not used.
parameters	list	Yes	List of parameter key-value pairs the user entered to Service Catalog as inputs for this operation.
tags	list	Yes	List of key-value-pairs the user entered to Service Catalog as tags to apply to the provisioned resources .

Workflow Result Notification:

Invoke the NotifyUpdateProvisionedProductEngineWorkflowResult API with the response object specified on the API details page.

Terminating

For the TerminateProvisionedProduct operation, Service Catalog delegates the actual terminating of resources to the engine. The engine is responsible for interfacing with your IaC solution of choice (such as Terraform) to terminate resources as defined in the artifact. The engine is also responsible for notifying Service Catalog of the result.

Service Catalog sends all Terminate requests to an Amazon SQS queue in your account named `ServiceCatalogExternalTerminateOperationQueue`.

Request syntax:

```
{
    "token": "string",
    "operation": "string",
    "provisionedProductId": "string",
    "provisionedProductName": "string",
    "recordId": "string",
    "launchRoleArn": "string",
    "identity": {
        "principal": "string",
        "awsAccountId": "string",
        "organizationId": "string"
    }
}
```

Field	Type	Required	Description
token	string	Yes	The token that identifies this operation. The token must be returned to Service Catalog to notify of execution results.

Field	Type	Required	Description
operation	string	Yes	This field must be `TERMINATE_PROVISIONED_PRODUCT` for this operation.
provisionedProductId	string	Yes	ID of the provisioned product.
provisionedProductName	string	Yes	Name of the provisioned product.
recordId	string	Yes	ID of the Service Catalog record for this operation.
launchRoleArn	string	Yes	Amazon Resource Name (ARN) for the IAM role to use for provisioning resources.
identity	string	No	The field is currently not used.

Workflow Result Notification:

Invoke the NotifyTerminateProvisionedProductEngineWorkflowResult API with the response object specified on the API details page.

Tagging

For managing tags through Resource Groups, your launch role need the following additional permission statements:

```
{
```

```
    "Effect": "Allow",
    "Action": [
        "resource-groups:CreateGroup",
        "resource-groups:ListGroupResources"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "tag:GetResources",
        "tag:GetTagKeys",
        "tag:GetTagValues",
        "tag:TagResources",
        "tag:UntagResources"
    ],
    "Resource": "*"
}
```

Note

The launch role also needs tagging permissions on the specific resources in the artifact, such as `ec2:CreateTags`.

Monitoring in AWS Service Catalog

You can monitor your AWS Service Catalog resources using Amazon CloudWatch, which collects and processes raw data from AWS Service Catalog into readable metrics. These statistics are recorded for a period of two weeks, so that you can access historical information and gain a better perspective on how your service is performing. AWS Service Catalog metric data is automatically sent to CloudWatch in 1-minute periods. For more information about CloudWatch, see the Amazon CloudWatch User Guide.

For a list of available metrics and dimensions, see AWS Service Catalog CloudWatch Metrics.

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Service Catalog and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. Before you start monitoring AWS Service Catalog, you should create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

Monitoring Tools

AWS provides various tools that you can use to monitor AWS Service Catalog. You can configure some of these tools to do the monitoring for you, while some of the tools require manual intervention. We recommend that you automate monitoring tasks as much as possible.

Automated Monitoring Tools

You can use Amazon CloudWatch alarms to monitor AWS Service Catalog and report disruptions.

CloudWatch alarms watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS)

topic or Amazon EC2 Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods. To learn how to create an alarm, see Creating Amazon CloudWatch Alarms. For more information on using Amazon CloudWatch metrics with AWS Service Catalog, see AWS Service Catalog CloudWatch Metrics.

AWS Service Catalog CloudWatch Metrics

You can monitor your AWS Service Catalog resources using Amazon CloudWatch, which collects and processes raw data from AWS Service Catalog into readable metrics. These statistics are recorded for a period of two weeks, so that you can access historical information and gain a better perspective on how your service is performing. AWS Service Catalog metric data is automatically sent to CloudWatch in 1-minute periods. For more information about CloudWatch, see the Amazon CloudWatch User Guide.

Topics

- Enabling CloudWatch Metrics
- Available Metrics and Dimensions
- Viewing AWS Service Catalog Metrics

Enabling CloudWatch Metrics

Amazon CloudWatch metrics are enabled by default.

Available Metrics and Dimensions

The metrics and dimensions that AWS Service Catalog sends to Amazon CloudWatch are listed below.

AWS Service Catalog Metrics

The `AWS/ServiceCatalog` namespace includes the following metrics.

Metric	Description
`ProvisionedProductLaunch`	The number of provisioned products launched for a given product and provisioning artifact in a specified time period. The dimensions are published as separate records in CloudWatch logs.

Metric	Description
	Units: Count Valid statistics: Minimum, Maximum, Sum, Average Dimensions: State, PPState, ProductId , ProvisioningArtifactId
ProductProvisioningOperation	The number of operations performed on product id, provisioningArtifactId . The dimensions are published as one record in CloudWatch logs. Units: Count Valid statistics: Minimum, Maximum, Sum, Average Dimensions: State, PPState, ProductId , ProvisioningArtifactId

Dimensions for AWS Service Catalog Metrics

AWS Service Catalog sends the following dimensions to Amazon CloudWatch.

Dimension	Description
PPState	This dimension filters the data you request for all provisioned products launched with this specified state. This helps you categorize your data by the state of launch. Valid State: AVAILABLE, TAINTED, ERROR
ProductId	This dimension filters the data you request for the identified product id only. This helps you to pinpoint an exact product from which to be launched.

Dimension	Description
`ProvisioningArtifactId`	This dimension filters the data you request for the identified provisioning artifact id only. This helps you to pinpoint an exact version of products from which to be launched.
`State`	This dimension filters the data you request for all provisioned products launched with this specified state. This helps you categorize your data by the state of launch. Valid State: SUCCEEDED, FAILED

Viewing AWS Service Catalog Metrics

You can view Amazon CloudWatch metrics in the Amazon CloudWatch console, which provides a fine-grained and customizable display of your resources, as well as the number of running tasks in a service.

Topics

- Viewing AWS Service Catalog Metrics in the Amazon CloudWatch Console

Viewing AWS Service Catalog Metrics in the Amazon CloudWatch Console

You can view AWS Service Catalog metrics in the Amazon CloudWatch console. The Amazon CloudWatch console provides a detailed view of AWS Service Catalog metrics, and you can tailor the views to suit your needs. For more information about Amazon CloudWatch, see the Amazon CloudWatch User Guide.

To view metrics in the Amazon CloudWatch console

1. Open the Amazon CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the **Metrics** section in the left navigation, choose **Service Catalog**.
3. Choose the metrics to view.

Logging AWS Service Catalog API calls using AWS CloudTrail

AWS Service Catalog is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS Service Catalog. CloudTrail captures all API calls for AWS Service Catalog as events. The calls captured include calls from the AWS Service Catalog console and code calls to the AWS Service Catalog API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS Service Catalog. 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 that was made to AWS Service Catalog, the IP address from which the request was made, who made the request, when it was made, and additional details.

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

AWS Service Catalog information in CloudTrail

CloudTrail is enabled on your AWS account when you create it. When activity occurs in AWS Service Catalog, that activity is recorded 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 AWS Service Catalog, 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 AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that 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 the following:

- Overview for creating a trail
- AWS CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for AWS CloudTrail
- Receiving AWS CloudTraillog files from multiple regions and Receiving AWS CloudTrail log files from multiple accounts

CloudTrail logs all AWS Service Catalog actions. For example, calls to the `CreatePortfolio`, `CreateProduct` and `UpdateProvisionedProduct` actions 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:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding AWS Service Catalog log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. 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 aren't an ordered stack trace of the public API calls, so they don't appear in any specific order. The following example shows an CloudTrail log entry that demonstrates the `CreateApplication` API.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "account",
        "arn": "arn:aws:iam::12345789012:user/dev-haw",
        "accountId": "12345789012",
        "accessKeyId": "keyId",
        "userName": "dev-haw"
    },
    "eventTime": "2020-09-23T21:07:58Z",
    "eventSource": "servicecatalog-appregistry.amazonaws.com",
    "eventName": "CreateApplication",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "205.251.233.48",
    "userAgent": "aws-cli/1.18.140 Python/3.6.11
 Linux/4.9.217-0.1.ac.205.84.332.metal1.x86_64 botocore/1.17.63",
    "requestParameters": {
        "name": "hawTestCT",
        "clientToken": "6f36d650-a086-47cf-810a-fbfab2f8ad33"
    },
```

```
    "responseElements": {
        "application": {
            "applicationArn": "arn:aws:servicecatalog:us-
east-1:12345789012:application/app-02ocuq2cie2328pv64ya78e22f",
            "applicationId": "app-02ocuq2cie2328pv64ya78e22f",
            "creationTime": 1600895277.775,
            "lastUpdateTime": 1600895277.775,
            "name": "hawTestCT",
            "tags": {}
        }
    },
    "requestID": "1b6ad353-3b06-421b-bcb4-00075a782762",
    "eventID": "0a2ca224-cdfd-4c4b-a4ed-163218ff5e2d",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "12345789012"
}
```

Console branding preferences

AWS Service Catalog allows administrators to specify console branding preferences for accounts. Administrators can use console branding to specify a company name, logo image, and a primary and secondary (accent) color for a variety of site components. These branding preferences are visible to both administrators and end-users when using the console.

Console branding preferences enhance an account's appearance and accomplish the following:

- Creates a seamless visual transition between the console and internal applications
- Distinguishes accounts used by different internal teams within the same company
- Differentiates accounts across multiple environments, such as development, staging, or production

> **Note**
>
> Administrators specify console branding preferences at the account level.

To specify console branding preferences

1. In the left navigation menu, choose **Preferences**.
2. Choose **Edit** for either the light mode or dark mode branding preferences.
3. Upload a **Logo**, enter a **Brand name**, and then select the **Primary color** and **Secondary color**.
4. Choose **Save**.

For a list of regions where AWS Service Catalog supports console branding, review AWS Region support for console branding.

AWS Region support for console branding preferences

AWS Service Catalog supports console branding preferences in the AWS Regions listed in the table below.

AWS Region name	AWS Region identity
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Milan)	eu-south-1
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1

AWS Region name	AWS Region identity
Middle East (Bahrain)	me-south-1
South America (São Paulo)	sa-east-1
AWS GovCloud (US-East)	us-gov-east-1
AWS GovCloud (US-West)	us-gov-west-1

Document History

The following table describes the important changes to the documentation for AWS Service Catalog. For notification about updates to this documentation, you can subscribe to an RSS feed.

- **API version**: 2014-11-12
- **Latest documentation update**: May 16, 2024

Change	Description	Date
External Engines for AWS Service Catalog	AWS Service Catalog adds new documentation for external engines. External engines are represented through an EXTERNAL product type. The EXTERNAL product type allows for the integration of third-party provisioning engines, such as Terraform. You can use external engines to extend the capabilities of Service Catalog beyond the native AWS CloudFormation templates, enabling the use of other instructure as code (IaC) tools. For more informati on, see External Engines for AWS Service Catalog.	May 16, 2024
Security IAM update	AWS Service Catalog updates the AWSServiceCatalogS yncServiceRolePoli cy policy to change codestar-connections to codeconnections . For	May 7, 2024

more information, see AWS managed policies for AWS Service Catalog AppRegistry.

Earlier Updates

The following table describes the documentation release history of AWS Service Catalog prior to April 25, 2024.

Feature	Description	Release date
AWS Service Catalog	To learn about Hashicorp 's changes to Terraform licensing and updating to the External product type, review Updating existing Terraform Open Source products and provisioned products to the External product type.	October 20, 2023
AWS Service Catalog	To learn about Sharing a portfolio with AWS Organizations and allowing AWS Service Catalog to sync with AWS Organizat ions, see the AWSServic eCatalogOrgsDataSy ncServiceRolePolicy policy and AWSServiceRoleForS erviceCatalogOrgsDataSync service-linked role.	April 14, 2023
AWS Service Catalog	To learn about managing git-connected products and allowing AWS Service Catalog to sync templates in an external repository to	November 18, 2022

Feature	Description	Release date
	your AWS Service Catalog products, see the AWSServiceCatalogSyncServiceRolePolicy policy and AWSServiceRoleForServiceCatalogSync service-linked role.	
AWS Service Catalog AppRegistry	To learn about how AppRegistry helps to store your AWS applications, their associated resource collections, and application attribute groups, see AWS Service Catalog AppRegistry.	June 15, 2022
AWS Service Management Connector	To learn about Connectors for Jira Service Management and ServiceNow, see AWS Service Management Connector.	June 9, 2022
Connector for Jira Service Management	To learn about the updates to the Connector for Jira Service Management, see AWS Service Management Connector for Jira Service Management.	May 25, 2021
Connector for ServiceNow	To learn about the updates to the Connector for ServiceNow, see AWS Service Management Connector for ServiceNow.	April 7, 2021

Feature	Description	Release date
Connector for ServiceNow	To learn about the updates to the Connector for ServiceNow, see AWS Service Management Connector for ServiceNow.	September 24, 2020
AWS Service Quotas	To learn about how AWS Service Catalog works with AWS Service Quotas, see AWS Service Catalog default service quotas.	March 24, 2020
Getting Started Library	To learn about the library of well-architected product templates offered by AWS Service Catalog, see Getting Started Library	March 10, 2020
Version guidance	To learn about product version guidance, see Version Guidance.	December 17, 2019
Connector for Jira Service Desk	To begin using the Connector for Jira Service Desk, see AWS Service Management Connector for Jira Service Desk.	November 21, 2019
Connector for ServiceNow	To learn about the updates to the Connector for ServiceNow, see AWS Service Management Connector for ServiceNow.	November 18, 2019

Feature	Description	Release date
New security chapter	To learn about security in AWS Service Catalog, see Security in AWS Service Catalog.	October 31, 2019
Changing provisioned product owner	To learn about how to change the owner of provisioned products, see Changing Provisioned Product Owner.	October 31, 2019
New resource update constraint	To learn about how to use the RESOURCE_UPDATE constraint to update tags in provisioned products, see AWS Service Catalog Tag Update Constraints.	April 17, 2019
Connector for ServiceNow	To begin using the Connector for ServiceNow, see AWS Service Management Connector for ServiceNow.	March 19, 2019
Support for AWS CloudFormation StackSets	To begin using AWS CloudFormation StackSets, see Using AWS CloudFormation StackSets.	November 14, 2018
Self-service actions	To begin using self-service actions, see AWS CloudFormation Service Actions.	October 17, 2018
Amazon CloudWatch metrics	To learn about Amazon CloudWatch metrics, see AWS Service CatalogAmazon CloudWatch.	September 26, 2018

Feature	Description	Release date
Support for TagOptions	To manage tags, see AWS Service Catalog TagOption Library.	June 28, 2017
Importing a portfolio	To import a portfolio that is shared from another AWS account, see Importing a Portfolio.	February 16, 2016
Updates to permissions information	To grant access to the end user console view, see Console access for end users.	February 16, 2016
Initial release	This is the initial release of the AWS Service Catalog Administrator Guide.	July 9, 2015