User Guide

AWS Proton

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Proton: User Guide

Copyright © 2024 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 AWS Proton? ... 1
Platform teams ... 1
Developers ... 2
Workflow ... 2
Setting up ... 4
Setting up with IAM ... 4
Sign up for AWS ... 5
Create an IAM user ... 5
Service roles ... 6
Setting up with AWS Proton ... 7
Setting up an Amazon S3 bucket ... 7
Setting up an AWS CodeStar connection ... 8
Setting up account CI/CD pipeline settings ... 8
Setting up the AWS CLI ... 11
Getting started ... 12
Prerequisites ... 12
Getting started workflow ... 13
Getting started with the console ... 14
Step 1: Open the AWS Proton console ... 15
Step 2: Prepare to use the example templates ... 15
Step 3: Create an environment template ... 15
Step 4: Create a service template ... 16
Step 5: Create an environment ... 17
Step 6: Optional - Create a service and deploy an application ... 18
Step 7: Clean up. ... 20
Getting started with the CLI ... 21
1. Register an environment template ... 21
2. Register a service template ... 22
3. Deploy an environment ... 23
4. Deploy a service ... 24
5. Clean up ... 26
Template library ... 27
How AWS Proton works ... 28
Objects ... 29

Provisioning methods ... 32
AWS-managed provisioning ... 34
CodeBuild provisioning ... 36
Self-managed provisioning ... 38
AWS Proton terminology ... 41
Template authoring and bundles ... 44
Template bundles ... 44
Parameters ... 46
Parameter types ... 46
Using parameters ... 47
Environment CloudFormation IaC parameters ... 51
Service CloudFormation IaC parameters ... 55
Component CloudFormation IaC parameters ... 58
CloudFormation parameter filters ... 61
CodeBuild provisioning parameters ... 69
Terraform IaC parameters ... 70
Infrastructure as code files ... 71
AWS CloudFormation IaC files ... 72
CodeBuild bundle ... 125
Terraform IaC files ... 131
Schema file ... 138
Environment schema requirements ... 139
Service schema requirements ... 143
Manifest and wrap up ... 146
Environment template bundle wrap up ... 148
Service template bundle wrap up ... 149
Template bundle considerations ... 150
Templates ... 151
Versions ... 152
Publish ... 154
Publish environment templates ... 154
Publish service templates ... 161
View templates ... 170
Update a template ... 174
Delete templates ... 176
Template sync configurations ... 180

Pushing a commit ... 180
Syncing service templates ... 180
Template sync considerations ... 181
Create ... 182
View ... 188
Edit ... 189
Delete ... 191
Service sync configurations ... 191
AWS Proton OPS file ... 192
Create ... 195
View ... 197
Edit ... 198
Delete ... 199
Environments ... 201
IAM Roles ... 201
AWS Proton service role ... 201
Create ... 202
Create and provision in the same account ... 204
Create in one account and provision in another ... 206
Self-managed provisioning ... 211
View ... 214
Update ... 215
Update an AWS managed provisioning environment ... 216
Update a self-managed provisioning environment ... 219
Cancel an environment deployment in progress ... 223
Delete ... 225
Account connections ... 227
Create an environment with environment account connections ... 229
Manage environment account connections ... 230
Customer-managed ... 237
Using customer-managed environments ... 237
CodeBuild provisioning role creation ... 239
Services ... 243
Create ... 243
What's in a service? ... 244
Service templates ... 244

Create a service ... 245
View ... 249
Edit ... 251
Edit service description ... 251
Add or remove service instances ... 253
Delete ... 260
View instances ... 261
Update instance ... 263
Update pipeline ... 269
Components ... 276
Components vs. other resources ... 278
AWS Proton console ... 279
AWS Proton API and AWS CLI ... 280
Component FAQ ... 280
Component states ... 282
Component IaC files ... 283
Using parameters with components ... 283
Authoring robust IaC files ... 283
Component AWS CloudFormation example ... 285
Administrator steps ... 285
Developer steps ... 288
Repositories ... 291
Create a repository link ... 292
View linked repository data ... 294
Delete a repository link ... 296
Monitoring ... 298
Automate AWS Proton with EventBridge ... 298
Event types ... 298
AWS Proton event examples ... 301
EventBridgeTutorial: Send Amazon Simple Notification Service alerts for AWS Proton service status changes ... 302
Prerequisites ... 303
Step 1: Create and subscribe to an Amazon SNS topic ... 303
Step 2: Register an event rule ... 303
Step 3: Test your event rule ... 305
Step 4: Clean up ... 306

AWS Proton dashboard ... 307
AWS Proton console ... 307
Security ... 310
Identity and Access Management ... 311
Audience ... 311
Authenticating with identities ... 312
Managing access using policies ... 315
How AWS Proton works with IAM ... 317
Policy examples ... 325
AWS managed policies ... 338
Using service-linked roles ... 354
Troubleshooting ... 362
Configuration and vulnerability analysis ... 363
Data protection ... 364
Server side encryption at rest ... 365
Encryption in transit ... 365
AWS Proton encryption key management ... 365
AWS Proton encryption context ... 365
Infrastructure security ... 367
VPC endpoints (AWS PrivateLink) ... 367
Logging and monitoring ... 369
Resilience ... 370
AWS Proton backups ... 370
Security best practices ... 371
Use IAM to control access ... 371
Do not embed credentials in your templates and template bundles ... 371
Use encryption to protect sensitive data ... 372
Use AWS CloudTrail to view and log API calls ... 372
Cross-service confused deputy prevention ... 372
Codebuild custom support ... 373
Updating the Environment Template ... 374
Tagging ... 378
AWS tagging ... 378
AWS Proton tagging ... 379
AWS Proton AWS managed tags ... 379
Tag propagation to provisioned resources ... 380

Customer managed tags 383
Create tags using the console and CLI 383
Create tags using the AWS Proton AWS CLI 385
Troubleshooting 386
Deployment errors that reference AWS CloudFormation dynamic parameters 386
AWS Proton quotas 388
Document history 389
AWS Glossary 394

What is AWS Proton?

AWS Proton is:

- **Automated infrastructure as code provisioning and deployment of serverless and container-based applications**

  The AWS Proton service is a two-pronged automation framework. As an administrator, you create *versioned service templates* that define standardized infrastructure and deployment tooling for serverless and container-based applications. As an application developer, you can select from the available *service templates* to automate your application or service deployments.

  AWS Proton identifies all existing *service instances* that are using an outdated template version for you. As an administrator, you can request AWS Proton to upgrade them with one click.
- **Standardized infrastructure**

  Platform teams can use AWS Proton and versioned infrastructure as code templates. They can use these templates to define and manage standard application stacks that contain the architecture, infrastructure resources, and the CI/CD software deployment pipeline.
- **Deployments integrated with CI/CD**

  When developers use the AWS Proton self-service interface to select a *service template*, they're selecting a standardized application stack definition for their code deployments. AWS Proton automatically provisions the resources, configures the CI/CD pipeline, and deploys the code into the defined infrastructure.

AWS Proton for platform teams

As an administrator, you or members of your platform team, create *environment templates* and *service templates* containing infrastructure as code. The *environment template* defines shared infrastructure used by multiple applications or resources. The *service template* defines the type of infrastructure that's needed to deploy and maintain a single application or microservice in an *environment*. An AWS Proton *service* is an instantiation of a *service template*, which normally includes several *service instances* and a *pipeline*. An AWS Proton *service instance* is an instantiation of a *service template* in a specific *environment*. You or others in your team can specify which *environment templates* are compatible with a given *service template*. For more information about *templates*, see AWS Proton templates.

You can use the following infrastructure as code providers with AWS Proton:

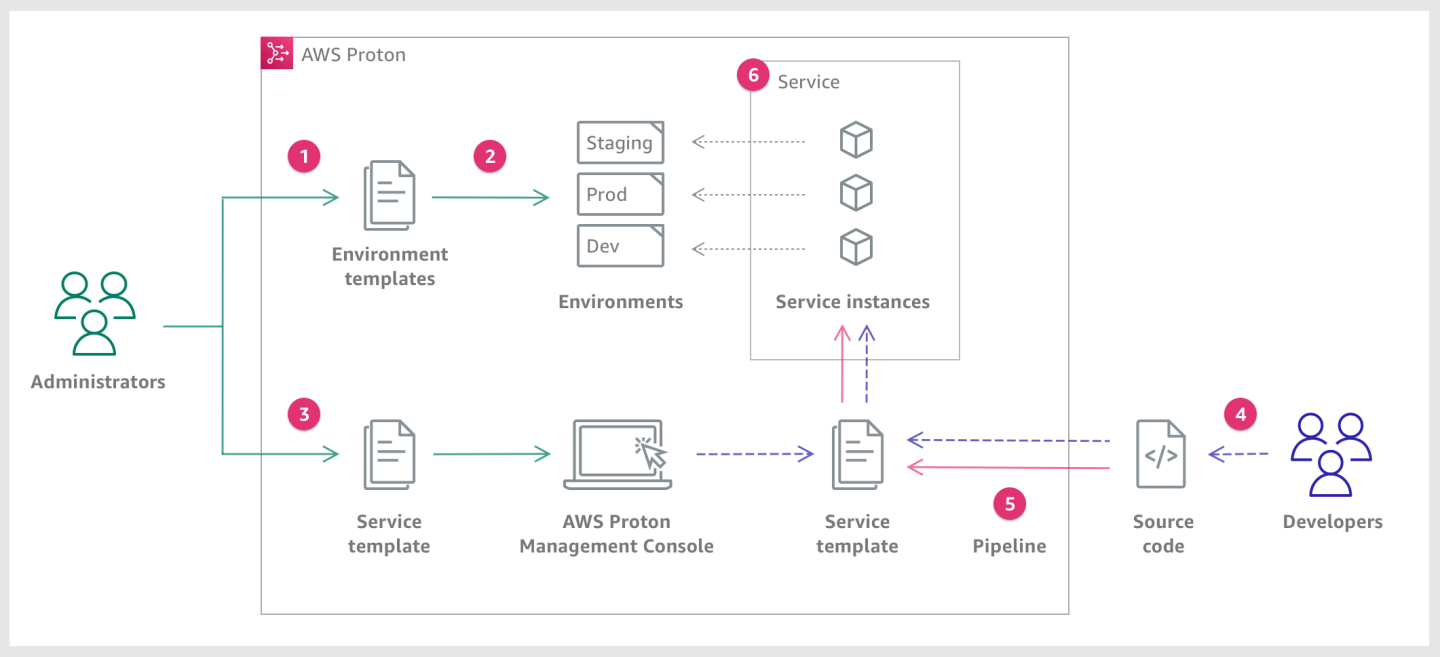
- AWS CloudFormation
- Terraform

AWS Proton for developers

As an application developer, you select a standardized *service template* that AWS Proton uses to create a *service* that deploys and manages your application in a *service instance*. An AWS Proton *service* is an instantiation of a *service template*, which normally includes several *service instances* and a *pipeline*.

AWS Proton workflow

The following diagram is a visualization of the main AWS Proton concepts discussed in the preceding paragraph. It also offers a high-level overview of what constitutes a simple AWS Proton workflow.



1 As an **Administrator**, you create and register an **Environment Template** with AWS Proton, which defines the shared resources.

2

AWS

Proton deploys one or more **Environments**, based on an **Environment Template**.

3

As an **Administrator**, you create and register a **Service Template** with AWS Proton, which defines the related infrastructure, monitoring, and CI/CD resources as well as compatible **Environment Templates**.

4

As a **Developer**, you select a registered **Service Template** and provide a link to your **Source code** repository.

5

AWS Proton provisions the **Service** with a **CI/CD Pipeline** for your **Service instances**.

6

AWS Proton provisions and manages the **Service** and the **Service Instances** that are running the **Source code** as was defined in the selected **Service Template**. A **Service Instance** is an instantiation of the selected **Service Template** in an **Environment** for a single stage of a **Pipeline** (for example Prod).

Setting up

Complete the tasks in this section so that you can create and register service and environment templates. You need these to deploy environments and services with AWS Proton.

> **Note**
>
> We're offering AWS Proton at no additional expense. You can create, register, and maintain service and environment templates at no charge. You can also count on AWS Proton to self-manage its own operations, such as storage, security, and deployment. The only expenses that you incur while using AWS Proton are the following.
>
> - Costs of deploying and using AWS Cloud resources that you instructed AWS Proton to deploy and maintain for you.
> - Costs of maintaining an AWS CodeStar connection to your code repository.
> - Costs of maintaining an Amazon S3 bucket, if you use a bucket to provide inputs to AWS Proton. You can avoid these costs if you switch to the section called "Template sync configurations" using Git repositories for your the section called "Template bundles".

Topics

- Setting up with IAM
- Setting up with AWS Proton

Setting up with IAM

When you sign up for AWS, your AWS account is automatically signed up for all services in AWS, including AWS Proton. You're charged only for the services and resources that you use.

> **Note**
>
> You and your team, including administrators and developers, must all be under the same account.

Sign up for AWS

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.

Create an IAM user

To create an administrator user, choose one of the following options.

Choose one way to manage your administrator	To	By	You can also
In IAM Identity Center (Recommended)	Use short-term credentials to access AWS. This aligns with the security best practices . For information about best practices , see Security best	Following the instructions in Getting started in the *AWS IAM Identity Center User Guide*.	Configure programmatic access by Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*.

Choose one way to manage your administrator	To	By	You can also
	practices in IAM in the *IAM User Guide*.		
In IAM (Not recommended)	Use long-term credentials to access AWS.	Following the instructions in Creating your first IAM admin user and user group in the *IAM User Guide*.	Configure programmatic access by Managing access keys for IAM users in the *IAM User Guide*.

Setting up AWS Proton service roles

There are a few IAM roles that you might want to create for different parts of your AWS Proton solution. You can create them in advance using the IAM console, or you can use the AWS Proton console to create them for you.

Create AWS Proton *environment roles* to allow AWS Proton to make API calls to other AWS services, like AWS CloudFormation, AWS CodeBuild, and various compute and storage services, on your behalf to provision resources for you. A *AWS-managed provisioning role* is required when an environment or any of the service instances running in it use AWS-managed provisioning. A *CodeBuild role* is required when an environment or any of its service instances use CodeBuild provisioning. To learn more about the AWS Proton environment roles, see the section called "IAM Roles". When you create an environment, you can use the AWS Proton console to choose an existing role for either of these two roles, or to create a role with administrative privileges for you.

Similarly, create AWS Proton *pipeline roles* to allow AWS Proton to make API calls to other services on your behalf to provision a CI/CD pipeline for you. To learn more about the AWS Proton pipeline roles, see the section called "Pipeline service roles". For more information about configuring CI/CD settings, see the section called "Setting up account CI/CD pipeline settings".

> **Note**
>
> Because we don't know which resources you will define in your AWS Proton templates, the roles that you create using the console have broad permissions and can be used as both the AWS Proton pipeline service roles and the AWS Proton service roles. For production deployments, we recommend that you scope down the permissions to the specific resources that will be deployed by creating customized policies for both the AWS Proton pipeline service roles and the AWS Proton environment service roles. You can create and customize these roles by using the AWS CLI or IAM. For more information, see Service roles for AWS Proton and Create a service.

Setting up with AWS Proton

If you want to use the AWS CLI to run AWS Proton APIs, verify that you have installed it. If you haven't installed it, see Setting up the AWS CLI.

AWS Proton specific configuration:

- **To create and manage templates:**
  - If you're using template sync configurations, set up an AWS CodeStar connection.
  - Otherwise, set up an Amazon S3 bucket.
- **To provision infrastructure:**
  - For self-managed provisioning, you must set up an AWS CodeStar connection.
- **(Optional) To provision pipelines:**
  - For AWS-managed provisioning and CodeBuild-based provisioning, set up pipeline roles.
  - For self-managed provisioning, set up a pipeline repository.

For more information about provisioning methods, see the section called "AWS-managed provisioning".

Setting up an Amazon S3 bucket

To set up an S3 bucket, follow the instructions at Create your first S3 bucket to set up an S3 bucket. Place your inputs to AWS Proton in the bucket where AWS Proton can retrieve them. These

inputs are known as template bundles. You can learn more about them in other sections of this guide.

Setting up an AWS CodeStar connection

To connect AWS Proton to a repository, you create an AWS CodeStar connection that activates a pipeline when a new commit is made on a third-party source code repository.

AWS Proton uses the connection to:

- Activate a service pipeline when a new commit is made on your repository source code.
- Make a pull request on an infrastructure as code repository.
- Create a new template minor or major version whenever a commit is pushed to a template repository that changes one of your templates, if the version doesn't already exist.

You can connect to Bitbucket, GitHub, GitHub Enterprise and GitHub Enterprise Server repositories with CodeConnections. For more information, see CodeConnections in the *AWS CodePipeline User Guide*.

To set up a CodeStar connection.

1. Open the AWS Proton console.
2. In the navigation pane, select **Settings** and then **Repository connections** to take you to the **Connections** page in **Developer Tools Settings**. The page displays a list of connections.
3. Choose **Create connection** and follow the instructions.

Setting up account CI/CD pipeline settings

AWS Proton can provision CI/CD pipelines for deploying application code into your service instances. The AWS Proton settings you need for pipeline provisioning depend on the provisioning method you choose for your pipeline.

AWS-managed and CodeBuild-based provisioning—set up pipeline roles

With AWS-managed provisioning and CodeBuild provisioning, AWS Proton provisions pipelines for you. Therefore, AWS Proton needs a service role that provides permissions for provisioning pipelines. Each one of these two provisioning methods uses its own service role. These roles are

shared across all AWS Proton service pipelines and you configure them once in your account settings.

To create pipeline service roles using the console

1. Open the AWS Proton console.
2. In the navigation pane, choose **Settings**, and then choose **Account settings**.
3. In the **Account CI/CD settings** page, choose **Configure**.
4. Do one of the following:
   - **To have AWS Proton create a pipeline service role for you**

     [To enable AWS-managed provisioning of pipelines] In the **Configure account settings** page, in the **AWS-managed provisioning pipeline role** section:

     a. Select **New service role**.

     b. Enter a name for the role, for example, **`myProtonPipelineServiceRole`**.

     c. Check the check box to agree to create an AWS Proton role with administrative privileges in your account.

     [To enable CodeBuild-based provisioning of pipelines] In the **Configure account settings** page, in the **CodeBuild pipeline role** section, choose **Existing service role**, and choose the service role that you created in the **CloudFormation pipeline role** section. Or, if you did not assign a CloudFormation pipeline role, repeat the previous three steps to create a new service role.

   - **To choose existing pipeline service roles**

     [To enable AWS-managed provisioning of pipelines] In the **Configure account settings** page, in the **AWS-managed provisioning pipeline role** section, choose **Existing service role**, and choose a service role in your AWS account.

     [To enable CodeBuild provisioning of pipelines] In the **Configure account settings** page, in the **CodeBuild pipeline provisioning role** section, choose **Existing service role**, and choose a service role in your AWS account.
5. Choose **Save changes**.

   Your new pipeline service role is displayed on the **Account settings** page.

Self-managed provisioning—set up a pipeline repository

With self-managed provisioning, AWS Proton sends a pull request (PR) to a provisioning repository that you have set up, and your automation code is responsible for provisioning pipelines. Therefore, AWS Proton doesn't need a service role to provision pipelines. Instead, it needs a registered provisioning repository. Your automation code in the repository has to assume an appropriate role that provides permissions for provisioning pipelines.

To register a pipeline provisioning repository using the console

1. Create a CI/CD pipeline provisioning repository if you haven't yet created one. For more information about pipelines in self-managed provisioning, see the section called "Self-managed provisioning".
2. In the navigation pane, choose **Settings**, and then choose **Account settings**.
3. In the **Account CI/CD settings** page, choose **Configure**.
4. In the **Configure account settings** page, in the **CI/CD pipeline repository** section:
   a. Select **New repository**, and then choose one of the repository providers.
   b. For **CodeStar connection**, choose one of your connections.

      > **Note**
      > If you don't yet have a connection to the relevant repository provider account, choose **Add a new CodeStar connection**, complete the connection creation process, and then choose the refresh button next to the **CodeStar connection** menu. You should now be able to choose your new connection in the menu.

   c. For **Repository name**, choose your pipeline provisioning repository. The drop-down menu shows the list of repositories in the provider account.
   d. For **Branch name**, choose one of the repository branches.
5. Choose **Save changes**.

   Your pipeline repository is displayed on the **Account settings** page.

Setting up the AWS CLI

To use the AWS CLI to make AWS Proton API calls, verify that you have installed the latest version of the AWS CLI. For more information, see Getting started with the AWS CLI in the *AWS Command Line Interface User Guide*. Then, to get started using the AWS CLI with AWS Proton, see the section called "Getting started with the CLI".

Getting started with AWS Proton

Before getting started, get set up to use AWS Proton and verify you have met the Getting started prerequisites.

Get started with AWS Proton by choosing one or more of the following paths:

- Follow a guided example console or CLI workflow through documentation links.
- Run through a guided example console workflow.
- Run through a guided example AWS CLI workflow.

Topics

- Prerequisites
- Getting started workflow
- Getting started with the AWS Management Console
- Getting started with the AWS CLI
- The AWS Proton template library

Prerequisites

Before you start using AWS Proton, make sure that the following prerequisites are met. For more information, see Setting up.

- You have an IAM account with administrator permissions. For more information, see Setting up with IAM.
- You have the AWS Proton service role and the AWS Proton pipeline service role are attached to your account. For more information, see Setting up AWS Proton service roles and Service roles for AWS Proton.
- You have an AWS CodeStar connection. For more information, see Setting up an AWS CodeStar connection.
- You're familiar with creating AWS CloudFormation templates and Jinja parameterization. For more information, see What is AWS CloudFormation? *in the AWS CloudFormation User Guide* and Jinja website.

- You have working knowledge of AWS infrastructure services.
- You're logged into your AWS account.

Getting started workflow

Learn to create template bundles, create and register templates, and create environments and services by following the example steps and links.

Before starting, verify that you created an AWS Proton service role.

If your service template includes an AWS Proton service pipeline, verify that you created an AWS CodeStar connection and a AWS Proton pipeline service role.

For more information, see The AWS Proton service API Reference.

Example: Getting started workflow

1. Refer to the diagram in How AWS Proton works for a high-level view of AWS Proton inputs and outputs.
2. Create an environment bundle and a service template bundle.
   a. Identify input parameters.
   b. Create a schema file.
   c. Create infrastructure as code (IaC) files.
   d. To wrap up your template bundle, create a manifest file and organize your IaC files, manifest files, and schema file in directories.
   e. Make your template bundle accessible to AWS Proton.
3. Create and register an environment template version with AWS Proton.

   When you use the console to create and register a template, a template version is automatically created.

   When you use the AWS CLI to create and register a template:

   a. Create an environment template.
   b. Create an environment template version.

For more information, see CreateEnvironmentTemplate and CreateEnvironmentTemplateVersion in the *AWS Proton API reference*.

4. Publish your environment template to make it available for use.

   For more information, see UpdateEnvironmentTemplateVersion in the *AWS Proton API reference*.

5. To create an environment, select a published environment template version and provide values for required inputs.

   For more information, see CreateEnvironment in the *AWS Proton API reference*.

6. Create and register a service template version with AWS Proton.

   When you use the console to create and register a template, a template version is automatically created.

   When you use the AWS CLI to create and register a template:

   a. Create a service template.

   b. Create a service template version.

   For more information, see CreateServiceTemplate and CreateServiceTemplateVersion in the *AWS Proton API reference*.

7. Publish your service template to make it available for use.

   For more information, see UpdateServiceTemplateVersion in the *AWS Proton API reference*.

8. To create a service, select a published service template version and provide values for required inputs.

   For more information, see CreateService in the *AWS Proton API reference*.

Getting started with the AWS Management Console

Get started with AWS Proton

- Create and view an environment template.

- Create, view, and publish a service template that uses the environment template that you just created.
- Create an environment and service (optional).
- Delete the service template, environment template, environment and service, if created.

Step 1: Open the AWS Proton console

- Open the AWS Proton console

Step 2: Prepare to use the example templates

1. Create a Codestar Connection to Github and name the connection my-proton-connection.
2. Navigate to https://github.com/aws-samples/aws-proton-cloudformation-sample-templates
3. Create a fork of the repository in your Github account.

Step 3: Create an environment template

In the navigation pane, choose **Environment templates**.

1. In the **Environment templates** page, choose **Create Environment template**.
2. In the **Create environment template** page, in the **Template options** section, choose **Create a template for provisioning new environments**.
3. In the **Template bundle source** section, choose **Sync a template bundle from Git**.
4. In the **Template definition repository** section, select **Choose a linked Git repository**.
5. Select **my-proton-connection** from the **Repository list**.
6. Select **main** from the **Branch list**.
7. In the **Proton environment template details** section.
   a. Enter the template name as **`fargate-env`**.
   b. Enter the environment template display name as **`My Fargate Environment`**.
   c. (Optional) Enter a description for the environment template.
8. (Optional) In the **Tags** section, choose **Add new tag** and enter a key and value to create a customer managed tag.

9. Choose **Create Environment template**.

   You're now on a new page that displays the status and details for your new environment template. These details include a list of AWS and customer managed tags. AWS Proton automatically generates AWS managed tags for you when you create AWS Proton resources. For more information, see AWS Proton resources and tagging.

10. The status of a new environment template status starts in the **Draft** state. You and others with `proton:CreateEnvironment` permissions can view and access it. Follow the next step to make the template available to others.
11. In the **Template versions** section, choose the radio button to the left of the minor version of the template you just created (1.0). As an alternative, you can choose **Publish** in the info alert banner and skip the next step.
12. In the **Template versions** section, choose **Publish**.
13. The template status changes to **Published**. Because it's the latest version of the template, it's the **Recommended** version.
14. In the navigation pane, select **Environment templates**.

    A new page displays a list of your environment templates along with template details.

Step 4: Create a service template

Create a service template.

1. In the navigation pane, choose **Service templates**.
2. In the **Service templates** page, choose **Create Service template**.
3. In the **Create service template** page, in the **Template bundle source** section, choose **Sync a template bundle from Git**.
4. In the **Template** section, select **Choose a linked Git repository**.
5. Select **my-proton-connection** from the **Repository list**.
6. Select **main** from the **Branch list**.
7. In the **Proton service template details** section.

   a. Enter the service template name as **`backend-fargate-svc`**.

   b. Enter the service template display name as **`My Fargate Service`**.

   c. (Optional) Enter a description for the service template.

8. In the **Compatible environment templates** section.
   - Check the check-box to the left of the environment template **My Fargate Environment** to select the compatible environment template for the new service template.
9. For **Encryption settings**, keep the defaults.
10. In the **Pipeline definition** section.
    - Keep the **This template includes a CI/CD pipeline** button selected.
11. Choose **Create service template**.

    You're now on a new page that displays the status and details for your new service template, including a list of AWS and customer managed tags.
12. The status of a new service template status starts in the **Draft** state. Only administrators can view and access it. To make the service template available for use by developers, follow the next step.
13. In the **Template versions** section, choose the radio button to the left of the minor version of the template you just created (1.0). As an alternative, you can choose **Publish** in the info alert banner and skip the next step.
14. In the **Template versions** section, choose **Publish**.
15. The template status changes to **Published**.

    The first minor version of your service template is published and available for use by developers. Because it's the latest version of the template, it's the **Recommended** version.
16. In the navigation pane, choose **Service templates**.

    A new page displays a list of your service templates and details.

Step 5: Create an environment

In the navigation pane, choose **Environments**.

1. Choose **Create environment**.
2. In the **Choose an environment template** page, select the template that you just created. It's named **My Fargate Environment**. Then, choose **Configure**.
3. In the **Configure environment** page, in the **Provisioning** section, choose **Provision through AWS Proton**.

4. In the **Deployment account** section, select **This AWS account**.
5. In **Environment Settings**, enter the environment name as **`my-fargate-environment`**.
6. In the **Environment roles** section, select **New service role** or, if you have already created an AWS Proton service role, select **Existing service role**.

   a. Select **New service role** to create a new role.

      i. Enter the **Environment role name** as **`MyProtonServiceRole`**.

      ii. Check the check box to agree to create an AWS Proton service role with administrative privileges for your account.

   b. Select **Existing service role** to use an existing role.

      - Select your role in the **Environment role name** drop down field.
7. Choose **Next**.
8. On the **Configure custom settings** page, use the defaults.
9. Choose **Next** and review your inputs.
10. Choose **Create**.

    View the environment details and status, as well as the AWS managed tags and customer managed tags for your environment.
11. In the navigation pane, choose **Environments**.

    A new page displays a list of your environments along with the status and other environment details.

Step 6: Optional - Create a service and deploy an application

1. Open the AWS Proton console.
2. In the navigation pane, choose **Services**.
3. In the **Services** page, choose **Create service**.
4. In the **Choose a service template** page, select the **My Fargate Service** template by choosing the radio button at the top-right corner of the template card.
5. Choose **Configure** at the lower right corner of the page.
6. In the **Configure service** page, in the **Service settings** section, enter the service name **`my-service`**.

7. (Optional) Enter a description for the service.
8. **In the Service repository settings section:**
   a. For **CodeStar connection**, choose your connection from the list.
   b. For **Repository name**, choose the name of your source code repository from the list.
   c. For **Branch name**, choose the name of your source code repository branch from the list.
9. (Optional) In the **Tags** section, choose **Add new tag** and enter a key and value to create a customer managed tag. Then choose **Next**.
10. In the **Configure custom settings** page, in the **Service instances** section, in the **New instance** section, follow the next steps to provide custom values for your service instance parameters.
    a. Enter the instance name **`my-app-service`**.
    b. Choose the environment **`my-fargate-environment`** for your service instance.
    c. Keep the defaults for the remaining instance parameters.
    d. Keep the defaults for **Pipeline inputs**.
    e. Choose **Next** and review your inputs.
    f. Choose **Create** and view your service status and details.
11. In the service details page, view the status of your service instance and pipeline by choosing the **Overview** and **Pipeline** tabs. On these pages you can also view AWS and customer managed tags. AWS Proton automatically creates AWS managed tags for you. Choose **Manage tags** to create and modify customer managed tags. For more information about tagging, see AWS Proton resources and tagging.
12. After the service is **Active**, in the **Overview** tab, in the **Service Instances** section, choose the name of your service instance, **my-app-service**.

    You are now on the service instance detail page.
13. To view your application, in the **Outputs** section, copy the **ServiceEndpoint** link to your browser.

    You see an AWS Proton graphic in the web page.
14. After the service is created, in the navigation pane, choose **Services** to view a list of your services.

Step 7: Clean up.

1. Open the AWS Proton console.
2. **Delete a service (if you created one)**
   a. In the navigation pane, choose **Services**.
   b. In the **Services** page, choose the service name **my-service**.

      You're now on the service detail page for **my-service**.
   c. In the upper right-hand corner of the page, choose **Actions** and then **Delete**.
   d. A modal prompts you to confirm the delete action.
   e. Follow the instructions and choose **Yes, delete**.
3. **Delete an environment**
   a. In the navigation pane, choose **Environments**.
   b. In the **Environments** page, select the radio button the left of the environment that you just created.
   c. Choose **Actions**, then **Delete**.
   d. A modal prompts you to confirm the delete action.
   e. Follow the instructions and choose **Yes, delete**.
4. **Delete a service template**
   a. In the navigation pane, choose **Service templates**.
   b. In the **Service templates** page, select the radio button to the left of service template **my-svc-template**.
   c. Choose **Actions**, then **Delete**.
   d. A modal prompts you to confirm the delete action.
   e. Follow the instructions and choose **Yes, delete**. This deletes the service template and all of its versions.
5. **Delete an environment template**
   a. In the navigation pane, choose **Environment templates**.
   b. In the **Environment templates** page, select the radio button to the left of **my-env-template**.
   c. Choose **Actions**, then **Delete**.

d. A modal prompts you to confirm the delete action.

e. Follow the instructions and choose **Yes, delete**. This deletes the environment template and all of its versions.

6. **Delete your Codestar Connection**

Getting started with the AWS CLI

To get started with AWS Proton using the AWS CLI, follow this tutorial. The tutorial demonstrates a public facing load-balanced AWS Proton service based on AWS Fargate. The tutorial also provisions a CI/CD pipeline that deploys a static website with a displayed image.

Before you start, be sure you are set up correctly. For details, see the section called "Prerequisites".

Step 1: Register an environment template

In this step, as an administrator, you register an example environment template, which contains an Amazon Elastic Container Service (Amazon ECS) cluster and an Amazon Virtual Private Cloud (Amazon VPC) with two public/private subnets.

To register an environment template

1. Fork the AWS Proton Sample CloudFormation Templates repository into your GitHub account or organization. This repository includes the environment and service templates that we use in this tutorial.

   Then, register your forked repository with AWS Proton. For more information, see the section called "Create a repository link".

2. Create an environment template.

   The environment template resource tracks environment template versions.

```
$ aws proton create-environment-template \
  --name "fargate-env" \
  --display-name "Public VPC Fargate" \
  --description "VPC with public access and ECS cluster"
```

3. Create a template sync configuration.

AWS Proton sets up a sync relationship between your repository and your environment template. It then creates template version 1.0 in DRAFT status.

```
$ aws proton create-template-sync-config \
  --template-name "fargate-env" \
  --template-type "ENVIRONMENT" \
  --repository-name "your-forked-repo" \
  --repository-provider "GITHUB" \
  --branch "your-branch" \
  --subdirectory "environment-templates/fargate-env"
```

4. Wait for the environment template version to be successfully registered.

   When this command returns with an exit status of `0`, version registration is complete. This is useful in scripts to ensure you can successfully run the command in the next step.

```
$ aws proton wait environment-template-version-registered \
  --template-name "fargate-env" \
  --major-version "1" \
  --minor-version "0"
```

5. Publish the environment template version to make it available for environment creation.

```
$ aws proton update-environment-template-version \
  --template-name "fargate-env" \
  --major-version "1" \
  --minor-version "0" \
  --status "PUBLISHED"
```

Step 2: Register a service template

In this step, as an administrator, you register an example service template, which contains all the resources required to provision an Amazon ECS Fargate service behind a load balancer and a CI/CD pipeline that uses AWS CodePipeline.

To register a service template

1. Create a service template.

   The service template resource tracks service template versions.

```
$ aws proton create-service-template \
  --name "load-balanced-fargate-svc" \
  --display-name "Load balanced Fargate service" \
  --description "Fargate service with an application load balancer"
```

2. Create a template sync configuration.

   AWS Proton sets up a sync relationship between your repository and your service template. It then creates template version 1.0 in DRAFT status.

```
$ aws proton create-template-sync-config \
  --template-name "load-balanced-fargate-svc" \
  --template-type "SERVICE" \
  --repository-name "your-forked-repo" \
  --repository-provider "GITHUB" \
  --branch "your-branch" \
  --subdirectory "service-templates/load-balanced-fargate-svc"
```

3. Wait for the service template version to be successfully registered.

   When this command returns with an exit status of `0`, version registration is complete. This is useful in scripts to ensure you can successfully run the command in the next step.

```
$ aws proton wait service-template-version-registered \
  --template-name "load-balanced-fargate-svc" \
  --major-version "1" \
  --minor-version "0"
```

4. Publish the service template version to make it available for service creation.

```
$ aws proton update-service-template-version \
  --template-name "load-balanced-fargate-svc" \
  --major-version "1" \
  --minor-version "0" \
  --status "PUBLISHED"
```

Step 3: Deploy an environment

In this step, as an administrator, you instantiate an AWS Proton environment from the environment template.

To deploy an environment

1. Get an example spec file for the environment template that you registered.

   You can download the file `environment-templates/fargate-env/spec/spec.yaml` from the template example repository. Alternatively, you can fetch the entire repository locally and run the **create-environment** command from the `environment-templates/fargate-env` directory.

2. Create an environment.

   AWS Proton reads input values from your environment spec, combines them with your environment template, and provisions environment resources in your AWS account using your AWS Proton service role.

```
$ aws proton create-environment \
  --name "fargate-env-prod" \
  --template-name "fargate-env" \
  --template-major-version 1 \
  --proton-service-role-arn "arn:aws:iam::123456789012:role/AWSProtonServiceRole" \
  --spec "file://spec/spec.yaml"
```

3. Wait for the environment to successfully deploy.

```
$ aws proton wait environment-deployed --name "fargate-env-prod"
```

Step 4: Deploy a service [application developer]

In the previous steps, an administrator registered and published a service template and deployed an environment. As an application developer, you can now create an AWS Proton service and deploy it into the AWS Proton environment

To deploy a service

1. Get an example spec file for the service template that the administrator registered.

   You can download the file `service-templates/load-balanced-fargate-svc/spec/spec.yaml` from the template example repository. Alternatively, you can fetch the entire repository locally and run the **create-service** command from the `service-templates/load-balanced-fargate-svc` directory.

2. Fork the AWS Proton Sample Services repository into your GitHub account or organization. This repository includes the application source code that we use in this tutorial.
3. Create a service.

   AWS Proton reads input values from your service spec, combines them with your service template, and provisions service resources in your AWS account in the environment that is specified in the spec. An AWS CodePipeline pipeline deploys your application code from the repository that you specify in the command.

```
$ aws proton create-service \
  --name "static-website" \
  --repository-connection-arn \
    "arn:aws:codestar-connections:us-east-1:123456789012:connection/your-codestar-connection-id" \
  --repository-id "your-GitHub-account/aws-proton-sample-services" \
  --branch-name "main" \
  --template-major-version 1 \
  --template-name "load-balanced-fargate-svc" \
  --spec "file://spec/spec.yaml"
```

4. Wait for the service to successfully deploy.

```
$ aws proton wait service-created --name "static-website"
```

5. Retrieve outputs and view your new website.

   Run the following command:

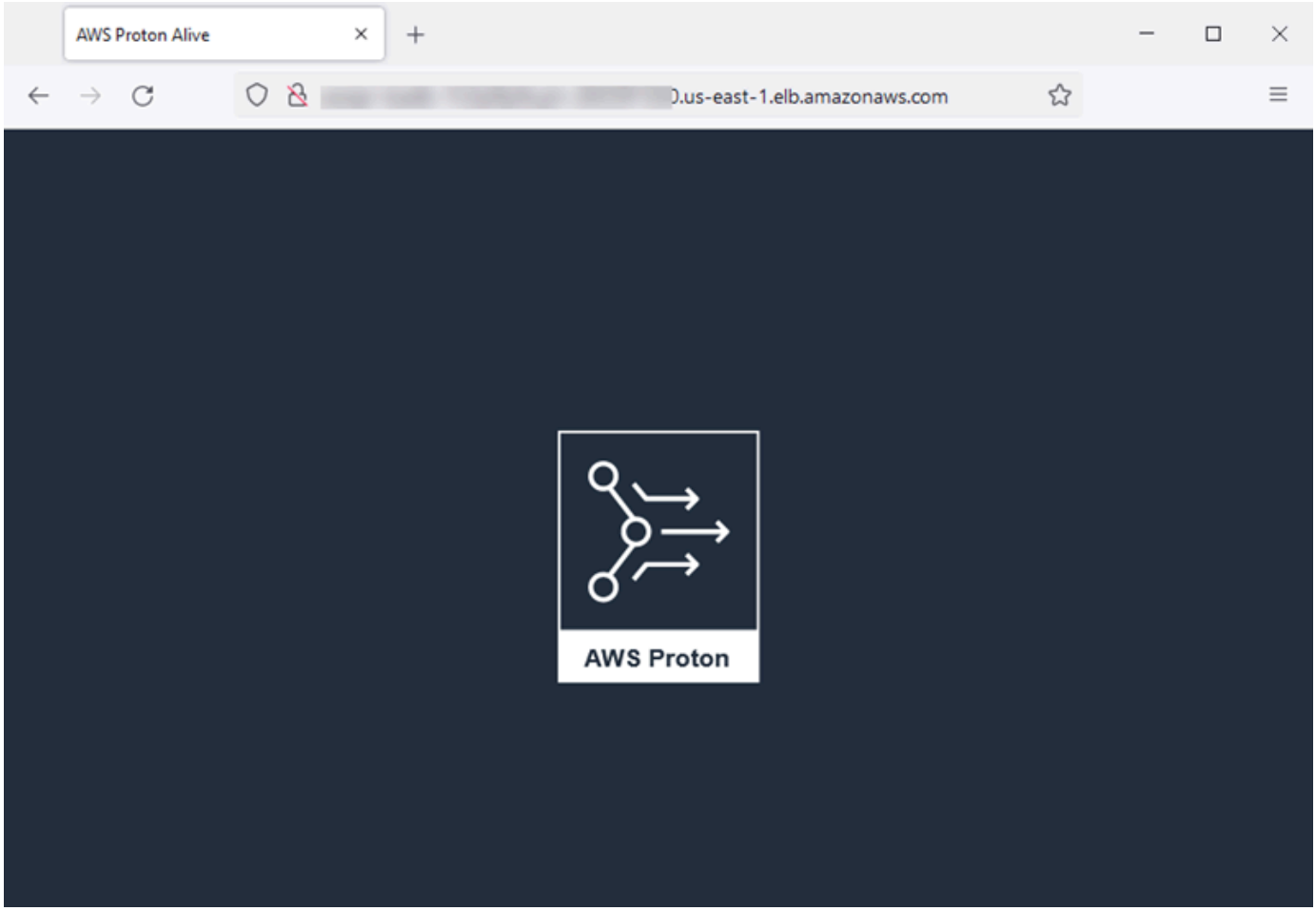
```
$ aws proton list-service-instance-outputs \
  --service-name "static-website" \
  --service-instance-name load-balanced-fargate-svc-prod
```

   The command's output should be similar to the following:

```
{
    "outputs": [
        {
            "key": "ServiceURL",
            "valueString": "http://your-service-endpoint.us-east-1.elb.amazonaws.com"
        }
```

```
    ]
}
```

The value of the `ServiceURL` instance output is the endpoint to your new service website. Use your browser to navigate to it. You should see the following graphic on a static page:



Step 5: Clean up (optional)

In this step, when you're done exploring the AWS resources that you created as part of this tutorial, and to save on costs associated with these resources, you delete them.

To delete tutorial resources

1. To delete the service, run the following command:

```
$ aws proton delete-service --name "static-website"
```

2. To delete the environment, run the following command:

```
$ aws proton delete-environment --name "fargate-env-prod"
```

3. To delete the service template, run the following commands:

```
$ aws proton delete-template-sync-config \
  --template-name "load-balanced-fargate-svc" \
  --template-type "SERVICE"
$ aws proton delete-service-template --name "load-balanced-fargate-svc"
```

4. To delete the environment template, run the following commands:

```
$ aws proton delete-template-sync-config \
  --template-name "fargate-env" \
  --template-type "ENVIRONMENT"
$ aws proton delete-environment-template --name "fargate-env"
```

The AWS Proton template library

The AWS Proton team maintains a library of template examples on GitHub. The library includes examples of infrastructure as code (IaC) files for many common environment and application infrastructure scenarios.

The template library is stored in two GitHub repositories:

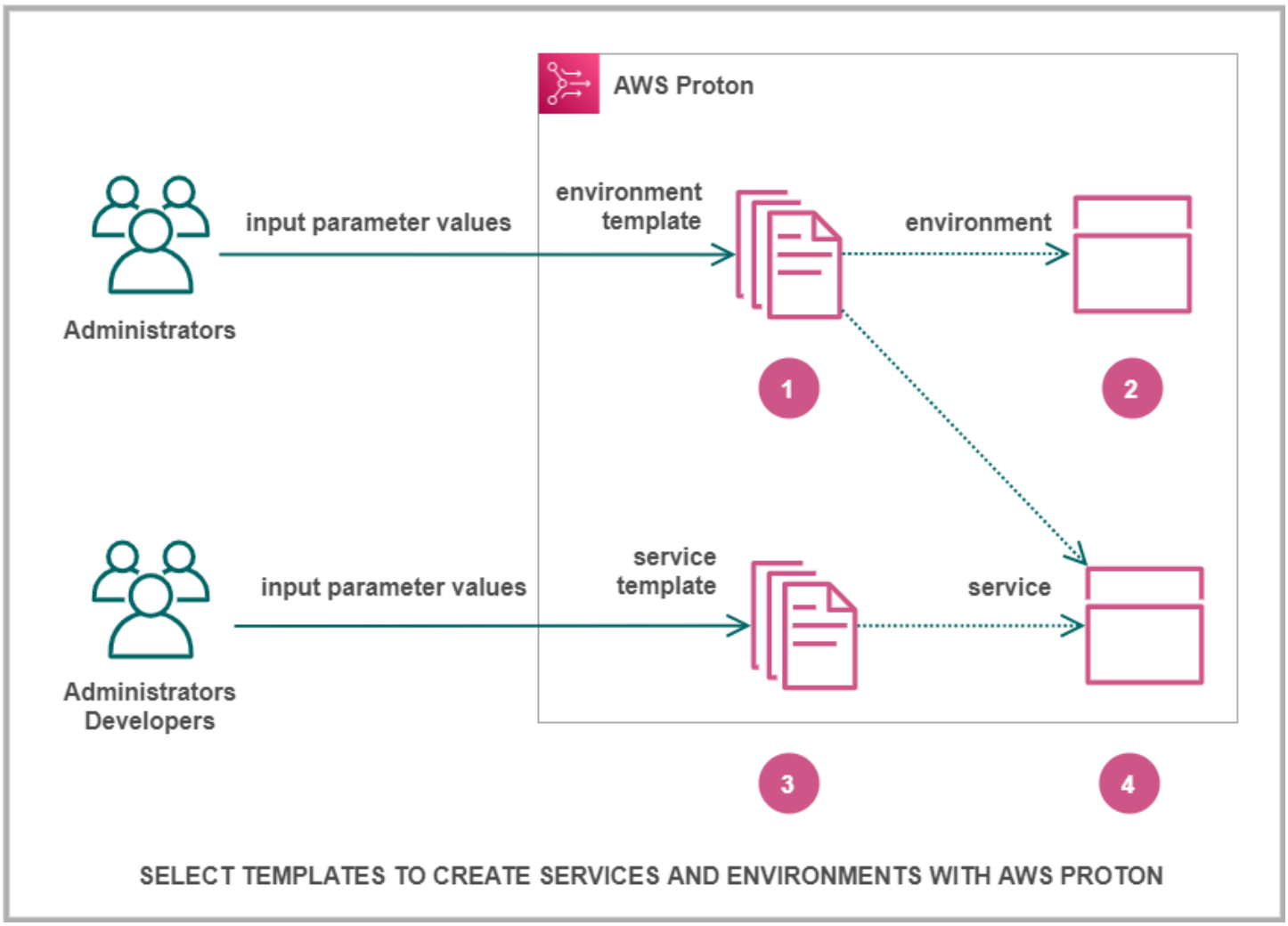
- aws-proton-cloudformation-sample-templates – Examples of template bundles that use *AWS CloudFormation* with Jinja as their IaC language. You can use these examples for AWS-managed provisioning environments.
- aws-proton-terraform-sample-templates – Examples of template bundles that use *Terraform* as their IaC language. You can use these examples for Self-managed provisioning environments.

Each one of these repositories has a `README` file with full information about the repository's content and structure. Each example has information about the use case that the template covers, the example's architecture, and the input parameters that the template takes.

You can use the templates in this library directly by forking one of the library's repositories into your GitHub account. Alternatively, use these examples as a starting point for developing your environment and service templates.

How AWS Proton works

With AWS Proton, you provision *environments*, and then *services* running in those environments. Environments and services are based on environment and service *templates*, respectively, that you choose in your AWS Proton versioned template library.



1

When you, as an administrator, select an environment template with AWS Proton, you provide values for required *input parameters*.

2

AWS Proton uses the environment template and parameter values to provision your environment.

3

When you, as a developer or administrator, select a service template with AWS Proton, you provide values for required input parameters. You also select an environment to deploy your application or service to.

4

AWS Proton uses the service template, and both your service and selected environment parameter values, to provision your service.

You provide values for the input parameters to customize your template for re-use and multiple use cases, applications, or services.

To make this work, you create environment or service template bundles and upload them to registered environment or service templates, respectively.

Template bundles contain everything AWS Proton needs to provision environments or services.

When you create an environment or service template, you upload a template bundle that contains the parametrized infrastructure as code (IaC) files that AWS Proton uses to provision environments or services.

When you select an environment or service template to create or update an environment or service, you provide values for the template bundle IaC file parameters.

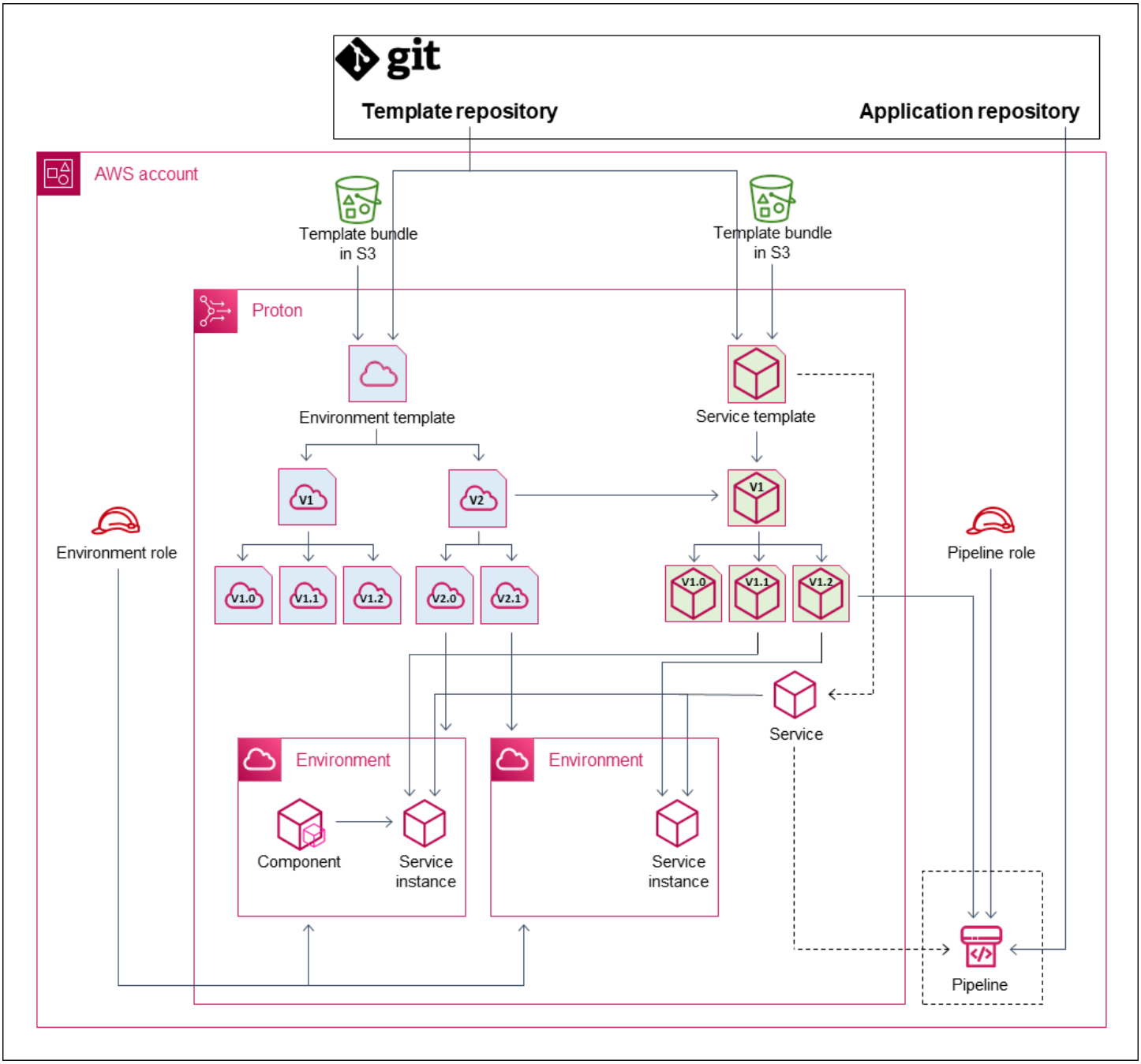
Topics

- AWS Proton objects
- How AWS Proton provisions infrastructure
- AWS Proton terminology

AWS Proton objects

The following diagram shows the main AWS Proton objects and their relationship to other AWS and third-party objects. The arrows represent the direction of data flow (the inverse direction of dependency).

We follow the diagram with brief descriptions and reference links for these AWS Proton objects.



- **Environment template** – A collection of environment template versions that can be used to create AWS Proton environments.

  For more information, see Template authoring and bundles and *Templates*.

- **Environment template version** – A specific version of an environment template. Takes a *template bundle* as input, either from an S3 bucket or from a Git repository. The bundle specifies Infrastructure as Code (IaC) and related input parameters for an AWS Proton environment.

For more information, see the section called "Versions", the section called "Publish", and the section called "Template sync configurations".

- **Environment** – The set of shared AWS infrastructure resources and access policies that AWS Proton services are deployed into. AWS resources are provisioned by using an environment template version invoked with specific parameter values. Access policies are provided in a service role.

  For more information, see *Environments*.

- **Service template** – A collection of service template versions that can be used to create AWS Proton services.

  For more information, see Template authoring and bundles and *Templates*.

- **Service template version** – A specific version of a service template. Takes a *template bundle* as input, either from an S3 bucket or from a Git repository. The bundle specifies Infrastructure as Code (IaC) and related input parameters for an AWS Proton service.

  A service template version also specifies these constraints on service instances based on the version:

  - **Compatible environment templates** – Instances can only run in environments based on these compatible environment templates.
  - **Supported component sources** – The types of components that developers can associate with instances.

  For more information, see the section called "Versions", the section called "Publish", and the section called "Template sync configurations".

- **Service** – A collection of service instances that run an application using resources specified in a service template, and optionally a CI/CD pipeline that deploys the application code into these instances.

  In the diagram, the dashed line from **Service template** means that the service passes the template through to service instances and the pipeline.

  For more information, see *Services*.

- **Service instance** – The set of AWS infrastructure resources that run an application in a specific AWS Proton environment. AWS resources are provisioned by using a service template version invoked with specific parameter values.

For more information, see *Services* and the section called "Update instance".

- **Pipeline** – An optional CI/CD pipeline that deploys an application into the instances of a service, with access policies to provision this pipeline. Access policies are provided in a service role. A service doesn't always have an associated AWS Proton pipeline—you can choose to manage your app code deployments outside of AWS Proton.

  In the diagram, the dashed line from **Service** and the dashed box around **Pipeline** mean that if you choose to manage your CI/CD deployments yourself, the AWS Proton pipeline may not be created, and your own pipeline may not be within your AWS account.

  For more information, see *Services* and the section called "Update pipeline".

- **Component** – A developer-defined extension to a service instance. Specifies additional AWS infrastructure resources that a particular application might need, in addition to the resources provided by the environment and the service instance. Platform teams control the infrastructure that a component can provision by attaching a component role to the environment.

  For more information, see *Components*.

How AWS Proton provisions infrastructure

AWS Proton can provision infrastructure in one of several ways:

- **AWS-managed provisioning** – AWS Proton calls the provisioning engine on your behalf. This method supports only AWS CloudFormation template bundles. For more information, see the section called "AWS CloudFormation IaC files".
- **CodeBuild provisioning** – AWS Proton uses AWS CodeBuild to run shell commands that you provide. Your commands can read inputs that AWS Proton provides, and are responsible for provisioning or deprovisioning infrastructure and generating output values. A template bundle for this method includes your commands in a manifest file and any programs, scripts, or other files that these commands may need.

  As an example to using CodeBuild provisioning, you can include code that uses the AWS Cloud Development Kit (AWS CDK) to provision AWS resources, and a manifest that installs the CDK and runs your CDK code.

  For more information, see the section called "CodeBuild bundle".

> **Note**
>
> You can use CodeBuild provisioning with environments and services. At this time you can't provision components this way.

- **Self-managed provisioning** – AWS Proton issues a pull request (PR) to a repository that you provide, where your own infrastructure deployment system runs the provisioning process. This method supports only Terraform template bundles. For more information, see the section called "Terraform IaC files".

AWS Proton determines and sets the provisioning method for each environment and service separately. When you create or update an environment or a service, AWS Proton examines the template bundle that you provide, and determines the provisioning method that the template bundle indicates. At the environment level, you provide the parameters that the environment and its potential services might need for their provisioning methods—AWS Identity and Access Management (IAM) roles, an environment account connection, or an infrastructure repository.

Developers who use AWS Proton to provision a service have the same experience regardless of provisioning method. Developers don't need to be aware of the provisioning method and don't need to change anything in the service provisioning process. The service template sets the provisioning method, and each environment that a developer deploys the service to provides the necessary parameters for service instance provisioning.

The following diagram summarizes some major traits of the different provisioning methods. The sections that follow the table provide details about each method.

Provisioning method	Templates	Provisioned by	Status tracked by
AWS-managed	manifest, schema, IaC file (CloudFormation)	AWS Proton (through CloudFormation)	AWS Proton (through CloudFormation)
CodeBuild	manifest (with commands), schema, command dependenc ies (e.g. AWS CDK code)	AWS Proton (through CodeBuild)	AWS Proton (your commands return status through CodeBuild)

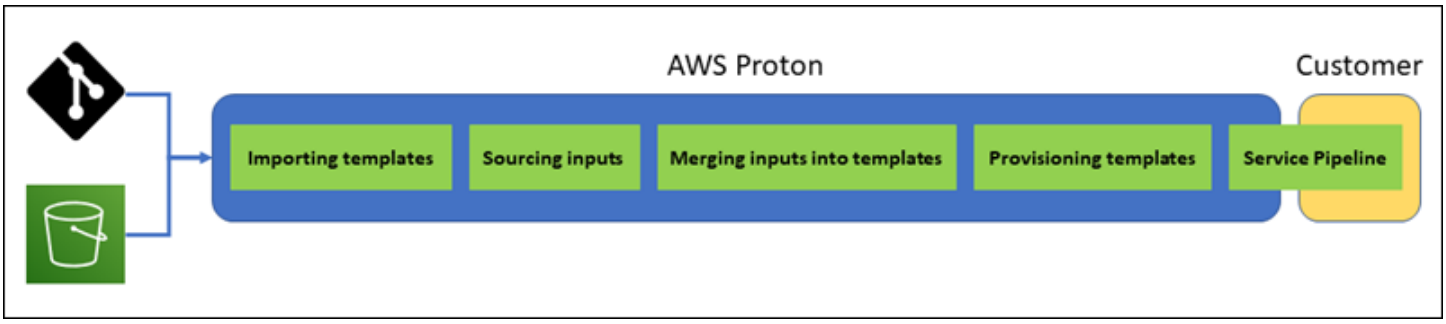
Provisioning method	Templates	Provisioned by	Status tracked by
self-managed	manifest, schema, IaC files (Terraform)	Your code (through Git actions)	Your code (passed to AWS through API call)

How AWS-managed provisioning works

When an environment or a service uses AWS-managed provisioning, infrastructure is provisioned as follows:

1. An AWS Proton customer (an administrator or a developer) creates the AWS Proton resource (an environment or a service). The customer selects a template for the resource and provides the required parameters. For more information, see the following section, the section called "Considerations for AWS-managed provisioning".
2. AWS Proton renders a complete AWS CloudFormation template for provisioning the resource.
3. AWS Proton calls AWS CloudFormation to start provisioning using the rendered template.
4. AWS Proton continuously monitors the AWS CloudFormation deployment.
5. When provisioning completes, AWS Proton reports back errors in case of failure, and captures provisioning outputs, like Amazon VPC ID, in case of success.

The following diagram shows that AWS Proton takes care of most of these steps directly.



Considerations for AWS-managed provisioning

- *Infrastructure provisioning role* – When an environment or any of the service instances running in it might use AWS-managed provisioning, an administrator needs to configure an IAM role (either directly or as part of an AWS Proton environment account connection). AWS Proton uses this role to provision the infrastructure of these AWS-managed provisioning resources. The role should

have permissions to use AWS CloudFormation to create all the resources that the templates of these resources include.

For more information, see the section called “IAM Roles” and the section called “Service role policy examples”.

- *Service provisioning* – When a developer deploys a service instance that uses AWS-managed provisioning to the environment, AWS Proton uses the role provided to that environment to provision infrastructure for the service instance. Developers don't see this role and can't change it.
- *Service with pipeline* – A service template that uses AWS-managed provisioning may include a pipeline definition written in the AWS CloudFormation YAML schema. AWS Proton also creates the pipeline by calling AWS CloudFormation. The role that AWS Proton uses to create a pipeline is separate from the role for each individual environment. This role is provided to AWS Proton separately, only once at the AWS account level, and it's used to provision and manage all AWS-managed pipelines. This role should have permissions to create pipelines and other resources that your pipelines need.

The following procedures show how to provide the pipeline role to AWS Proton.

AWS Proton console

To provide the pipeline role

1. In the AWS Proton console, on the navigation pane, choose **Settings > Account settings**, and then choose **Configure**.
2. Use the **Pipeline AWS-managed role** section to configure a new or existing pipeline role for AWS-managed provisioning.

AWS Proton API

To provide the pipeline role

1. Use the UpdateAccountSettings API action.
2. Provide the Amazon Resource Name (ARN) of your pipeline service role in the `pipelineServiceRoleArn` parameter.

AWS CLI

To provide the pipeline role

Run the following command:

```
$ aws proton update-account-settings \
    --pipeline-service-role-arn \
        "arn:aws:iam::123456789012:role/my-pipeline-role"
```

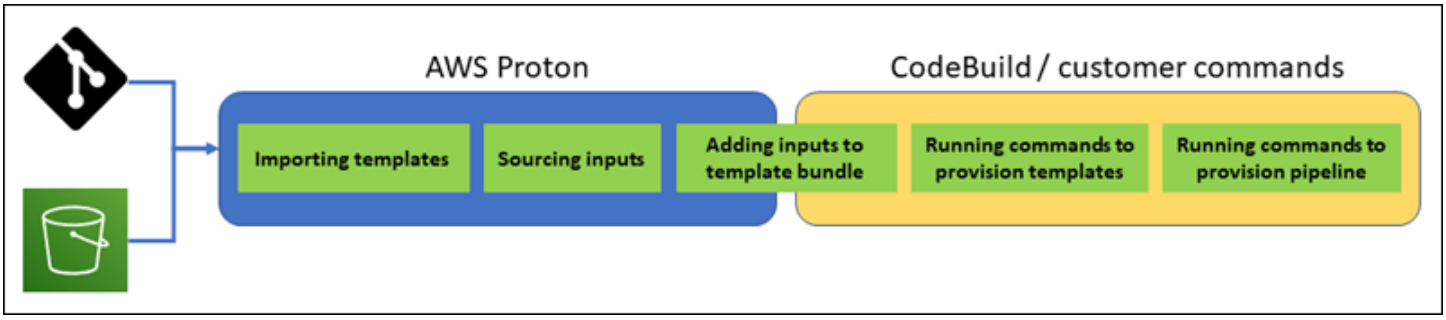
How CodeBuild provisioning works

When an environment or a service uses CodeBuild provisioning, infrastructure is provisioned as follows:

1. An AWS Proton customer (an administrator or a developer) creates the AWS Proton resource (an environment or a service). The customer selects a template for the resource and provides the required parameters. For more information, see the following section, the section called "Considerations for CodeBuild provisioning".
2. AWS Proton renders an input file with input parameter values for provisioning the resource.
3. AWS Proton calls CodeBuild to start a job. The CodeBuild job runs the customer shell commands specified in the template. These commands provision the desired infrastructure, while optionally reading input values.
4. When provisioning completes, the final customer command returns the provisioning status to CodeBuild and calls the NotifyResourceDeploymentStatusChange AWS Proton API action to provide outputs, like Amazon VPC ID, if any exist.

> **Important**
>
> Be sure that your commands correctly return the provisioning status to CodeBuild and provide the outputs. If they don't, AWS Proton can't properly track the provisioning status and can't provide correct outputs to service instances.

The following diagram illustrates the steps that AWS Proton performs and the steps that your commands perform within a CodeBuild job.



Considerations for CodeBuild provisioning

- *Infrastructure provisioning role* – When an environment or any of the service instances running in it might use CodeBuild-based provisioning, an administrator needs to configure an IAM role (either directly or as part of an AWS Proton environment account connection). AWS Proton uses this role to provision the infrastructure of these CodeBuild provisioning resources. The role should have permissions to use CodeBuild to create all the resources that your commands in the templates of these resources provision.

  For more information, see the section called "IAM Roles" and the section called "Service role policy examples".
- *Service provisioning* – When a developer deploys a service instance that uses CodeBuild provisioning to the environment, AWS Proton uses the role provided to that environment to provision infrastructure for the service instance. Developers don't see this role and can't change it.
- *Service with pipeline* – A service template that uses CodeBuild provisioning may include commands to provision a pipeline. AWS Proton also creates the pipeline by calling CodeBuild. The role that AWS Proton uses to create a pipeline is separate from the role for each individual environment. This role is provided to AWS Proton separately, only once at the AWS account level, and it's used to provision and manage all CodeBuild-based pipelines. This role should have permissions to create pipelines and other resources that your pipelines need.

  The following procedures show how to provide the pipeline role to AWS Proton.

  AWS Proton console

  To provide the pipeline role

  1. In the AWS Proton console, on the navigation pane, choose **Settings > Account settings**, and then choose **Configure**.

2. Use the **Codebuild pipeline provisioning role** section to configure a new or existing pipeline role for CodeBuild provisioning.

AWS Proton API

To provide the pipeline role

1. Use the UpdateAccountSettings API action.
2. Provide the Amazon Resource Name (ARN) of your pipeline service role in the `pipelineCodebuildRoleArn` parameter.

AWS CLI

To provide the pipeline role

Run the following command:

```
$ aws proton update-account-settings \
    --pipeline-codebuild-role-arn \
        "arn:aws:iam::123456789012:role/my-pipeline-role"
```

How self-managed provisioning works

When an environment is configured to use self-managed provisioning, infrastructure is provisioned as follows:

1. An AWS Proton customer (an administrator or a developer) creates the AWS Proton resource (an environment or a service). The customer selects a template for the resource and provides the required parameters. For an environment, the customer also provides a linked infrastructure repository. For more information, see the following section, the section called "Considerations for self-managed provisioning".
2. AWS Proton renders a complete Terraform template. It consists of one or more Terraform files, potentially in multiple folders, and a `.tfvars` variables file. AWS Proton writes parameter values provided on the resource creation call into this variables file.
3. AWS Proton submits a PR to the infrastructure repository with the rendered Terraform template.

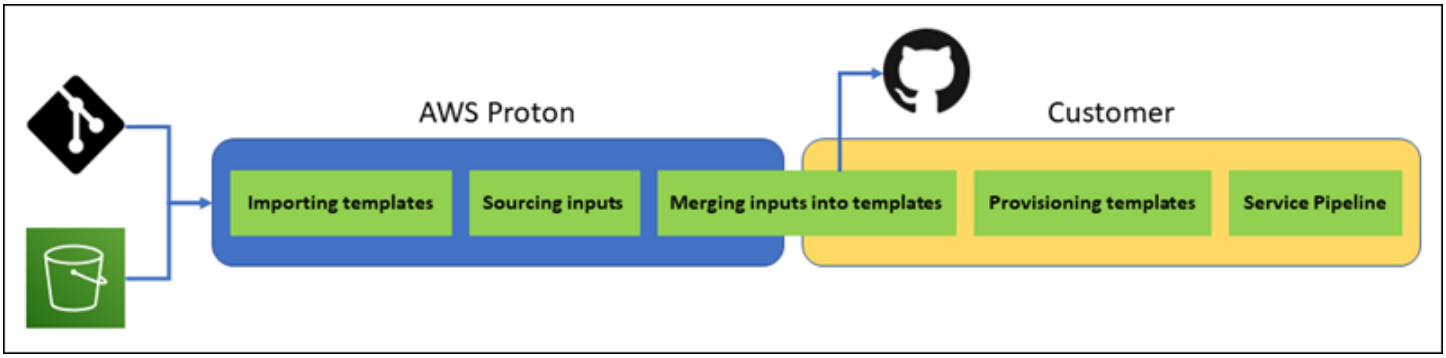
4. When the customer (administrator or developer) merges the PR, the customer's automation triggers the provisioning engine to start provisioning infrastructure using the merged template.

> **Note**
>
> If the customer (administrator or developer) closes the PR, AWS Proton recognizes the PR as closed and marks the deployment as cancelled.

5. When provisioning completes, the customer's automation calls the NotifyResourceDeploymentStatusChange AWS Proton API action to indicate completion, provide the status (success or failure), and provide outputs, like Amazon VPC ID, if any exist.

> **Important**
>
> Be sure that your automation code calls back into AWS Proton with the provisioning status and outputs. If it doesn't, AWS Proton might consider the provisioning as pending for longer than it should, and keep showing **In progress** status.

The following diagram illustrates the steps that AWS Proton performs and the steps that your own provisioning system performs.



Considerations for self-managed provisioning

- *Infrastructure repository* – When an administrator configures an environment for self-managed provisioning, they need to provide a linked infrastructure repository. AWS Proton submits PRs to this repository to provision the infrastructure of the environment and all the service instances that are deployed to it. The customer-owned automation action in the repository should assume an IAM role with permissions to create all the resources that your environment and service templates include, and an identity that reflects the destination AWS account. For an example

GitHub Action that assumes a role, see Assuming a Role in the *"Configure AWS Credentials" Action For GitHub Actions* documentation.

- *Permissions* – Your provisioning code has to authenticate with an account as necessary (for example, authenticate to an AWS account) and provide resource provisioning authorization (for example, provide a role).
- *Service provisioning* – When a developer deploys a service instance that uses self-managed provisioning to the environment, AWS Proton submits a PR to the repository that is associated with the environment to provision infrastructure for the service instance. Developers don't see the repository and can't change it.

> **Note**
>
> Developers creating services use the same process regardless of provisioning method, and the difference is abstracted from them. However, with self-managed provisioning developers might experience slower response, because they need to wait until someone (which might not be themselves) merges the PR in the infrastructure repository before provisioning can start.

- *Service with pipeline* – A service template for an environment with self-managed provisioning may include a pipeline definition (for example, an AWS CodePipeline pipeline), written in Terraform HCL. To enable AWS Proton to provision these pipelines, an administrator provides a linked pipeline repository to AWS Proton. When provisioning a pipeline, the customer-owned automation action in the repository should assume an IAM role with permissions to provision the pipeline, and an identity that reflects the destination AWS account. The pipeline repository and role are separate from those used for each individual environment. The linked repository is provided to AWS Proton separately, only once at the AWS account level, and it's used to provision and manage all pipelines. The role should have permissions to create pipelines and other resources that your pipelines need.

  The following procedures show how to provide the pipeline repository and role to AWS Proton.

  AWS Proton console

  To provide the pipeline role

  1. In the AWS Proton console, on the navigation pane, choose **Settings > Account settings**, and then choose **Configure**.
  2. Use the **CI/CD pipeline repository** section to configure a new or existing repository link.

AWS Proton API

To provide the pipeline role

1. Use the UpdateAccountSettings API action.
2. Provide the provider, name, and branch of your pipeline repository in the `pipelineProvisioningRepository` parameter.

AWS CLI

To provide the pipeline role

Run the following command:

```
$ aws proton update-account-settings \
    --pipeline-provisioning-repository \
        "provider=GITHUB,name=my-pipeline-repo-name,branch=my-branch"
```

- *Deletion of self-managed provisioned resources* – Terraform modules may include configuration elements that are necessary for Terraform operation, in addition to resource definitions. Therefore, AWS Proton can't delete all the Terraform files for an environment or service instance. Instead, AWS Proton marks the files for deletion and updated a flag in the PR metadata. Your automation can read that flag and use it to trigger a terraform destroy command.

AWS Proton terminology

Environment template

Defines shared infrastructure, such as a VPC or cluster, that is used by multiple applications or resources.

Environment template bundle

A collection of files that you upload to create and register an environment template in AWS Proton. An environment template bundle contains the following:

1. A schema file that defines infrastructure as code input parameters.
2. An infrastructure as code (IaC) file that defines shared infrastructure, such as a VPC or cluster, that is used by multiple applications or resources.

3. A manifest file that lists the IaC file.

Environment

Provisioned shared infrastructure, such as a VPC or cluster, that is used by multiple applications or resources.

Service template

Defines the type of infrastructure that's needed to deploy and maintain an application or microservice in an environment.

Service template bundle

A collection of files that you upload to create and register a service template in AWS Proton. A service template bundle contains the following:

1. A schema file that defines infrastructure as code (IaC) input parameters.
2. An IaC file that defines the infrastructure that's needed to deploy and maintain an application or microservice in an environment.
3. A manifest file that lists the IaC file.
4. Optional
   a. An IaC file that defines the service pipeline infrastructure.
   b. A manifest file that lists the IaC file.

Service

Provisioned infrastructure that's needed to deploy and maintain an application or microservice in an environment.

Service instance

Provisioned infrastructure that supports an application or microservice in an environment.

Service pipeline

Provisioned infrastructure that supports a pipeline.

Template version

A major or minor version of a template. For more information, see Versioned templates.

Input parameters

Defined in a schema file and used in an infrastructure as code (IaC) file so that the IaC file can be used repeatably and for a variety of use cases.

Schema file

Defines infrastructure as code file input parameters.

Spec file

Specifies values for infrastructure as code file input parameters, as defined in a schema file.

Manifest file

Lists an infrastructure as code file.

Authoring templates and creating bundles for AWS Proton

AWS Proton provisions resources for you based on infrastructure as code (IaC) files. You describe infrastructure in reusable IaC files. To make the files reusable for different environments and applications, you author them as *templates*, define input parameters, and use these parameters in IaC definitions. When you later create a provisioning resource (environment, service instance, or component), AWS Proton uses a rendering engine, which combines input values with a template to create an IaC file that is ready to provision.

Administrators author most templates as *template bundles*, and then upload and register them into AWS Proton. The remainder of this page discusses these AWS Proton template bundles. *Directly defined components* are an exception—developers create them and provide IaC template files directly. For more information about components, see *Components*.

Topics

- Template bundles
- AWS Proton parameters
- AWS Proton infrastructure as code files
- Schema file
- Wrap up template files for AWS Proton
- Template bundle considerations

Template bundles

As an administrator, you create and register templates with AWS Proton. You use these templates to create environments and services. When you create a service, AWS Proton provisions and deploys service instances to selected environments. For more information, see AWS Proton for platform teams.

To create and register a template in AWS Proton, you upload a template bundle that contains the infrastructure as code (IaC) files that AWS Proton needs to provision and environment or service.

A *template bundle* contains the following:

- An Infrastructure as code (IaC) file with a manifest YAML file that lists the *IaC file*.
- A schema YAML file for your IaC file input parameter definitions.

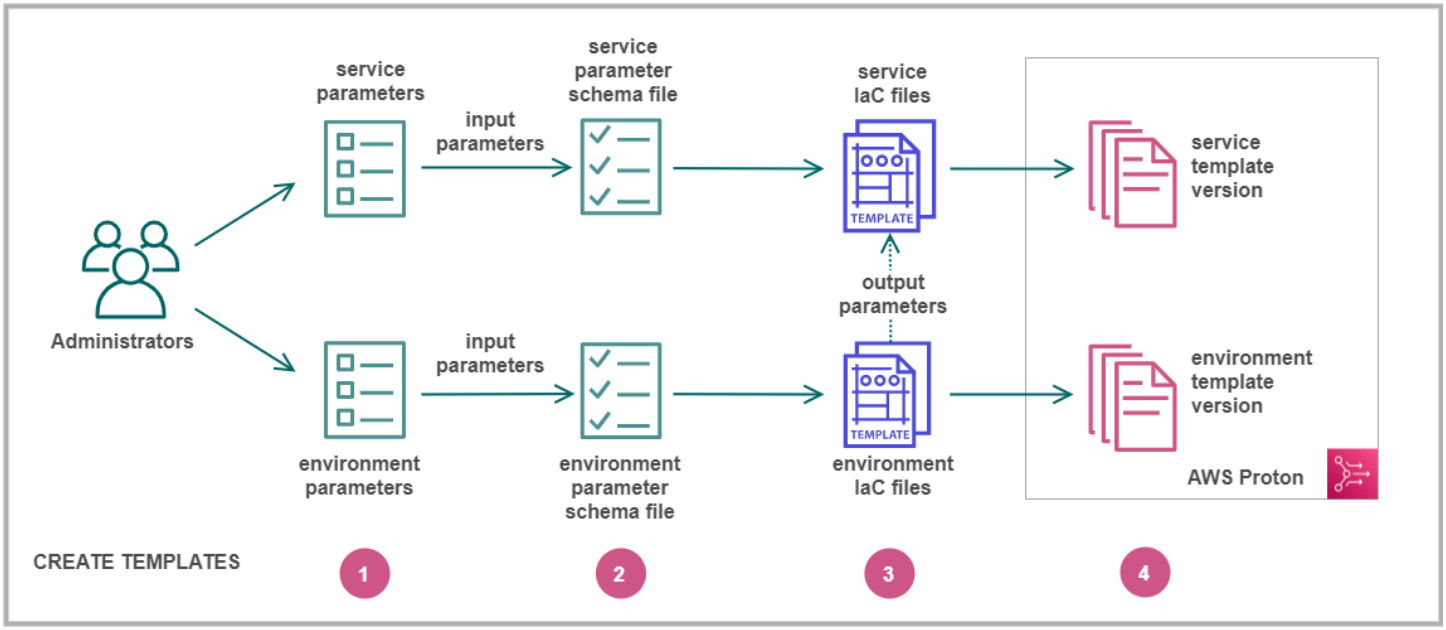
A CloudFormation environment template bundle contains one IaC file.

A CloudFormation service template bundle contains one IaC file for service instance definitions and another optional IaC file for a pipeline definition.

Terraform environment and service template bundles can contain multiple IaC files each.

AWS Proton requires an input parameter schema file. When you use AWS CloudFormation to create your IaC files, you use Jinja syntax to reference your input parameters. AWS Proton provides parameter namespaces that you can use to reference parameters in your IaC files.

The following diagram shows an example of steps that you can take to create a *template* for AWS Proton.



1

Identify input parameters.

2

Create a schema file to define your input parameters.

3

Create IaC files that reference your input parameters. You can reference environment IaC file *outputs* as *inputs* for your service IaC files.

4

Register a template version with AWS Proton and upload your template bundle.

AWS Proton parameters

You can define and use parameters in your infrastructure as code (IaC) files to make them flexible and re-usable. You read a parameter value in your IaC files by referring to the parameter's name in the AWS Proton *parameter namespace*. AWS Proton injects parameter values into the rendered IaC files that it generates during resource provisioning. To process AWS CloudFormation IaC parameters, AWS Proton uses Jinja. To process Terraform IaC parameters, AWS Proton generates a Terraform parameter value file and relies on the parametrization ability built into HCL.

With CodeBuild provisioning, AWS Proton generates an input file that your code can import. The file is a JSON or HCL file, depending on a property in your template's manifest. For more information, see the section called “CodeBuild provisioning parameters”.

You can refer to parameters in your environment, service, and component IaC files or provisioning code with the following requirements:

- The length of each parameter name doesn't exceed 100 characters.
- The length of the parameter namespace and resource name combined doesn't exceed the character limit for the resource name.

AWS Proton provisioning fails if these quotas are exceeded.

Parameter types

The following parameter types are available to you for reference in AWS Proton IaC files:

Input parameter

Environments and service instances can take input parameters that you define in a schema file that you associate with the environment or service template. You can refer to a resource's input

parameters in the resource's IaC file. Component IaC files can refer to input parameters of the service instance that the component is attached to.

AWS Proton checks input parameter names against your schema file, and matches them with the parameters that are referenced in your IaC files to inject the input values that you provide in a spec file during resource provisioning.

Output parameter

You can define outputs in any of your IaC files. An output can be, for example, a name, ID, or ARN of one of the resources that the template provisions, or it can be a way to pass through one of the template's inputs. You can refer to these outputs in IaC files of other resources.

In CloudFormation IaC files,define output parameters in the `Outputs:` block. In a Terraform IaC file, define each output parameter using an `output` statement.

Resource parameter

AWS Proton automatically creates AWS Proton resource parameters. These parameters expose properties of the AWS Proton resource object. An example of a resource parameter is `environment.name`.

Using AWS Proton parameters in your IaC files

To read a parameter value in an IaC file, you refer to the parameter's name in the AWS Proton parameter namespace. For AWS CloudFormation IaC files, you use *Jinja* syntax and surround the parameter with pairs of curly braces and quotation marks.

The following table shows the reference syntax for each supported template language, with an example.

Template language	Syntax	Example: environment input named "VPC"
CloudFormation	`"{{ parameter-name }}"`	`"{{ environment.inputs.VPC }}"`
Terraform	`var.parameter-name`	`var.environment.inputs.VPC` Generated Terraform variable definitions

> **Note**
>
> If you use CloudFormation dynamic parameters in your IaC file, you must escape them to prevent Jinja misinterpretation errors. For more information, see Troubleshooting AWS Proton

The following table lists namespace names for all AWS Proton resource parameters. Each template file type can use a different subset of the parameter namespace.

Template file	Parameter type	Parameter name	Description
Environment	resource	`environment.` **`name`**	Environment name
	input	`environment.inputs.` *`input-name`*	Schema-defined environment inputs
Service	resource	`environment.` **`name`** `environment.` **`account_id`**	Environment name and AWS account ID
	output	`environment.outputs.` *`output-name`*	Environment IaC file outputs
	resource	`service.`**`branch_name`** `service.`**`name`** `service.`**`repository_connection_arn`** `service.`**`repository_id`**	Service name and code repository
	resource	`service_instance.` **`name`**	Service instance name
	input	`service_instance.inputs.` *`input-name`*	Schema-defined service instance inputs

Template file	Parameter type	Parameter name	Description
	resource	`service_instance.components.default.` **`name`**	Attached default component name
	output	`service_instance.components.default.outputs.` *`output-name`*	Attached default component IaC file outputs
Pipeline	resource	`service_instance.environment.` **`name`** `service_instance.environment.` **`account_id`**	Service instance environment name and AWS account ID
	output	`service_instance.environment.outputs.` *`output-name`*	Service instance environment IaC file outputs
	input	`pipeline.inputs.` *`input-name`*	Schema-defined pipeline inputs
	resource	`service.`**`branch_name`** `service.`**`name`** `service.`**`repository_connection_arn`** `service.`**`repository_id`**	Service name and code repository
	input	`service_instance.inputs.` *`input-name`*	Schema-defined service instance inputs
	collection	`{% for service_instance in` **`service_instances`** `%}...{% endfor %}`	A collection of service instances that you can loop through

Template file	Parameter type	Parameter name	Description
Component	resource	`environment.` **`name`** `environment.` **`account_id`**	Environment name and AWS account account ID
	output	`environment.outputs.` *`output-name`*	Environment IaC file outputs
	resource	`service.`**`branch_name`** `service.`**`name`** `service.`**`repository_connection_arn`** `service.`**`repository_id`**	Service name and code repository (attached components)
	resource	`service_instance.` **`name`**	Service instance name (attached components)
	input	`service_instance.inputs.` *`input-name`*	Schema-defined service instance inputs (attached components)
	resource	`component.` **`name`**	Component name

For more information and examples, see the subtopics about parameters in IaC template files for different resource types and template languages.

Topics

- Environment CloudFormation IaC file parameter details and examples
- Service CloudFormation IaC file parameter details and examples
- Component CloudFormation IaC file parameter details and examples
- Parameter filters for CloudFormation IaC files
- CodeBuild provisioning parameter details and examples

- Terraform infrastructure as code (IaC) file parameter details and examples

Environment CloudFormation IaC file parameter details and examples

You can define and reference parameters in your environment infrastructure as code (IaC) files. For a detailed description of AWS Proton parameters, parameter types, the parameter namespace, and how to use parameters in your IaC files, see the section called "Parameters".

Define environment parameters

You can define both input and output parameters for environment IaC files.

- **Input parameters** – Define environment input parameters in your schema file.

  The following list includes examples of environment input parameters for typical use cases.

  - VPC CIDR values
  - Load balancer settings
  - Database settings
  - A health check timeout

  As an administrator, you can provide values for input parameters when you create an environment:

  - Use the console to fill out a schema-based form that AWS Proton provides.
  - Use the CLI to provide a spec that includes the values.
- **Output parameters** – Define environment outputs in your environment IaC files. You can then refer to these outputs in IaC files of other resources.

Read parameter values in environment IaC files

You can read parameters related to the environment in environment IaC files. You read a parameter value by referencing the parameter's name in the AWS Proton parameter namespace.

- **Input parameters** – Read an environment input value by referencing `environment.inputs.`*`input-name`*.
- **Resource parameters** – Read AWS Proton resource parameters by referencing names such as `environment.name`.

> **Note**
>
> No output parameters of other resources are available to environment IaC files.

Example environment and service IaC files with parameters

The following example demonstrates parameter definition and reference in an environment IaC file. The example then shows how environment output parameters defined in the environment IaC file can be referenced in a service IaC file.

Example Environment CloudFormation IaC file

Note the following in this example:

- The `environment.inputs.` namespace refers to environment input parameters.
- The Amazon EC2 Systems Manager (SSM) parameter `StoreInputValue` concatenates the environment inputs.
- The `MyEnvParameterValue` output exposes the same input parameter concatenation as an output parameter. Three additional output parameters also expose the input parameters individually.
- Six additional output parameters expose resources that the environment provisions.

```
Resources:
  StoreInputValue:
    Type: AWS::SSM::Parameter
    Properties:
      Type: String
      Value: "{{ environment.inputs.my_sample_input }}
 {{ environment.inputs.my_other_sample_input}}
 {{ environment.inputs.another_optional_input }}"
              # input parameter references

# These output values are available to service infrastructure as code files as outputs,
 when given the
# the 'environment.outputs' namespace, for example,
 service_instance.environment.outputs.ClusterName.
Outputs:
  MyEnvParameterValue:                                          # output definition
    Value: !GetAtt StoreInputValue.Value
```

```
  MySampleInputValue:                                          # output definition
    Value: "{{ environment.inputs.my_sample_input }}"          #   input parameter
 reference
  MyOtherSampleInputValue:                                     # output definition
    Value: "{{ environment.inputs.my_other_sample_input }}"    #   input parameter
 reference
  AnotherOptionalInputValue:                                   # output definition
    Value: "{{ environment.inputs.another_optional_input }}"   #   input parameter
 reference
  ClusterName:                                                 # output definition
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'                                   #   provisioned resource
  ECSTaskExecutionRole:                                        # output definition
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'                  #   provisioned resource
  VpcId:                                                       # output definition
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'                                          #   provisioned resource
  PublicSubnetOne:                                             # output definition
    Description: Public subnet one
    Value: !Ref 'PublicSubnetOne'                              #   provisioned resource
  PublicSubnetTwo:                                             # output definition
    Description: Public subnet two
    Value: !Ref 'PublicSubnetTwo'                              #   provisioned resource
  ContainerSecurityGroup:                                      # output definition
    Description: A security group used to allow Fargate containers to receive traffic
    Value: !Ref 'ContainerSecurityGroup'                       #   provisioned resource
```

Example Service CloudFormation IaC file

The `environment.outputs.` namespace refers to environment outputs from an environment IaC file. For example, the name `environment.outputs.ClusterName` reads the value of the `ClusterName` environment output parameter.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy a service on AWS Fargate, hosted in a public subnet, and accessible
 via a public load balancer.
Mappings:
  TaskSize:
    x-small:
      cpu: 256
      memory: 512
    small:
```

```
      cpu: 512
      memory: 1024
    medium:
      cpu: 1024
      memory: 2048
    large:
      cpu: 2048
      memory: 4096
    x-large:
      cpu: 4096
      memory: 8192
Resources:
  # A log group for storing the stdout logs from this service's containers
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: '{{service_instance.name}}' # resource parameter

  # The task definition. This is a simple metadata description of what
  # container to run, and what resource requirements it has.
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: '{{service_instance.name}}' # resource parameter
      Cpu: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, cpu] # input
 parameter
      Memory: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, memory]
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn: '{{environment.outputs.ECSTaskExecutionRole}}'  # output
 reference to an environment infrastructure code file
      TaskRoleArn: !Ref "AWS::NoValue"
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'  # resource parameter
          Cpu: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, cpu]
          Memory: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, memory]
          Image: '{{service_instance.inputs.image}}'
          PortMappings:
            - ContainerPort: '{{service_instance.inputs.port}}' # input parameter
          LogConfiguration:
            LogDriver: 'awslogs'
            Options:
              awslogs-group: '{{service_instance.name}}' # resource parameter
```

```
              awslogs-region: !Ref 'AWS::Region'
              awslogs-stream-prefix: '{{service_instance.name}}' # resource parameter

  # The service_instance. The service is a resource which allows you to run multiple
  # copies of a type of task, and gather up their logs and metrics, as well
  # as monitor the number of running tasks and replace any that have crashed
  Service:
    Type: AWS::ECS::Service
    DependsOn: LoadBalancerRule
    Properties:
      ServiceName: '{{service_instance.name}}'  # resource parameter
      Cluster: '{{environment.outputs.ClusterName}}' # output reference to an
 environment infrastructure as code file
      LaunchType: FARGATE
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 75
      DesiredCount: '{{service_instance.inputs.desired_count}}'# input parameter
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups:
            - '{{environment.outputs.ContainerSecurityGroup}}' # output reference to an
 environment infrastructure as code file
          Subnets:
            - '{{environment.outputs.PublicSubnetOne}}' # output reference to an
 environment infrastructure as code file
            - '{{environment.outputs.PublicSubnetTwo}}' # output reference to an
 environment infrastructure as code file
      TaskDefinition: !Ref 'TaskDefinition'
      LoadBalancers:
        - ContainerName: '{{service_instance.name}}'  # resource parameter
          ContainerPort: '{{service_instance.inputs.port}}' # input parameter
          TargetGroupArn: !Ref 'TargetGroup'
[...]
```

Service CloudFormation IaC file parameter details and examples

You can define and reference parameters in your service and pipeline infrastructure as code (IaC) files. For a detailed description of AWS Proton parameters, parameter types, the parameter namespace, and how to use parameters in your IaC files, see the section called “Parameters”.

Define service parameters

You can define both input and output parameters for service IaC files.

- **Input parameters** – Define service instance input parameters in your schema file.

  The following list includes examples of service input parameters for typical use cases.

  - Port
  - Task size
  - Image
  - Desired count
  - Docker file
  - Unit test command

  You provide values for input parameters when you create a service:

  - Use the console to fill out a schema-based form that AWS Proton provides.
  - Use the CLI to provide a spec that includes the values.
- **Output parameters** – Define service instance outputs in your service IaC files. You can then refer to these outputs in IaC files of other resources.

Read parameter values in service IaC files

You can read parameters related to the service and to other resources in service IaC files. You read a parameter value by referencing the parameter's name in the AWS Proton parameter namespace.

- **Input parameters** – Read a service instance input value by referencing `service_instance.inputs.`*`input-name`*.
- **Resource parameters** – Read AWS Proton resource parameters by referencing names such as `service.name`, `service_instance.name`, and `environment.name`.
- **Output parameters** – Read outputs of other resources by referencing `environment.outputs.`*`output-name`* or `service_instance.components.default.outputs.`*`output-name`*.

Example service IaC file with parameters

The following example is a snippet from a service CloudFormation IaC file. The `environment.outputs.` namespace refers to outputs from the environment IaC file. The `service_instance.inputs.` namespace refers to service instance input parameters. The `service_instance.name` property refers to an AWS Proton resource parameter.

```
Resources:
  StoreServiceInstanceInputValue:
    Type: AWS::SSM::Parameter
    Properties:
      Type: String
      Value: "{{ service.name }} {{ service_instance.name }}
 {{ service_instance.inputs.my_sample_service_instance_required_input }}
 {{ service_instance.inputs.my_sample_service_instance_optional_input }}
 {{ environment.outputs.MySampleInputValue }}
 {{ environment.outputs.MyOtherSampleInputValue }}"
              #  resource parameter references              # input parameter
 references
                                             # output references to an environment
 infrastructure as code file
Outputs:
  MyServiceInstanceParameter:                                                #
 output definition
    Value: !Ref StoreServiceInstanceInputValue
  MyServiceInstanceRequiredInputValue:                                       #
 output definition
    Value: "{{ service_instance.inputs.my_sample_service_instance_required_input }}"  #
 input parameter reference
  MyServiceInstanceOptionalInputValue:                                       #
 output definition
    Value: "{{ service_instance.inputs.my_sample_service_instance_optional_input }}"  #
 input parameter reference
  MyServiceInstancesEnvironmentSampleOutputValue:                            #
 output definition
    Value: "{{ environment.outputs.MySampleInputValue }}"                    #
 output reference to an environment IaC file
  MyServiceInstancesEnvironmentOtherSampleOutputValue:                       #
 output definition
    Value: "{{ environment.outputs.MyOtherSampleInputValue }}"               #
 output reference to an environment IaC file
```

Component CloudFormation IaC file parameter details and examples

You can define and reference parameters in your component infrastructure as code (IaC) files. For a detailed description of AWS Proton parameters, parameter types, the parameter namespace, and how to use parameters in your IaC files, see the section called "Parameters". For more information about components, see *Components*.

Define component output parameters

You can define output parameters in your component IaC files. You can then refer to these outputs in service IaC files.

> **Note**
>
> You can't define inputs for component IaC files. Attached components can get inputs from the service instance that they are attached to. Detached components don't have inputs.

Read parameter values in component IaC files

You can read parameters related to the component and to other resources in component IaC files. You read a parameter value by referencing the parameter's name in the AWS Proton parameter namespace.

- **Input parameters** – Read an attached service instance input value by referencing `service_instance.inputs.`*`input-name`*.
- **Resource parameters** – Read AWS Proton resource parameters by referencing names such as `component.name`, `service.name`, `service_instance.name`, and `environment.name`.
- **Output parameters** – Read environment outputs by referencing `environment.outputs.`*`output-name`*.

Example component and service IaC files with parameters

The following example shows a component that provisions an Amazon Simple Storage Service (Amazon S3) bucket and related access policy and exposes the Amazon Resource Names (ARNs) of both resources as component outputs. A service IaC template adds the component outputs as container environment variables of an Amazon Elastic Container Service (Amazon ECS) task to make the outputs available to code running in the container, and adds the bucket access policy

to the task's role. The bucket name is based on the names of the environment, service, service instance, and component, meaning that the bucket is coupled with a specific instance of the component template extending a specific service instance. Developers can create multiple custom components based on this component template, to provision Amazon S3 buckets for different service instances and functional needs.

The example shows how you use Jinja `{{ ... }}` syntax to refer to component and other resource parameters in your service IaC file. You can use `{% if ... %}` statements to add blocks of statements only when a component is attached to the service instance. The `proton_cfn_*` keywords are *filters* that you can use to sanitize and format output parameter values. For more information about filters, see the section called "CloudFormation parameter filters".

As an administrator, you author the service IaC template file.

Example service CloudFormation IaC file using a component

```
# service/instance_infrastructure/cloudformation.yaml

Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      TaskRoleArn: !Ref TaskRole
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'
          # ...
          {% if service_instance.components.default.outputs | length > 0 %}
          Environment:
            {{ service_instance.components.default.outputs |
                proton_cfn_ecs_task_definition_formatted_env_vars }}
          {% endif %}

  # ...

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
        {{ service_instance.components.default.outputs
            | proton_cfn_iam_policy_arns }}
```

```
  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

As a developer, you author the component IaC template file.

Example component CloudFormation IaC file

```
# cloudformation.yaml

# A component that defines an S3 bucket and a policy for accessing the bucket.
Resources:
  S3Bucket:
    Type: 'AWS::S3::Bucket'
    Properties:
      BucketName: '{{environment.name}}-{{service.name}}-{{service_instance.name}}-
{{component.name}}'
  S3BucketAccessPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      PolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Action:
              - 's3:Get*'
              - 's3:List*'
              - 's3:PutObject'
            Resource: !GetAtt S3Bucket.Arn
Outputs:
  BucketName:
    Description: "Bucket to access"
    Value: !GetAtt S3Bucket.Arn
  BucketAccessPolicyArn:
    Value: !Ref S3BucketAccessPolicy
```

When AWS Proton renders an AWS CloudFormation template for your service instance and replaces all parameters with actual values, the template might look like the following file.

Example service instance CloudFormation rendered IaC file

```
Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      TaskRoleArn: !Ref TaskRole
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'
          # ...
          Environment:
            - Name: BucketName
              Value: arn:aws:s3:us-
east-1:123456789012:environment_name-service_name-service_instance_name-component_name
            - Name: BucketAccessPolicyArn
              Value: arn:aws:iam::123456789012:policy/cfn-generated-policy-name
  # ...

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
        - arn:aws:iam::123456789012:policy/cfn-generated-policy-name

  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

Parameter filters for CloudFormation IaC files

When you make references to AWS Proton parameters in your AWS CloudFormation IaC files, you can use Jinja modifiers known as *filters* to validate, filter, and format parameter values before they get inserted into the rendered template. Filter validations are particularly useful when referring to component output parameters, because component creation and attachment are done by developers, and an administrator using component outputs in a service instance template might want to verify their existence and validity. However, you can use filters in any Jinja IaC file.

The following sections describe and define the available parameter filters, and provide examples. AWS Proton defines most of these filters. The `default` filter is a Jinja built-in filter.

Format environment properties for Amazon ECS tasks

Declaration

```
dict # proton_cfn_ecs_task_definition_formatted_env_vars (raw: boolean = True) # YAML
 list of dicts
```

Description

This filter formats a list of outputs to be used in an Environment property in the `ContainerDefinition` section of an Amazon Elastic Container Service (Amazon ECS) task definition.

Set `raw` to `False` to also validate the parameter value. In this case, the value is required to match the regular expression `^[a-zA-Z0-9_-]*$`. If the value fails this validation, template rendering fails.

Example

With the following custom component template:

```
Resources:
  # ...
Outputs:
  Output1:
    Description: "Example component output 1"
    Value: hello
  Output2:
    Description: "Example component output 2"
    Value: world
```

And the following service template:

```
Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      # ...
      ContainerDefinitions:
```

```
        - Name: MyServiceName
          # ...
          Environment:
            {{ service_instance.components.default.outputs
              | proton_cfn_ecs_task_definition_formatted_env_vars }}
```

The rendered service template is as follows:

```
Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      # ...
      ContainerDefinitions:
        - Name: MyServiceName
          # ...
          Environment:
            - Name: Output1
              Value: hello
            - Name: Output2
              Value: world
```

Format environment properties for Lambda functions

Declaration

```
dict # proton_cfn_lambda_function_formatted_env_vars (raw: boolean = True) # YAML dict
```

Description

This filter formats a list of outputs to be used in an Environment property in the `Properties` section of an AWS Lambda function definition.

Set `raw` to `False` to also validate the parameter value. In this case, the value is required to match the regular expression `^[a-zA-Z0-9_-]*$`. If the value fails this validation, template rendering fails.

Example

With the following custom component template:

```
Resources:
```

```
  # ...
Outputs:
  Output1:
    Description: "Example component output 1"
    Value: hello
  Output2:
    Description: "Example component output 2"
    Value: world
```

And the following service template:

```
Resources:
  Lambda:
    Type: AWS::Lambda::Function
    Properties:
      Environment:
        Variables:
          {{ service_instance.components.default.outputs
            | proton_cfn_lambda_function_formatted_env_vars }}
```

The rendered service template is as follows:

```
Resources:
  Lambda:
    Type: AWS::Lambda::Function
    Properties:
      Environment:
        Variables:
          Output1: hello
          Output2: world
```

Extract IAM policy ARNs to include in IAM roles

Declaration

```
dict # proton_cfn_iam_policy_arns # YAML list
```

Description

This filter formats a list of outputs to be used in a ManagedPolicyArns property in the `Properties` section of an AWS Identity and Access Management (IAM) role definition. The filter uses the regular

expression `^arn:[a-zA-Z-]+:iam::\d{12}:policy/` to extract valid IAM policy ARNs from the list of output parameters. You can use this filter to append policies in output parameter values to an IAM role definition in a service template.

Example

With the following custom component template:

```
Resources:
  # ...
  ExamplePolicy1:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
  ExamplePolicy2:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...

  # ...

Outputs:
  Output1:
    Description: "Example component output 1"
    Value: hello
  Output2:
    Description: "Example component output 2"
    Value: world
  PolicyArn1:
    Description: "ARN of policy 1"
    Value: !Ref ExamplePolicy1
  PolicyArn2:
    Description: "ARN of policy 2"
    Value: !Ref ExamplePolicy2
```

And the following service template:

```
Resources:

  # ...

  TaskRole:
    Type: AWS::IAM::Role
```

```
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
        {{ service_instance.components.default.outputs
            | proton_cfn_iam_policy_arns }}

  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

The rendered service template is as follows:

```
Resources:

  # ...

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
        - arn:aws:iam::123456789012:policy/cfn-generated-policy-name-1
        - arn:aws:iam::123456789012:policy/cfn-generated-policy-name-2

  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

Sanitize property values

Declaration

```
string # proton_cfn_sanitize # string
```

Description

This is a general purpose filter. Use it to validate the safety of a parameter value. The filter validates that the value either matches the regular expression `^[a-zA-Z0-9_-]*$` or is a valid Amazon Resource Name (ARN). If the value fails this validation, template rendering fails.

Example

With the following custom component template:

```
Resources:
  # ...
Outputs:
  Output1:
    Description: "Example of valid output"
    Value: "This-is_valid_37"
  Output2:
    Description: "Example incorrect output"
    Value: "this::is::incorrect"
  SomeArn:
    Description: "Example ARN"
    Value: arn:aws:some-service::123456789012:some-resource/resource-name
```

- The following reference in a service template:

  ```
  # ...
    {{ service_instance.components.default.outputs.Output1
      | proton_cfn_sanitize }}
  ```

  Renders as follows:

  ```
  # ...
    This-is_valid_37
  ```

- The following reference in a service template:

  ```
  # ...
    {{ service_instance.components.default.outputs.Output2
      | proton_cfn_sanitize }}
  ```

  Results with the following rendering error:

```
Illegal character(s) detected in "this::is::incorrect". Must match regex ^[a-zA-
Z0-9_-]*$ or be a valid ARN
```

- The following reference in a service template:

```
# ...
  {{ service_instance.components.default.outputs.SomeArn
    | proton_cfn_sanitize }}
```

  Renders as follows:

```
# ...
  arn:aws:some-service::123456789012:some-resource/resource-name
```

Provide default values for nonexistent references

Description

The `default` filter provides a default value when a namespace reference doesn't exist. Use it to write robust templates that can render without failure even when the parameter you refer to is missing.

Example

The following reference in a service template causes template rendering to fail if the service instance doesn't have an attached directly defined (default) component, or if the attached component doesn't have an output named `test`.

```
# ...
  {{ service_instance.components.default.outputs.test }}
```

To avoid this issue, add the `default` filter.

```
# ...
  {{ service_instance.components.default.outputs.test | default("[optional-value]") }}
```

CodeBuild provisioning parameter details and examples

You can define parameters in your templates for CodeBuild-based AWS Proton resources and reference these parameters in your provisioning code. For a detailed description of AWS Proton parameters, parameter types, the parameter namespace, and how to use parameters in your IaC files, see the section called “Parameters”.

> **Note**
>
> You can use CodeBuild provisioning with environments and services. At this time you can't provision components this way.

Input parameters

When you create an AWS Proton resource, like an environment or a service, you provide values for input parameters that are defined in your template's schema file. When the resource that you create uses CodeBuild provisioning, AWS Proton renders these input values into an input file. Your provisioning code can import and get parameter values from this file.

For an example of a CodeBuild templates, see the section called “CodeBuild bundle”. For more information about manifest files, see the section called “Manifest and wrap up”.

The following example is a JSON input file generated during CodeBuild-based provisioning of a service instance.

Example: using the AWS CDK with CodeBuild provisioning

```
{
  "service_instance": {
    "name": "my-service-staging",
    "inputs": {
      "port": "8080",
      "task_size": "medium"
    }
  },
  "service": {
    "name": "my-service"
  },
  "environment": {
    "account_id": "123456789012",
```

```
    "name": "my-env-staging",
    "outputs": {
      "vpc-id": "hdh2323423"
    }
  }
}
```

Output parameters

To communicate resource provisioning outputs back to AWS Proton, your provisioning code can generate a JSON file named `proton-outputs.json` with values for output parameters defined in your template's schema file. For example, the `cdk deploy` command has the `--outputs-file` argument that instructs the AWS CDK to generate a JSON file with provisioning outputs. If your resource uses the AWS CDK, specify the following command in your CodeBuild template manifest:

```
aws proton notify-resource-deployment-status-change
```

AWS Proton looks for this JSON file. If the file exists after your provisioning code successfully completes, AWS Proton reads output parameter values from it.

Terraform infrastructure as code (IaC) file parameter details and examples

You can include Terraform input variables in `variable.tf` files in your template bundle. You can also create a schema to create AWS Proton managed variables. AWS Proton creates variable `.tf files` from your schema file. For more information, see the section called “Terraform IaC files”.

To reference your schema defined AWS Proton variables in your infrastructure `.tf files`, you use the AWS Proton namespaces shown in the *Parameters and namespaces for Terraform IaC* table. For example, you can use `var.environment.inputs.vpc_cidr`. Inside quotation marks, surround these variables with single brackets and add a dollar sign in front of the first brace (for example, “`${var.environment.inputs.vpc_cidr}`”).

The following example shows how to use namespaces to include AWS Proton parameters in an environment `.tf file`.

```
terraform {
  required_providers {
    aws = {
```

```
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
  // This tells terraform to store the state file in s3 at the location
  // s3://terraform-state-bucket/tf-os-sample/terraform.tfstate
  backend "s3" {
    bucket = "terraform-state-bucket"
    key    = "tf-os-sample/terraform.tfstate"
    region = "us-east-1"
  }
}

// Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
  default_tags {
    tags = var.proton_tags
  }
}

resource "aws_ssm_parameter" "my_ssm_parameter" {
  name  = "my_ssm_parameter"
  type  = "String"
  // Use the Proton environment.inputs. namespace
  value = var.environment.inputs.ssm_parameter_value
}
```

AWS Proton infrastructure as code files

The primary parts of the template bundle are *infrastructure as code (IaC) files* that define the infrastructure resources and properties that you want to provision. AWS CloudFormation and other infrastructure as code engines use these types of files to provision infrastructure resources.

> **Note**
>
> An IaC file can also be used independently of template bundles, as a direct input to *directly defined components*. For more information about components, see *Components*.

AWS Proton currently supports two types of IaC files:

- *CloudFormation files* – Used for *AWS-managed provisioning*. AWS Proton uses Jinja on top of the CloudFormation template file format for parametrization.
- *Terraform HCL files* – Used for *Self-managed provisioning*. HCL natively supports parametrization.

You can't provision AWS Proton resources using a combination of provisioning methods. You must use one or the other. You can't deploy an AWS-managed provisioning service to a self-managed provisioning environment, or vice versa.

For more information, see the section called "Provisioning methods", *Environments*, *Services*, and *Components*.

AWS CloudFormation IaC files

Learn how to use AWS CloudFormation infrastructure as code files with AWS Proton. AWS CloudFormation is an infrastructure as code (IaC) service that helps you model and set up your AWS resources. You define your infrastructure resources in templates, using Jinja on top of the CloudFormation template file format for parametrization. AWS Proton expands parameters and renders the full CloudFormation template. CloudFormation provisions the defined resources as CloudFormation stack. For more information, see What is AWS CloudFormation in *the AWS CloudFormation User Guide*.

AWS Proton supports AWS-managed provisioning for CloudFormation IaC.

Start with your own existing infrastructure as code files

You can adapt *your own existing* infrastructure as code (IaC) files for use with AWS Proton.

The following AWS CloudFormation examples, Example 1, and Example 2, represent *your own existing* CloudFormation IaC files. CloudFormation can use these files to create two different CloudFormation stacks.

In Example 1, the CloudFormation IaC file is configured to provision infrastructure to be shared among container applications. In this example, input parameters are added so that you can use the same IaC file to create multiple sets of provisioned infrastructure. Each set can have different names along with a different set of VPC and subnet CIDR values. As either an administrator or a developer, you provide values for these parameters when you use an IaC file to provision infrastructure resources with CloudFormation. For your convenience, these input parameters are marked with comments and referenced multiple times in the example. The *outputs* are defined at the end of the template. They can be referenced in other CloudFormation IaC files.

In Example 2, the CloudFormation IaC file is configured to deploy an application to the infrastructure that's provisioned from *Example 1*. The parameters are commented for your convenience.

Example 1: CloudFormation IaC file

```
AWSTemplateFormatVersion: '2010-09-09'
Description: AWS Fargate cluster running containers in a public subnet. Only supports
             public facing load balancer, and public service discovery namespaces.
Parameters:
   VpcCIDR:       # input parameter
        Description: CIDR for VPC
        Type: String
        Default: "10.0.0.0/16"
   SubnetOneCIDR: # input parameter
        Description: CIDR for SubnetOne
        Type: String
        Default: "10.0.0.0/24"
   SubnetTwoCIDR: # input parameters
        Description: CIDR for SubnetTwo
        Type: String
        Default: "10.0.1.0/24"
Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      EnableDnsSupport: true
      EnableDnsHostnames: true
      CidrBlock:
        Ref: 'VpcCIDR'

  # Two public subnets, where containers will have public IP addresses
  PublicSubnetOne:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
         Fn::Select:
         - 0
         - Fn::GetAZs: {Ref: 'AWS::Region'}
      VpcId: !Ref 'VPC'
      CidrBlock:
         Ref: 'SubnetOneCIDR'
      MapPublicIpOnLaunch: true
```

```
  PublicSubnetTwo:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
         Fn::Select:
         - 1
         - Fn::GetAZs: {Ref: 'AWS::Region'}
      VpcId: !Ref 'VPC'
      CidrBlock:
        Ref: 'SubnetTwoCIDR'
      MapPublicIpOnLaunch: true

  # Setup networking resources for the public subnets. Containers
  # in the public subnets have public IP addresses and the routing table
  # sends network traffic via the internet gateway.
  InternetGateway:
    Type: AWS::EC2::InternetGateway
  GatewayAttachement:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref 'VPC'
      InternetGatewayId: !Ref 'InternetGateway'
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref 'VPC'
  PublicRoute:
    Type: AWS::EC2::Route
    DependsOn: GatewayAttachement
    Properties:
      RouteTableId: !Ref 'PublicRouteTable'
      DestinationCidrBlock: '0.0.0.0/0'
      GatewayId: !Ref 'InternetGateway'
  PublicSubnetOneRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetOne
      RouteTableId: !Ref PublicRouteTable
  PublicSubnetTwoRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetTwo
      RouteTableId: !Ref PublicRouteTable
```

```
  # ECS Resources
  ECSCluster:
    Type: AWS::ECS::Cluster

  # A security group for the containers we will run in Fargate.
  # Rules are added to this security group based on what ingress you
  # add for the cluster.
  ContainerSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the Fargate containers
      VpcId: !Ref 'VPC'

  # This is a role which is used by the ECS tasks themselves.
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
        - Effect: Allow
          Principal:
            Service: [ecs-tasks.amazonaws.com]
          Action: ['sts:AssumeRole']
      Path: /
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy'

# These output values will be available to other templates to use.
Outputs:
  ClusterName:                                                # output
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-ECSCluster"
  ECSTaskExecutionRole:                                        # output
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-ECSTaskExecutionRole"
  VpcId:                                                       # output
    Description: The ID of the VPC that this stack is deployed in
```

```
    Value: !Ref 'VPC'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-VPC"
  PublicSubnetOne:                                          # output
    Description: Public subnet one
    Value: !Ref 'PublicSubnetOne'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-PublicSubnetOne"
  PublicSubnetTwo:                                          # output
    Description: Public subnet two
    Value: !Ref 'PublicSubnetTwo'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-PublicSubnetTwo"
  ContainerSecurityGroup:                                   # output
    Description: A security group used to allow Fargate containers to receive traffic
    Value: !Ref 'ContainerSecurityGroup'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-ContainerSecurityGroup"
```

Example 2: CloudFormation IaC file

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy a service on AWS Fargate, hosted in a public subnet, and accessible
 via a public load balancer.
Parameters:
    ContainerPortInput:  # input parameter
        Description: The port to route traffic to
        Type: Number
        Default: 80
    TaskCountInput:      # input parameter
        Description: The default number of Fargate tasks you want running
        Type: Number
        Default: 1
    TaskSizeInput:       # input parameter
        Description: The size of the task you want to run
        Type: String
        Default: x-small
    ContainerImageInput: # input parameter
        Description: The name/url of the container image
```

```
        Type: String
        Default: "public.ecr.aws/z9d2n7e1/nginx:1.19.5"
    TaskNameInput:       # input parameter
        Description: Name for your task
        Type: String
        Default: "my-fargate-instance"
    StackName:           # input parameter
        Description: Name of the environment stack to deploy to
        Type: String
        Default: "my-fargate-environment"
Mappings:
  TaskSizeMap:
    x-small:
      cpu: 256
      memory: 512
    small:
      cpu: 512
      memory: 1024
    medium:
      cpu: 1024
      memory: 2048
    large:
      cpu: 2048
      memory: 4096
    x-large:
      cpu: 4096
      memory: 8192
Resources:
  # A log group for storing the stdout logs from this service's containers
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName:
        Ref: 'TaskNameInput' # input parameter

  # The task definition. This is a simple metadata description of what
  # container to run, and what resource requirements it has.
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: !Ref 'TaskNameInput'
      Cpu: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', cpu]
      Memory: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', memory]
      NetworkMode: awsvpc
```

```
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn:
        Fn::ImportValue:
          !Sub "${StackName}-ECSTaskExecutionRole"    # output parameter from another
CloudFormation template
              awslogs-region: !Ref 'AWS::Region'
              awslogs-stream-prefix: !Ref 'TaskNameInput'


 # The service_instance. The service is a resource which allows you to run multiple
 # copies of a type of task, and gather up their logs and metrics, as well
 # as monitor the number of running tasks and replace any that have crashed
 Service:
   Type: AWS::ECS::Service
   DependsOn: LoadBalancerRule
   Properties:
     ServiceName: !Ref 'TaskNameInput'
     Cluster:
       Fn::ImportValue:
         !Sub "${StackName}-ECSCluster"  # output parameter from another
CloudFormation template
     LaunchType: FARGATE
     DeploymentConfiguration:
       MaximumPercent: 200
       MinimumHealthyPercent: 75
     DesiredCount: !Ref 'TaskCountInput'
     NetworkConfiguration:
       AwsvpcConfiguration:
         AssignPublicIp: ENABLED
         SecurityGroups:
           - Fn::ImportValue:
               !Sub "${StackName}-ContainerSecurityGroup"    # output parameter from
another CloudFormation template
         Subnets:
           - Fn::ImportValue:r CloudFormation template
     TaskRoleArn: !Ref "AWS::NoValue"
     ContainerDefinitions:
       - Name: !Ref 'TaskNameInput'
         Cpu: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', cpu]
         Memory: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', memory]
         Image: !Ref 'ContainerImageInput'             # input parameter
         PortMappings:
           - ContainerPort: !Ref 'ContainerPortInput'  # input parameter
```

```
        LogConfiguration:
          LogDriver: 'awslogs'
          Options:
            awslogs-group: !Ref 'TaskNameInput'
              !Sub "${StackName}-PublicSubnetOne"    # output parameter from another
CloudFormation template
          - Fn::ImportValue:
              !Sub "${StackName}-PublicSubnetTwo"    # output parameter from another
CloudFormation template
     TaskDefinition: !Ref 'TaskDefinition'
     LoadBalancers:
       - ContainerName: !Ref 'TaskNameInput'
         ContainerPort: !Ref 'ContainerPortInput'  # input parameter
         TargetGroupArn: !Ref 'TargetGroup'

 # A target group. This is used for keeping track of all the tasks, and
 # what IP addresses / port numbers they have. You can query it yourself,
 # to use the addresses yourself, but most often this target group is just
 # connected to an application load balancer, or network load balancer, so
 # it can automatically distribute traffic across all the targets.
 TargetGroup:
   Type: AWS::ElasticLoadBalancingV2::TargetGroup
   Properties:
     HealthCheckIntervalSeconds: 6
     HealthCheckPath: /
     HealthCheckProtocol: HTTP
     HealthCheckTimeoutSeconds: 5
     HealthyThresholdCount: 2
     TargetType: ip
     Name: !Ref 'TaskNameInput'
     Port: !Ref 'ContainerPortInput'
     Protocol: HTTP
     UnhealthyThresholdCount: 2
     VpcId:
       Fn::ImportValue:
         !Sub "${StackName}-VPC"    # output parameter from another CloudFormation
template

 # Create a rule on the load balancer for routing traffic to the target group
 LoadBalancerRule:
   Type: AWS::ElasticLoadBalancingV2::ListenerRule
   Properties:
     Actions:
```

```
        - TargetGroupArn: !Ref 'TargetGroup'
          Type: 'forward'
      Conditions:
        - Field: path-pattern
          Values:
            - '*'
      ListenerArn: !Ref PublicLoadBalancerListener
      Priority: 1

  # Enable autoscaling for this service
  ScalableTarget:
    Type: AWS::ApplicationAutoScaling::ScalableTarget
    DependsOn: Service
    Properties:
      ServiceNamespace: 'ecs'
      ScalableDimension: 'ecs:service:DesiredCount'
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - Fn::ImportValue:
                !Sub "${StackName}-ECSCluster"
            - !Ref 'TaskNameInput'
      MinCapacity: 1
      MaxCapacity: 10
      RoleARN: !Sub arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService

  # Create scaling policies for the service
  ScaleDownPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - !Ref 'TaskNameInput'
            - down
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
```

```
          - - service
            - Fn::ImportValue:
                !Sub "${StackName}-ECSCluster"  # output parameter from another
CloudFormation template
            - !Ref 'TaskNameInput'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalUpperBound: 0
            ScalingAdjustment: -1
        MetricAggregationType: 'Average'
        Cooldown: 60

  ScaleUpPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - !Ref 'TaskNameInput'
            - up
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - Fn::ImportValue:
                !Sub "${StackName}-ECSCluster"
            - !Ref 'TaskNameInput'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalLowerBound: 0
            MetricIntervalUpperBound: 15
            ScalingAdjustment: 1
          - MetricIntervalLowerBound: 15
            MetricIntervalUpperBound: 25
            ScalingAdjustment: 2
```

```
          - MetricIntervalLowerBound: 25
            ScalingAdjustment: 3
        MetricAggregationType: 'Average'
        Cooldown: 60

  # Create alarms to trigger these policies
  LowCpuUsageAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmName:
        Fn::Join:
          - '-'
          - - low-cpu
            - !Ref 'TaskNameInput'
      AlarmDescription:
        Fn::Join:
          - ' '
          - - "Low CPU utilization for service"
            - !Ref 'TaskNameInput'
      MetricName: CPUUtilization
      Namespace: AWS/ECS
      Dimensions:
        - Name: ServiceName
          Value: !Ref 'TaskNameInput'
        - Name: ClusterName
          Value:
            Fn::ImportValue:
              !Sub "${StackName}-ECSCluster"
      Statistic: Average
      Period: 60
      EvaluationPeriods: 1
      Threshold: 20
      ComparisonOperator: LessThanOrEqualToThreshold
      AlarmActions:
        - !Ref ScaleDownPolicy

  HighCpuUsageAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmName:
        Fn::Join:
          - '-'
          - - high-cpu
            - !Ref 'TaskNameInput'
```

```
      AlarmDescription:
        Fn::Join:
          - ' '
          - - "High CPU utilization for service"
            - !Ref 'TaskNameInput'
      MetricName: CPUUtilization
      Namespace: AWS/ECS
      Dimensions:
        - Name: ServiceName
          Value: !Ref 'TaskNameInput'
        - Name: ClusterName
          Value:
            Fn::ImportValue:
              !Sub "${StackName}-ECSCluster"
      Statistic: Average
      Period: 60
      EvaluationPeriods: 1
      Threshold: 70
      ComparisonOperator: GreaterThanOrEqualToThreshold
      AlarmActions:
        - !Ref ScaleUpPolicy

  EcsSecurityGroupIngressFromPublicALB:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      Description: Ingress from the public ALB
      GroupId:
        Fn::ImportValue:
          !Sub "${StackName}-ContainerSecurityGroup"
      IpProtocol: -1
      SourceSecurityGroupId: !Ref 'PublicLoadBalancerSG'

  # Public load balancer, hosted in public subnets that is accessible
  # to the public, and is intended to route traffic to one or more public
  # facing services. This is used for accepting traffic from the public
  # internet and directing it to public facing microservices
  PublicLoadBalancerSG:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the public facing load balancer
      VpcId:
        Fn::ImportValue:
          !Sub "${StackName}-VPC"
      SecurityGroupIngress:
```

```
          # Allow access to ALB from anywhere on the internet
          - CidrIp: 0.0.0.0/0
            IpProtocol: -1

  PublicLoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Scheme: internet-facing
      LoadBalancerAttributes:
      - Key: idle_timeout.timeout_seconds
        Value: '30'
      Subnets:
        # The load balancer is placed into the public subnets, so that traffic
        # from the internet can reach the load balancer directly via the internet
 gateway
        - Fn::ImportValue:
            !Sub "${StackName}-PublicSubnetOne"
        - Fn::ImportValue:
            !Sub "${StackName}-PublicSubnetTwo"
      SecurityGroups: [!Ref 'PublicLoadBalancerSG']

  PublicLoadBalancerListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    DependsOn:
      - PublicLoadBalancer
    Properties:
      DefaultActions:
        - TargetGroupArn: !Ref 'TargetGroup'
          Type: 'forward'
      LoadBalancerArn: !Ref 'PublicLoadBalancer'
      Port: 80
      Protocol: HTTP
# These output values will be available to other templates to use.
Outputs:
  ServiceEndpoint:        # output
    Description: The URL to access the service
    Value: !Sub "http://${PublicLoadBalancer.DNSName}"
```

You can adapt these files for use with AWS Proton.

Bring your infrastructure as code to AWS Proton

With slight modifications, you can use Example 1 as an infrastructure as code (IaC) file for an environment template bundle that AWS Proton uses to deploy an environment (as shown in Example 3).

Instead of using the CloudFormation parameters, you use Jinja syntax to reference parameters that you have defined in an Open API based schema file. These input parameters are commented for your convenience and referenced multiple times in the IaC file. This way, AWS Proton can audit and check parameter values. It can also match and insert output parameter values in one IaC file to parameters in another IaC file.

As administrator, you can add the AWS Proton `environment.inputs.` namespace to the input parameters. When you reference environment IaC file outputs in a service IaC file, you can add the `environment.outputs.` namespace to the outputs (for example, `environment.outputs.ClusterName`). Last, you surround them with curly braces and quotation marks.

With these modifications, your CloudFormation IaC files can be used by AWS Proton.

Example 3: AWS Proton environment infrastructure as code file

```
AWSTemplateFormatVersion: '2010-09-09'
Description: AWS Fargate cluster running containers in a public subnet. Only supports
             public facing load balancer, and public service discovery prefixes.
Mappings:
  # The VPC and subnet configuration is passed in via the environment spec.
  SubnetConfig:
    VPC:
      CIDR: '{{ environment.inputs.vpc_cidr}}'        # input parameter
    PublicOne:
      CIDR: '{{ environment.inputs.subnet_one_cidr}}' # input parameter
    PublicTwo:
      CIDR: '{{ environment.inputs.subnet_two_cidr}}' # input parameter
Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      EnableDnsSupport: true
      EnableDnsHostnames: true
      CidrBlock: !FindInMap ['SubnetConfig', 'VPC', 'CIDR']
```

```
  # Two public subnets, where containers will have public IP addresses
  PublicSubnetOne:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
         Fn::Select:
         - 0
         - Fn::GetAZs: {Ref: 'AWS::Region'}
      VpcId: !Ref 'VPC'
      CidrBlock: !FindInMap ['SubnetConfig', 'PublicOne', 'CIDR']
      MapPublicIpOnLaunch: true

  PublicSubnetTwo:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
         Fn::Select:
         - 1
         - Fn::GetAZs: {Ref: 'AWS::Region'}
      VpcId: !Ref 'VPC'
      CidrBlock: !FindInMap ['SubnetConfig', 'PublicTwo', 'CIDR']
      MapPublicIpOnLaunch: true

  # Setup networking resources for the public subnets. Containers
  # in the public subnets have public IP addresses and the routing table
  # sends network traffic via the internet gateway.
  InternetGateway:
    Type: AWS::EC2::InternetGateway
  GatewayAttachement:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref 'VPC'
      InternetGatewayId: !Ref 'InternetGateway'
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref 'VPC'
  PublicRoute:
    Type: AWS::EC2::Route
    DependsOn: GatewayAttachement
    Properties:
      RouteTableId: !Ref 'PublicRouteTable'
      DestinationCidrBlock: '0.0.0.0/0'
      GatewayId: !Ref 'InternetGateway'
```

```
  PublicSubnetOneRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetOne
      RouteTableId: !Ref PublicRouteTable
  PublicSubnetTwoRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetTwo
      RouteTableId: !Ref PublicRouteTable

  # ECS Resources
  ECSCluster:
    Type: AWS::ECS::Cluster

  # A security group for the containers we will run in Fargate.
  # Rules are added to this security group based on what ingress you
  # add for the cluster.
  ContainerSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the Fargate containers
      VpcId: !Ref 'VPC'

  # This is a role which is used by the ECS tasks themselves.
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
        - Effect: Allow
          Principal:
            Service: [ecs-tasks.amazonaws.com]
          Action: ['sts:AssumeRole']
      Path: /
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy'

# These output values are available to service infrastructure as code files as outputs,
 when given the
# the 'service_instance.environment.outputs.' namespace, for example,
 service_instance.environment.outputs.ClusterName.

Outputs:
```

```
  ClusterName:                                            # output
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'
  ECSTaskExecutionRole:                                   # output
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'
  VpcId:                                                  # output
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'
  PublicSubnetOne:                                        # output
    Description: Public subnet one
    Value: !Ref 'PublicSubnetOne'
  PublicSubnetTwo:                                        # output
    Description: Public subnet two
    Value: !Ref 'PublicSubnetTwo'
  ContainerSecurityGroup:                                 # output
    Description: A security group used to allow Fargate containers to receive traffic
    Value: !Ref 'ContainerSecurityGroup'
```

The IaC files in Example 1 and Example 3 produce slightly different CloudFormation stacks. Parameters are displayed differently in the stack template files. The *Example 1* CloudFormation stack template file displays the parameter labels (keys) in the stack template view. The *Example 3* AWS Proton CloudFormation infrastructure stack template file displays the parameter values. AWS Proton input parameters *don't* appear in the console CloudFormation stack parameters view.

In Example 4, the AWS Proton service IaC file corresponds with Example 2.

Example 4: AWS Proton service instance IaC file

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy a service on AWS Fargate, hosted in a public subnet, and accessible
 via a public load balancer.
Mappings:
  TaskSize:
    x-small:
      cpu: 256
      memory: 512
    small:
      cpu: 512
      memory: 1024
    medium:
      cpu: 1024
      memory: 2048
```

```
    large:
      cpu: 2048
      memory: 4096
    x-large:
      cpu: 4096
      memory: 8192
Resources:
  # A log group for storing the stdout logs from this service's containers
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: '{{service_instance.name}}' # resource parameter

  # The task definition. This is a simple metadata description of what
  # container to run, and what resource requirements it has.
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: '{{service_instance.name}}'
      Cpu: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, cpu] # input
 parameter
      Memory: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, memory]
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn: '{{environment.outputs.ECSTaskExecutionRole}}' # output from an
 environment infrastructure as code file
      TaskRoleArn: !Ref "AWS::NoValue"
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'
          Cpu: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, cpu]
          Memory: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, memory]
          Image: '{{service_instance.inputs.image}}'
          PortMappings:
            - ContainerPort: '{{service_instance.inputs.port}}' # input parameter
          LogConfiguration:
            LogDriver: 'awslogs'
            Options:
              awslogs-group: '{{service_instance.name}}'
              awslogs-region: !Ref 'AWS::Region'
              awslogs-stream-prefix: '{{service_instance.name}}'

  # The service_instance. The service is a resource which allows you to run multiple
  # copies of a type of task, and gather up their logs and metrics, as well
```

```
  # as monitor the number of running tasks and replace any that have crashed
  Service:
    Type: AWS::ECS::Service
    DependsOn: LoadBalancerRule
    Properties:
      ServiceName: '{{service_instance.name}}'
      Cluster: '{{environment.outputs.ClusterName}}' # output from an environment
infrastructure as code file
      LaunchType: FARGATE
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 75
      DesiredCount: '{{service_instance.inputs.desired_count}}'       # input parameter
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups:
            - '{{environment.outputs.ContainerSecurityGroup}}' # output from an
environment infrastructure as code file
          Subnets:
            - '{{environment.outputs.PublicSubnetOne}}'        # output from an
environment infrastructure as code file
            - '{{environment.outputs.PublicSubnetTwo}}'
      TaskDefinition: !Ref 'TaskDefinition'
      LoadBalancers:
        - ContainerName: '{{service_instance.name}}'
          ContainerPort: '{{service_instance.inputs.port}}'
          TargetGroupArn: !Ref 'TargetGroup'

  # A target group. This is used for keeping track of all the tasks, and
  # what IP addresses / port numbers they have. You can query it yourself,
  # to use the addresses yourself, but most often this target group is just
  # connected to an application load balancer, or network load balancer, so
  # it can automatically distribute traffic across all the targets.
  TargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      HealthCheckIntervalSeconds: 6
      HealthCheckPath: /
      HealthCheckProtocol: HTTP
      HealthCheckTimeoutSeconds: 5
      HealthyThresholdCount: 2
      TargetType: ip
      Name: '{{service_instance.name}}'
```

```
      Port: '{{service_instance.inputs.port}}'
      Protocol: HTTP
      UnhealthyThresholdCount: 2
      VpcId: '{{environment.outputs.VpcId}}' # output from an environment
 infrastructure as code file

  # Create a rule on the load balancer for routing traffic to the target group
  LoadBalancerRule:
    Type: AWS::ElasticLoadBalancingV2::ListenerRule
    Properties:
      Actions:
        - TargetGroupArn: !Ref 'TargetGroup'
          Type: 'forward'
      Conditions:
        - Field: path-pattern
          Values:
            - '*'
      ListenerArn: !Ref PublicLoadBalancerListener
      Priority: 1

  # Enable autoscaling for this service
  ScalableTarget:
    Type: AWS::ApplicationAutoScaling::ScalableTarget
    DependsOn: Service
    Properties:
      ServiceNamespace: 'ecs'
      ScalableDimension: 'ecs:service:DesiredCount'
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - '{{environment.outputs.ClusterName}}' # output from an environment
 infrastructure as code file
            - '{{service_instance.name}}'
      MinCapacity: 1
      MaxCapacity: 10
      RoleARN: !Sub arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService

  # Create scaling policies for the service
  ScaleDownPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
```

```
    Properties:
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - '{{service_instance.name}}'
            - down
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - '{{environment.outputs.ClusterName}}'
            - '{{service_instance.name}}'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalUpperBound: 0
            ScalingAdjustment: -1
        MetricAggregationType: 'Average'
        Cooldown: 60

  ScaleUpPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - '{{service_instance.name}}'
            - up
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - '{{environment.outputs.ClusterName}}'
            - '{{service_instance.name}}'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
```

```
      AdjustmentType: 'ChangeInCapacity'
      StepAdjustments:
        - MetricIntervalLowerBound: 0
          MetricIntervalUpperBound: 15
          ScalingAdjustment: 1
        - MetricIntervalLowerBound: 15
          MetricIntervalUpperBound: 25
          ScalingAdjustment: 2
        - MetricIntervalLowerBound: 25
          ScalingAdjustment: 3
      MetricAggregationType: 'Average'
      Cooldown: 60

# Create alarms to trigger these policies
LowCpuUsageAlarm:
  Type: AWS::CloudWatch::Alarm
  Properties:
    AlarmName:
      Fn::Join:
        - '-'
        - - low-cpu
          - '{{service_instance.name}}'
    AlarmDescription:
      Fn::Join:
        - ' '
        - - "Low CPU utilization for service"
          - '{{service_instance.name}}'
    MetricName: CPUUtilization
    Namespace: AWS/ECS
    Dimensions:
      - Name: ServiceName
        Value: '{{service_instance.name}}'
      - Name: ClusterName
        Value:
          '{{environment.outputs.ClusterName}}'
    Statistic: Average
    Period: 60
    EvaluationPeriods: 1
    Threshold: 20
    ComparisonOperator: LessThanOrEqualToThreshold
    AlarmActions:
      - !Ref ScaleDownPolicy

HighCpuUsageAlarm:
```

```
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmName:
        Fn::Join:
          - '-'
          - - high-cpu
            - '{{service_instance.name}}'
      AlarmDescription:
        Fn::Join:
          - ' '
          - - "High CPU utilization for service"
            - '{{service_instance.name}}'
      MetricName: CPUUtilization
      Namespace: AWS/ECS
      Dimensions:
        - Name: ServiceName
          Value: '{{service_instance.name}}'
        - Name: ClusterName
          Value:
            '{{environment.outputs.ClusterName}}'
      Statistic: Average
      Period: 60
      EvaluationPeriods: 1
      Threshold: 70
      ComparisonOperator: GreaterThanOrEqualToThreshold
      AlarmActions:
        - !Ref ScaleUpPolicy

  EcsSecurityGroupIngressFromPublicALB:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      Description: Ingress from the public ALB
      GroupId: '{{environment.outputs.ContainerSecurityGroup}}'
      IpProtocol: -1
      SourceSecurityGroupId: !Ref 'PublicLoadBalancerSG'

  # Public load balancer, hosted in public subnets that is accessible
  # to the public, and is intended to route traffic to one or more public
  # facing services. This is used for accepting traffic from the public
  # internet and directing it to public facing microservices
  PublicLoadBalancerSG:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the public facing load balancer
```

```
      VpcId: '{{environment.outputs.VpcId}}'
      SecurityGroupIngress:
          # Allow access to ALB from anywhere on the internet
          - CidrIp: 0.0.0.0/0
            IpProtocol: -1

  PublicLoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Scheme: internet-facing
      LoadBalancerAttributes:
      - Key: idle_timeout.timeout_seconds
        Value: '30'
      Subnets:
        # The load balancer is placed into the public subnets, so that traffic
        # from the internet can reach the load balancer directly via the internet
 gateway
        - '{{environment.outputs.PublicSubnetOne}}'
        - '{{environment.outputs.PublicSubnetTwo}}'
      SecurityGroups: [!Ref 'PublicLoadBalancerSG']

  PublicLoadBalancerListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    DependsOn:
      - PublicLoadBalancer
    Properties:
      DefaultActions:
        - TargetGroupArn: !Ref 'TargetGroup'
          Type: 'forward'
      LoadBalancerArn: !Ref 'PublicLoadBalancer'
      Port: 80
      Protocol: HTTP
Outputs:
  ServiceEndpoint:         # output
    Description: The URL to access the service
    Value: !Sub "http://${PublicLoadBalancer.DNSName}"
```

In Example 5, the AWS Proton pipeline IaC file provisions the pipeline infrastructure to support the service instances provisioned by Example 4.

Example 5: AWS Proton service pipeline IaC file

```
Resources:
```

```
  ECRRepo:
    Type: AWS::ECR::Repository
    DeletionPolicy: Retain
  BuildProject:
    Type: AWS::CodeBuild::Project
    Properties:
      Artifacts:
        Type: CODEPIPELINE
      Environment:
        ComputeType: BUILD_GENERAL1_SMALL
        Image: aws/codebuild/amazonlinux2-x86_64-standard:3.0
        PrivilegedMode: true
        Type: LINUX_CONTAINER
        EnvironmentVariables:
        - Name: repo_name
          Type: PLAINTEXT
          Value: !Ref ECRRepo
        - Name: service_name
          Type: PLAINTEXT
          Value: '{{ service.name }}'    # resource parameter
      ServiceRole:
        Fn::GetAtt:
          - PublishRole
          - Arn
      Source:
        BuildSpec:
          Fn::Join:
            - ""
            - - >-
                {
                  "version": "0.2",
                  "phases": {
                    "install": {
                      "runtime-versions": {
                        "docker": 18
                      },
                      "commands": [
                        "pip3 install --upgrade --user awscli",
                        "echo
 'f6bd1536a743ab170b35c94ed4c7c4479763356bd543af5d391122f4af852460  yq_linux_amd64' >
 yq_linux_amd64.sha",
                        "wget https://github.com/mikefarah/yq/releases/download/3.4.0/
yq_linux_amd64",
                        "sha256sum -c yq_linux_amd64.sha",
```

```
                        "mv yq_linux_amd64 /usr/bin/yq",
                        "chmod +x /usr/bin/yq"
                      ]
                    },
                    "pre_build": {
                      "commands": [
                        "cd $CODEBUILD_SRC_DIR",
                        "$(aws ecr get-login --no-include-email --region
 $AWS_DEFAULT_REGION)",
                        "{{ pipeline.inputs.unit_test_command }}",    # input parameter
                      ]
                    },
                    "build": {
                      "commands": [
                        "IMAGE_REPO_NAME=$repo_name",
                        "IMAGE_TAG=$CODEBUILD_BUILD_NUMBER",
                        "IMAGE_ID=
             - Ref: AWS::AccountId
             - >-
               .dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:
$IMAGE_TAG",
                        "docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG -f
 {{ pipeline.inputs.dockerfile }} .",     # input parameter
                        "docker tag $IMAGE_REPO_NAME:$IMAGE_TAG $IMAGE_ID;",
                        "docker push $IMAGE_ID"
                      ]
                    },
                    "post_build": {
                      "commands": [
                        "aws proton --region $AWS_DEFAULT_REGION get-service --name
 $service_name | jq -r .service.spec > service.yaml",
                        "yq w service.yaml 'instances[*].spec.image' \"$IMAGE_ID\" >
 rendered_service.yaml"
                      ]
                    }
                  },
                  "artifacts": {
                    "files": [
                      "rendered_service.yaml"
                    ]
                  }
                }
        Type: CODEPIPELINE
      EncryptionKey:
```

```
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
{% for service_instance in service_instances %}
  Deploy{{loop.index}}Project:
    Type: AWS::CodeBuild::Project
    Properties:
      Artifacts:
        Type: CODEPIPELINE
      Environment:
        ComputeType: BUILD_GENERAL1_SMALL
        Image: aws/codebuild/amazonlinux2-x86_64-standard:3.0
        PrivilegedMode: false
        Type: LINUX_CONTAINER
        EnvironmentVariables:
        - Name: service_name
          Type: PLAINTEXT
          Value:  '{{service.name}}'          # resource parameter
        - Name: service_instance_name
          Type: PLAINTEXT
          Value: '{{service_instance.name}}'  # resource parameter
      ServiceRole:
        Fn::GetAtt:
          - DeploymentRole
          - Arn
      Source:
        BuildSpec: >-
          {
            "version": "0.2",
            "phases": {
              "build": {
                "commands": [
                  "pip3 install --upgrade --user awscli",
                  "aws proton --region $AWS_DEFAULT_REGION update-service-instance
 --deployment-type CURRENT_VERSION --name $service_instance_name --service-name
 $service_name --spec file://rendered_service.yaml",
                  "aws proton --region $AWS_DEFAULT_REGION wait service-instance-
deployed --name $service_instance_name --service-name $service_name"
                ]
              }
            }
          }
        Type: CODEPIPELINE
      EncryptionKey:
```

```
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
{% endfor %}
  # This role is used to build and publish an image to ECR
  PublishRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codebuild.amazonaws.com
        Version: "2012-10-17"
  PublishRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - logs:CreateLogGroup
              - logs:CreateLogStream
              - logs:PutLogEvents
            Effect: Allow
            Resource:
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/
                    - Ref: BuildProject
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
```

```
                - :log-group:/aws/codebuild/
                - Ref: BuildProject
                - :*
        - Action:
            - codebuild:CreateReportGroup
            - codebuild:CreateReport
            - codebuild:UpdateReport
            - codebuild:BatchPutTestCases
          Effect: Allow
          Resource:
            Fn::Join:
              - ""
              - - "arn:"
                - Ref: AWS::Partition
                - ":codebuild:"
                - Ref: AWS::Region
                - ":"
                - Ref: AWS::AccountId
                - :report-group/
                - Ref: BuildProject
                - -*
        - Action:
            - ecr:GetAuthorizationToken
          Effect: Allow
          Resource: "*"
        - Action:
            - ecr:BatchCheckLayerAvailability
            - ecr:CompleteLayerUpload
            - ecr:GetAuthorizationToken
            - ecr:InitiateLayerUpload
            - ecr:PutImage
            - ecr:UploadLayerPart
          Effect: Allow
          Resource:
            Fn::GetAtt:
              - ECRRepo
              - Arn
        - Action:
            - proton:GetService
          Effect: Allow
          Resource: "*"
        - Action:
            - s3:GetObject*
            - s3:GetBucket*
```

```
              - s3:List*
              - s3:DeleteObject*
              - s3:PutObject*
              - s3:Abort*
            Effect: Allow
            Resource:
              - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - PipelineArtifactsBucket
                        - Arn
                    - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action:
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
        Version: "2012-10-17"
      PolicyName: PublishRoleDefaultPolicy
      Roles:
        - Ref: PublishRole

  DeploymentRole:
    Type: AWS::IAM::Role
    Properties:
```

```
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codebuild.amazonaws.com
        Version: "2012-10-17"
  DeploymentRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - logs:CreateLogGroup
              - logs:CreateLogStream
              - logs:PutLogEvents
            Effect: Allow
            Resource:
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/Deploy*Project*
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/Deploy*Project:*
          - Action:
              - codebuild:CreateReportGroup
              - codebuild:CreateReport
              - codebuild:UpdateReport
              - codebuild:BatchPutTestCases
            Effect: Allow
            Resource:
              Fn::Join:
```

```
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":codebuild:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - :report-group/Deploy*Project
                  - -*
            - Action:
                - proton:UpdateServiceInstance
                - proton:GetServiceInstance
              Effect: Allow
              Resource: "*"
            - Action:
                - s3:GetObject*
                - s3:GetBucket*
                - s3:List*
              Effect: Allow
              Resource:
                - Fn::GetAtt:
                    - PipelineArtifactsBucket
                    - Arn
                - Fn::Join:
                    - ""
                    - - Fn::GetAtt:
                          - PipelineArtifactsBucket
                          - Arn
                      - /*
            - Action:
                - kms:Decrypt
                - kms:DescribeKey
              Effect: Allow
              Resource:
                Fn::GetAtt:
                  - PipelineArtifactsBucketEncryptionKey
                  - Arn
            - Action:
                - kms:Decrypt
                - kms:Encrypt
                - kms:ReEncrypt*
                - kms:GenerateDataKey*
              Effect: Allow
              Resource:
```

```
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
        Version: "2012-10-17"
      PolicyName: DeploymentRoleDefaultPolicy
      Roles:
        - Ref: DeploymentRole
  PipelineArtifactsBucketEncryptionKey:
    Type: AWS::KMS::Key
    Properties:
      KeyPolicy:
        Statement:
          - Action:
              - kms:Create*
              - kms:Describe*
              - kms:Enable*
              - kms:List*
              - kms:Put*
              - kms:Update*
              - kms:Revoke*
              - kms:Disable*
              - kms:Get*
              - kms:Delete*
              - kms:ScheduleKeyDeletion
              - kms:CancelKeyDeletion
              - kms:GenerateDataKey
              - kms:TagResource
              - kms:UntagResource
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
```

```
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - PipelineRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - PublishRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - PublishRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - DeploymentRole
                - Arn
          Resource: "*"
        - Action:
```

```
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Principal:
              AWS:
                Fn::GetAtt:
                  - DeploymentRole
                  - Arn
            Resource: "*"
        Version: "2012-10-17"
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
  PipelineArtifactsBucket:
    Type: AWS::S3::Bucket
    Properties:
      VersioningConfiguration:
        Status: Enabled
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              KMSMasterKeyID:
                Fn::GetAtt:
                  - PipelineArtifactsBucketEncryptionKey
                  - Arn
              SSEAlgorithm: aws:kms
      PublicAccessBlockConfiguration:
        BlockPublicAcls: true
        BlockPublicPolicy: true
        IgnorePublicAcls: true
        RestrictPublicBuckets: true
    UpdateReplacePolicy: Retain
    DeletionPolicy: Retain
  PipelineArtifactsBucketEncryptionKeyAlias:
    Type: AWS::KMS::Alias
    Properties:
      AliasName: 'alias/codepipeline-encryption-key-{{ service.name }}'
      TargetKeyId:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
```

```
  PipelineRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codepipeline.amazonaws.com
        Version: "2012-10-17"
  PipelineRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - s3:GetObject*
              - s3:GetBucket*
              - s3:List*
              - s3:DeleteObject*
              - s3:PutObject*
              - s3:Abort*
            Effect: Allow
            Resource:
              - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - PipelineArtifactsBucket
                        - Arn
                    - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
```

```
          - Action: codestar-connections:*
            Effect: Allow
            Resource: "*"
          - Action: sts:AssumeRole
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineBuildCodePipelineActionRole
                - Arn
          - Action: sts:AssumeRole
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineDeployCodePipelineActionRole
                - Arn
        Version: "2012-10-17"
      PolicyName: PipelineRoleDefaultPolicy
      Roles:
        - Ref: PipelineRole
  Pipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      RoleArn:
        Fn::GetAtt:
          - PipelineRole
          - Arn
      Stages:
        - Actions:
            - ActionTypeId:
                Category: Source
                Owner: AWS
                Provider: CodeStarSourceConnection
                Version: "1"
              Configuration:
                ConnectionArn: '{{ service.repository_connection_arn }}'
                FullRepositoryId: '{{ service.repository_id }}'
                BranchName: '{{ service.branch_name }}'
              Name: Checkout
              OutputArtifacts:
                - Name: Artifact_Source_Checkout
              RunOrder: 1
          Name: Source
        - Actions:
            - ActionTypeId:
```

```
                Category: Build
                Owner: AWS
                Provider: CodeBuild
                Version: "1"
              Configuration:
                ProjectName:
                  Ref: BuildProject
              InputArtifacts:
                - Name: Artifact_Source_Checkout
              Name: Build
              OutputArtifacts:
                - Name: BuildOutput
              RoleArn:
                Fn::GetAtt:
                  - PipelineBuildCodePipelineActionRole
                  - Arn
              RunOrder: 1
          Name: Build {%- for service_instance in service_instances %}
        - Actions:
            - ActionTypeId:
                Category: Build
                Owner: AWS
                Provider: CodeBuild
                Version: "1"
              Configuration:
                ProjectName:
                  Ref: Deploy{{loop.index}}Project
              InputArtifacts:
                - Name: BuildOutput
              Name: Deploy
              RoleArn:
                Fn::GetAtt:
                  - PipelineDeployCodePipelineActionRole
                  - Arn
              RunOrder: 1
          Name: 'Deploy{{service_instance.name}}'
{%- endfor %}
      ArtifactStore:
        EncryptionKey:
          Id:
            Fn::GetAtt:
              - PipelineArtifactsBucketEncryptionKey
              - Arn
          Type: KMS
```

```
        Location:
          Ref: PipelineArtifactsBucket
        Type: S3
    DependsOn:
      - PipelineRoleDefaultPolicy
      - PipelineRole
  PipelineBuildCodePipelineActionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
        Version: "2012-10-17"
  PipelineBuildCodePipelineActionRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - codebuild:BatchGetBuilds
              - codebuild:StartBuild
              - codebuild:StopBuild
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - BuildProject
                - Arn
        Version: "2012-10-17"
      PolicyName: PipelineBuildCodePipelineActionRoleDefaultPolicy
      Roles:
        - Ref: PipelineBuildCodePipelineActionRole
  PipelineDeployCodePipelineActionRole:
    Type: AWS::IAM::Role
    Properties:
```

```
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
        Version: "2012-10-17"
  PipelineDeployCodePipelineActionRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - codebuild:BatchGetBuilds
              - codebuild:StartBuild
              - codebuild:StopBuild
            Effect: Allow
            Resource:
              Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":codebuild:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - ":project/Deploy*"
        Version: "2012-10-17"
      PolicyName: PipelineDeployCodePipelineActionRoleDefaultPolicy
      Roles:
        - Ref: PipelineDeployCodePipelineActionRole
Outputs:
  PipelineEndpoint:
    Description: The URL to access the pipeline
    Value: !Sub "https://${AWS::Region}.console.aws.amazon.com/codesuite/codepipeline/
pipelines/${Pipeline}/view?region=${AWS::Region}"
```

```
                ]
              }
            }
          }
        Type: CODEPIPELINE
      EncryptionKey:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
{% endfor %}
  # This role is used to build and publish an image to ECR
  PublishRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codebuild.amazonaws.com
        Version: "2012-10-17"
  PublishRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - logs:CreateLogGroup
              - logs:CreateLogStream
              - logs:PutLogEvents
            Effect: Allow
            Resource:
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/
                    - Ref: BuildProject
              - Fn::Join:
                  - ""
```

```
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":logs:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - :log-group:/aws/codebuild/
                  - Ref: BuildProject
                  - :*
          - Action:
              - codebuild:CreateReportGroup
              - codebuild:CreateReport
              - codebuild:UpdateReport
              - codebuild:BatchPutTestCases
            Effect: Allow
            Resource:
              Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":codebuild:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - :report-group/
                  - Ref: BuildProject
                  - -*
          - Action:
              - ecr:GetAuthorizationToken
            Effect: Allow
            Resource: "*"
          - Action:
              - ecr:BatchCheckLayerAvailability
              - ecr:CompleteLayerUpload
              - ecr:GetAuthorizationToken
              - ecr:InitiateLayerUpload
              - ecr:PutImage
              - ecr:UploadLayerPart
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - ECRRepo
                - Arn
          - Action:
```

```
              - proton:GetService
            Effect: Allow
            Resource: "*"
          - Action:
              - s3:GetObject*
              - s3:GetBucket*
              - s3:List*
              - s3:DeleteObject*
              - s3:PutObject*
              - s3:Abort*
            Effect: Allow
            Resource:
              - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - PipelineArtifactsBucket
                        - Arn
                    - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action:
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
        Version: "2012-10-17"
      PolicyName: PublishRoleDefaultPolicy
```

```
      Roles:
        - Ref: PublishRole

  DeploymentRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codebuild.amazonaws.com
        Version: "2012-10-17"
  DeploymentRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - logs:CreateLogGroup
              - logs:CreateLogStream
              - logs:PutLogEvents
            Effect: Allow
            Resource:
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/Deploy*Project*
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/Deploy*Project:*
          - Action:
              - codebuild:CreateReportGroup
```

```
              - codebuild:CreateReport
              - codebuild:UpdateReport
              - codebuild:BatchPutTestCases
            Effect: Allow
            Resource:
              Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":codebuild:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - :report-group/Deploy*Project
                  - -*
          - Action:
              - proton:UpdateServiceInstance
              - proton:GetServiceInstance
            Effect: Allow
            Resource: "*"
          - Action:
              - s3:GetObject*
              - s3:GetBucket*
              - s3:List*
            Effect: Allow
            Resource:
              - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - PipelineArtifactsBucket
                        - Arn
                    - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action:
```

```
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
        Version: "2012-10-17"
      PolicyName: DeploymentRoleDefaultPolicy
      Roles:
        - Ref: DeploymentRole
  PipelineArtifactsBucketEncryptionKey:
    Type: AWS::KMS::Key
    Properties:
      KeyPolicy:
        Statement:
          - Action:
              - kms:Create*
              - kms:Describe*
              - kms:Enable*
              - kms:List*
              - kms:Put*
              - kms:Update*
              - kms:Revoke*
              - kms:Disable*
              - kms:Get*
              - kms:Delete*
              - kms:ScheduleKeyDeletion
              - kms:CancelKeyDeletion
              - kms:GenerateDataKey
              - kms:TagResource
              - kms:UntagResource
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
```

```
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Principal:
              AWS:
                Fn::GetAtt:
                  - PipelineRole
                  - Arn
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Principal:
              AWS:
                Fn::GetAtt:
                  - PublishRole
                  - Arn
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Principal:
              AWS:
                Fn::GetAtt:
                  - PublishRole
                  - Arn
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
            Effect: Allow
            Principal:
```

```
              AWS:
                Fn::GetAtt:
                  - DeploymentRole
                  - Arn
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Principal:
              AWS:
                Fn::GetAtt:
                  - DeploymentRole
                  - Arn
            Resource: "*"
        Version: "2012-10-17"
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
  PipelineArtifactsBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              KMSMasterKeyID:
                Fn::GetAtt:
                  - PipelineArtifactsBucketEncryptionKey
                  - Arn
              SSEAlgorithm: aws:kms
      PublicAccessBlockConfiguration:
        BlockPublicAcls: true
        BlockPublicPolicy: true
        IgnorePublicAcls: true
        RestrictPublicBuckets: true
    UpdateReplacePolicy: Retain
    DeletionPolicy: Retain
  PipelineArtifactsBucketEncryptionKeyAlias:
    Type: AWS::KMS::Alias
    Properties:
      AliasName: 'alias/codepipeline-encryption-key-{{ service.name }}'     # resource
 parameter
      TargetKeyId:
```

```
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
  PipelineRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codepipeline.amazonaws.com
        Version: "2012-10-17"
  PipelineRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - s3:GetObject*
              - s3:GetBucket*
              - s3:List*
              - s3:DeleteObject*
              - s3:PutObject*
              - s3:Abort*
            Effect: Allow
            Resource:
              - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - PipelineArtifactsBucket
                        - Arn
                    - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
```

```
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action: codestar-connections:*
            Effect: Allow
            Resource: "*"
          - Action: sts:AssumeRole
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineBuildCodePipelineActionRole
                - Arn
          - Action: sts:AssumeRole
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineDeployCodePipelineActionRole
                - Arn
        Version: "2012-10-17"
      PolicyName: PipelineRoleDefaultPolicy
      Roles:
        - Ref: PipelineRole
  Pipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      RoleArn:
        Fn::GetAtt:
          - PipelineRole
          - Arn
      Stages:
        - Actions:
            - ActionTypeId:
                Category: Source
                Owner: AWS
                Provider: CodeStarSourceConnection
                Version: "1"
              Configuration:
                ConnectionArn: '{{ service.repository_connection_arn }}'   # resource
parameter
                FullRepositoryId: '{{ service.repository_id }}'            # resource
parameter
```

```
                BranchName: '{{ service.branch_name }}'                    # resource
parameter
              Name: Checkout
              OutputArtifacts:
                - Name: Artifact_Source_Checkout
              RunOrder: 1
          Name: Source
        - Actions:
            - ActionTypeId:
                Category: Build
                Owner: AWS
                Provider: CodeBuild
                Version: "1"
              Configuration:
                ProjectName:
                  Ref: BuildProject
              InputArtifacts:
                - Name: Artifact_Source_Checkout
              Name: Build
              OutputArtifacts:
                - Name: BuildOutput
              RoleArn:
                Fn::GetAtt:
                  - PipelineBuildCodePipelineActionRole
                  - Arn
              RunOrder: 1
          Name: Build {%- for service_instance in service_instances %}
        - Actions:
            - ActionTypeId:
                Category: Build
                Owner: AWS
                Provider: CodeBuild
                Version: "1"
              Configuration:
                ProjectName:
                  Ref: Deploy{{loop.index}}Project
              InputArtifacts:
                - Name: BuildOutput
              Name: Deploy
              RoleArn:
                Fn::GetAtt:
                  - PipelineDeployCodePipelineActionRole
                  - Arn
              RunOrder: 1
```

```
          Name: 'Deploy{{service_instance.name}}'         # resource parameter
{%- endfor %}
      ArtifactStore:
        EncryptionKey:
          Id:
            Fn::GetAtt:
              - PipelineArtifactsBucketEncryptionKey
              - Arn
          Type: KMS
        Location:
          Ref: PipelineArtifactsBucket
        Type: S3
    DependsOn:
      - PipelineRoleDefaultPolicy
      - PipelineRole
  PipelineBuildCodePipelineActionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
        Version: "2012-10-17"
  PipelineBuildCodePipelineActionRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - codebuild:BatchGetBuilds
              - codebuild:StartBuild
              - codebuild:StopBuild
            Effect: Allow
            Resource:
              Fn::GetAtt:
```

```
                - BuildProject
                - Arn
        Version: "2012-10-17"
      PolicyName: PipelineBuildCodePipelineActionRoleDefaultPolicy
      Roles:
        - Ref: PipelineBuildCodePipelineActionRole
  PipelineDeployCodePipelineActionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
        Version: "2012-10-17"
  PipelineDeployCodePipelineActionRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - codebuild:BatchGetBuilds
              - codebuild:StartBuild
              - codebuild:StopBuild
            Effect: Allow
            Resource:
              Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":codebuild:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - ":project/Deploy*"
        Version: "2012-10-17"
```

```
      PolicyName: PipelineDeployCodePipelineActionRoleDefaultPolicy
      Roles:
        - Ref: PipelineDeployCodePipelineActionRole
Outputs:
  PipelineEndpoint:
    Description: The URL to access the pipeline
    Value: !Sub "https://${AWS::Region}.console.aws.amazon.com/codesuite/codepipeline/
pipelines/${Pipeline}/view?region=${AWS::Region}"
```

CodeBuild provisioning template bundle

With CodeBuild provisioning, instead of using IaC templates to render IaC files and run them using an IaC provisioning engine, AWS Proton simply runs your shell commands. To do that, AWS Proton creates an AWS CodeBuild project for the environment, in the environment account, and starts a job to run your commands for each AWS Proton resource creation or update. When you author a template bundle, you provide a manifest that specifies infrastructure provisioning and deprovisioning commands, and any programs, scripts, and other files that these commands may need. Your commands can read inputs that AWS Proton provides, and are responsible for provisioning or deprovisioning infrastructure and generating output values.

The manifest also specifies how AWS Proton should render the input file that your code can input and get input values from. It can be rendered into JSON or HCL. For more information about input parameters, see the section called “CodeBuild provisioning parameters”. For more information about manifest files, see the section called “Manifest and wrap up”.

> **Note**
>
> You can use CodeBuild provisioning with environments and services. At this time you can't provision components this way.

Example: using the AWS CDK with CodeBuild provisioning

As an example to using CodeBuild provisioning, you can include code that uses the AWS Cloud Development Kit (AWS CDK) to provision (*deploy*) and deprovision (*destroy*) AWS resources, and a manifest that installs the CDK and runs your CDK code.

The following sections list example files you can include in a CodeBuild provisioning template bundle that provisions an environment using the AWS CDK.

Manifest

The following manifest file specifies CodeBuild provisioning, and includes the commands necessary to install and use the AWS CDK, output file processing, and reporting outputs back to AWS Proton.

Example infrastructure/manifest.yaml

```
infrastructure:
  templates:
    - rendering_engine: codebuild
      settings:
        image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
        runtimes:
          nodejs: 16
        provision:
          - npm install
          - npm run build
          - npm run cdk bootstrap
          - npm run cdk deploy -- --require-approval never --outputs-file proton-
outputs.json
          - jq 'to_entries | map_values(.value) | add | to_entries | map({key:.key,
 valueString:.value})' < proton-outputs.json > outputs.json
          - aws proton notify-resource-deployment-status-change --resource-arn
 $RESOURCE_ARN --status IN_PROGRESS --outputs file://./outputs.json
        deprovision:
          - npm install
          - npm run build
          - npm run cdk destroy
        project_properties:
          VpcConfig:
            VpcId: "{{ environment.inputs.codebuild_vpc_id }}"
            Subnets: "{{ environment.inputs.codebuild_subnets }}"
            SecurityGroupIds: "{{ environment.inputs.codebuild_security_groups }}"
```

Schema

The following schema file defines parameters for the environment. Your AWS CDK code can refer to values of these parameters during deployment.

Example schema/schema.yaml

```
schema:
```

```
  format:
    openapi: "3.0.0"
  environment_input_type: "MyEnvironmentInputType"
  types:
    MyEnvironmentInputType:
      type: object
      description: "Input properties for my environment"
      properties:
        my_sample_input:
          type: string
          description: "This is a sample input"
          default: "hello world"
        my_other_sample_input:
          type: string
          description: "Another sample input"
      required:
        - my_other_sample_input
```

AWS CDK files

The following files are an example to a Node.js CDK project.

Example infrastructure/package.json

```
{
  "name": "ProtonEnvironment",
  "version": "0.1.0",
  "bin": {
    "ProtonEnvironmente": "bin/ProtonEnvironment.js"
  },
  "scripts": {
    "build": "tsc",
    "watch": "tsc -w",
    "test": "jest",
    "cdk": "cdk"
  },
  "devDependencies": {
    "@types/jest": "^28.1.7",
    "@types/node": "18.7.6",
    "jest": "^28.1.3",
    "ts-jest": "^28.0.8",
    "aws-cdk": "2.37.1",
    "ts-node": "^10.9.1",
```

```
    "typescript": "~4.7.4"
  },
  "dependencies": {
    "aws-cdk-lib": "2.37.1",
    "constructs": "^10.1.77",
    "source-map-support": "^0.5.21"
  }
}
```

Example infrastructure/tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2018",
    "module": "commonjs",
    "lib": [
      "es2018"
    ],
    "declaration": true,
    "strict": true,
    "noImplicitAny": true,
    "strictNullChecks": true,
    "noImplicitThis": true,
    "alwaysStrict": true,
    "noUnusedLocals": false,
    "noUnusedParameters": false,
    "noImplicitReturns": true,
    "noFallthroughCasesInSwitch": false,
    "inlineSourceMap": true,
    "inlineSources": true,
    "experimentalDecorators": true,
    "strictPropertyInitialization": false,
    "resolveJsonModule": true,
    "esModuleInterop": true,
    "typeRoots": [
      "./node_modules/@types"
    ]
  },
  "exclude": [
    "node_modules",
    "cdk.out"
  ]
}
```

Example infrastructure/cdk.json

```
{
  "app": "npx ts-node --prefer-ts-exts bin/ProtonEnvironment.ts",
  "outputsFile": "proton-outputs.json",
  "watch": {
    "include": [
      "**"
    ],
    "exclude": [
      "README.md",
      "cdk*.json",
      "**/*.d.ts",
      "**/*.js",
      "tsconfig.json",
      "package*.json",
      "yarn.lock",
      "node_modules",
      "test"
    ]
  },
  "context": {
    "@aws-cdk/aws-apigateway:usagePlanKeyOrderInsensitiveId": true,
    "@aws-cdk/core:stackRelativeExports": true,
    "@aws-cdk/aws-rds:lowercaseDbIdentifier": true,
    "@aws-cdk/aws-lambda:recognizeVersionProps": true,
    "@aws-cdk/aws-cloudfront:defaultSecurityPolicyTLSv1.2_2021": true,
    "@aws-cdk-containers/ecs-service-extensions:enableDefaultLogDriver": true,
    "@aws-cdk/aws-ec2:uniqueImdsv2TemplateName": true,
    "@aws-cdk/core:target-partitions": [
      "aws",
      "aws-cn"
    ]
  }
}
```

Example infrastructure/bin/ProtonEnvironment.ts

```
#!/usr/bin/env node
import 'source-map-support/register';
import * as cdk from 'aws-cdk-lib';
import { ProtonEnvironmentStack } from '../lib/ProtonEnvironmentStack';
```

```
const app = new cdk.App();
new ProtonEnvironmentStack(app, 'ProtonEnvironmentStack', {});
```

Example infrastructure/lib/ProtonEnvironmentStack.ts

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as cdk from 'aws-cdk-lib';
import * as ssm from 'aws-cdk-lib/aws-ssm';
import input from '../proton-inputs.json';

export class ProtonEnvironmentStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, { ...props, stackName: process.env.STACK_NAME });

    const ssmParam = new ssm.StringParameter(this, "ssmParam", {
      stringValue: input.environment.inputs.my_sample_input,
      parameterName: `${process.env.STACK_NAME}-Param`,
      tier: ssm.ParameterTier.STANDARD
    })

    new cdk.CfnOutput(this, 'ssmParamOutput', {
      value: ssmParam.parameterName,
      description: 'The name of the ssm parameter',
      exportName: `${process.env.STACK_NAME}-Param`
    });
  }
}
```

Rendered input file

When you create an environment using a CodeBuild-based provisioning template, AWS Proton renders an input file with input parameter values that you provided. Your code can refer to these values. The following file is an example to a rendered input file.

Example infrastructure/proton-inputs.json

```
{
  "environment": {
    "name": "myenv",
    "inputs": {
      "my_sample_input": "10.0.0.0/16",
      "my_other_sample_input": "11.0.0.0/16"
```

```
    }
  }
}
```

Terraform IaC files

Learn how to use Terraform infrastructure as code (IaC) files with AWS Proton. Terraform is a widely used open-source IaC engine that was developed by HashiCorp. Terraform modules are developed in HashiCorp's HCL language, and support several backend infrastructures providers, including Amazon Web Services.

AWS Proton supports self-managed provisioning for Terraform IaC.

For a complete example of a provisioning repository that responds to pull requests and implements infrastructure provisioning, see Terraform OpenSource GitHub Actions automation template for AWS Proton on GitHub.

How self-managed provisioning works with Terraform IaC template bundle files:

1. When you create an environment from Terraform template bundles, AWS Proton compiles your `.tf` files with console or `spec file` input parameters.
2. It makes a pull request to merge the compiled IaC files to repository that you have registered with AWS Proton.
3. If the request is approved, AWS Proton waits on provisioning status that you provide.
4. If the request is rejected, the environment creation is cancelled.
5. If the pull request times out, environment creation *isn't* complete.

AWS Proton with Terraform IaC considerations:

- AWS Proton doesn’t manage your Terraform provisioning.
- You must register a provisioning repository with AWS Proton. AWS Proton makes pull requests on this repository.
- You must create a CodeStar connection to connect AWS Proton with your provisioning repository.
- To provision from AWS Proton compiled IaC files, you must respond to AWS Proton pull requests. AWS Proton makes pull requests after environment and service create and update actions. For more information, see AWS Proton environments and AWS Proton services.

- To provision a pipeline from AWS Proton compiled IaC files, you must create a CI/CD pipeline repository.
- Your pull request based provisioning automation must include steps to notify AWS Proton of any provisioned AWS Proton resource status changes. You can use the AWS Proton NotifyResourceDeploymentStatusChange API.
- You can't deploy services, pipelines, and components created from CloudFormation IaC files to environments created from Terraform IaC files.
- You can't deploy services, pipelines, and components created from Terraform IaC files to environments created from CloudFormation IaC files.

When preparing your Terraform IaC files for AWS Proton, you attach namespaces to your input variables, as shown in the following examples. For more information, see Parameters.

Example 1: AWS Proton environment Terraform IaC file

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
  // This tells terraform to store the state file in s3 at the location
  // s3://terraform-state-bucket/tf-os-sample/terraform.tfstate
  backend "s3" {
    bucket = "terraform-state-bucket"
    key    = "tf-os-sample/terraform.tfstate"
    region = "us-east-1"
  }
}

// Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
  default_tags {
    tags = var.proton_tags
  }
}

resource "aws_ssm_parameter" "my_ssm_parameter" {
```

```
  name  = "my_ssm_parameter"
  type  = "String"
  // Use the Proton environment.inputs. namespace
  value = var.environment.inputs.ssm_parameter_value
}
```

Compiled infrastructure as code

When you create an environment or service, AWS Proton compiles your infrastructure as code files with console or `spec file` inputs. It creates `proton.`*`resource-type`*`.variables.tf` and `proton.auto.tfvars.json` files for your inputs that can be used by Terraform, as shown in the following examples. These files are located in a specified repository in a folder that matches the environment or service instance name.

The example shows how AWS Proton includes tags in the variable definition and variable values, and how you can propagate these AWS Proton tags to provisioned resources. For more information, see the section called “Tag propagation to provisioned resources”.

Example 2: compiled IaC files for an environment named "dev".

dev/environment.tf:

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
  // This tells terraform to store the state file in s3 at the location
  // s3://terraform-state-bucket/tf-os-sample/terraform.tfstate
  backend "s3" {
    bucket = "terraform-state-bucket"
    key    = "tf-os-sample/terraform.tfstate"
    region = "us-east-1"
  }
}

// Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
  default_tags {
```

```
    tags = var.proton_tags
  }
}

resource "aws_ssm_parameter" "my_ssm_parameter" {
  name  = "my_ssm_parameter"
  type  = "String"
  // Use the Proton environment.inputs. namespace
  value = var.environment.inputs.ssm_parameter_value
}
```

dev/proton.environment.variables.tf:

```
variable "environment" {
  type = object({
    inputs = map(string)
    name = string
  })
}

variable "proton_tags" {
  type = map(string)
  default = null
}
```

dev/proton.auto.tfvars.json:

```
{
  "environment": {
    "name": "dev",
    "inputs": {
      "ssm_parameter_value": "MyNewParamValue"
    }
  }

  "proton_tags" : {
    "proton:account" : "123456789012",
    "proton:template" : "arn:aws:proton:us-east-1:123456789012:environment-template/
fargate-env",
    "proton:environment" : "arn:aws:proton:us-east-1:123456789012:environment/dev"
  }
}
```

Repository paths

AWS Proton uses console or spec inputs from environment or service create actions to find the repository and path where it is to locate the compiled IaC files. The input values are passed to namespaced input parameters.

AWS Proton supports two repository path layouts. In the following examples, the paths are named by the namespaced resource parameters from two environments. Each environment has service instances of two services, and the service instances of one of the services have directly defined components.

Resource type	Name parameter	=	Resource name
Environment	`environment.name`		`"env-prod"`
Environment	`environment.name`		`"env-staged"`
Service	`service.name`		`"service-one"`
Service instance	`service_instance.name`	=	`"instance-one-prod"`
Service instance	`service_instance.name`		`"instance-one-staged"`
Service	`service.name`		`"service-two"`
Service instance	`service_instance.name`		`"instance-two-prod"`

Resource type	Name parameter	=	Resource name
Component	`service_instance.components.default.name`		`"component-prod"`
Service instance	`service_instance.name`		`"instance-two-staged"`
Component	`service_instance.components.default.name`		`"component-staged"`

Layout 1

If AWS Proton finds the specified repository with an `environments` folder, it creates a folder that includes the compiled IaC files and is named with the `environment.name`.

If AWS Proton finds the specified repository with an `environments` folder that contains a folder name that matches a service instance compatible environment name, it creates a folder that includes the compiled instance IaC files and is named with the `service_instance.name`.

```
/repo
    /environments
        /env-prod                           # environment folder
            main.tf
            proton.environment.variables.tf
            proton.auto.tfvars.json

            /service-one-instance-one-prod    # instance folder
                main.tf
                proton.service_instance.variables.tf
                proton.auto.tfvars.json

            /service-two-instance-two-prod    # instance folder
                main.tf
                proton.service_instance.variables.tf
                proton.auto.tfvars.json
```

```
            /component-prod                   # component folder
                main.tf
                proton.component.variables.tf
                proton.auto.tfvars.json

        /env-staged                           # environment folder
            main.tf
            proton.variables.tf
            proton.auto.tfvars.json

            /service-one-instance-one-staged  # instance folder
                main.tf
                proton.service_instance.variables.tf
                proton.auto.tfvars.json

            /service-two-instance-two-staged  # instance folder
                main.tf
                proton.service_instance.variables.tf
                proton.auto.tfvars.json

            /component-staged                 # component folder
                main.tf
                proton.component.variables.tf
                proton.auto.tfvars.json
```

Layout 2

If AWS Proton finds the specified repository without an `environments` folder, it creates an `environment.name` folder where it locates the compiled environment IaC files.

If AWS Proton finds the specified repository with a folder name that matches a service instance compatible environment name, it creates a `service_instance.name` folder where it locates the compiled instance IaC files.

```
/repo
    /env-prod                             # environment folder
        main.tf
        proton.environment.variables.tf
        proton.auto.tfvars.json

        /service-one-instance-one-prod    # instance folder
            main.tf
            proton.service_instance.variables.tf
```

```
            proton.auto.tfvars.json

        /service-two-instance-two-prod    # instance folder
            main.tf
            proton.service_instance.variables.tf
            proton.auto.tfvars.json

        /component-prod                   # component folder
            main.tf
            proton.component.variables.tf
            proton.auto.tfvars.json

    /env-staged                           # environment folder
        main.tf
        proton.variables.tf
        proton.auto.tfvars.json

        /service-one-instance-one-staged  # instance folder
            main.tf
            proton.service_instance.variables.tf
            proton.auto.tfvars.json

        /service-two-instance-two-staged  # instance folder
            main.tf
            proton.service_instance.variables.tf
            proton.auto.tfvars.json

        /component-staged                 # component folder
            main.tf
            proton.component.variables.tf
            proton.auto.tfvars.json
```

Schema file

As an administrator, when you use the Open API Data Models (schemas) section to define a parameter schema YAML file for your template bundle, AWS Proton can validate parameter value inputs against the requirements that you defined in your schema.

For more information about formats and available keywords, see the Schema object section of the OpenAPI.

Schema requirements for environment template bundles

Your schema must follow the Data Models (schemas) section of the OpenAPI in the YAML format. It must also be a part of your environment template bundle.

For your environment schema, you must include the formatted headers to establish that you're using the Data Models (schemas) section of the Open API. In the following environment schema examples, these headers appear in the first three lines.

An `environment_input_type` must be included and defined with a name that you provide. In the following examples, this is defined on line 5. By defining this parameter, you associate it with an AWS Proton environment resource.

To follow the Open API schema model, you must include `types`. In the following example, this is line 6.

Following `types`, you must define an `environment_input_type` type. You define the input parameters for your environment as properties of the `environment_input_type`. You must include at least one property with a name that matches at least one parameter that's listed in the environment infrastructure as code (IaC) file that's associated with schema.

When you create an environment and provide customized parameter values, AWS Proton uses the schema file to match, validate, and inject them into the curly braced parameters in the associated CloudFormation IaC file. For each property (parameter), provide a `name` and `type`. Optionally, also provide a `description`, `default`,and `pattern`.

The defined parameters for the following example *standard* environment template schema include `vpc_cidr`, `subnet_one_cidr`, and `subnet_two_cidr` with the `default` keyword and default values. When you create an environment with this environment template bundle schema, you can accept the default values or provide your own. If a parameter *doesn't* have a default value and is listed as a `required` property (parameter), you must provide values for it when you create an environment.

The second example *standard* environment template schema lists the `required` parameter `my_other_sample_input`.

You can create a schema for two types of environment templates. For more information, see Register and publish templates.

- ***Standard* environment templates**

In the following example, an environment input type is defined with a description and input properties. This schema example can be used with the AWS Proton CloudFormation IaC file shown in Example 3.

Example schema for a *standard* environment template:

```
schema:                           # required
  format:                         # required
    openapi: "3.0.0"              # required
  # required              defined by administrator
  environment_input_type: "PublicEnvironmentInput"
  types:                          # required
    # defined by administrator
    PublicEnvironmentInput:
      type: object
      description: "Input properties for my environment"
      properties:
        vpc_cidr:                  # parameter
          type: string
          description: "This CIDR range for your VPC"
          default: 10.0.0.0/16
          pattern: ([0-9]{1,3}\.){3}[0-9]{1,3}($|/(16|24))
        subnet_one_cidr:           # parameter
          type: string
          description: "The CIDR range for subnet one"
          default: 10.0.0.0/24
          pattern: ([0-9]{1,3}\.){3}[0-9]{1,3}($|/(16|24))
        subnet_two_cidr:           # parameter
          type: string
          description: "The CIDR range for subnet one"
          default: 10.0.1.0/24
          pattern: ([0-9]{1,3}\.){3}[0-9]{1,3}($|/(16|24))
```

Example schema for a *standard* environment template that includes a `required` parameter:

```
schema:                           # required
  format:                         # required
    openapi: "3.0.0"              # required
  # required              defined by administrator
  environment_input_type: "MyEnvironmentInputType"
  types:                          # required
```

```
    # defined by administrator
    MyEnvironmentInputType:
      type: object
      description: "Input properties for my environment"
      properties:
        my_sample_input:           # parameter
          type: string
          description: "This is a sample input"
          default: "hello world"
        my_other_sample_input:     # parameter
          type: string
          description: "Another sample input"
        another_optional_input:    # parameter
          type: string
          description: "Another optional input"
          default: "!"
      required:
        - my_other_sample_input
```

- ***Customer managed*** **environment templates**

  In the following example, the schema only includes a list of outputs that replicate the outputs from the IaC that you used to provision your *customer managed* infrastructure. You need to define output value types as *strings only* (*not* lists, arrays or other types). For example, the next code snippet shows the outputs section of an external AWS CloudFormation template. This is from the template shown in Example 1. It can be used to create external *customer managed* infrastructure for an AWS Proton Fargate service created from Example 4.

  > **Important**
  >
  > As an administrator, you must ensure that your provisioned and managed infrastructure and all output parameters are compatible with the associated *customer managed* environment templates. AWS Proton can't account for changes on your behalf because these changes aren't visible to AWS Proton. Inconsistencies result in failures.

  Example CloudFormation IaC file outputs for a *customer managed* environment template:

```
// Cloudformation Template Outputs
[...]
Outputs:
```

```
  ClusterName:
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'
  ECSTaskExecutionRole:
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'
  VpcId:
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'
[...]
```

The schema for the corresponding AWS Proton *customer managed* environment template bundle is shown in the following example. Each output value is defined as a string.

Example schema for a *customer managed* environment template:

```
schema:                           # required
  format:                         # required
    openapi: "3.0.0"              # required
  # required              defined by administrator
  environment_input_type: "EnvironmentOutput"
  types:                          # required
    # defined by administrator
    EnvironmentOutput:
      type: object
      description: "Outputs of the environment"
      properties:
        ClusterName:              # parameter
          type: string
          description: "The name of the ECS cluster"
        ECSTaskExecutionRole:     # parameter
          type: string
          description: "The ARN of the ECS role"
        VpcId:                    # parameter
          type: string
          description: "The ID of the VPC that this stack is deployed in"
[...]
```

Schema requirements for service template bundles

Your schema must follow the Data Models (schemas) section of the OpenAPI in YAML format as shown in the following examples. You must provide a schema file in your service template bundle.

In the following service schema examples, you must include the formatted headers. In the following example, this is in the first three lines. This is to establish that you're using the Data Models (schemas) section of the Open API.

A `service_input_type` must be included and defined with a name that you provide. In the following example, this is in line 5. This associates the parameters with an AWS Proton service resource.

An AWS Proton service pipeline is included by default when you use the console or the CLI to create a service. When you include a service pipeline for your service, you must include `pipeline_input_type` with a name that you provide. In the following example, this is in line 7. *Don't* include this parameter if you *aren't* including an AWS Proton service pipeline. For more information, see Register and publish templates.

To follow the Open API schema model, you must include `types` In the following example, this is in line 9.

Following `types`, you must define a `service_input_type` type. You define the input parameters for your service as properties of the `service_input_type`. You must include at least one property with a name that matches at least one parameter listed in the service infrastructure as code (IaC) file that is associated with schema.

To define a service pipeline, below your `service_input_type` definition, you must define a `pipeline_input_type`. As above, you must include at least one property with a name that matches at least one parameter listed in a pipeline IaC file that is associated with schema. *Don't* include this definition if you *aren't* including an AWS Proton service pipeline.

When you, as an administrator or developer, create a service and provide customized parameter values, AWS Proton uses the schema file to match, validate, and inject them into the associated CloudFormation IaC file's curly braced parameters. For each property (parameter), provide a `name` and a `type`. Optionally, also provide a `description`, `default`, and `pattern`.

The defined parameters for the example schema include `port`, `desired_count`, `task_size` and `image` with the `default` keyword and default values. When you create a service with this service template bundle schema, you can accept the default values or provide your own. The parameter

`unique_name` is also included in the example and *doesn't* have a default value. It is listed as a `required` property (parameter). You, as administrator or developer, must provide values for `required` parameters when you create services.

If you want to create a service template with a service pipeline, include the `pipeline_input_type` in your schema.

Example service schema file for a service that includes an AWS Proton service pipeline.

This schema example can be used with the AWS Proton IaC files shown in Example 4 and Example 5. A service pipeline is included.

```
schema:                           # required
  format:                         # required
    openapi: "3.0.0"              # required
  # required           defined by administrator
  service_input_type: "LoadBalancedServiceInput"
  # only include if including AWS Proton service pipeline, defined by administrator
  pipeline_input_type: "PipelineInputs"

  types:                          # required
    # defined by administrator
    LoadBalancedServiceInput:
      type: object
      description: "Input properties for a loadbalanced Fargate service"
      properties:
        port:                     # parameter
          type: number
          description: "The port to route traffic to"
          default: 80
          minimum: 0
          maximum: 65535
        desired_count:            # parameter
          type: number
          description: "The default number of Fargate tasks you want running"
          default: 1
          minimum: 1
        task_size:                # parameter
          type: string
          description: "The size of the task you want to run"
          enum: ["x-small", "small", "medium", "large", "x-large"]
          default: "x-small"
        image:                    # parameter
```

```
          type: string
          description: "The name/url of the container image"
          default: "public.ecr.aws/z9d2n7e1/nginx:1.19.5"
          minLength: 1
          maxLength: 200
        unique_name:               # parameter
          type: string
          description: "The unique name of your service identifier. This will be used
 to name your log group, task definition and ECS service"
          minLength: 1
          maxLength: 100
      required:
        - unique_name
    # defined by administrator
    PipelineInputs:
      type: object
      description: "Pipeline input properties"
      properties:
        dockerfile:                # parameter
          type: string
          description: "The location of the Dockerfile to build"
          default: "Dockerfile"
          minLength: 1
          maxLength: 100
        unit_test_command:         # parameter
          type: string
          description: "The command to run to unit test the application code"
          default: "echo 'add your unit test command here'"
          minLength: 1
          maxLength: 200
```

If you want to create a service template without a service pipeline, *don't* include the `pipeline_input_type` in your schema, as shown in the following example.

Example service schema file for a service that *doesn't* include an AWS Proton service pipeline

```
schema:                            # required
  format:                          # required
    openapi: "3.0.0"               # required
  # required            defined by administrator
  service_input_type: "MyServiceInstanceInputType"

  types:                           # required
```

```
    # defined by administrator
    MyServiceInstanceInputType:
      type: object
      description: "Service instance input properties"
      required:
        - my_sample_service_instance_required_input
      properties:
        my_sample_service_instance_optional_input:   # parameter
          type: string
          description: "This is a sample input"
          default: "hello world"
        my_sample_service_instance_required_input:   # parameter
          type: string
          description: "Another sample input"
```

Wrap up template files for AWS Proton

After preparing your environment and service infrastructure as code (IaC) files and their respective schema files, you must organize them in directories. You must also create a manifest YAML file. The manifest file lists the IaC files in a directory, the rendering engine, and the template language used to develop the IaC in this template.

> **Note**
>
> A manifest file can also be used independently of template bundles, as a direct input to *directly defined components*. In this case, it always specifies a single IaC template file, for both CloudFormation and Terraform. For more information about components, see *Components*.

The manifest file needs to adhere to the format and content shown in the following example.

CloudFormation manifest file format:

With CloudFormation, you list a single file.

```
infrastructure:
  templates:
    - file: "cloudformation.yaml"
      rendering_engine: jinja
```

```
      template_language: cloudformation
```

Terraform manifest file format:

With terraform, you can explicitly list a single file or use the wildcard `*` to list each of the files in a directory.

> **Note**
>
> The wildcard only includes files whose names end with `.tf`. Other files are ignored.

```
infrastructure:
  templates:
    - file: "*"
      rendering_engine: hcl
      template_language: terraform
```

CodeBuild-based provisioning manifest file format:

With CodeBuild-based provisioning, you specify provisioning and deprovisioning shell commands.

> **Note**
>
> In addition to the manifest, your bundle should include any files that your commands depend on.

The following example manifest uses CodeBuild-based provisioning to provision (*deploy*) and deprovision (*destroy*) resources using the AWS Cloud Development Kit (AWS CDK) (AWS CDK). The template bundle should also include the CDK code.

During provisioning, AWS Proton creates an input file with values for input parameters that you defined in the template's schema with the name `proton-input.json`.

```
infrastructure:
  templates:
    - rendering_engine: codebuild
      settings:
        image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
```

```
        runtimes:
          nodejs: 16
        provision:
          - npm install
          - npm run build
          - npm run cdk bootstrap
          - npm run cdk deploy -- --require-approval never --outputs-file proton-
outputs.json
          - jq 'to_entries | map_values(.value) | add | to_entries | map({key:.key,
 valueString:.value})' < proton-outputs.json > outputs.json
          - aws proton notify-resource-deployment-status-change --resource-arn
 $RESOURCE_ARN --status IN_PROGRESS --outputs file://./outputs.json
        deprovision:
          - npm install
          - npm run build
          - npm run cdk destroy
        project_properties:
          VpcConfig:
            VpcId: "{{ environment.inputs.codebuild_vpc_id }}"
            Subnets: "{{ environment.inputs.codebuild_subnets }}"
            SecurityGroupIds: "{{ environment.inputs.codebuild_security_groups }}"
```

After you set up the directories and manifest files for your environment or service template bundle, you gzip the directories into a tar ball and upload them to an Amazon Simple Storage Service (Amazon S3) bucket where AWS Proton can retrieve them, or to a template sync Git repository.

When you create a minor version of an environment or a service template that you registered with AWS Proton, you provide the path to your environment or service template bundle tar ball that's located in your S3 bucket. AWS Proton saves it with the new template minor version. You can select the new template minor version to create or update environments or services with AWS Proton.

Environment template bundle wrap up

There are two types of environment template bundles that you create for AWS Proton.

- To create an environment template bundle for a *standard* environment template, organize the schema, infrastructure as code (IaC) files and manifest file in directories as shown in the following environment template bundle directory structure.
- To create an environment template bundle for a *customer managed* environment template, provide only the schema file and directory. *Don't* include the infrastructure directory and files. AWS Proton throws an error if the infrastructure directory and files are included.

For more information, see Register and publish templates.

CloudFormation environment template bundle directory structure:

```
/schema
  schema.yaml
/infrastructure
  manifest.yaml
  cloudformation.yaml
```

Terraform environment template bundle directory structure:

```
/schema
  schema.yaml
/infrastructure
  manifest.yaml
  environment.tf
```

Service template bundle wrap up

To create a service template bundle, you must organize the schema, infrastructure as code (IaC) files, and manifest files into directories as shown in the service template bundle directory structure example.

If you *don't* include a service pipeline in your template bundle, *don't* include the pipeline directory and files and set `"pipelineProvisioning": "CUSTOMER_MANAGED"` when you create the service template that is to be associated with this template bundle.

> **Note**
>
> You can't modify `pipelineProvisioning` after the service template is created.

For more information, see Register and publish templates.

CloudFormation service template bundle directory structure:

```
/schema
  schema.yaml
/instance_infrastructure
```

```
  manifest.yaml
  cloudformation.yaml
/pipeline_infrastructure
  manifest.yaml
  cloudformation.yaml
```

Terraform service template bundle directory structure:

```
/schema
  schema.yaml
/instance_infrastructure
  manifest.yaml
  instance.tf
/pipeline_infrastructure
  manifest.yaml
  pipeline.tf
```

Template bundle considerations

- **Infrastructure as code (IaC) files**

  AWS Proton audits templates for the correct file format. However, AWS Proton doesn't check for template development, dependency, and logic errors. For example, assume that you specified the creation of an Amazon S3 bucket in your AWS CloudFormation IaC file as part of your service or environment template. A service is created based on those templates. Now, suppose at some point you want to delete the service. If the specified S3 bucket *isn't* empty and the CloudFormation IaC file *doesn't* mark it as `Retain` in the `DeletionPolicy`, AWS Proton fails on the service delete operation.
- **Bundle file size limits and format**
  - Bundle file size, count, and name size limits can be found at AWS Proton quotas.
  - The template bundle directories of files are gzipped into a tar ball and located in an Amazon Simple Storage Service (Amazon S3) bucket.
  - Each file in the bundle must be a valid formatted YAML file.
- **S3 bucket template bundle encryption**

  If you want to encrypt sensitive data in your template bundles at rest in your S3 bucket, use SSE-S3 or SSE-KMS keys to allow AWS Proton to retrieve them.

AWS Proton templates

To add your template bundle to your AWS Proton template library, create a template minor version and register it with AWS Proton. When creating the template, provide the name of the Amazon S3 bucket and path for your template bundle. After templates are published, they can be selected by platform team members and developers. After they're selected, AWS Proton uses the template to create and provision infrastructure and applications.

As an administrator, you can create and register an environment template with AWS Proton. This environment template can then be used to deploy multiple environments. For example, it can be used to deploy "dev," "staging," and "prod" environments. The "dev" environment might include a VPC with private subnets and a restrictive access policy to all resources. Environment outputs can be used as inputs for services.

You can create and register environment templates to create two different types of environments. Both you and developers can use AWS Proton to deploy services to both types.

- Register and publish a *standard* environment template that AWS Proton uses to create a *standard* environment that provisions and manages the environment infrastructure.
- Register and publish a *customer managed* environment template that AWS Proton uses to create a customer managed environment that connects to your existing provisioned infrastructure. AWS Proton *doesn't* manage your existing provisioned infrastructure.

You can create and register service templates with AWS Proton to deploy services to environments. An AWS Proton environment must be created before a service can be deployed to it.

The following list describes how you create and manage templates with AWS Proton.

- (Optional) Prepare an IAM role to control developer access to AWS Proton API calls and AWS Proton IAM service roles. For more information, see the section called “IAM Roles”.
- Compose a template bundle. For more information, see Template bundles.
- Create and register a template with AWS Proton after the template bundle is composed, compressed, and saved in an Amazon S3 bucket. You can do this either in the console or by using the AWS CLI.
- Test and use the template to create and manage AWS Proton provisioned resources after it's registered with AWS Proton.

- Create and manage major and minor versions of the template throughout the life of the template.

You can manage template versions manually or with template sync configurations:

- Use the AWS Proton console and AWS CLI to create a new minor or major version.
- Create a template sync configuration that lets AWS Proton automatically create a new minor or major version when it detects a change to your template bundle in a repository that you define.

For additional information, see the *The AWS Proton Service API Reference*.

Topics

- Versioned templates
- Register and publish templates
- View template data
- Update a template
- Delete templates
- Template sync configurations
- Service sync configurations

Versioned templates

As an administrator or a member of a platform team, you define, create, and manage a library of versioned templates that are used to provision infrastructure resources. There are two types of template versions—minor versions and major versions.

- *Minor versions* – Changes to the template that have a backward compatible schema. These changes don't require the developer to provide new information when updating to the new template version.

  When you attempt to make a minor version change, AWS Proton makes a best-effort attempt to determine whether the schema of the new version is backward compatible with the previous minor versions of the template. If the new schema isn't backward compatible, AWS Proton fails the registration of the new minor version.

> **Note**
>
> Compatibility is determined solely based on schema. AWS Proton doesn't check if the template bundle infrastructure as code (IaC) file is backward compatible with the previous minor versions. For example, AWS Proton doesn't check if the new IaC file causes breaking changes for the applications that are running on the infrastructure provisioned by a previous minor version of the template.

- *Major versions* – Changes to the template that may not be not backward compatible. These changes typically require new inputs from the developer and often involve template schema changes.

  You may sometimes choose to designate a backward compatible change as a major version based on your team's operational model.

The way AWS Proton determines if a template version request is for a minor or major version depends on the way template changes are tracked:

- When you explicitly make a request to create a new template version, you request a major version by specifying a major version number, and you request a minor version by not specifying a major version number.
- When you use template sync (and therefore you don't make explicit template version requests), AWS Proton attempts to create new minor versions for template changes that occur in the existing YAML file. AWS Proton creates a major version when you create a new directory for the new template change (for example, move from v1 to v2).

> **Note**
>
> A new minor version registration based on template sync still fails if AWS Proton determines that the change isn't backward compatible.

When you publish a new version of a template, it becomes the **Recommended** version if it's the highest major and minor version. New AWS Proton resources are created using the new recommended version, and AWS Proton prompts administrators to use the new version and to update existing AWS Proton resources that are using an outdated version.

Register and publish templates

You can register and publish environment and service templates with AWS Proton, as described in the following sections.

You can create a new version of a template with the console or AWS CLI.

Alternatively, you can use the console or AWS CLI to create a template and configure a configure a template sync for it. This configuration lets AWS Proton sync from template bundles located in registered git repositories that you have defined. Whenever a commit is pushed to your repository that changes one of your template bundles, a new minor or major version of your template is created, if the version doesn't already exist. To learn more about template sync configuration prerequisites and requirements, see Template sync configurations.

Register and publish environment templates

You can register and publish the following types of environment templates.

- Register and publish a *standard* environment template that AWS Proton uses to deploy and manage environment infrastructure.
- Register and publish a *customer managed* environment template that AWS Proton uses to connect to your existing provisioned infrastructure that you manage. AWS Proton *doesn't* manage your existing provisioned infrastructure.

> **Important**
>
> As an administrator, ensure that your provisioned and managed infrastructure and all output parameters are compatible with associated *customer managed* environment templates. AWS Proton can't account for changes on your behalf because these changes aren't visible to AWS Proton. Inconsistencies result in failures.

You can use the console or the AWS CLI to register and publish an environment template.

AWS Management Console

Use the console to register and publish a new environment template.

1. In the AWS Proton console, choose **Environment templates**.

2. Choose **Create environment template**.
3. In the **Create environment template** page, in the **Template options** section, choose one of the two available template options.
   - **Create a template for provisioning new environments**.
   - **Create a template to use provisioned infrastructure that you manage**.
4. If you chose **Create a template for provisioning new environments**, in the **Template bundle source** section, choose one of the three available template bundle source options. To learn more about requirements and prerequisites for syncing templates, see Template sync configurations.
   - **Use one of our sample template bundles**.
   - **Use your own template bundle**.
   - **Sync templates from Git**.
5. **Provide a path to a template bundle.**
   a. If you chose **Use one of our sample template bundles**:

      In the **Sample template bundle** section, select a sample template bundle.

   b. If you chose **Sync templates from Git**, in the **Source code** section:

      i. Select the repository for your template sync configuration.

      ii. Enter the name of the repository branch to sync from.

      iii. (Optional) Enter name of a directory to limit the search for your template bundle.

   c. Otherwise, in the **S3 bundle location** section, provide a path to your template bundle.
6. In the **Template details** section.
   a. Enter a **Template name**.

   b. (Optional) Enter a **Template display name**.

   c. (Optional) Enter a **Template description** for the environment template.
7. (Optional) Check the check box for **Customize encryption settings (advanced)** in the **Encryption settings** section to provide your own encryption key.
8. (Optional) In the **Tags** section, choose **Add new tag** and enter a key and value to create a customer managed tag.
9. Choose **Create Environment template**.

You're now on a new page that displays the status and details for your new environment template. These details include a list of AWS and customer managed tags. AWS Proton automatically generates AWS managed tags for you when you create AWS Proton resources. For more information, see AWS Proton resources and tagging.

10. The status of a new environment template status starts in the **Draft** state. You and others with `proton:CreateEnvironment` permissions can view and access it. Follow the next step to make the template available to others.
11. In the **Template versions** section, choose the radio button to the left of the minor version of the template you just created (1.0). As an alternative, you can choose **Publish** in the info alert and skip the next step.
12. In the **Template versions** section, choose **Publish**.
13. The template status changes to **Published**. Because it's the latest version of the template, it's the **Recommended** version.
14. In the navigation pane, select **Environment templates** to view a list of your environment templates and details.

Use the console to register new major and minor versions of an environment template.

For more information, see Versioned templates.

1. In the AWS Proton console, choose **Environment Templates**.
2. In the list of environment templates, choose the name of the environment template that you want to create a major or minor version for.
3. In the environment template detail view, choose **Create new version** in the **Template versions** section.
4. In the **Create a new environment template version** page, in the **Template bundle source** section, choose one of the two available template bundle source options.
   - **Use one of our sample template bundles**.
   - **Use your own template bundle**.
5. Provide a path to the selected template bundle.
   - If you chose **Use one of our sample template bundles**, in the **Sample template bundle** section, select a sample template bundle.

- If you chose **Use your own template bundle**, in the **S3 bundle location** section, choose the path to your template bundle.

6. In the **Template details** section.

   a. (Optional) Enter a **Template display name**.

   b. (Optional) Enter a **Template description** for the service template.

7. In the **Template details** section, choose one of the following options.

   - To create a minor version, keep the check box **Check to create a new major version** empty.
   - To create a major version, check the check box **Check to create a new major version**.

8. Continue through the console steps to create the new minor or major version and choose **Create new version**.

AWS CLI

Use the CLI to register and publish a new environment template as shown in the following steps.

1. Create a *standard* OR *customer managed* environment template by specifying the region, name, display name (optional), and description (optional).

   a. Create a *standard* environment template.

      Run the following command:

```
$ aws proton create-environment-template \
    --name "simple-env" \
    --display-name "Fargate" \
    --description "VPC with public access"
```

      Response:

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env",
        "createdAt": "2020-11-11T23:02:45.336000+00:00",
```

```
        "description": "VPC with public access",
        "displayName": "VPC",
        "lastModifiedAt": "2020-11-11T23:02:45.336000+00:00",
        "name": "simple-env"
    }
}
```

b. Create a *customer managed* environment template by adding the `provisioning` parameter with value `CUSTOMER_MANAGED`.

Run the following command:

```
$ aws proton create-environment-template \
    --name "simple-env" \
    --display-name "Fargate" \
    --description "VPC with public access" \
    --provisioning "CUSTOMER_MANAGED"
```

Response:

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env",
        "createdAt": "2020-11-11T23:02:45.336000+00:00",
        "description": "VPC with public access",
        "displayName": "VPC",
        "lastModifiedAt": "2020-11-11T23:02:45.336000+00:00",
        "name": "simple-env",
        "provisioning": "CUSTOMER_MANAGED"
    }
}
```

2. **Create a minor version 0 of major version 1 of the environment template**

This and the remaining steps are the same for both the *standard* and *customer managed* environment templates.

Include the template name, major version, and the S3 bucket name and key for the bucket that contains your environment template bundle.

Run the following command:

```
$ aws proton create-environment-template-version \
    --template-name "simple-env" \
    --description "Version 1" \
    --source s3="{bucket=your_s3_bucket, key=your_s3_key}"
```

Response:

```
{
    "environmentTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env:1.0",
        "createdAt": "2020-11-11T23:02:47.763000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:47.763000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "REGISTRATION_IN_PROGRESS",
        "templateName": "simple-env"
    }
}
```

3. Use the get command to check the registrations status.

   Run the following command:

```
$ aws proton get-environment-template-version \
    --template-name "simple-env" \
    --major-version "1" \
    --minor-version "0"
```

   Response:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env:1.0",
        "createdAt": "2020-11-11T23:02:47.763000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:47.763000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
```

```
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 environment_input_type: \"MyEnvironmentInputType\"\n  types:\n
 MyEnvironmentInputType:\n      type: object\n      description: \"Input
 properties for my environment\"\n      properties:\n        my_sample_input:\n
          type: string\n          description: \"This is a sample input\"\n
      default: \"hello world\"\n        my_other_sample_input:\n          type:
 string\n          description: \"Another sample input\"\n      required:\n
   - my_other_sample_input\n",
        "status": "DRAFT",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

4. Publish of minor version 0 of major version 1 of the environment template by providing the template name and the major and minor version. This version is the `Recommended` version.

   Run the following command:

```
$ aws proton update-environment-template-version \
    --template-name "simple-env" \
    --major-version "1" \
    --minor-version "0" \
    --status "PUBLISHED"
```

   Response:

```
{
    "environmentTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env:1.0",
        "createdAt": "2020-11-11T23:02:47.763000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:54.610000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 environment_input_type: \"MyEnvironmentInputType\"\n  types:\n
 MyEnvironmentInputType:\n      type: object\n      description: \"Input
 properties for my environment\"\n      properties:\n        my_sample_input:\n
          type: string\n          description: \"This is a sample input\"\n
```

```
      default: \"hello world\"\n        my_other_sample_input:\n          type:
 string\n          description: \"Another sample input\"\n      required:\n
   - my_other_sample_input\n",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

After creating a new template using the AWS CLI, you can view a list of AWS and customer managed tags. AWS Proton automatically generates AWS managed tags for you. You can also modify and create customer managed tags using the AWS CLI. For more information, see AWS Proton resources and tagging.

Run the following command:

```
$ aws proton list-tags-for-resource \
    --resource-arn "arn:aws:proton:region-id:123456789012:environment-
template/simple-env"
```

Register and publish service templates

When you create a service template version, you specify a list of compatible environment templates. That way, when developers select a service template, they have options for which environment to deploy their service to.

Before creating a service from a service template or before publishing a service template, confirm that environments are deployed from the listed compatible environment templates.

You *can't* update a service to the new major version if it's deployed to an environment that was built from a removed compatible environment template.

To add or remove compatible environment templates for a service template version, you create a new major version of it.

You can use the console or the AWS CLI to register and publish a service template.

AWS Management Console

Use the console to register and publish a new service template.

1. In the AWS Proton console, choose **Service templates**.
2. Choose **Create service template**.
3. In the **Create service template** page, in the **Template bundle source** section, choose one of the available template options.
   - **Use your own template bundle**.
   - **Sync templates from Git**.
4. **Provide a path to a template bundle.**
   a. If you chose **Sync templates from Git**, in the **Source code repository** section:
      i. Select the repository for your template sync configuration.
      ii. Enter the name of the repository branch to sync from.
      iii. (Optional) Enter name of a directory to limit the search for your template bundle.
   b. Otherwise, in the **S3 bundle location** section, provide a path to your template bundle.
5. In the **Template details** section.
   a. Enter a **Template name**.
   b. (Optional) Enter a **Template display name**.
   c. (Optional) Enter a **Template description** for the service template.
6. In the **Compatible environment templates** section, choose from a list of compatible environment templates.
7. (Optional) In the **Encryption settings** section, choose **Customize encryption settings (advanced)** to provide your own encryption key.
8. (Optional) In the **Pipeline** section:

   If you aren't including a service pipeline definition in your service template, uncheck the **Pipeline - optional** check box at the bottom of the page. You *can't* change this after the service template is created. For more information, see Template bundles.

9. (Optional) In the **Supported component sources** section, for **Component sources**, choose **Directly defined** to enable attachment of directly defined components to your service instances.
10. (Optional) In the **Tags** section, choose **Add new tag** and enter a key and value to create a customer managed tag.
11. Choose **Create a service template**.

    You're now on a new page that displays the status and details for your new service template. These details include a list of AWS and customer managed tags. AWS Proton automatically generates AWS managed tags for you when you create AWS Proton resources. For more information, see AWS Proton resources and tagging.
12. The status of a new service template status starts in the **Draft** state. You and others with `proton:CreateService` permissions can view and access it. Follow the next step to make the template available to others.
13. In the **Template versions** section, choose the radio button to the left of the minor version of the template you just created (1.0). As an alternative, you can choose **Publish** in the info alert and skip the next step.
14. In the **Template versions** section, choose **Publish**.
15. The template status changes to **Published**. Because it's the latest version of the template, it's the **Recommended** version.
16. In the navigation pane, select **Service templates** to view a list of your service templates and details.

Use the console to register new major and minor versions of a service template.

For more information, see Versioned templates.

1. In the AWS Proton console, choose **Service Templates**.
2. In the list of service templates, choose the name of the service template that you want to create a major or minor version for.
3. In the service template detail view, choose **Create new version** in the **Template versions** section.
4. In the **Create a new service template version** page, in the **Bundle source** section, select **Use your own template bundle**.
5. In the **S3 bundle location** section, choose the path to your template bundle.

6. In the **Template details** section.
   a. (Optional) Enter a **Template display name**.
   b. (Optional) Enter a **Template description** for the service template.
7. In the **Template details** section, choose one of the following options.
   - To create a minor version, keep the check box **Check to create a new major version** empty.
   - To create a major version, check the check box **Check to create a new major version**.
8. Continue through the console steps to create the new minor or major version and choose **Create new version**.

AWS CLI

To create service template that deploys a service without a service pipeline, add the parameter and value `--pipeline-provisioning "CUSTOMER_MANAGED"` to the `create-service-template` command. Configure your template bundles as described in Template bundles creation and Schema requirements for service template bundles.

> **Note**
> You can't modify `pipelineProvisioning` after the service template is created.

1. **Use the CLI to register and publish a new service template, with or without a service pipeline, as shown in the following steps.**
   a. **Create a service template with a service pipeline using the CLI.**

      Specify the name, display name (optional), and description (optional).

      Run the following command:

```
$ aws proton create-service-template \
    --name "fargate-service" \
    --display-name "Fargate" \
    --description "Fargate-based Service"
```

      Response:

```
{
    "serviceTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/
fargate-service",
        "createdAt": "2020-11-11T23:02:55.551000+00:00",
        "description": "Fargate-based Service",
        "displayName": "Fargate",
        "lastModifiedAt": "2020-11-11T23:02:55.551000+00:00",
        "name": "fargate-service"
    }
}
```

b. **Create a service template without a service pipeline.**

Add `--pipeline-provisioning`.

Run the folllowing command:

```
$ aws proton create-service-template \
    --name "fargate-service" \
    --display-name "Fargate" \
    --description "Fargate-based Service" \
    --pipeline-provisioning "CUSTOMER_MANAGED"
```

Response:

```
{
    "serviceTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/
fargate-service",
        "createdAt": "2020-11-11T23:02:55.551000+00:00",
        "description": "Fargate-based Service",
        "displayName": "Fargate",
        "lastModifiedAt": "2020-11-11T23:02:55.551000+00:00",
        "name": "fargate-service",
        "pipelineProvisioning": "CUSTOMER_MANAGED"
    }
}
```

2. **Create a minor version 0 of major version 1 of the service template.**

   Include the template name, compatible environment templates, major version, and the S3 bucket name and key for the bucket that contains your service template bundle.

   Run the following command:

```
$ aws proton create-service-template-version \
    --template-name "fargate-service" \
    --description "Version 1" \
    --source s3="{bucket=your_s3_bucket, key=your_s3_key}" \
    --compatible-environment-templates '[{"templateName":"simple-
env","majorVersion":"1"}]'
```

   Response:

```
{
    "serviceTemplateMinorVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
                "templateName": "simple-env"
            }
        ],
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:57.912000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "REGISTRATION_IN_PROGRESS",
        "templateName": "fargate-service"
    }
}
```

3. Use the *get* command to check the registrations status.

   Run the folllowing command:

```
$ aws proton get-service-template-version \
```

```
    --template-name "fargate-service" \
    --major-version "1" \
    --minor-version "0"
```

Response:

```
{
    "serviceTemplateMinorVersion": {
        "arn": "arn:aws:proton:us-east-1:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
                "templateName": "simple-env"
            }
        ],
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:57.912000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 pipeline_input_type: \"MyPipelineInputType\"\n  service_input_type:
 \"MyServiceInstanceInputType\"\n\n  types:\n    MyPipelineInputType:\n
    type: object\n      description: \"Pipeline input properties\"\n
 required:\n        - my_sample_pipeline_required_input\n      properties:\n
        my_sample_pipeline_optional_input:\n          type: string\n
   description: \"This is a sample input\"\n          default: \"hello world
\"\n        my_sample_pipeline_required_input:\n          type: string\n
     description: \"Another sample input\"\n\n    MyServiceInstanceInputType:
\n      type: object\n      description: \"Service instance input properties
\"\n      required:\n        - my_sample_service_instance_required_input\n
    properties:\n        my_sample_service_instance_optional_input:\n
   type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_sample_service_instance_required_input:\n
         type: string\n          description: \"Another sample input\"",
        "status": "DRAFT",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

4. Publish the service template by using the update command to change the status to `"PUBLISHED"`.

   Run the following command:

```
$ aws proton update-service-template-version \
    --template-name "fargate-service" \
    --description "Version 1" \
    --major-version "1" \
    --minor-version "0" \
    --status "PUBLISHED"
```

   Response:

```
{
    "serviceTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
                "templateName": "simple-env"
            }
        ],
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:57.912000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 pipeline_input_type: \"MyPipelineInputType\"\n  service_input_type:
 \"MyServiceInstanceInputType\"\n\n  types:\n    MyPipelineInputType:\n
    type: object\n      description: \"Pipeline input properties\"\n
 required:\n        - my_sample_pipeline_required_input\n      properties:\n
        my_sample_pipeline_optional_input:\n          type: string\n
 description: \"This is a sample input\"\n          default: \"hello pipeline
\"\n        my_sample_pipeline_required_input:\n          type: string\n
     description: \"Another sample input\"\n\n    MyServiceInstanceInputType:
\n      type: object\n      description: \"Service instance input properties
\"\n      required:\n        - my_sample_service_instance_required_input\n
    properties:\n        my_sample_service_instance_optional_input:\n
```

```
    type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_sample_service_instance_required_input:\n
         type: string\n          description: \"Another sample input\"\n",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

5. Check that AWS Proton has published version 1.0 by using the get command to retrieve service template detail data.

   Run the following command:

```
$ aws proton get-service-template-version \
    --template-name "fargate-service" \
    --major-version "1" \
    --minor-version "0"
```

   Response:

```
{
    "serviceTemplateMinorVersion": {
        "arn": "arn:aws:proton:us-east-1:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
                "templateName": "simple-env"
            }
        ],
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:03:04.767000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 pipeline_input_type: \"MyPipelineInputType\"\n  service_input_type:
 \"MyServiceInstanceInputType\"\n\n  types:\n    MyPipelineInputType:\n
    type: object\n      description: \"Pipeline input properties\"\n
 required:\n        - my_sample_pipeline_required_input\n      properties:\n
        my_sample_pipeline_optional_input:\n          type: string\n
   description: \"This is a sample input\"\n          default: \"hello world
```

```
\"\n        my_sample_pipeline_required_input:\n          type: string\n
     description: \"Another sample input\"\n\n    MyServiceInstanceInputType:
\n      type: object\n      description: \"Service instance input properties
\"\n      required:\n        - my_sample_service_instance_required_input\n
    properties:\n        my_sample_service_instance_optional_input:\n
   type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_sample_service_instance_required_input:\n
         type: string\n          description: \"Another sample input\"",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

View template data

You can view lists of templates with details and view individual templates with detail data by using the AWS Proton console and AWS CLI.

Customer managed environment template data includes the `provisioned` parameter with the value `CUSTOMER_MANAGED`.

If a service template *doesn't* include a service pipeline, the service template data includes the `pipelineProvisioning` parameter with the value `CUSTOMER_MANAGED`.

For more information, see Register and publish templates.

You can use the console or the AWS CLI to list and view template data.

AWS Management Console

Use the console to list and view templates.

1. To view a list of templates, choose **(Environment or Service) templates**.
2. To view detail data choose the name of a template.

   View the detail data of the template, a list of the major and minor versions of the template, a list of the AWS Proton resources that were deployed using template versions and template tags.

   The recommended major version and minor version is labeled as **Recommended**.

AWS CLI

Use the AWS CLI to list and view templates.

Run the following command:

```
$ aws proton get-environment-template-version \
    --template-name "simple-env" \
    --major-version "1" \
    --minor-version "0"
```

Response:

```
{
    "environmentTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/simple-
env:1.0",
        "createdAt": "2020-11-10T18:35:08.293000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-10T18:35:11.162000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 environment_input_type: \"MyEnvironmentInputType\"\n  types:\n
 MyEnvironmentInputType:\n      type: object\n      description: \"Input properties
 for my environment\"\n      properties:\n        my_sample_input:\n
  type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_other_sample_input:\n          type: string
\n          description: \"Another sample input\"\n      required:\n        -
 my_other_sample_input\n",
        "status": "DRAFT",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

Run the following command:

```
$ aws proton list-environment-templates
```

Response:

```
{
    "templates": [
        {
            "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env-3",
            "createdAt": "2020-11-10T18:35:05.763000+00:00",
            "description": "VPC with Public Access",
            "displayName": "VPC",
            "lastModifiedAt": "2020-11-10T18:35:05.763000+00:00",
            "name": "simple-env-3",
            "recommendedVersion": "1.0"
        },
        {
            "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env-1",
            "createdAt": "2020-11-10T00:14:06.881000+00:00",
            "description": "Some SSM Parameters",
            "displayName": "simple-env-1",
            "lastModifiedAt": "2020-11-10T00:14:06.881000+00:00",
            "name": "simple-env-1",
            "recommendedVersion": "1.0"
        }
    ]
}
```

View a minor version of a service template.

Run the following command:

```
$ aws proton get-service-template-version \
    --template-name "fargate-service" \
    --major-version "1" \
    --minor-version "0"
```

Response:

```
{
    "serviceTemplateMinorVersion": {
        "arn": "arn:aws:proton:us-east-1:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
```

```
                "templateName": "simple-env"
            }
        ],
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:57.912000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 pipeline_input_type: \"MyPipelineInputType\"\n  service_input_type:
 \"MyServiceInstanceInputType\"\n\n  types:\n    MyPipelineInputType:\n
    type: object\n      description: \"Pipeline input properties\"\n
 required:\n        - my_sample_pipeline_required_input\n      properties:\n
        my_sample_pipeline_optional_input:\n          type: string\n
 description: \"This is a sample input\"\n          default: \"hello world\"\n
  my_sample_pipeline_required_input:\n          type: string\n          description:
 \"Another sample input\"\n\n    MyServiceInstanceInputType:\n      type: object
\n      description: \"Service instance input properties\"\n      required:\n
       - my_sample_service_instance_required_input\n      properties:\n
 my_sample_service_instance_optional_input:\n          type: string\n
 description: \"This is a sample input\"\n          default: \"hello world\"\n
     my_sample_service_instance_required_input:\n          type: string\n
 description: \"Another sample input\"",
        "status": "DRAFT",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

View a service template without a service pipeline as shown in the next example command and response.

Run the following command:

```
$ aws proton get-service-template \
    --name "simple-svc-template-cli"
```

Response:

```
{
    "serviceTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/simple-svc-
template-cli",
```

```
        "createdAt": "2021-02-18T15:38:57.949000+00:00",
        "displayName": "simple-svc-template-cli",
        "lastModifiedAt": "2021-02-18T15:38:57.949000+00:00",
        "status": "DRAFT",
        "name": "simple-svc-template-cli",
        "pipelineProvisioning": "CUSTOMER_MANAGED"
    }
}
```

Update a template

You can update a template as described in the following list.

- Edit the `description` or `display name` of a template when you use either the console or AWS CLI. You *can't* edit the `name` of a template.
- Update the status of a template minor version when you use either the console or AWS CLI. You can only change the status from `DRAFT` to `PUBLISHED`.
- Edit the display name and description of a minor or major version of a template when you use the AWS CLI.

AWS Management Console

Edit a template description and display name using the console as described in the following steps.

In the list of templates.

1. In the AWS Proton console, choose **(Environment or Service) Templates**.
2. In the list of templates, choose the radio button to the left of the template that you want to update the description or display name for.
3. Choose **Actions** and then **Edit**.
4. In the **Edit (environment or service) template** page, in the **Template details** section, enter your edits in the form and choose **Save changes**.

Change the status of a minor version of a template using the console to publish a template as described in the following. You can only change the status from `DRAFT` to `PUBLISHED`.

In the (environment or service) template detail page.

1. In the AWS Proton console, choose **(Environment or Service) templates**.
2. In the list of templates, choose the name of the template that you want to update the status of a minor version from **Draft** to **Published**.
3. In the (environment or service) template detail page, in the **Template versions** section, select the radio button to the left of the minor version that you want to publish.
4. Choose **Publish** in the **Template versions** section. The status changes from **Draft** to **Published**.

AWS CLI

The following example command and response shows how you can edit the description of an environment template.

Run the following command.

```
$ aws proton update-environment-template \
    --name "simple-env" \
    --description "A single VPC with public access"
```

Response:

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/simple-
env",
        "createdAt": "2020-11-28T22:02:10.651000+00:00",
        "description": "A single VPC with public access",
        "displayName": "simple-env",
        "lastModifiedAt": "2020-11-29T16:11:18.956000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 environment_input_type: \"MyEnvironmentInputType\"\n  types:\n
 MyEnvironmentInputType:\n      type: object\n      description: \"Input properties
 for my environment\"\n      properties:\n        my_sample_input:\n
  type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_other_sample_input:\n          type: string
```

```
\n          description: \"Another sample input\"\n      required:\n        -
 my_other_sample_input\n",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

You can also use the AWS CLI to update service templates. See Register and publish service templates, step 5, for an example of updating the status of a minor version of a service template.

Delete templates

Templates can be deleted using the console and AWS CLI.

You can delete a minor version of an environment template if there are no environments deployed to that version.

You can delete a minor version of a service template if there are no service instances or pipelines deployed to that version. Your pipeline can be deployed to a different template version than your service instance. For example, if your service instance is updated to version 1.1 from 1.0 and your pipeline is still deployed to version 1.0, you can't delete service template 1.0.

AWS Management Console

You can use the console to delete the entire template or individual minor and major versions of a template.

Use the console to delete templates as follows.

> **Note**
>
> **When using the console to delete templates.**
>
> - When you delete the entire template, you also delete the major and minor versions of the template.

In the list of (environment or service) templates.

1. In the AWS Proton console, choose **(Environment or Service) Templates**.
2. In the list of templates, select the radio button to the left of the template you want to delete.

   You can only delete an entire template if there are no AWS Proton resources deployed to its versions.
3. Choose **Actions** and then **Delete** to delete the entire template.
4. A modal prompts you to confirm the delete action.
5. Follow the instructions and choose **Yes, delete**.

In the (environment or service) template detail page.

1. In the AWS Proton console, choose **(Environment or Service) Templates**.
2. In the list of templates, choose the name of the template that you want to entirely delete or delete individual major or minor versions of it.
3. **To delete the entire template.**

   You can only delete an entire template if there are no AWS Proton resources deployed to its versions.

   a. Choose **Delete**, top right corner of page.

   b. A modal prompts you to confirm the delete action.

   c. Follow the instructions and choose **Yes, delete**.
4. **To delete major or minor versions of a template.**

   You can only delete a minor version of a template if there are no AWS Proton resources deployed to that version.

   a. In the **Template versions** section, select the radio button to the left of the version that you want to delete.

   b. Choose **Delete** in the **Template versions** section.

   c. A modal prompts you to confirm the delete action.

   d. Follow the instructions and choose **Yes, delete**.

AWS CLI

AWS CLI template delete operations *don't* include the deletion of other versions of a template. When using the AWS CLI, delete templates with the following conditions.

- Delete an entire template if no minor or major versions of the template exist.
- Delete a major version when you delete the last remaining minor version.
- Delete a minor version of a template if there are no AWS Proton resources deployed to that version.
- Delete the recommended minor version of a template if no other minor versions of the template exist and there are no AWS Proton resources deployed to that version.

The following example commands and responses show how to use the AWS CLI to delete templates.

Run the following command:

```
$ aws proton delete-environment-template-version \
    --template-name "simple-env" \
    --major-version "1" \
    --minor-version "0"
```

Response:

```
{
    "environmentTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/simple-
env:1.0",
        "createdAt": "2020-11-11T23:02:47.763000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:54.610000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

Run the following command:

```
$ aws proton delete-environment-template \
    --name "simple-env"
```

Response:

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/simple-
env",
        "createdAt": "2020-11-11T23:02:45.336000+00:00",
        "description": "VPC with Public Access",
        "displayName": "VPC",
        "lastModifiedAt": "2020-11-12T00:23:22.339000+00:00",
        "name": "simple-env",
        "recommendedVersion": "1.0"
    }
}
```

Run the following command:

```
$ aws proton delete-service-template-version \
    --template-name "fargate-service" \
    --major-version "1" \
    --minor-version "0"
```

Response:

```
{
    "serviceTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [{"majorVersion": "1", "templateName":
 "simple-env"}],
        "createdAt": "2020-11-28T22:07:05.798000+00:00",
        "lastModifiedAt": "2020-11-28T22:19:05.368000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
```

```
}
```

Template sync configurations

Learn how to configure a template to let AWS Proton sync from template bundles located in registered git repositories that you define. When a commit is pushed to your repository, AWS Proton checks for changes to your repository template bundles. If it detects a template bundle change, a new minor or major version of its template is created, if the version doesn't already exist. AWS Proton currently supports GitHub, GitHub Enterprise, and BitBucket.

Pushing a commit to a synced template bundle

When you push a commit to a branch that's being tracked by one of your templates, AWS Proton clones your repository and determines what templates it needs to sync. It scans the files in your directory to find directories matching the convention of `{template-name}/{major-version}/`.

After AWS Proton determines which templates and major versions are associated with your repository and branch, it starts trying to sync all of those templates in parallel.

During each sync to a particular template, AWS Proton first checks to see if the contents of the template directory changed since the last successful sync. If the contents didn't change, AWS Proton skips registering a duplicate bundle. This ensures that a new template minor version is created if the content of the template bundle changes. If the contents of the template bundle changed, the bundle is registered with AWS Proton.

After the template bundle is registered, AWS Proton monitors the registration status until the registration is complete.

Only one sync can occur to a particular template minor and major version at a single given time. Any commits that might have been pushed while a sync was in progress are batched. The batched commits are synced after the previous sync attempt is complete.

Syncing service templates

AWS Proton can sync both environment and service templates from your git repository. To sync your service templates you add an additional file named `.template-registration.yaml` to each major version directory in your template bundle. This file contains additional details that AWS Proton needs when it creates a service template version for you following a commit: *compatible environments* and *supported component sources*.

The file's full path is `service-template-name/major-version/.template-registration.yaml`. For more information, see the section called "Syncing service templates".

Template sync configuration considerations

Review the following considerations for using template sync configurations.

- Repositories must be no larger than 250 MB.
- To configure template sync, first link the repository to AWS Proton. For more information, see the section called "Create a repository link".
- When a new template version is created from a synced template, it's in the `DRAFT` state.
- A new minor version of a template is created if one of the following is true:
  - The template bundle contents are different from those of the last synced template minor version.
  - The last previously synced template minor version was deleted.
- Syncing can't be paused.
- Both new minor or major versions are automatically synced.
- New top-level templates can't be created by template sync configurations.
- You can't sync to one template from multiple repositories with a template sync configuration.
- You can't use tags instead of branches.
- When you create a service template, you specify compatible environment templates.
- You can create an environment template and add it as a compatible environment for your service template in the same commit.
- Syncs to a single template major version are run one at a time. During a sync, if any new commits are detected, they're batched and applied at the end of active sync. Syncs to different template major versions happen in parallel.
- If you change the branch your templates are syncing from, any ongoing syncs from the old branch first complete. Then syncing begins from the new branch.
- If you change the repository your templates sync from, any ongoing syncs from the old repository might fail or run to completion. It depends on which stage of the sync they're in.

For more information, see the *The AWS Proton Service API Reference*.

Topics

- Create a template sync configuration
- View template sync configuration details
- Edit a template sync configuration
- Delete a template sync configuration

Create a template sync configuration

Learn how to create a template sync configuration with AWS Proton.

Create a template sync configuration prerequisites:

- You've linked a repository with AWS Proton.
- A template bundle is located in your repository.

The repository link consists of the following:

- An CodeConnections connection that gives AWS Proton permission to access your repository and subscribe to its notifications.
- A service linked role. When you link your repository, the service linked role is created for you.

Before you create your first template sync configuration, push a template bundle to your repository as shown in the following directory layout.

```
/templates/                                                 # subdirectory (optional)
/templates/my-env-template/                                 # template name
/templates/my-env-template/v1/                              # template version
/templates/my-env-template/v1/infrastructure/               # template bundle
/templates/my-env-template/v1/schema/
```

After you create your first template sync configuration, new template versions are automatically created when you push a commit that adds an updated template bundle under a new version (for example, under `/my-env-template/v2/`).

```
/templates/                                                 # subdirectory (optional)
/templates/my-env-template/                                 # template name
/templates/my-env-template/v1/                              # template version
/templates/my-env-template/v1/infrastructure/               # template bundle
```

```
/templates/my-env-template/v1/schema/
/templates/my-env-template/v2/
/templates/my-env-template/v2/infrastructure/
/templates/my-env-template/v2/schema/
```

You can include new template bundle versions for one or more sync configured templates in a single commit. AWS Proton creates a new template version for each new template bundle version that was included in the commit.

After you created the template sync configuration, you can still manually create new versions of the template in the console or with the AWS CLI by uploading template bundles from an S3 bucket. Template syncing only works in one direction: from your repository to AWS Proton. Manually created template versions *aren't* synced.

After you set up a template sync configuration, AWS Proton listens for changes to your repository. Whenever a change is pushed, it looks for any directory that has the same name as your template. It then looks inside that directory for any directories that look like major versions. AWS Proton registers the template bundle to the corresponding template major version. The new versions are always in the `DRAFT` state. You can publish the new versions with the console or AWS CLI.

For example, suppose you have a template that's called `my-env-template` configured to sync from `my-repo/templates` on branch `main` with the following layout.

```
/code
/code/service.go
README.md
/templates/
/templates/my-env-template/
/templates/my-env-template/v1/
/templates/my-env-template/v1/infrastructure/
/templates/my-env-template/v1/schema/
/templates/my-env-template/v2/
/templates/my-env-template/v2/infrastructure/
/templates/my-env-template/v2/schema/
```

AWS Proton syncs the contents of `/templates/my-env-template/v1/` to `my-env-template:1` and the contents of `/templates/my-env-template/v2/` to `my-env-template:2`. If they don't already exist, it creates these major versions.

AWS Proton found the first directory that matched the template name. You can limit the directories AWS Proton searches by specifying a `subdirectoryPath` when you create or edit

a template sync configuration. For example, you can specify `/production-templates/` for `subdirectoryPath`.

You can create a template sync configuration using the console or CLI.

AWS Management Console

Create a template and template sync configuration using the console.

1. In the AWS Proton console, choose **Environment templates**.
2. Choose **Create environment template**.
3. In the **Create environment template** page, in the **Template options** section, choose **Create a template for provisioning new environments**.
4. In the **Template bundle source** section, choose **Sync templates from Git**.
5. In the **Source code repository** section:
   a. For **Repository**, select the linked repository that contains your template bundle.
   b. For **Branch**, select a repository branch to sync from.
   c. (Optional) For **Template bundle directory**, enter the name of a directory to scope down the search for your template bundle.
6. In the **Template details** section.
   a. Enter a **Template name**.
   b. (Optional) Enter a **Template display name**.
   c. (Optional) Enter a **Template description** for the environment template.
7. (Optional) Check the checkbox for **Customize encryption settings (advanced)** in the **Encryption settings** section to provide your own encryption key.
8. (Optional) In the **Tags** section, choose **Add new tag** and enter a key and value to create a customer managed tag.
9. Choose **Create Environment template**.

   You're now on a new page that displays the status and details for your new environment template. These details include a list of AWS managed and customer managed tags. AWS Proton automatically generates AWS managed tags for you when you create AWS Proton resources. For more information, see AWS Proton resources and tagging.

10. In the template detail page, choose the **Sync** tab to view template sync configuration detail data.
11. Choose the **Template versions** tab to view template versions with status details.
12. The status of a new environment template status starts in the **Draft** state. You and others with `proton:CreateEnvironment` permissions can view and access it. Follow the next step to make the template available to others.
13. In the **Template versions** section, choose the radio button to the left of the minor version of the template that you just created (1.0). As an alternative, you can choose **Publish** in the info alert and skip the next step.
14. In the **Template versions** section, choose **Publish**.
15. The template status changes to **Published**. It's the latest and **Recommended** version of the template.
16. In the navigation pane, select **Environment templates** to view a list of your environment templates and details.

The procedure for creating a service template and template sync configuration is similar.

AWS CLI

Create a template and template sync configuration using the AWS CLI.

1. **Create a template. In this example, an environment template is created.**

   Run the following command.

```
$ aws proton create-environment-template \
    --name "env-template"
```

   The response is as follows.

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:us-east-1:123456789012:environment-template/env-template",
        "createdAt": "2021-11-07T23:32:43.045000+00:00",
        "displayName": "env-template",
        "lastModifiedAt": "2021-11-07T23:32:43.045000+00:00",
        "name": "env-template",
```

```
        "status": "DRAFT",
        "templateName": "env-template"
    }
}
```

2. **Create your template sync configuration with AWS CLI by providing the following:**

   - The template that you want to sync to. After you have created the template sync configuration, you can still create new versions from it manually in the console or with the AWS CLI.
   - The template name.
   - The template type.
   - The linked repository that you want to sync from.
   - The linked repository provider.
   - The branch where the template bundle is located.
   - (Optional) The path to the directory containing your template bundle. By default, AWS Proton looks for the first directory that matches your template name.

   Run the following command.

```
$ aws proton create-template-sync-config \
    --template-name "env-template" \
    --template-type "ENVIRONMENT" \
    --repository-name "myrepos/templates" \
    --repository-provider "GITHUB" \
    --branch "main" \
    --subdirectory "env-template/"
```

   The response is as follows.

```
{
    "templateSyncConfigDetails": {
        "branch": "main",
        "repositoryName": "myrepos/templates",
        "repositoryProvider": "GITHUB",
        "subdirectory": "templates",
        "templateName": "env-template",
        "templateType": "ENVIRONMENT"
    }
```

```
}
```

3. **To publish your template version, see Register and publish templates.**

Syncing service templates

The preceding examples show how you can sync environment templates. Service templates are similar. To sync service templates you add an additional file named `.template-registration.yaml` to each major version directory in your template bundle. This file contains additional details that AWS Proton needs when it creates a service template version for you following a commit. When you explicitly create a service template version using the AWS Proton console or API, you provide these details as inputs, and this file replaces these inputs for template sync.

```
./templates/                                                # subdirectory (optional)
 /templates/my-svc-template/                                # service template name
 /templates/my-svc-template/v1/                             # service template version
 /templates/my-svc-template/v1/.template-registration.yaml  # service template version
 properties
 /templates/my-svc-template/v1/instance_infrastructure/     # template bundle
 /templates/my-svc-template/v1/schema/
```

The `.template-registration.yaml` file contains the following details:

- **Compatible environments** [required] – Environments based on these environment templates and major versions are compatible with services based on this service template version.
- **Supported component sources** [optional] – Components using these sources are compatible with services based on this service template version. If not specified, components can't be attached to these services. For more information about components, see *Components*.

The file's YAML syntax is as follows:

```
compatible_environments:
  - env-templ-name:major-version
  - ...
supported_component_sources:
  - DIRECTLY_DEFINED
```

Specify one or more environment template / major version combinations. Specifying `supported_component_sources` is optional, and the only supported value is `DIRECTLY_DEFINED`.

Example .template-registration.yaml

In this example, the service template version is compatible with major versions 1 and 2 of the `my-env-template` environment template. It's also compatible with the major versions 1 and 3 of the `another-env-template` environment template. The file doesn't specify `supported_component_sources`, so components can't be attached to services based on this service template version.

```
compatible_environments:
  - my-env-template:1
  - my-env-template:2
  - another-env-template:1
  - another-env-template:3
```

> **Note**
>
> Previously, AWS Proton defined a different file, `.compatible-envs`, for specifying compatible environments. AWS Proton still supports that file and its format for backward compatibility. We don't recommend using it anymore, because it isn't extensible and can't support newer features like components.

View template sync configuration details

View template sync configuration detail data using the console or CLI.

AWS Management Console

Use the console to view template sync configuration details.

1. In the navigation pane, choose **(Environment or Service) templates**.
2. To view detail data, choose the name of a template that you created a template sync configuration for.
3. In the detail page for the template, select the **Sync** tab to view the template sync configuration detail data.

AWS CLI

Use the AWS CLI to view a synced template.

Run the following command.

```
$ aws proton get-template-sync-config \
    --template-name "svc-template" \
    --template-type "SERVICE"
```

The response is as follows.

```
{
    "templateSyncConfigDetails": {
        "branch": "main",
        "repositoryProvider": "GITHUB",
        "repositoryName": "myrepos/myrepo",
        "subdirectory": "svc-template",
        "templateName": "svc-template",
        "templateType": "SERVICE"
    }
}
```

Use the AWS CLI to get template sync status.

For `template-version`, enter the template major version.

Run the following command.

```
$ aws proton get-template-sync-status \
    --template-name "env-template" \
    --template-type "ENVIRONMENT" \
    --template-version "1"
```

Edit a template sync configuration

You can edit any of the template sync configuration parameters except `template-name` and `template-type`.

Learn to edit a template sync configuration using the console or CLI.

AWS Management Console

Edit a template sync configuration branch using the console.

In the list of templates.

1. In the AWS Proton console, choose **(Environment or Service) Templates**.
2. In the list of templates, choose the name of the template with the template sync configuration that you want to edit.
3. In the template detail page, choose the **Template sync** tab.
4. In the **Template sync details** section, choose **Edit**.
5. In the **Edit** page, in the **Source code repository** section, for **Branch**, select a branch, and then choose **Save configuration**.

AWS CLI

The following example command and response shows how you can edit a template sync configuration branch using the CLI.

Run the following command.

```
$ aws proton update-template-sync-config \
    --template-name "env-template" \
    --template-type "ENVIRONMENT" \
    --repository-provider "GITHUB" \
    --repository-name "myrepos/templates" \
    --branch "fargate" \
    --subdirectory "env-template"
```

The response is as follows.

```
{
    "templateSyncConfigDetails": {
        "branch": "fargate",
        "repositoryProvider": "GITHUB",
        "repositoryName": "myrepos/myrepo",
        "subdirectory": "templates",
        "templateName": "env-template",
        "templateType": "ENVIRONMENT"
    }
```

```
}
```

You can similarly use the AWS CLI to update synced service templates.

Delete a template sync configuration

Delete a template sync configuration using the console or CLI.

AWS Management Console

Delete a template sync configuration using the console.

1. In the template details page, choose the **Sync** tab.
2. In the **Sync details** section, choose **Disconnect**.

AWS CLI

The following example commands and responses show how to use the AWS CLI to delete synced template configurations.

Run the following command.

```
$ aws proton delete-template-sync-config \
    --template-name "env-template" \
    --template-type "ENVIRONMENT"
```

The response is as follows.

```
{
    "templateSyncConfig": {
        "templateName": "env-template",
        "templateType": "ENVIRONMENT"
    }
}
```

Service sync configurations

With service sync, you can configure and deploy your AWS Proton services using Git. You can use service sync to manage initial deployments and updates to your AWS Proton service with a

configuration defined in a Git repository. Through Git, you can use features like version tracking and pull requests to configure, manage, and deploy your services. Service sync combines AWS Proton and Git to help you provision standardized infrastructure that is defined and managed through AWS Proton templates. It manages service definitions in your Git repository and reduces tool switching. Compared to using Git alone, the standardization of templates and deployment in AWS Proton helps you spend less time managing your infrastructure. AWS Proton also provides higher transparency and auditability for both developers and platform teams.

AWS Proton OPS file

The `proton-ops` file defines where AWS Proton finds the spec file that's used to update your service instance. It also defines what order to update service instances in and when to promote changes from one instance to another.

The `proton-ops` file supports syncing a service instance using the spec file, or multiple spec files, found in your linked repository. You can do this by defining a sync block in the `proton-ops` file, like in the following example.

Example ./configuration/proton-ops.yaml:

```
sync:
  services:
      frontend-svc:
          alpha:
              branch: dev
              spec: ./frontend-svc/test/frontend-spec.yaml
          beta:
              branch: dev
              spec: ./frontend-svc/test/frontend-spec.yaml
          gamma:
              branch: pre-prod
              spec: ./frontend-svc/pre-prod/frontend-spec.yaml
          prod-one:
              branch: prod
              spec: ./frontend-svc/prod/frontend-spec-second.yaml
          prod-two:
              branch: prod
              spec: ./frontend-svc/prod/frontend-spec-second.yaml
          prod-three:
              branch: prod
              spec: ./frontend-svc/prod/frontend-spec-second.yaml
```

In the preceding example, `frontend-svc` is the service name, and `alpha`, `beta`, `gamma`, `prod-one`, `prod-two`, and `prod-three` are the instances.

The `spec` file can be all of the instances or a subset of the instances that are defined within the `proton-ops` file. However, at minimum, it must have the instance defined within the branch and the spec it's syncing from. If instances aren't defined in the `proton-ops` file, with the specific branch and `spec` file location, service sync won't create or update those instances.

The following examples show what the `spec` files look like. Remember, the `proton-ops` file is synced from these `spec` files.

Example `./frontend-svc/test/frontend-spec.yaml`:

```
proton: "ServiceSpec"
instances:
- name: "alpha"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
- name: "beta"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
```

Example `./frontend-svc/pre-prod/frontend-spec.yaml`:

```
proton: "ServiceSpec"
instances:
- name: "gamma"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
```

Example `./frontend-svc/prod/frontend-spec-second.yaml`:

```
proton: "ServiceSpec"
instances:
- name: "prod-one"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
- name: "prod-two"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
- name: "prod-three"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
```

If an instance doesn't sync, and there's a continuing issue when trying to sync it, calling the `GetServiceInstanceSyncStatus` API may help in resolving the issue.

> **Note**
>
> Customers using service sync are still restricted by AWS Proton limits.

Blockers

By syncing your service using AWS Proton service sync, you can update your service spec and create and update service instances from your Git repository. However, there may be moments where you need to update a service or instance manually through the AWS Management Console or AWS CLI.

AWS Proton helps avoid overwriting any manual changes you make through the AWS Management Console or AWS CLI, such as updating a service instance or deleting a service instance. To achieve

this, AWS Proton automatically creates a service sync blocker by disabling service sync when it detects a manual change.

To get all the blockers associated with a service, you must do the following in order for each `serviceInstance` associated to the service:

- Call the `getServiceSyncBlockerSummary` API with only the `serviceName`.
- Call the `getServiceSyncBlockerSummary` API with the `serviceName` and `serviceInstanceName`.

This returns a list of the most recent blockers and the status associated with them. If any blockers are marked **ACTIVE**, you must resolve them by calling the `UpdateServiceSyncBlocker` API with the `blockerId` and `resolvedReason` for each one.

If you manually update or create a service instance, AWS Proton creates a service sync blocker on the service instance. AWS Proton continues to sync all other service instances, but disables the syncing of this service instance until the blocker is resolved. If you delete a service instance from a service, AWS Proton creates a service sync blocker on the service. This prevents AWS Proton from syncing any of the service instances until the blocker has been resolved.

After you have all the active blockers, you must resolve them by calling the `UpdateServiceSyncBlocker` API with the `blockerId` and `resolvedReason` for each of the active blockers.

Using the AWS Management Console, you can determine if a service sync is disabled by navigating to AWS Proton and choosing the **Service Sync** tab. If the service or service instances are blocked, an **Enable** button appears. To enable service sync, choose **Enable**.

Topics

- Create a service sync configuration
- View configuration details for a service sync
- Edit a service sync configuration
- Delete a service sync configuration

Create a service sync configuration

You can create a service sync configuration using the console or AWS CLI.

AWS Management Console

1. On the **Choose a service template** page, select a template and choose **Configure**.
2. On the **Configure service** page, in the **Service details** section, enter a new **Service name**.
3. (Optional) Enter a description for the service.
4. In the **Application source code repository** section, choose **Choose a linked Git repository** to select a repository you've already linked with AWS Proton. If you don't already have a linked repository, choose **Link another Git repository** and follow the instructions in Create a link to your repository.
5. For **Repository**, choose the name of your source code repository from the list.
6. For **Branch**, choose the name of the repository branch for your source code from the list.
7. (Optional) In the **Tags** section, choose **Add new tag** and enter a key and value to create a customer managed tag.
8. Choose **Next**.
9. On the **Configure service instances** page, in the **Service definition source** section, select **Sync your service from Git**.
10. In the **Service definition files** section, if you want AWS Proton to create your `proton-ops` file, select **I want AWS Proton to create the files**. With this option, AWS Proton creates the `spec` and `proton-ops` file in the locations you specify. Select **I am providing my own files** to create your own OPS file.
11. In the **Service definition repository** section, choose **Choose a linked Git repository** to select a repository you've already linked with AWS Proton.
12. For **Repository name**, choose the name of your source code repository from the list.
13. For **`proton-ops` file branch**, choose the name of your branch from the list where AWS Proton will put your OPS and spec file.
14. In the **Service instances** section, each field is automatically filled based on the values in the `proton-ops` file.
15. Choose **Next** and review your inputs.
16. Choose **Create**.

AWS CLI

Create a service sync configuration using the AWS CLI

- Run the following command.

```
$ aws proton create-service-sync-config \
    --resource "service-arn" \
    --repository-provider "GITHUB" \
    --repository "example/proton-sync-service" \
    --ops-file-branch "main" \
    --proton-ops-file "./configuration/custom-proton-ops.yaml" (optional)
```

The response is as follows.

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

View configuration details for a service sync

You can view the configuration details data for a service sync using the console or AWS CLI.

AWS Management Console

Use the console to view configuration details for a service sync

1. In the navigation pane, choose **Services**.
2. To view detail data, choose the name of a service that you created a service sync configuration for.
3. In the detail page for the service, select the **Service sync** tab to view the configuration detail data for the service sync.

AWS CLI

Use the AWS CLI to get a synced service.

Run the following command.

```
$ aws proton get-service-sync-config \
    --service-name "service name"
```

The response is as follows.

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

Use the AWS CLI to get the service sync status.

Run the following command.

```
$ aws proton get-service-sync-status \
    --service-name "service name"
```

Edit a service sync configuration

You can edit a service sync configuration using the console or AWS CLI.

AWS Management Console

Edit a service sync configuration using the console.

1. In the navigation pane, choose **Services**.
2. To view detail data, choose the name of a service that you created a service sync configuration for.

3. On the service detail page, choose the **Service sync** tab.
4. In the **Service sync** section, choose **Edit**.
5. On the **Edit** page, update the information you want to edit and then choose **Save**.

AWS CLI

The following example command and response shows how you can edit a service sync configuration using the AWS CLI.

Run the following command.

```
$ aws proton update-service-sync-config \
    --service-name "service name" \
    --repository-provider "GITHUB" \
    --repository "example/proton-sync-service" \
    --ops-file-branch "main" \
    --ops-file "./configuration/custom-proton-ops.yaml"
```

The response is as follows.

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

Delete a service sync configuration

You can delete a service sync configuration using the console or AWS CLI.

AWS Management Console

Delete a service sync configuration using the console

1. On the service details page, choose the **Service sync** tab.

2. In the **Service sync details** section, choose **Disconnect** to disconnect your repository. After your repository is disconnected, we no longer sync the service from that repository.

AWS CLI

The following example commands and responses show how to use the AWS CLI to delete service synced configurations.

Run the following command.

```
$ aws proton delete-service-sync-config \
    --service-name "service name"
```

The response is as follows.

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

> **Note**
>
> Service sync doesn't delete service instances. It only deletes the configuration.

AWS Proton environments

For AWS Proton, an environment represents the set of shared resources and policies that AWS Proton services are deployed into. They can contain any resources that are expected to be shared across AWS Proton service instances. These resources can include VPCs, clusters, and shared load balancers or API Gateways. An AWS Proton environment must be created before a service can be deployed to it.

This section describes how to manage environments using create, view, update, and delete operations. For >additional information, see the The AWS Proton Service API Reference.

Topics

- IAM Roles
- Create an environment
- View environment data
- Update an environment
- Delete an environment
- Environment account connections
- Customer-managed environments
- CodeBuild provisioning role creation

IAM Roles

With AWS Proton, you supply the IAM roles and AWS KMS keys for the AWS resources that you own and manage. These are later applied to and used by resources owned and managed by developers. You create an IAM role to control your developer team's access to the AWS Proton API.

AWS Proton service role

When you create a new environment, you provide a related IAM service role. The role contains all permissions that are necessary to update all provisioned infrastructure defined in both the environment templates and the service templates. For role examples, see AWS Proton service role for provisioning using AWS CloudFormation. If you use environment account connections and environment accounts, you create the role in a selected environment account. For more

information, see Create an environment in one account and provision in another account and Environment account connections.

How you provide this service role, and who assumes the role, depends on your environment's provisioning method.

- *AWS-managed provisioning* – You provide the role to AWS Proton, either directly while creating an environment, or indirectly through account connections. AWS Proton assumes the role in the relevant account to provision environment and service infrastructure.
- *Self-managed provisioning* – It's your responsibility to configure your provisioning automation to assume an appropriate role using appropriate credentials when a pull request (PR) triggers a provisioning action. For an example GitHub Action that assumes a role, see Assuming a Role in the *"Configure AWS Credentials" Action For GitHub Actions* documentation.

For more information about provisioning methods, see the section called “Provisioning methods”.

Create an environment

Learn to create AWS Proton environments.

You can create an AWS Proton environment in one of two ways:

- Create, manage, and provision a standard environment by using a *standard environment template*. AWS Proton provisions infrastructure for your environment.
- Connect AWS Proton to customer-managed infrastructure by using a *customer-managed environment template*. You provision your own shared resources outside of AWS Proton, and then you provide provisioning outputs that AWS Proton can use.

You can choose one of several provisioning approaches when you create an environment.

- *AWS managed provisioning* – Create, manage, and provision an environment in a single account. AWS Proton provisions your environment.

  This method only supports CloudFormation infrastructure code (IaC) templates.
- *AWS managed provisioning to another account* – In a single management account, create and manage an environment that's provisioned in another account with environment account connections. AWS Proton provisions your environment in the other account. For more

information, see Create an environment in one account and provision in another account and Environment account connections.

This method only supports CloudFormation IaC templates.

- *Self-managed provisioning* – AWS Proton submits provisioning pull requests to a linked repository with your own provisioning infrastructure.

  This method only supports Terraform IaC templates.

- *CodeBuild provisioning* – AWS Proton uses AWS CodeBuild to run shell commands that you provide. Your commands can read inputs that AWS Proton provides, and are responsible for provisioning or deprovisioning infrastructure and generating output values. A template bundle for this method includes your commands in a manifest file and any programs, scripts, or other files that these commands may need.

  As an example to using CodeBuild provisioning, you can include code that uses the AWS Cloud Development Kit (AWS CDK) to provision AWS resources, and a manifest that installs the CDK and runs your CDK code.

  For more information, see the section called "CodeBuild bundle".

> **Note**
>
> You can use CodeBuild provisioning with environments and services. At this time you can't provision components this way.

With AWS managed provisioning (both in the same account and to another account), AWS Proton makes direct calls to provision your resources.

With self-managed provisioning, AWS Proton makes pull requests to provide compiled IaC files that your IaC engine uses to provision resources.

For more information, see the section called "Provisioning methods", the section called "Template bundles", and the section called "Environment schema requirements".

Topics

- Create and provision a standard environment in the same account
- Create an environment in one account and provision in another account

- Create and provision an environment using self-managed provisioning

Create and provision a standard environment in the same account

Use the console or AWS CLI to create and provision an environment in a single account. Provisioning is managed by AWS.

AWS Management Console

Use the console to create and provision an environment in a single account

1. In the AWS Proton console, choose **Environments**.
2. Choose **Create environment**.
3. In the **Choose an environment template** page, select a template and choose **Configure**.
4. In the **Configure environment** page, in the **Provisioning** section, choose **AWS managed provisioning**.
5. In the **Deployment account** section, choose **This AWS account**.
6. In the **Configure environment** page, in the **Environment settings** section, enter an **Environment name**.
7. (Optional) Enter a description for the environment.
8. In the **Environment roles** section, select the AWS Proton service role that you created as part of Setting up AWS Proton service roles.
9. (Optional) In the **Component role** section, select a service role that enables directly defined components to run in the environment and scopes down the resources that they can provision. For more information, see *Components*.
10. (Optional) In the **Tags** section, choose **Add new tag** and enter a key and value to create a customer managed tag.
11. Choose **Next**.
12. In the **Configure environment custom settings** page, you must enter values for the `required` parameters. You can enter values for the `optional` parameters or use the defaults when given.
13. Choose **Next** and review your inputs.
14. Choose **Create**.

View the environment details and status, as well as the AWS managed tags and customer managed tags for your environment.

15. In the navigation pane, choose **Environments**.

    A new page displays a list of your environments along with the status and other environment details.

AWS CLI

Use the AWS CLI to create and provision an environment in a single account.

To create an environment, you specify the AWS Proton service role ARN, path to your spec file, environment name, environment template ARN, the major and minor versions, and description (optional).

The next examples shows a YAML formatted spec file that specifies values for two inputs that are defined in the environment template schema file. You can use the `get-environment-template-minor-version` command to view the environment template schema.

```
proton: EnvironmentSpec
spec:
  my_sample_input: "the first"
  my_other_sample_input: "the second"
```

Create an environment by running the following command.

```
$ aws proton create-environment \
    --name "MySimpleEnv" \
    --template-name simple-env \
    --template-major-version 1 \
    --proton-service-role-arn "arn:aws:iam::123456789012:role/AWSProtonServiceRole"
 \
    --spec "file://env-spec.yaml"
```

Response:

```
{
```

```
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2020-11-11T23:03:05.405000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2020-11-11T23:03:05.405000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/ProtonServiceRole",
        "templateName": "simple-env"
    }
}
```

After you create a new environment, you can view a list of AWS and customer managed tags as shown in the following example command. AWS Proton automatically generates AWS managed tags for you. You can also modify and create customer managed tags using the AWS CLI. For more information, see AWS Proton resources and tagging.

Command:

```
$ aws proton list-tags-for-resource \
    --resource-arn "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv"
```

Create an environment in one account and provision in another account

Use the console or AWS CLI to create a standard environment in a management account that provisions environment infrastructure in another account. Provisioning is managed by AWS.

Before using the console or CLI, complete the following steps.

1. Identify the AWS account IDs for the management and environment account, and copy them for later use.
2. In the environment account, create an AWS Proton service role with minimum permissions for the environment to create. For more information, see AWS Proton service role for provisioning using AWS CloudFormation.

AWS Management Console

Use the console create an environment in one account and provision in another.

1. **In the environment account, create an environment account connection, and use it to send a request to connect to the management account.**

   a. In AWS Proton console, choose **Environment account connections** in the navigation pane.

   b. In the **Environment account connections** page, choose **Request to connect**.

   > **Note**
   >
   > Verify that the account ID that's listed in the **Environment account connection** page heading matches your pre-identified environment account ID.

   c. In the **Request to connect** page, in the **Environment role** section, select **Existing service role** and the name of the service role that you created for the environment.

   d. In the **Connect to management account** section, enter the **Management account ID** and an **Environment name** for your AWS Proton environment. Copy the name for later use.

   e. Choose **Request to connect** at the lower right corner of the page.

   f. Your request shows as pending in the **Environment connections sent to a management account** table and a modal shows how to accept the request from the management account.

2. **In the management account, accept a request to connect from the environment account.**

   a. Log in to your management account and choose **Environment account connections** in the AWS Proton console.

   b. In the **Environment account connections** page, in the **Environment account connection requests** table, select the environment account connection with the environment account ID that matches your pre-identified environment account ID.

   > **Note**
   >
   > Verify that the account ID that's listed in the **Environment account connection** page heading matches your pre-identified management account ID.

c. Choose **Accept**. The status changes from PENDING to CONNECTED.

3. **In the management account, create an environment.**

   a. In the navigation pane, choose **Environment templates**.
   b. In the **Environment templates** page, choose **Create environment template**.
   c. In the **Choose an environment template** page, choose an environment template.
   d. In the **Configure environment** page, in the **Provisioning** section, choose **AWS managed provisioning**.
   e. In the **Deployment account** section, choose **Another AWS account;**.
   f. In the **Environment details** section, select your **Environment account connection** and **Environment name**.
   g. Choose **Next**.
   h. Fill out the forms and choose **Next** until you reach the **Review and Create** page.
   i. Review and choose **Create environment**.

AWS CLI

Use the AWS CLI to create an environment in one account and provision in another.

In the environment account, create an environment account connection and request to connect by running the following command.

```
$ aws proton create-environment-account-connection \
    --environment-name "simple-env-connected" \
    --role-arn "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role" \
    --management-account-id "111111111111"
```

Response:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
```

```
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "PENDING"
    }
}
```

In the management account, accept the environment account connection request by running the following command.

```
$ aws proton accept-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

Response:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

View your environment account connection by running the following command.

```
$ aws proton get-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

Response:

```
{
    "environmentAccountConnection": {
```

```
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

In the management account, create an environment by running the following command.

```
$ aws proton create-environment \
    --name "simple-env-connected" \
    --template-name simple-env-template \
    --template-major-version "1" \
    --template-minor-version "1" \
    --spec "file://simple-env-template/specs/original.yaml" \
    --environment-account-connection-id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

Response:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:111111111111:environment/simple-env-
connected",
        "createdAt": "2021-04-28T23:02:57.944000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "environmentAccountConnectionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "lastDeploymentAttemptedAt": "2021-04-28T23:02:57.944000+00:00",
        "name": "simple-env-connected",
        "templateName": "simple-env-template"
    }
}
```

Create and provision an environment using self-managed provisioning

When you use self-managed provisioning, AWS Proton submits provisioning pull requests to a linked repository with your own provisioning infrastructure. The pull requests start your own workflow, which calls AWS services; to provision infrastructure.

Self-managed provisioning considerations:

- Before you create an environment, set up a repository resource directory for self-managed provisioning. For more information, see AWS Proton infrastructure as code files.
- After you create the environment, AWS Proton waits to receive asynchronous notifications regarding the status of your infrastructure provisioning. Your provisioning code must use the AWS Proton `NotifyResourceStateChange` API to send these asynchronous notifications to AWS Proton.

You can use self-managed provisioning in the console or with the AWS CLI. The following examples show how you can use self-managed provisioning with Terraform.

AWS Management Console

Use the console to create a Terraform environment using self-managed provisioning.

1. In the AWS Proton console, choose **Environments**.
2. Choose **Create environment**.
3. In the **Choose an environment template** page, select a Terraform template and choose **Configure**.
4. In the **Configure environment** page, in the **Provisioning** section, choose **Self-managed provisioning**.
5. In the **Provisioning repository details** section:

   a. If you haven't yet linked your provisioning repository to AWS Proton, choose **New repository**, choose one of the repository providers, and then, for **CodeStar connection**, choose one of your connections.

   > **Note**
   >
   > If you don't yet have a connection to the relevant repository provider account, choose **Add a new CodeStar connection**. Then, create a connection, and then

choose the refresh button next to the **CodeStar connection** menu. You should now be able to choose your new connection in the menu.

If you've already linked your repository to AWS Proton, choose **Existing repository**.

b. For **Repository name**, choose a repository. The drop-down menu shows linked repositories for **Existing repository** or the list of repositories in the provider account for **New repository**.

c. For **Branch name**, choose one of the repository branches.

6. In the **Environment settings** section, enter an **Environment name**.
7. (Optional) Enter a description for the environment.
8. (Optional) In the **Tags** section, choose **Add new tag** and enter a key and value to create a customer managed tag.
9. Choose **Next**.
10. In the **Configure environment custom settings** page, you must enter values for the `required` parameters. You can enter values for the `optional` parameters or use the defaults when given.
11. Choose **Next** and review your inputs.
12. Choose **Create** to send a pull request.

    - If you approve the pull request, the deployment is in progress.
    - If you reject the pull request, the environment creation is cancelled.
    - If the pull request times out, environment creation *isn't* complete.
13. View the environment details and status, as well as the AWS managed tags and customer managed tags for your environment.
14. In the navigation pane, choose **Environments**.

    A new page displays a list of your environments along with the status and other environment details.

AWS CLI

When you create an environment using self-managed provisioning, you *add* the `provisioningRepository` parameter and omit the `ProtonServiceRoleArn` and `environmentAccountConnectionId` parameters.

Use the AWS CLI to create a Terraform environment with self-managed provisioning.

1. Create an environment and send a pull request to the repository for review and approval.

   The next examples shows a YAML formatted spec file that defines the values for two inputs based on the environment template schema file. You can use the `get-environment-template-minor-version` command to view the environment template schema.

   Spec:

```
proton: EnvironmentSpec
spec:
  ssm_parameter_value: "test"
```

   Create an environment by running the following command.

```
$ aws proton create-environment \
    --name "pr-environment" \
    --template-name "pr-env-template" \
    --template-major-version "1" \
    --provisioning-repository="branch=main,name=myrepos/env-
repo,provider=GITHUB" \
    --spec "file://env-spec.yaml"
```

   Response:>

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/pr-
environment",
        "createdAt": "2021-11-18T17:06:58.679000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-11-18T17:06:58.679000+00:00",
        "name": "pr-environment",
        "provisioningRepository": {
            "arn": "arn:aws:proton:region-id:123456789012:repository/
github:myrepos/env-repo",
            "branch": "main",
            "name": "myrepos/env-repo",
            "provider": "GITHUB"
        },
```

```
        "templateName": "pr-env-template"
    }
```

2. Review the request.

   - If you approve the request, provisioning is in progress.
   - If you reject the request, the environment creation is cancelled.
   - If the pull request times out, environment creation *isn't* complete.

3. Asynchronously provide provisioning status to AWS Proton. The following example notifies AWS Proton of a successful provisioning.

```
$ aws proton notify-resource-deployment-status-change \
    --resource-arn "arn:aws:proton:region-id:123456789012:environment/pr-environment" \
    --status "SUCCEEDED"
```

View environment data

You can view environment detail data using either the AWS Proton console or the AWS CLI.

AWS Management Console

You can view lists of environments with details and individual environments with detail data by using the AWS Proton console.

1. To view a list of your environments, choose **Environments** in the navigation pane.
2. To view detail data, choose the name of an environment.

   View your environment detail data.

AWS CLI

Use the AWS CLI *get* or *list* environment details.

Run the following command:

```
$ aws proton get-environment \
    --name "MySimpleEnv"
```

Response:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2020-11-11T23:03:05.405000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "lastDeploymentAttemptedAt": "2020-11-11T23:03:05.405000+00:00",
        "lastDeploymentSucceededAt": "2020-11-11T23:03:05.405000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\nspec:\n  my_sample_input: \"the first\"\n
 my_other_sample_input: \"the second\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "simple-env"
    }
}
```

Update an environment

If the AWS Proton environment is associated with an environment account connection, *don't* update or include the `protonServiceRoleArn` parameter to update or connect to an environment account connection.

You can only update to a new environment account connection if both of the following is true:

- The environment account connection was created in the same environment account that the current environment account connection was created in.
- >The environment account connection is associated with the current environment.

If the environment *isn't* associated with an environment account connection, *don't* update or include the `environmentAccountConnectionId` parameter.

You can update either the `environmentAccountConnectionId` or `protonServiceRoleArn` parameter and value. You can't update both.

If your environment uses self-managed provisioning, *don't* update the `provisioning-repository` parameter and *omit* the `environmentAccountConnectionId` and `protonServiceRoleArn` parameters.

There are four modes for updating an environment as described in the following list. When using the AWS CLI, the `deployment-type` field defines the mode. When using the console, these modes map to the **Edit**, **Update**, **Update minor**, and **Update major** actions that drop down from **Actions**.

`NONE`

In this mode, a deployment *doesn't* occur. Only the requested metadata parameters are updated.

`CURRENT_VERSION`

In this mode, the environment is deployed and updated with the new spec that you provide. Only requested parameters are updated. *Don't* include minor or major version parameters when you use this `deployment-type`.

`MINOR_VERSION`

In this mode, the environment is deployed and updated with the published, recommended (latest) minor version of the current major version in use by default. You can also specify a different minor version of the current major version in use.

`MAJOR_VERSION`

In this mode, the environment is deployed and updated with the published, recommended (latest) major and minor version of the current template by default. You can also specify a different major version that is higher than the major version in use and a minor version (optional).

Topics

- Update an AWS managed provisioning environment
- Update a self-managed provisioning environment
- Cancel an environment deployment in progress

Update an AWS managed provisioning environment

Standard provisioning is only supported by environments that provision with AWS CloudFormation.

Use the console or AWS CLI to update your environment.

AWS Management Console

Update an environment using the console as shown in the following steps.

1. **Choose 1 of the following 2 steps.**
   a. **In the list of environments.**
      i. In the AWS Proton console, choose **Environments**.
      ii. In the list of environments, choose the radio button to the left of the environment that you want to update.
   b. **In the console environment detail page.**
      i. In the AWS Proton console, choose **Environments**.
      ii. In the list of environments, choose the name of the environment that you want to update.
2. **Choose 1 of the next 4 steps to update your environment.**
   a. **To make an edit that doesn't require environment deployment.**
      i. For example, to change a description.

         Choose **Edit**.
      ii. Fill out the form and choose **Next**.
      iii. Review your edit and choose **Update**.
   b. **To make updates to metadata inputs only.**
      i. Choose **Actions** and then **Update**.
      ii. Fill out the form and choose **Edit**.
      iii. Fill out the forms and choose **Next** until you reach the **Review** page.
      iv. Review your updates and choose **Update**.
   c. **To make an update to a new minor version of its environment template.**
      i. Choose **Actions** and then **Update minor**.
      ii. Fill out the form and choose **Next**.
      iii. Fill out the forms and choose **Next** until you reach the **Review** page.

iv. Review your updates and choose **Update**.

d. **To make an update to a new major version of its environment template.**

i. Choose **Actions** and then **Update major**.

ii. Fill out the form and choose **Next**.

iii. Fill out the forms and choose **Next** until you reach the **Review** page.

iv. Review your updates and choose **Update**.

AWS CLI

Use the AWS Proton AWS CLI to update an environment to a new minor version.

Run the following command to update your environment:

```
$ aws proton update-environment \
        --name "MySimpleEnv" \
        --deployment-type "MINOR_VERSION" \
        --template-major-version "1" \
        --template-minor-version "1" \
        --proton-service-role-arn arn:aws:iam::123456789012:role/service-
role/ProtonServiceRole \
        --spec "file:///spec.yaml"
```

Response:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-04-02T17:48:26.307000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:29:55.472000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "simple-env"
    }
}
```

Run the following command to get and confirm the status:

```
$ aws proton get-environment \
        --name "MySimpleEnv"
```

Response:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "environmentName": "MySimpleEnv",
        "lastDeploymentAttemptedAt": "2021-04-02T17:48:26.307000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\n\nspec:\n  my_sample_input: hello\n
 my_other_sample_input: everybody\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```

Update a self-managed provisioning environment

Self-managed provisioning is only supported by environments that provision with Terraform.

Use the console or AWS CLI to update your environment.

AWS Management Console

Update an environment using the console as shown in the following steps.

1. **Choose 1 of the following 2 steps.**

    a. **In the list of environments.**

        i. In the AWS Proton console, choose **Environments**.

ii. In the list of environments, choose the radio button to the left of the environment template that you want to update.

b. **In the console environment detail page.**

i. In the AWS Proton console, choose **Environments**.

ii. In the list of environments, choose the name of the environment that you want to update.

2. **Choose 1 of the next 4 steps to update your environment.**

a. **To make an edit that doesn't require environment deployment.**

i. For example, to change a description.

Choose **Edit**.

ii. Fill out the form and choose **Next**.

iii. Review your edit and choose **Update**.

b. **To make updates to metadata inputs only.**

i. Choose **Actions** and then **Update**.

ii. Fill out the form and choose **Edit**.

iii. Fill out the forms and choose **Next** until you reach the **Review** page.

iv. Review your updates and choose **Update**.

c. **To make an update to a new minor version of its environment template.**

i. Choose **Actions** and then **Update minor**.

ii. Fill out the form and choose **Next**.

iii. Fill out the forms and choose **Next** until you reach the **Review** page.

iv. Review your updates and choose **Update**.

d. **To make an update to a new major version of its environment template.**

i. Choose **Actions** and then **Update major**.

ii. Fill out the form and choose **Next**.

iii. Fill out the forms and choose **Next** until you reach the **Review** page.

iv. Review your updates and choose **Update**.

AWS CLI

Use the AWS CLI to update a Terraform environment to a new minor version with self-managed provisioning.

1. Run the following command to update your environment:

```
$ aws proton update-environment \
    --name "pr-environment" \
    --deployment-type "MINOR_VERSION" \
    --template-major-version "1" \
    --template-minor-version "1" \
    --provisioning-repository "branch=main,name=myrepos/env-
repo,provider=GITHUB" \
    --spec "file://env-spec-mod.yaml"
```

Response:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/pr-
environment",
        "createdAt": "2021-11-18T21:09:15.745000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-11-18T21:25:41.998000+00:00",
        "lastDeploymentSucceededAt": "2021-11-18T21:09:15.745000+00:00",
        "name": "pr-environment",
        "provisioningRepository": {
            "arn": "arn:aws:proton:region-id:123456789012:repository/
github:myrepos/env-repo",
            "branch": "main",
            "name": "myrepos/env-repo",
            "provider": "GITHUB"
        },
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "pr-env-template"
    }
}
```

2. Run the following command to get and confirm the status:

```
$ aws proton get-environment \
    --name "pr-environment"
```

Response:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/pr-
environment",
        "createdAt": "2021-11-18T21:09:15.745000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "lastDeploymentAttemptedAt": "2021-11-18T21:25:41.998000+00:00",
        "lastDeploymentSucceededAt": "2021-11-18T21:25:41.998000+00:00",
        "name": "pr-environment",
        "provisioningRepository": {
            "arn": "arn:aws:proton:region-id:123456789012:repository/
github:myrepos/env-repo",
            "branch": "main",
            "name": "myrepos/env-repo",
            "provider": "GITHUB"
        },
        "spec": "proton: EnvironmentSpec\nspec:\n   ssm_parameter_value: \"test
\"\n ssm_another_parameter_value: \"update\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "pr-env-template"
    }
}
```

3. Review the pull request that was sent by AWS Proton.

   - If you approve the request, provisioning is in progress.
   - If you reject the request, the environment creation is cancelled.
   - If the pull request times out, environment creation isn't complete.

4. Provide provisioning status to AWS Proton.

```
$ aws proton notify-resource-deployment-status-change \
    --resource-arn "arn:aws:proton:region-id:123456789012:environment/pr-
environment" \
    --status "SUCCEEDED"
```

Cancel an environment deployment in progress

You can attempt to cancel an environment update deployment if the `deploymentStatus` is in `IN_PROGRESS`. AWS Proton attempts to cancel the deployment. Successful cancellation *isn't* guaranteed.

When you cancel an update deployment, AWS Proton attempts to cancel the deployment as listed in the following steps.

With AWS-managed provisioning, AWS Proton does the following:

- Sets the deployment state to `CANCELLING`.
- Stops the deployment in progress and deletes any new resources that were created by the deployment when `IN_PROGRESS`.
- Sets the deployment state to `CANCELLED`.
- Reverts the state of the resource to what it was before the deployment was started.

With self-managed provisioning, AWS Proton does the following:

- Attempts to close the pull request to prevent merging the changes to your repository.
- Sets the deployment state to `CANCELLED` if the pull request was successfully closed.

For instructions on how to cancel an environment deployment, see CancelEnvironmentDeployment in the *AWS Proton API Reference*.

You can use the console or CLI to cancel environments that are in progress.

AWS Management Console

Use the console to cancel an environment update deployment as shown in the following steps.

1. In the AWS Proton console, choose **Environments** in the navigation pane.
2. In the list of environments, choose the name of the environment with the deployment update that you want to cancel.
3. If your update deployment status is **In progress**, in the environment detail page, choose **Actions** and then **Cancel deployment**.

4. A modal prompts you to confirm that you want to cancel. Choose **Cancel deployment**.
5. Your update deployment status is set to **Cancelling** and then **Cancelled** to complete the cancellation.

AWS CLI

Use the AWS Proton AWS CLI to cancel an IN_PROGRESS environment update deployment to a new minor version 2.

A wait condition is included in the template used for this example so that the cancellation starts before the update deployment succeeds.

Run the following command to cancel the update:

```
$ aws proton cancel-environment-deployment \
        --environment-name "MySimpleEnv"
```

Response:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "CANCELLING",
        "lastDeploymentAttemptedAt": "2021-04-02T18:15:10.243000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\n\nspec:\n  my_sample_input: hello\n
 my_other_sample_input: everybody\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```

Run the following command to get and confirm the status:"

```
$ aws proton get-environment \
```

```
        --name "MySimpleEnv"
```

Response:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "CANCELLED",
        "deploymentStatusMessage": "User initiated cancellation.",
        "lastDeploymentAttemptedAt": "2021-04-02T18:15:10.243000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\n\nspec:\n  my_sample_input: hello\n
 my_other_sample_input: everybody\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```

Delete an environment

You can delete an AWS Proton environment by using the AWS Proton console or the AWS CLI.

> **Note**
>
> You can't delete an environment that has any associated component. To delete such an environment, you should first delete all components that are running in the environment. For more information about components, see *Components*.

AWS Management Console

Delete an environment using the console as described in the following two options.

In the list of environments.

1. In the AWS Proton console, choose **Environments**.

2. In the list of environments, select the radio button to the left of the environment that you want to delete.
3. Choose **Actions** and then **Delete**.
4. A modal prompts you to confirm the delete action.
5. Follow the instructions and choose **Yes, delete**.

In the environment detail page.

1. In the AWS Proton console, choose **Environments**.
2. In the list of environments, choose the name of the environment that you want to delete.
3. In the environment detail page, choose **Actions** and then **Delete**.
4. A modal prompts you to confirm that you want to delete.
5. Follow the instructions and choose **Yes, delete**.

AWS CLI

Use the AWS CLI to delete an environment.

Don't delete an environment if services or service instances are deployed to the environment.

Run the following command:

```
$ aws proton delete-environment \
    --name "MySimpleEnv"
```

Response:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "DELETE_IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-04-02T17:48:26.307000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/ProtonServiceRole",
        "templateMajorVersion": "1",
```

```
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```

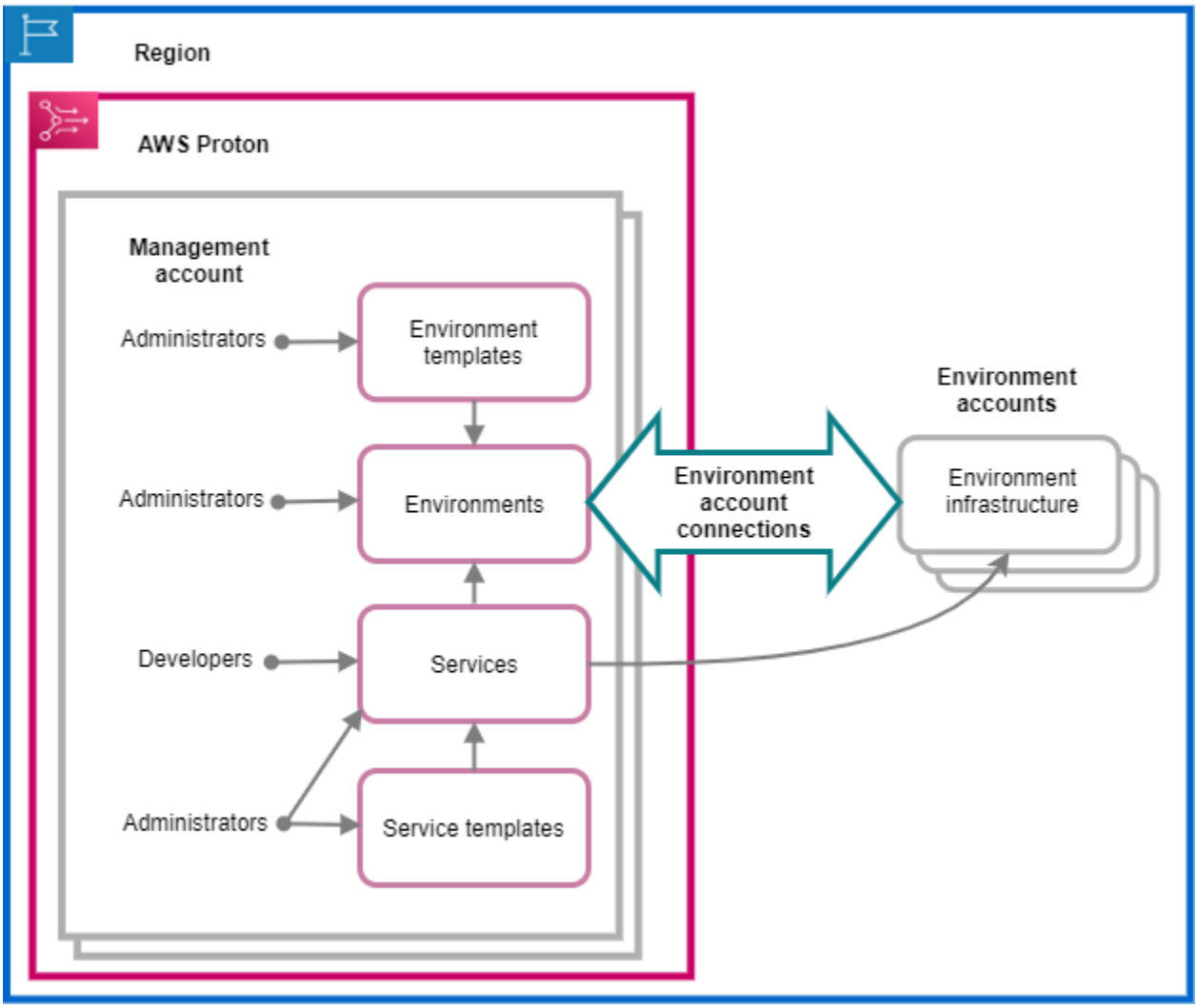
Environment account connections

Overview

Learn how to create and manage an AWS Proton environment in one account and provision its infrastructure resources in another account. This can help improve visibility and efficiency at scale. Environment account connections only support standard provisioning with AWS CloudFormation infrastructure as code.

> **Note**
>
> The information in this topic is relevant to environments that are configured with *AWS managed provisioning*. With environments configured with *self-managed provisioning*, AWS Proton doesn't directly provision your infrastructure. Instead, it sends pull requests (PRs) to your repository for provisioning. It's your responsibility to ensure that your automation code assumes the right identity and role.
> For more information about provisioning methods, see the section called “Provisioning methods”.

Terminology



With AWS Proton *environment account connections*, you can create an AWS Proton environment from one account and provision its infrastructure in another account.

Management account

The single account where you, as an administrator, create an AWS Proton environment that provisions infrastructure resources in another *environment account*.

Environment account

An account that environment infrastructure is provisioned in, when you create an AWS Proton environment in another account.

Environment account connection

A secure bi-directional connection between a *management account* and an *environment account*. It maintains authorization and permissions as described further in the following sections.

When you create an environment account connection in an environment account in a specific Region, only the management accounts in the same Region can see and use the environment account connection. This means that the AWS Proton environment created in the management account and the environment infrastructure provisioned in the environment account must be in the same Region.

Environment account connection considerations

- You need an environment account connection for each environment that you want to provision in an environment account.
- For information about environment account connection quotas, see AWS Proton quotas.

Tagging

In the environment account, use the console or the AWS CLI to view and manage environment account connection customer managed tags. AWS managed tags *aren't* generated for environment account connections. For more information, see *Tagging*.

Create an environment in one account and provision its infrastructure in another account

To create and provision an environment from a single management account, set up an environment account for an environment that you plan to create.

Start in the environment account and create connection.

In the environment account, create an AWS Proton service role that's scoped down to only the permissions that are needed for provisioning your environment infrastructure resources. For more information, see AWS Proton service role for provisioning using AWS CloudFormation.

Then, create and send an environment account connection request to your management account. When the request is accepted, AWS Proton can use the associated IAM role that permits environment resource provisioning in the associated environment account.

In the management account, accept or reject the environment account connection.

In the management account, accept or reject the environment account connection request. You *can't* delete an environment account connection from your management account.

If you accept the request, the AWS Proton can use the associated IAM role that permits resource provisioning in the associated environment account.

The environment infrastructure resources are provisioned in the associated environment account. You can only use AWS Proton APIs to access and manage your environment and its infrastructure resources, from your management account. For more information, see Create an environment in one account and provision in another account and Update an environment.

After you reject a request, you *can't* accept or use the rejected environment account connection.

> **Note**
>
> You *can't* reject an environment account connection that's connected to an environment. To reject the environment account connection, you must first delete the associated environment.

In the environment account, access the provisioned infrastructure resources.

In the environment account, you can view and access the provisioned infrastructure resources. For example, you can use CloudFormation API actions to monitor and clean up stacks if needed. You can't use the AWS Proton API actions to access or manage the AWS Proton environment that was used to provision the infrastructure resources.

In the environment account, you can delete environment account connections that you have created in the environment account. You *can't* accept or reject them. If you delete an environment account connection that's in use by an AWS Proton environment, AWS Proton won't be able to manage the environment infrastructure resources until a new environment connection is accepted for the environment account and named environment. You're responsible for cleaning up provisioned resources that remain without an environment connection.

Use the console or CLI to manage environment account connections

You can use the console or CLI to create and manage environment account connections.

AWS Management Console

Use the console to create an environment account connection and send a request to the management account as shown in the next steps.

1. Decide on a name for the environment that you plan to create in your management account or choose the name of an existing environment that requires an environment account connection.
2. In an environment account, in the AWS Proton console, choose **Environment account connections** in the navigation pane.
3. In the **Environment account connections** page, choose **Request to connect**.

   > **Note**
   >
   > Verify the account ID that's listed in the **Environment account connection** page heading. Make sure that it matches the account ID of the environment account that you want your named environment to provision in.

4. In the **Request to connect** page:

   a. In the **Connect to management account** section, enter the **Management account ID** and the **Environment name** that you entered in step 1.

   b. In the **Environment role** section, choose **New service role** and AWS Proton automatically creates a new role for you. Or, select **Existing service role** and the name of the service role that you created previously.

   > **Note**
   >
   > The role that AWS Proton automatically creates for you has broad permissions. We recommend that you scope down the role to the permissions required to provision your environment infrastructure resources. For more information, see AWS Proton service role for provisioning using AWS CloudFormation.

   c. (Optional) In the **Tags** section, choose **Add new tag** to create a customer managed tag for your environment account connection.

   d. Choose **Request to connect**.

5. Your request shows as pending in the **Environment connections sent to a management account** table and a modal lets you know how to accept the request from the management account.

Accept or reject an environment account connection request.

1. In a management account, in the AWS Proton console, choose **Environment account connections** in the navigation pane.
2. In the **Environment account connections** page, in the **Environment account connection requests** table, choose the environment connection request to accept or reject.

> **Note**
>
> Verify the account ID that's listed in the **Environment account connection** page heading. Make sure that it matches the account ID of the management account that's associated with the environment account connection to reject. After you reject this environment account connection, you *can't* accept or use the rejected environment account connection.

3. Choose **Reject** or **Accept**.
   - If you selected **Reject**, the status changes from *pending* to *rejected*.
   - If you selected **Accept**, the status changes from *pending* to *connected*.

Delete an environment account connection.

1. In an environment account, in the AWS Proton console, choose **Environment account connections** in the navigation pane.

> **Note**
>
> Verify the account ID that's listed in the **Environment account connection** page heading. Make sure that it matches the account ID of the management account that's associated with the environment account connection to reject. After you delete this environment account connection, AWS Proton *can't* manage the environment infrastructure resources in the environment account. It can only

manage it after a new environment account connection for the environment account and named environment is accepted by the management account.

2. In the **Environment account connections** page, in the **Sent requests to connect to management account** section, choose **Delete**.
3. A modal prompts you to confirm that you want to delete. Choose **Delete**.

AWS CLI

Decide on a name for the environment that you plan to create in your management account or choose the name of an existing environment that requires an environment account connection.

Create an environment account connection in an environment account.

Run the following command:

```
$ aws proton create-environment-account-connection \
    --environment-name "simple-env-connected" \
    --role-arn "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role" \
    --management-account-id "111111111111"
```

Response:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "PENDING"
    }
}
```

Accept or reject an environment account connection in a management account as shown in the following command and response.

> **Note**
>
> If you reject this environment account connection, you won't be able to accept or use the rejected environment account connection.

If you specify **Reject**, the status changes from *pending* to *rejected*.

If you specify **Accept**, the status changes from *pending* to *connected*.

Run the following command to accept the environment account connection:

```
$ aws proton accept-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

Response:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

Run the following command to reject the environment account connection:

```
$ aws proton reject-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

Response:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:us-east-1:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "status": "REJECTED",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-reject",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role"
    }
}
```

View an environment account's connections. You can *get* or *list* environment account connections.

Run the following get command:

```
$ aws proton get-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

Response:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
```

```
}
```

Delete an environment account connection in an environment account.

> **Note**
>
> If you delete this environment account connection, AWS Proton won't be able to manage the environment infrastructure resources in the environment account until a new environment connection has been accepted for the environment account and named environment. You're responsible for cleaning up provisioned resources that remain without an environment connection.

Run the following command:

```
$ aws proton delete-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

Response:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:us-east-1:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

Customer-managed environments

With customer-managed environments, you can use existing infrastructure, like a VPC, that you already have deployed as your AWS Proton environment. While using customer-managed environments, you can provision your own shared resources outside of AWS Proton. However, you can still allow AWS Proton to consume relevant provisioning outputs as inputs for AWS Proton services when they are deployed. If the outputs can change, AWS Proton is able to accept updates. AWS Proton is unable to change the environment directly, though, since the provisioning is managed outside of AWS Proton.

After the environment is created, you're responsible for providing the same outputs to AWS Proton that would have been created if AWS Proton had made the environment, such as Amazon ECS cluster names or Amazon VPC IDs.

With this functionality, you can deploy and update AWS Proton service resources from an AWS Proton service template to this environment. However, the environment itself isn't modified through template updates in AWS Proton. You're responsible for executing updates to the environment and updating those outputs in AWS Proton.

You can have multiple environments in a single account that are a mix of AWS Proton managed and customer-managed environments. You can also link a second account and use an AWS Proton template in the primary account to execute deployments and updates to environments and services in that second, linked account.

How to use customer-managed environments

The first thing administrators need to do is register an imported, customer-managed environment template. Don't supply manifests or infrastructure files in the template bundle. Only supply the schema.

The schema below outlines a list of outputs using the open API format and replicates the outputs from an AWS CloudFormation template.

> **Important**
>
> Only string inputs are allowed for the outputs.

The following example is a snippet of the output sections of an AWS CloudFormation template for a corresponding Fargate template.

```
Outputs:
  ClusterName:
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'
  ECSTaskExecutionRole:
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'
  VpcId:
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'
[...]
```

The schema for the corresponding AWS Proton imported environment is similar to the following. Don't supply default values in the schema.

```
schema:
  format:
    openapi: "3.0.0"
  environment_input_type: "EnvironmentOutput"
  types:
    EnvironmentOutput:
      type: object
      description: "Outputs of the environment"
      properties:
        ClusterName:
          type: string
          description: "The name of the ECS cluster"
        ECSTaskExecutionRole:
          type: string
          description: "The ARN of the ECS role"
        VpcId:
          type: string
          description: "The ID of the VPC that this stack is deployed in"
[...]
```

At the time of registering the template, you indicate that this template is imported and provides the Amazon S3 bucket location for the bundle. AWS Proton validates that the schema only contains `environment_input_type` and no AWS CloudFormation template parameters before putting the template in draft.

You provide the following to create an imported environment.

- An IAM role to use when making deployments.
- A specification with the values for the required outputs.

You can provide both of these through either the console or the AWS CLI using a process similar to the deployment of a regular environment.

CodeBuild provisioning role creation

Infrastructure as a Code (IaaC) tools like AWS CloudFormation and Terraform require permissions for the many different types of AWS resources. For example, if an IaaC template declares an Amazon S3 bucket, it needs permissions to create, read, update, and delete Amazon S3 buckets. It's considered a security best practice to limit roles to the minimal permissions required. Given the breadth of AWS resources, it’s challenging to create least-privilege policies for IaaC templates, especially when the resources being managed by those templates can change later. For example, in your latest edits to a template being managed by AWS Proton, you add an RDS database resource.

Configuring the right permissions helps make deployments of your IaC smooth. AWS Proton CodeBuild Provisioning executes arbitrary customer-supplied CLI commands in a CodeBuild project located in the customer’s account. Typically, these commands create and delete infrastructure using an Infrastructure as Code (IaaC) tool such as AWS CDK. When an AWS resource deploys whose template uses CodeBuild Provisioning, AWS will start a build in a CodeBuild project managed by AWS. A role is passed to CodeBuild, which CodeBuild assumes to execute commands. This role, called the CodeBuild Provisioning Role, is provided by the customer and contains permissions required to provision infrastructure. It's meant to be assumed only by CodeBuild and even AWS Proton can't assume it.

Creating the role

The CodeBuild Provisioning role can be created in the IAM console or in the AWS CLI. To create it in the AWS CLI:

```
aws iam create-role --role-name AWSProtonCodeBuildProvisioning --assume-role-
policy-document '{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Principal":
{"Service":"codebuild.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
aws iam attach-role-policy --role-name AWSProtonCodeBuildProvisioning --policy-arn
 arn:aws:iam::aws:policy/AWSProtonCodeBuildProvisioningBasicAccess
```

This also attaches the `AWSProtonCodeBuildProvisioningBasicAccess`, which contains the minimal permissions needed by the CodeBuild service to run a build.

If you prefer to use the console, please ensure the following when you create the role:

1. For trusted entity, select AWS service and then select CodeBuild.
2. In the Add permissions step, select `AWSProtonCodeBuildProvisioningBasicAccess` and any other policies you want to attach.

Administrator Access

If you attach the `AdministratorAccess` policy to the CodeBuild Provisioning Role, it will guarantee that any IaaC template won't fail due to lack of permissions. It also means that anyone who can create an Environment Template or Service Template can perform administrator-level actions, even if that user isn't an administrator. AWS Proton doesn't recommend using `AdministatorAccess` with the CodeBuild Provisioning Role. If you decide to use `AdministratorAccess` with the CodeBuild Provisioning Role, do so in a sandbox environment.

You can create a role with `AdministratorAccess` in the IAM console or by executing this command:

```
aws iam create-role --role-name AWSProtonCodeBuildProvisioning --assume-role-
policy-document '{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Principal":
{"Service":"codebuild.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
aws iam attach-role-policy --role-name AWSProtonCodeBuildProvisioning --policy-arn
 arn:aws:iam::aws:policy/AdministratorAccess
```

Creating a Minimally-Scoped Role

If you want to create a role with minimum permissions, there are multiple approaches:

- Deploy with admin permissions, then scope down the role. We recommend using IAM Access Analyzer.
- Use managed policies to give access to the services you plan on using.

AWS CDK

If you're using AWS CDK with AWS Proton, and you've run `cdk bootstrap` on each environment account/Region, then there already exists a role for `cdk deploy`. In this case, attach the following policy to the CodeBuild Provisioning Role:

```
{
    "Action": "sts:AssumeRole",
    "Resource": [
        "arn:aws:iam::account-id:role/cdk-*-deploy-role-*",
        "arn:aws:iam::account-id:role/cdk-*-file-publishing-role-*"
    ],
    "Effect": "Allow"
}
```

Custom VPC

If you decide to run CodeBuild in a custom VPC, you'll need the following permissions in your CodeBuild role:

```
{
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterface"
    ],
    "Resource": [
        "arn:aws:ec2:region:account-id:network-interface/*",
        "arn:aws:ec2:region:account-id:subnet/*",
        "arn:aws:ec2:region:account-id:security-group/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:DeleteNetworkInterface"
    ],
    "Resource": [
        "arn:aws:ec2:region:account-id:*/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeNetworkInterfaces",
```

```
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeVpcs"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterfacePermission"
    ],
    "Resource": "arn:aws:ec2:region:account-id:network-interface/*",
    "Condition": {
        "StringEquals": {
            "ec2:AuthorizedService": "codebuild.amazonaws.com"
        }
    }
}
```

You could also use the `AmazonEC2FullAccess` managed policy, although that includes permissions that you may not need. To attach the managed policy using the CLI:

```
aws iam create-role --role-name AWSProtonCodeBuildProvisioning --assume-role-
policy-document '{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Principal":
{"Service":"codebuild.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
aws iam attach-role-policy --role-name AWSProtonCodeBuildProvisioning --policy-arn
 arn:aws:iam::aws:policy/AdministratorAccess
```

AWS Proton services

An AWS Proton service is an instantiation of a service template, normally including several service instances and a pipeline. An AWS Proton service instance is an instantiation of a service template in a specific environment. A service template is a complete definition of the infrastructure and optional service pipeline for an AWS Proton service.

After you deploy your service instances, you can update them by source code pushes that prompt the CI/CD pipeline or by updating the service to new versions of its service template. AWS Proton prompts you when new versions of its service template become available so you can update your services to a new version. When your service is updated, AWS Proton re-deploys the service and service instances.

This chapter shows how to manage services by using create, view, update and delete operations. For additional information, see the *The AWS Proton Service API Reference*.

Topics

- Create a service
- View service data
- Edit a service
- Delete a service
- View service instance data
- Update a service instance
- Update a service pipeline

Create a service

To deploy an application with AWS Proton, as a developer, you create a service and provide the following inputs.

1. The name of an AWS Proton service template that's published by the platform team.
2. A name for the service.
3. The number of service instances that you want to deploy.
4. A selection of environments that you want to use.

5. A connection to your code repository if you're using a service template that includes a service pipeline (optional).

What's in a service?

When you create an AWS Proton service, you can choose from two different types of service templates:

- A service template that includes a service pipeline (default).
- A service template that *doesn't* include a service pipeline.

You must create at least one service instance when you create your service.

A service instance and optional pipeline are associated with a service. You can only create or delete a pipeline within the context of service *create* and *delete* actions. To learn how to add and remove instances from a service, see Edit a service.

> **Note**
>
> Your environment is configured for either AWS- or self-managed provisioning. AWS Proton provisions services in an environment using the same provisioning method as the environment uses. The developer creating or updating service instances doesn't see the difference and their experience is the same in both case.
> For more information about provisioning methods, see the section called “Provisioning methods”.

Service templates

Both major and minor versions of service templates are available. When you use the console, you select the latest `Recommended` major and minor version of the service template. When you use the AWS CLI and you specify only the major version of the service template, you implicitly specify its latest `Recommended` minor version.

The following describes the difference between major and minor template versions and their use.

- New versions of a template become `Recommended` as soon as they're approved by a member of the platform team. This means that new services are created using that version, and you're prompted to update existing services to the new version.
- Through AWS Proton, the platform team can automatically update service instances to a new minor version of a service template. Minor versions must be backward compatible.
- Because major versions require you to provide new inputs as part of the update process, you need to update your service to a major version of its service template. Major versions *aren't* backward compatible.

Create a service

The following procedures show how to use the AWS Proton console or AWS CLI to create a service with or without a service pipeline.

AWS Management Console

Create a service as shown in the following console steps.

1. In the AWS Proton console, choose **Services**.
2. Choose **Create service**.
3. In the **Choose a service template** page, select a template and choose **Configure**.

   When you *don't* want to use an enabled pipeline, choose a template marked with *Excludes pipeline* for your service.
4. In the **Configure service** page, in the **Service settings** section, enter an **Service name**.
5. (Optional) Enter a description for the service.
6. **In the Service repository settings section:**

   a. For **CodeStar Connection**, choose your connection from the list.

   b. For **Repository ID**, choose the name of your source code repository from the list.

   c. For **Branch name**, choose the name of your source code repository branch from the list.
7. (Optional) In the **Tags** section, choose **Add new tag** and enter a key and value to create a customer managed tag.
8. Choose **Next**.

9. In the **Configure custom settings** page, in the **Service instances** section, in the **New instance** section. You must enter values for the `required` parameters. You can enter values for the `optional` parameters or use the defaults when given.
10. In the **Pipeline inputs** section, you must enter values for the `required` parameters. You can enter values for the `optional` parameters or use the defaults when given.
11. Choose **Next** and review your inputs.
12. Choose **Create**.

    View the service details and status, as well as the AWS managed tags and customer managed tags for your service.
13. In the navigation pane, choose **Services**.

    A new page displays a list of your services along with the status and other service details.

AWS CLI

When you use the AWS CLI, you specify service inputs in a YAML formatted `spec` file, `.aws-proton/service.yaml`, located in your source code directory.

You can use the CLI `get-service-template-minor-version` command to view the schema required and optional parameters that you provide values for in your spec file.

If you want to use a service template that has `pipelineProvisioning: "CUSTOMER_MANAGED"`, *don't* include the `pipeline:` section in your spec and *don't* include `-repository-connection-arn`, `-repository-id`, and `-branch-name` parameters in your `create-service` command.

Create a service with a service pipeline as shown in the following CLI steps.

1. **Set up the service role for the pipeline as shown in the following CLI example command.**

   Command:

```
$ aws proton update-account-settings \
        --pipeline-service-role-arn
 "arn:aws:iam::123456789012:role/AWSProtonServiceRole"
```

2. The following listing shows an example spec, based on the service template schema, that includes the service pipeline and instance inputs.

   Spec:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_required_input: "hello"
  my_sample_pipeline_optional_input: "bye"

instances:
  - name: "acme-network-dev"
    environment: "ENV_NAME"
    spec:
      my_sample_service_instance_required_input: "hi"
      my_sample_service_instance_optional_input: "ho"
```

Create a service with a pipeline as shown in the following CLI example command and response.

Command:

```
$ aws proton create-service \
        --name "MySimpleService" \
        --branch-name "mainline" \
        --template-major-version "1" \
        --template-name "fargate-service" \
        --repository-connection-arn "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111" \
        --repository-id "myorg/myapp" \
        --spec "file://spec.yaml"
```

Response:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService",
        "createdAt": "2020-11-18T19:50:27.460000+00:00",
        "lastModifiedAt": "2020-11-18T19:50:27.460000+00:00",
        "name": "MySimpleService",
```

```
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "myorg/myapp",
        "status": "CREATE_IN_PROGRESS",
        "templateName": "fargate-service"
    }
}
```

Create a service without a service pipeline as shown in the following CLI example command and response.

The following shows an example spec that *doesn't* include service pipeline inputs.

Spec:

```
proton: ServiceSpec

instances:
  - name: "acme-network-dev"
    environment: "ENV_NAME"
    spec:
      my_sample_service_instance_required_input: "hi"
      my_sample_service_instance_optional_input: "ho"
```

To create a service *without* a provisioned service pipeline, you provide the path to a `spec.yaml` and you *don't* include repository parameters as shown in the following CLI example command and response.

Command:

```
$ aws proton create-service \
        --name "MySimpleServiceNoPipeline" \
        --template-major-version "1" \
        --template-name "fargate-service" \
        --spec "file://spec-no-pipeline.yaml"
```

Response:

```
{
    "service": {
```

```
        "arn": "arn:aws:proton:region-id:123456789012:service/
MySimpleServiceNoPipeline",
        "createdAt": "2020-11-18T19:50:27.460000+00:00",
        "lastModifiedAt": "2020-11-18T19:50:27.460000+00:00",
        "name": "MySimpleServiceNoPipeline",
        "status": "CREATE_IN_PROGRESS",
        "templateName": "fargate-service-no-pipeline"
    }
}
```

View service data

You can view and list service detail data using the AWS Proton console or the AWS CLI.

AWS Management Console

List and view service details using the AWS Proton console as shown in the following steps.

1. To view a list of your services, choose **Services** in the navigation pane.
2. To view detail data, choose the name of a service.

   View your service detail data.

AWS CLI

View the details of a service with a service pipeline as shown in the following CLI example command and response.

Command:

```
$ aws proton get-service \
    --name "simple-svc"
```

Response:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc",
        "branchName": "mainline",
```

```
        "createdAt": "2020-11-28T22:40:50.512000+00:00",
        "lastModifiedAt": "2020-11-28T22:44:51.207000+00:00",
        "name": "simple-svc",
        "pipeline": {
            "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/
pipeline/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "createdAt": "2020-11-28T22:40:50.512000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "lastDeploymentAttemptedAt": "2020-11-28T22:40:50.512000+00:00",
            "lastDeploymentSucceededAt": "2020-11-28T22:40:50.512000+00:00",
            "spec": "proton: ServiceSpec\npipeline:\n
 my_sample_pipeline_required_input: hello\n  my_sample_pipeline_optional_input:
 bye\ninstances:\n- name: instance-svc-simple\n  environment: my-simple-
env\n  spec:\n    my_sample_service_instance_required_input: hi\n
 my_sample_service_instance_optional_input: ho\n",
            "templateMajorVersion": "1",
            "templateMinorVersion": "1",
            "templateName": "svc-simple"
        },
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
        "repositoryId": "myorg/myapp",
        "spec": "proton: ServiceSpec\npipeline:\n
 my_sample_pipeline_required_input: hello\n  my_sample_pipeline_optional_input:
 bye\ninstances:\n- name: instance-svc-simple\n  environment: my-simple-
env\n  spec:\n    my_sample_service_instance_required_input: hi\n
 my_sample_service_instance_optional_input: ho\n",
        "status": "ACTIVE",
        "templateName": "svc-simple"
    }
}
```

View the details of a service without a service pipeline as shown in the following CLI example command and response.

Command:

```
$ aws proton get-service \
    --name "simple-svc-no-pipeline"
```

Response:

```
{
```

```
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc-without-
pipeline",
        "createdAt": "2020-11-28T22:40:50.512000+00:00",
        "lastModifiedAt": "2020-11-28T22:44:51.207000+00:00",
        "name": "simple-svc-without-pipeline",
        "spec": "proton: ServiceSpec\ninstances:\n- name: instance-svc-simple\n
 environment: my-simple-env\n  spec:\n    my_sample_service_instance_required_input:
 hi\n    my_sample_service_instance_optional_input: ho\n",
        "status": "ACTIVE",
        "templateName": "svc-simple-no-pipeline"
    }
}
```

Edit a service

You can make the following edits to an AWS Proton service.

- Edit the service description.
- Edit a service by adding and removing service instances.

Edit service description

You can use the console or the AWS CLI to edit a service description.

AWS Management Console

Edit a service using the console as described in the following steps.

In the list of services.

1. In the AWS Proton console, choose **Services**.
2. In the list of services, choose the radio button to the left of the service that you want to update.
3. Choose **Edit**.
4. In the **Configure service** page, fill out the form and choose **Next**.
5. In the **Configure custom settings** page, choose **Next**.
6. Review your edits and choose **Save changes**.

In the service detail page.

1. In the AWS Proton console, choose **Services**.
2. In the list of services, choose the name of the service that you want to edit.
3. In the service detail page, choose **Edit**.
4. In the **Configure service** page, fill out the form and choose **Next**.
5. In the **Configure custom settings** page, fill out the form and choose **Next**.
6. Review your edits and choose **Save changes**.

AWS CLI

Edit a description as shown in the following CLI example command and response.

Command:

```
$ aws proton update-service \
    --name "MySimpleService" \
    --description "Edit by updating description"
```

Response:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService",
        "branchName": "main",
        "createdAt": "2021-03-12T22:39:42.318000+00:00",
        "description": "Edit by updating description",
        "lastModifiedAt": "2021-03-12T22:44:21.975000+00:00",
        "name": "MySimpleService",
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "my-repository/myorg-myapp",
        "status": "ACTIVE",
        "templateName": "fargate-service"
    }
}
```

Edit a service to add or remove service instances

For an AWS Proton service, you can add or delete service instances by submitting an edited spec. The following conditions must be met for a successful request:

- Your service and pipeline aren't already being edited or deleted when you submit the edit request.
- Your edited spec doesn't include edits that modify the service pipeline or edits to existing service instances that *aren't* to be deleted.
- Your edited spec doesn't remove any existing service instance that has an attached component. To delete such a service instance, you should first update the component to detach it from its service instance. For more information about components, see *Components*.

Deletion-failed instances are service instances in the `DELETE_FAILED` state. When you request a service edit, AWS Proton attempts to remove the deletion-failed instances for you, as part of the edit process. If any of your service instances failed to delete, there might still be resources that are associated with the instances, even though they aren't visible from the console or AWS CLI. Check your deletion-failed instance infrastructure resources and clean them up so that AWS Proton can remove them for you.

For the quota of service instances for a service, see AWS Proton quotas. You also must maintain at least 1 service instance for your service after it's created. During the update process, AWS Proton makes a count of the existing service instances and the instances to be added or removed. Deletion-failed instances are included in this count and you must account for them when you edit your `spec`.

Use the console or AWS CLI to add or remove service instances

AWS Management Console

Edit your service to add or remove service instances using the console.

In the AWS Proton console

1. In the navigation pane, choose **Services**.
2. Select the service that you want to edit.
3. Choose **Edit**.

4. (Optional) On the **Configure service** page, edit the service name or description, and then choose **Next**.
5. On the **Configure custom settings** page, choose **Delete** to delete a service instance and choose **Add new instance** to add a service instance and fill out the form.
6. Choose **Next**.
7. Review your update and choose **Save changes**.
8. A modal asks you to verify deletion of service instances. Follow the instructions and choose **Yes, delete**.
9. On the service detail page, view the status details for your service.

AWS CLI

Add and delete service instances with an edited `spec` as shown in the following AWS CLI example commands and responses.

When you use the CLI, your `spec` must *exclude* the service instances to delete and *include* both the service instances to add and the existing service instances that you *haven't* marked for deletion.

The following listing shows the example `spec` before the edit and a list of the service instances deployed by the spec. This spec was used in the previous example for editing a service description.

Spec:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "abc"
  my_sample_pipeline_required_input: "123"

instances:
  - name: "my-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_optional_input: "def"
      my_sample_service_instance_required_input: "456"
  - name: "my-other-instance"
    environment: "simple-env"
```

```
    spec:
      my_sample_service_instance_required_input: "789"
```

The following example CLI `list-service-instances` command and response shows the active instances prior to adding or deleting a service instance.

Command:

```
$ aws proton list-service-instances \
    --service-name "MySimpleService"
```

Response:

```
{
    "serviceInstances": [
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService/
service-instance/my-other-instance",
            "createdAt": "2021-03-12T22:39:42.318000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentName": "simple-env",
            "lastDeploymentAttemptedAt": "2021-03-12T22:39:43.109000+00:00",
            "lastDeploymentSucceededAt": "2021-03-12T22:39:43.109000+00:00",
            "name": "my-other-instance",
            "serviceName": "example-svc",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
            "templateName": "fargate-service"
        },
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService/
service-instance/my-instance",
            "createdAt": "2021-03-12T22:39:42.318000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentName": "simple-env",
            "lastDeploymentAttemptedAt": "2021-03-12T22:39:43.160000+00:00",
            "lastDeploymentSucceededAt": "2021-03-12T22:39:43.160000+00:00",
            "name": "my-instance",
            "serviceName": "example-svc",
            "serviceTemplateArn": "arn:aws:proton:region-id:123456789012:service-
template/fargate-service",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
```

```
            "templateName": "fargate-service"
        }
    ]
}
```

The following listing shows the example edited `spec` used to delete and add an instance. The existing instance named `my-instance` is removed and a new instance named `yet-another-instance` is added.

Spec:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "abc"
  my_sample_pipeline_required_input: "123"

instances:
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
  - name: "yet-another-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

You can use `"${Proton::CURRENT_VAL}"` to indicate which parameter values to preserve from the original `spec`, if the values exist in the `spec`. Use `get-service` to view the original `spec` for a service, as described in [View service data](#).

The following listing shows how you can use `"${Proton::CURRENT_VAL}"` to ensure that your `spec` *doesn't* include parameter values changes for the existing services instances to remain.

Spec:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "${Proton::CURRENT_VAL}"
```

```
  my_sample_pipeline_required_input: "${Proton::CURRENT_VAL}"

instances:
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "${Proton::CURRENT_VAL}"
  - name: "yet-another-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

The next listing shows the CLI command and response to edit the service.

Command:

```
$ aws proton update-service
    --name "MySimpleService" \
    --description "Edit by adding and deleting a service instance" \
    --spec "file://spec.yaml"
```

Response:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService",
        "branchName": "main",
        "createdAt": "2021-03-12T22:39:42.318000+00:00",
        "description": "Edit by adding and deleting a service instance",
        "lastModifiedAt": "2021-03-12T22:55:48.169000+00:00",
        "name": "MySimpleService",
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "my-repository/myorg-myapp",
        "status": "UPDATE_IN_PROGRESS",
        "templateName": "fargate-service"
    }
}
```

The following `list-service-instances` command and response confirms that the existing instance named `my-instance` is removed and a new instance named `yet-another-instance` is added.

Command:

```
$ aws proton list-service-instances \
    --service-name "MySimpleService"
```

Response:

```
{
    "serviceInstances": [
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService/
service-instance/yet-another-instance",
            "createdAt": "2021-03-12T22:39:42.318000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentName": "simple-env",
            "lastDeploymentAttemptedAt": "2021-03-12T22:56:01.565000+00:00",
            "lastDeploymentSucceededAt": "2021-03-12T22:56:01.565000+00:00",
            "name": "yet-another-instance",
            "serviceName": "MySimpleService",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
            "templateName": "fargate-service"
        },
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService/
service-instance/my-other-instance",
            "createdAt": "2021-03-12T22:39:42.318000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentName": "simple-env",
            "lastDeploymentAttemptedAt": "2021-03-12T22:39:43.109000+00:00",
            "lastDeploymentSucceededAt": "2021-03-12T22:39:43.109000+00:00",
            "name": "my-other-instance",
            "serviceName": "MySimpleService",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
            "templateName": "fargate-service"
        }
    ]
}
```

What happens when you add or remove service instances

After you submit a service edit to delete and add service instances, AWS Proton takes the following actions.

- Sets the service to `UPDATE_IN_PROGRESS`.
- If the service has a pipeline, sets its status to `IN_PROGRESS` and blocks pipeline actions.
- Sets any service instances that are to be deleted to `DELETE_IN_PROGRESS`.
- Blocks service actions.
- Blocks actions on service instances that are marked for deletion.
- Creates new service instances.
- Deletes instances that you listed for deletion.
- Attempts to remove deletion-failed instances.
- After additions and deletions are complete, re-provisions the service pipeline (if there is one), sets your service to `ACTIVE` and enables service and pipeline actions.

AWS Proton attempts to re-mediate failure modes as follows.

- If one or more service instances *failed to be created*, AWS Proton tries to de-provision all of the newly created service instances and reverts the `spec` to the previous state. It *doesn't* delete any service instances and it *doesn't* modify the pipeline in any way.
- If one or more service instances *failed to be deleted*, AWS Proton re-provisions the pipeline without the deleted instances. The `spec` is updated to include the added instances and to exclude the instances that were marked for deletion.
- If the *pipeline fails provisioning*, a rollback *isn't* attempted and both the service and pipeline reflect a failed update state.

Tagging and service edits

When you add service instances as part of your service edit, AWS managed tags propagate to and are automatically created for the new instances and provisioned resources. If you create new tags, those tags are only applied to the new instances. Existing service customer managed tags also propagate to the new instances. For more information, see AWS Proton resources and tagging.

Delete a service

You can delete an AWS Proton service, with its instances and pipeline, by using the AWS Proton console or the AWS CLI.

You can't delete a service that has any service instance with an attached component. To delete such a service, you should first update all attached components to detach them from their service instances. For more information about components, see *Components*.

AWS Management Console

Delete a service using the console as described in the following steps.

In the service detail page.

1. In the AWS Proton console, choose **Services**.
2. In the list of services, choose the name of the service that you want to delete.
3. On the service detail page, choose **Actions** and then **Delete**.
4. A modal prompts you to confirm the delete action.
5. Follow the instructions and choose **Yes, delete**.

AWS CLI

Delete a service as shown in the following CLI example command and response.

Command:

```
$ aws proton delete-service \
    --name "simple-svc"
```

Response:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc",
        "branchName": "mainline",
        "createdAt": "2020-11-28T22:40:50.512000+00:00",
        "description": "Edit by updating description",
        "lastModifiedAt": "2020-11-29T00:30:39.248000+00:00",
        "name": "simple-svc",
```

```
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "myorg/myapp",
        "status": "DELETE_IN_PROGRESS",
        "templateName": "fargate-service"
    }
}
```

View service instance data

Learn to view AWS Proton service instance detail data. You can use the console or the AWS CLI.

A service instance belongs to a service. You can only create or delete an instance within the context of service edit, create and delete actions. To learn how to add and remove instances from a service, see Edit a service.

AWS Management Console

List and view service instance details using the AWS Proton console as shown in the following steps.

1. To view a list of your service instances, choose **Services instances** in the navigation pane.
2. To view detail data, choose the name of a service instance.

   View your service instance detail data.

AWS CLI

List and view service instance details as shown in the following CLI example commands and responses.

Command:

```
$ aws proton list-service-instances
```

Response:

```
{
    "serviceInstances": [
```

```
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/
service-instance/instance-one",
            "createdAt": "2020-11-28T22:40:50.512000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentArn": "arn:aws:proton:region-id:123456789012:environment/
simple-env",
            "lastDeploymentAttemptedAt": "2020-11-28T22:40:50.512000+00:00",
            "lastDeploymentSucceededAt": "2020-11-28T22:40:50.512000+00:00",
            "name": "instance-one",
            "serviceName": "simple-svc",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
            "templateName": "fargate-service"
        }
    ]
}
```

Command:

```
$ aws proton get-service-instance \
    --name "instance-one" \
    --service-name "simple-svc"
```

Response:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2020-11-28T22:40:50.512000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "environmentName": "simple-env",
        "lastDeploymentAttemptedAt": "2020-11-28T22:40:50.512000+00:00",
        "lastDeploymentSucceededAt": "2020-11-28T22:40:50.512000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "spec": "proton: ServiceSpec\npipeline:\n
 my_sample_pipeline_optional_input: hello world\n
 my_sample_pipeline_required_input: pipeline up\ninstances:\n- name: instance-one\n
 environment: my-simple-env\n  spec:\n    my_sample_service_instance_optional_input:
 Ola\n    my_sample_service_instance_required_input: Ciao\n",
        "templateMajorVersion": "1",
```

```
        "templateMinorVersion": "0",
        "templateName": "svc-simple"
    }
}
```

Update a service instance

Learn to update an AWS Proton service instance and cancel the update.

A service instance belongs to a service. You can only create or delete an instance within the context of service edit, create and delete actions. To learn how to add and remove instances from a service, see Edit a service.

There are four modes for updating a service instance as described in the following list. When using the AWS CLI, the `deployment-type` field defines the mode. When using the console, these modes map to the **Edit** and the **Update to latest minor version** and **Update to latest major version** actions that drop down from **Actions** in the service instance detail page.

`NONE`

In this mode, a deployment *doesn't* occur. Only the requested metadata parameters are updated.

`CURRENT_VERSION`

In this mode, the service instance is deployed and updated with the new spec that you provide. Only requested parameters are updated. *Don't* include minor or major version parameters when you use this `deployment-type`.

`MINOR_VERSION`

In this mode, the service instance is deployed and updated with the published, recommended (latest) minor version of the current major version in use by default. You can also specify a different minor version of the current major version in use.

`MAJOR_VERSION`

In this mode, the service instance is deployed and updated with the published, recommended (latest) major and minor version of the current template by default. You can also specify

a different major version that is higher than the major version in use and a minor version (optional).

You can attempt to cancel a service instance update deployment if the `deploymentStatus` is `IN_PROGRESS`. AWS Proton attempts to cancel the deployment. Successful cancellation *isn't* guaranteed.

When you cancel an update deployment, AWS Proton attempts to cancel the deployment as listed in the following steps.

- Sets the deployment state to `CANCELLING`.
- Stops the deployment in process and deletes any new resources that were created by the deployment when `IN_PROGRESS`.
- Sets the deployment state to `CANCELLED`.
- Reverts the state of the resource to what it was before the deployment was started.

For more information on cancelling a service instance deployment, see CancelServiceInstanceDeployment in the *AWS Proton API Reference*.

Use the console or AWS CLI to make updates or cancel update deployments.

AWS Management Console

Update a service instance using the console by following these steps.

1. In the AWS Proton console, choose **Service instances** in the navigation pane.
2. In the list of service instances, choose the name of the service instance that you want to update.
3. Choose **Actions** and then choose one of the update options, **Edit** to update spec or **Actions** and then **Update to latest minor version**, or **Update to latest major version**.
4. Fill out each form and choose **Next** until you reach the **Review** page.
5. Review your edits and choose **Update**.

AWS CLI

Update a service instance to a new minor version as shown in the CLI example commands and responses.

When you update your service instance with a modified `spec`, you can use `"${Proton::CURRENT_VAL}"` to indicate which parameter values to preserve from the original `spec`, if the values exist in the `spec`. Use `get-service` to view the original `spec` for a service instance, as described in View service data.

The following example shows how you can use `"${Proton::CURRENT_VAL}"` in a `spec`.

Spec:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "${Proton::CURRENT_VAL}"
  my_sample_pipeline_required_input: "${Proton::CURRENT_VAL}"

instances:
  - name: "my-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_optional_input: "${Proton::CURRENT_VAL}"
      my_sample_service_instance_required_input: "${Proton::CURRENT_VAL}"
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

Command: to update

```
$ aws proton update-service-instance \
    --name "instance-one" \
    --service-name "simple-svc" \
    --spec "file://service-spec.yaml" \
    --template-major-version "1" \
    --template-minor-version "1" \
    --deployment-type "MINOR_VERSION"
```

Response:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
```

```
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "environmentName": "arn:aws:proton:region-id:123456789012:environment/
simple-env",
        "lastDeploymentAttemptedAt": "2021-04-02T21:38:00.823000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:29:59.962000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "svc-simple"
    }
}
```

Command: to get and confirm status

```
$ aws proton get-service-instance \
    --name "instance-one" \
    --service-name "simple-svc"
```

Response:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "environmentName": "simple-env",
        "lastDeploymentAttemptedAt": "2021-04-02T21:38:00.823000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:38:00.823000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def\"\n
      my_sample_service_instance_required_input: \"456\"\n  - name: \"my-
other-instance\"\n    environment: \"kls-simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "svc-simple"
```

```
    }
}
```

AWS Management Console

Cancel a service instance deployment using the console as shown in the following steps.

1. In the AWS Proton console, choose **Service instances** in the navigation pane.
2. In the list of service instances, choose the name of the service instance with the deployment update that you want to cancel.
3. If your update deployment status is **In progress**, in the service instance detail page, choose **Actions** and then **Cancel deployment**.
4. A modal asks you to confirm the cancellation. Choose **Cancel deployment**.
5. Your update deployment status is set to **Cancelling** and then **Cancelled** to complete the cancellation.

AWS CLI

Cancel an IN_PROGRESS service instance deployment update to new minor version 2 as shown in the following CLI example commands and responses.

A wait condition is included in the template used for this example so that the cancellation starts before the update deployment succeeds.

Command: to cancel

```
$ aws proton cancel-service-instance-deployment \
    --service-instance-name "instance-one" \
    --service-name "simple-svc"
```

Response:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "CANCELLING",
```

```
        "environmentName": "simple-env",
        "lastDeploymentAttemptedAt": "2021-04-02T21:45:15.406000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:38:00.823000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "spec": "proton: ServiceSpec\npipeline:\n
 my_sample_pipeline_optional_input: abc\n  my_sample_pipeline_required_input:
 '123'\ninstances:\n- name: my-instance\n  environment: MySimpleEnv
\n  spec:\n    my_sample_service_instance_optional_input: def\n
 my_sample_service_instance_required_input: '456'\n- name: my-other-instance\n
 environment: MySimpleEnv\n  spec:\n    my_sample_service_instance_required_input:
 '789'\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "svc-simple"
    }
}
```

Command: to get and confirm status

```
$ aws proton get-service-instance \
    --name "instance-one" \
    --service-name "simple-svc"
```

Response:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "CANCELLED",
        "deploymentStatusMessage": "User initiated cancellation.",
        "environmentName": "simple-env",
        "lastDeploymentAttemptedAt": "2021-04-02T21:45:15.406000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:38:00.823000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def\"\n
      my_sample_service_instance_required_input: \"456\"\n  - name: \"my-
```

```
other-instance\"\n    environment: \"kls-simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "svc-simple"
    }
}
```

Update a service pipeline

Learn to update an AWS Proton service pipeline and cancel the update.

A service pipeline belongs to a service. You can only create or delete a pipeline within the context of service create and delete actions.

There are four modes for updating a service pipeline as described in the following list. When using the AWS CLI, the `deployment-type` field defines the mode. When you use the console, these modes map to the **Edit pipeline** and **Update to recommended version**.

`NONE`

In this mode, a deployment *doesn't* occur. Only the requested metadata parameters are updated.

`CURRENT_VERSION`

In this mode, the service pipeline is deployed and updated with the new spec that you provide. Only requested parameters are updated. *Don't* include minor or major version parameters when you use this `deployment-type`.

`MINOR_VERSION`

In this mode, the service pipeline is deployed and updated with the published, recommended (latest) minor version of the current major version in use by default. You can also specify a different minor version of the current major version in use.

`MAJOR_VERSION`

In this mode, the service pipeline is deployed and updated with the published, recommended (latest) major and minor version of the current template by default. You can also specify

a different major version that is higher than the major version in use and a minor version (optional).

You can attempt to cancel a service pipeline update deployment if the `deploymentStatus` is `IN_PROGRESS`. AWS Proton attempts to cancel the deployment. Successful cancellation isn't guaranteed.

When you cancel an update deployment, AWS Proton attempts to cancel the deployment as listed in the following steps.

- Sets the deployment state to `CANCELLING`.
- Stops the deployment in process and deletes any new resources that were created by the deployment when `IN_PROGRESS`.
- Sets the deployment state to `CANCELLED`.
- Reverts the state of the resource to what it was before the deployment was started.

For more information on cancelling a service pipeline deployment, see CancelServicePipelineDeployment in the *AWS Proton API Reference*.

Use the console or AWS CLI to make updates or cancel update deployments.

AWS Management Console

Update a service pipeline using the console as described in the following steps.

1. In the AWS Proton console, choose **Services**.
2. In the list of services, choose the name of the service that you want to update the pipeline for.
3. There are two tabs on the service detail page, **Overview** and **Pipeline**. Choose **Pipeline**.
4. If you want to update specs, choose **Edit Pipeline** and fill out each form and choose **Next** until you complete the final form and then choose **Update pipeline**.

   If you want to update to a new version and there's an **information icon** that indicates a new version is available at **Pipeline template**, choose the name of the new template version.

   a. Choose **Update to recommended version**.

b. Fill out each form and choose **Next** until you complete the final form and choose **Update**.

AWS CLI

Update a service pipeline to a new minor version as shown in the following CLI example commands and responses.

When you update your service pipeline with a modified `spec`, you can use `"${Proton::CURRENT_VAL}"` to indicate which parameter values to preserve from the original `spec`, if the values exist in the `spec`. Use `get-service` to view the original `spec` for a service pipeline, as described in View service data.

The following example shows how you can use `"${Proton::CURRENT_VAL}"` in a `spec`.

Spec:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "${Proton::CURRENT_VAL}"
  my_sample_pipeline_required_input: "${Proton::CURRENT_VAL}"

instances:
  - name: "my-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_optional_input: "${Proton::CURRENT_VAL}"
      my_sample_service_instance_required_input: "${Proton::CURRENT_VAL}"
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

Command: to update

```
$ aws proton update-service-pipeline \
    --service-name "simple-svc" \
    --spec "file://service-spec.yaml" \
    --template-major-version "1" \
    --template-minor-version "1" \
```

```
    --deployment-type "MINOR_VERSION"
```

Response:

```
{
    "pipeline": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/pipeline/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-04-02T21:39:28.991000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:29:59.962000+00:00",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"my-instance\"\n    environment: \"MySimpleEnv
\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"MySimpleEnv\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "svc-simple"
    }
}
```

Command: to get and confirm status

```
$ aws proton get-service \
    --name "simple-svc"
```

Response:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc",
        "branchName": "main",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "lastModifiedAt": "2021-04-02T21:30:54.364000+00:00",
        "name": "simple-svc",
        "pipeline": {
            "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/
pipeline",
```

```
            "createdAt": "2021-04-02T21:29:59.962000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "lastDeploymentAttemptedAt": "2021-04-02T21:39:28.991000+00:00",
            "lastDeploymentSucceededAt": "2021-04-02T21:39:28.991000+00:00",
            "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
            "templateMajorVersion": "1",
            "templateMinorVersion": "1",
            "templateName": "svc-simple"
        },
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "repo-name/myorg-myapp",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "status": "ACTIVE",
        "templateName": "svc-simple"
    }
}
```

AWS Management Console

Cancel a service pipeline deployment using the console as shown in the following steps.

1. In the AWS Proton console, choose **Services** in the navigation pane.
2. In the list of services, choose the name of the service that has the pipeline with the deployment update that you want to cancel.
3. In the service detail page, choose the **Pipeline** tab.
4. If your update deployment status is **In progress**, in the service pipeline detail page, choose **Cancel deployment**.

5. A modal asks you to confirm the cancellation. Choose **Cancel deployment**.
6. Your update deployment status is set to **Cancelling** and then **Cancelled** to complete the cancellation.

AWS CLI

Cancel an IN_PROGRESS service pipeline deployment update to minor version 2 as shown in the following CLI example commands and responses.

A wait condition is included in the template used for this example so that the cancellation starts before the update deployment succeeds.

Command: to cancel

```
$ aws proton cancel-service-pipeline-deployment \
    --service-name "simple-svc"
```

Response:

```
{
    "pipeline": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/pipeline",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "CANCELLING",
        "lastDeploymentAttemptedAt": "2021-04-02T22:02:45.095000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:39:28.991000+00:00",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "svc-simple"
    }
}
```

Command: to get and confirm status

```
$ aws proton get-service \
    --name "simple-svc"
```

Response:

```
{
```

```
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc",
        "branchName": "main",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "lastModifiedAt": "2021-04-02T21:30:54.364000+00:00",
        "name": "simple-svc",
        "pipeline": {
            "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/
pipeline",
            "createdAt": "2021-04-02T21:29:59.962000+00:00",
            "deploymentStatus": "CANCELLED",
            "deploymentStatusMessage": "User initiated cancellation.",
            "lastDeploymentAttemptedAt": "2021-04-02T22:02:45.095000+00:00",
            "lastDeploymentSucceededAt": "2021-04-02T21:39:28.991000+00:00",
            "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
            "templateMajorVersion": "1",
            "templateMinorVersion": "1",
            "templateName": "svc-simple"
        },
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "repo-name/myorg-myapp",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "status": "ACTIVE",
        "templateName": "svc-simple"
    }
}
```

AWS Proton components

Components are a type of AWS Proton resource. They add flexibility to service templates. Components provide platform teams with a mechanism to extend core infrastructure patterns, and define safeguards that empower developers to manage aspects of their application infrastructure.

In AWS Proton administrators define standard infrastructure that is used across development teams and applications. However, development teams might need to include additional resources for their specific use cases, like Amazon Simple Queue Service (Amazon SQS) queues or Amazon DynamoDB tables. These application-specific resources might change frequently, particularly during early application development. Maintaining these frequent changes in administrator authored templates might be hard to manage and scale—administrators would need to maintain many more templates without real administrator added value. The alternative—letting application developers author templates for their applications—isn't ideal either, because it takes away administrators' ability to standardize the main architecture components, like AWS Fargate tasks. This is where components come in.

With a component, a developer can add supplemental resources to their application, above and beyond what administrators defined in environment and service templates. The developer then attaches the component to a service instance. AWS Proton provisions infrastructure resources defined by the component just like it provisions resources for environments and service instances.

A component can read service instance inputs and provide outputs to the service instance, for a fully integrated experience. For example, if the component adds an Amazon Simple Storage Service (Amazon S3) bucket for use by a service instance, the component template can take the environment and service instance names into account for naming the bucket. When AWS Proton renders the service template to provision a service instance, the service instance can refer to the bucket and use it.

The components that AWS Proton currently supports are *directly defined components*. You pass the Infrastructure as Code (IaC) file that defines the component's infrastructure directly to the AWS Proton API or console. This is different than an environment or service, where you define IaC in a template bundle and register the bundle as a template resource, then use a template resource to create the environment or service.

> **Note**
>
> Directly defined components allow developers to define extra infrastructure and provision it. AWS Proton provisions all directly defined components running in the same environment using the same AWS Identity and Access Management (IAM) role.

An administrator can control what developers can do with components in two ways:

- *Supported component sources* – An administrator can allow the attachment of components to service instances based on a property of AWS Proton service template versions. By default, developers can't attach components to service instances.

  For more information about this property, see the supportedComponentSources parameter of the CreateServiceTemplateVersion API action in the *AWS Proton API Reference*.

  > **Note**
  >
  > When you use template sync, AWS Proton creates service template versions implicitly when you commit changes to a service template bundle in a repository. In this case, instead of specifying supported component sources during service template version creation, you specify this property in a file associated with each service template major version. For more information, see the section called "Syncing service templates".

- *Component roles* – An administrator can assign a component role to an environment. AWS Proton assumes this role when it provisions infrastructure defined by directly defined component in the environment. Therefore, the component role scopes down the infrastructure that developers can add using directly defined components in the environment. In the absence of the component role, developers can't create directly defined components in the environment.

  For more information about assigning a component role, see the componentRoleArn parameter of the CreateEnvironment API action in the *AWS Proton API Reference*.

  > **Note**
  >
  > Component roles aren't used in Self-managed provisioning environments.

Topics

- How do components compare to other AWS Proton resources?
- Components in the AWS Proton console
- Components in the AWS Proton API and AWS CLI
- Component frequently asked questions
- Component states
- Component infrastructure as code files
- Component AWS CloudFormation example

How do components compare to other AWS Proton resources?

In many ways, components are similar to other AWS Proton resources. Their infrastructure is defined in an IaC template file, authored in either AWS CloudFormation YAML or Terraform HCL format. AWS Proton can provision component infrastructure using either AWS-managed provisioning or self-managed provisioning.

Components are, however, different from other AWS Proton resources in a few ways:

- *Detached state* – Components are designed to be attached to service instances and to extend their infrastructure, but can also be in a *detached* state, in which they aren't attached to any service instance. For more information about component states, see the section called "Component states".
- *No schema* – Components don't have an associated schema like template bundles have. Component inputs are defined by a service. A component can consume inputs when it is attached to a service instance.
- *No customer-managed components* – AWS Proton always provisions component infrastructure for you. There isn't a *bring your own resources* version of components. For more information about customer-managed environments, see the section called "Create".
- *No template resource* – Directly defined components don't have an associated template resource similar to environment and service templates. You provide an IaC template file directly to the component. Similarly, you directly provide a manifest that defines the template language and rendering engine for provisioning the component's infrastructure. You author the template file and the manifest in a way similar to authoring a template bundle. However, with directly defined components, there's no requirement to store IaC files as bundles in particular locations, and you don't create a template resource in AWS Proton out of IaC files.

- *No CodeBuild-based provisioning* – You can't provision directly defined components using your own custom provisioning script, known as *CodeBuild-based provisioning*. For more information, see the section called "CodeBuild provisioning".

Components in the AWS Proton console

Use the AWS Proton console to create, update, view, and use AWS Proton components.

The following console pages are related to components. We include direct links to top level console pages.

- Components – View the list of components in your AWS account. You can create new components, and update or delete existing components. Choose a component name on the list to view its details page.

  Similar lists exist also on the **Environment details** and **Service instance details** pages. These lists show only the components associated with the resource that is being viewed. When you create a component from one of these lists, AWS Proton pre-selects the associated environment on the **Create component** page.
- **Component details** – To view the component details page, choose a component name on the Components list.

  On the details page, view the component details and status, and update or delete the component. View and manage lists of outputs (for example, provisioned resource ARNs), provisioned AWS CloudFormation stacks, and assigned tags.
- Create component – Create a component. Enter the component name and description, choose the associated resources, specify the component source IaC file, and assign tags.
- **Update component** – To update a component, select the component on the Components list, and then, on the **Actions** menu, choose **Update component**. Alternatively, on the **Component details** pages, choose **Update**.

  You can update most of the component's details. You can't update the component name. And you can choose whether or not to redeploy the component after a successful update.
- **Configure environment** – When you create or update an environment, you can specify a **Component role**. This role controls the ability to run directly defined components in the environment and provides permissions for provisioning them.

- **Create new service template version** – When you create a service template version, you can specify **Supported component sources** for the template version. This controls the ability to attach components to service instances of services based on this template version.

Components in the AWS Proton API and AWS CLI

Use the AWS Proton API or the AWS CLI to create, update, view, and use AWS Proton components.

The following API actions directly manage AWS Proton component resources.

- CreateComponent – Create an AWS Proton component.
- DeleteComponent – Delete an AWS Proton component.
- GetComponent – Get detailed data for a component.
- ListComponentOutputs – Get a list of component Infrastructure as Code (IaC) outputs.
- ListComponentProvisionedResources – List provisioned resources for a component with details.
- ListComponents – List components with summary data. You can filter the result list by environment, service, or a single service instance.

The following API actions of other AWS Proton resources have some functionality related to components.

- CreateEnvironment, UpdateEnvironment – Use `componentRoleArn` to specify the Amazon Resource Name (ARN) of the IAM service role that AWS Proton uses when provisioning directly defined components in this environment. It determines the scope of infrastructure that a directly defined component can provision.
- CreateServiceTemplateVersion – Use `supportedComponentSources` to specify supported component sources. Components with supported sources can be attached to service instances based on this service template version.

Component frequently asked questions

What is the lifecycle of a component?

Components can be in an *attached* or *detached* state. They are designed to be attached to a service instance and enhance its infrastructure most of the time. Detached components are in a

transitional state that enables you to delete a component or attach it to another service instance in a controlled and safe way. For more information, see the section called “Component states”.

Why can't I delete my attached components?

Solution: To delete an attached component, update the component to detach it from the service instance, validate service instance stability, and then delete the component.

Why is this required? Attached components provide extra infrastructure that your application needs to perform its runtime functions. The service instance might be using component outputs to detect and use resources of this infrastructure. Deleting the component, thereby removing its infrastructure resources, could be disruptive to the attached service instance.

As an added safety measure, AWS Proton requires that you update the component and detach it from its service instance before you can delete it. You can then validate your service instance to ensure that it continues to deploy and function properly. If you detect an issue, you can quickly reattach the component to the service instance, then work to fix the issue. When you're confident that your service instance is clear of any dependency on the component, you can safely delete the component.

Why can't I change a component's attached service instance directly?

Solution: To change attachment, update the component to detach it from the service instance, validate component and service instance stability, then attach the component to the new service instance.

Why is this required? A component is designed to be attached to a service instance. Your component might use service instance inputs for infrastructure resource naming and configuration. Changing the attached service instance could be disruptive to the component (in addition to possible disruption to the service instance, as described in the previous FAQ, Why can't I delete my attached components?). For example, it might cause renaming, and possibly even replacement, of resources defined in the component's IaC template.

As an added safety measure, AWS Proton requires that you update the component and detach it from its service instance before you can attach it to another service instance. You can then validate the stability of both the component and the service instance before attaching the component to the new service instance.

Component states

AWS Proton components can be in two fundamentally different states:

- *Attached* – The component is attached to a service instance. It defines infrastructure that supports the runtime functionality of the service instance. The component extends the infrastructure defined in environment and service templates with developer-defined infrastructure.

  A typical component is in the attached state throughout most of the useful part of its lifecycle.
- *Detached* – The component is associated with an AWS Proton environment, and isn't attached to any service instance in the environment.

  This is a transitional state for extending the lifetime of a component beyond a single service instance.

The following table provides a top level comparison of the different component states.

	Attached	**Detached**
State's main purpose	To extend the infrastructure of a service instance.	To maintain the component's infrastructure between service instance attachments.
Associated with	A service instance and an environment	An environment
Key specific properties	• Service name • Service instance name • Spec	• Environment name
Can be deleted	✕ No	✓ Yes
Can be updated to another service instance	✕ No	✓ Yes
Can read inputs	✓ Yes	✕ No

A component's main purpose is to be attached to a service instance and extend its infrastructure with additional resources. An attached component can read inputs from the service instance according to the spec. You can't directly delete the component or attach it to a different service instance. You can't delete its service instance or the related service and environment, either. To do any of these things, update the component to detach it from its service instance first.

To maintain the component's infrastructure beyond the lifetime of a single service instance, you update the component and detach it from its service instance by removing the service and service instance names. This detached state is a transitional state. The component has no inputs. Its infrastructure stays provisioned and you can update it. You can delete resources that the component was associated with when it was attached (service instance, service). You can delete the component or update it to be attached to a service instance again.

Component infrastructure as code files

Component infrastructure as code (IaC) files are similar to those for other AWS Proton resources. Learn here about some details that are specific to components. For complete information about authoring IaC files for AWS Proton, see *Template authoring and bundles*.

Using parameters with components

The AWS Proton parameter namespace includes some parameters that a service IaC file can reference to get an associated component's name and outputs. The namespace also includes parameters that a component IaC file can reference to get inputs, outputs, and resource values from the environment, service, and service instance that the component is associated with.

A components doesn't have inputs of its own—it gets its inputs from the service instance it's attached to. A component can also read environment outputs.

For more information about using parameters in component and associated service IaC files, see the section called “Component CloudFormation IaC parameters”. For general information about AWS Proton parameters and a complete reference of the parameter namespace, see the section called “Parameters”.

Authoring robust IaC files

As an administrator, when you create a service template version, you can decide if you want to allow service instances created from the template version to have attached components. See the supportedComponentSources parameter of the CreateServiceTemplateVersion API action in the

AWS Proton API Reference. However, for any future service instance, the person who creates the instance, decides whether or not to attach a component to it, and (in the case of directly defined components) authors the component IaC is typically a different person—a developer using your service template. Therefore, you can't guarantee that a component would be attached to a service instance. You also can't guarantee the existence of specific component output names or the validity and safety of the values of these outputs.

AWS Proton and the Jinja syntax help you work around these issues and author robust service templates that render without failure in the following ways:

- *AWS Proton parameter filters* – When you refer to component output properties, you can use *parameter filters*—modifiers that validate, filter, and format parameter values. For more information and examples, see the section called "CloudFormation parameter filters".
- *Single property default* – When you refer to a single resource or output property of a component, you can guarantee that rendering your service template won't fail by using the `default` filter, with or without a default value. If the component, or a specific output parameter you're referring to, doesn't exist, the default value (or an empty string, if you haven't specified a default value) is rendered instead, and rendering succeeds. For more information, see the section called "Provide default values".

  Examples:

  - `{{ service_instance.components.default.name | default("") }}`
  - `{{ service_instance.components.default.outputs.my-output | default("17") }}`

  > **Note**
  >
  > Do not confuse the `.default` part of the namespace, which designates directly defined components, with the `default` filter, which provides a default value when referenced property doesn't exist.

- *Entire object reference* – When you refer to the entire component, or to the collection of a component's outputs, AWS Proton returns an empty object, `{}`, and therefore guarantees that rendering your service template won't fail. You don't have to use any filter. Be sure to make the reference in a context that can take an empty object, or use an `{{ if .. }}` condition to test for an empty object.

  Examples:

- `{{ service_instance.components.default }}`
- `{{ service_instance.components.default.outputs }}`

Component AWS CloudFormation example

Here is a complete example of an AWS Proton directly defined component and how you can use it in an AWS Proton service. The component provisions an Amazon Simple Storage Service (Amazon S3) bucket and related access policy. The service instance can refer to this bucket and use it. The bucket name is based on the names of the environment, service, service instance, and component, meaning that the bucket is coupled with a specific instance of the component template extending a specific service instance. Developers can create multiple components based on this component template, to provision Amazon S3 buckets for different service instances and functional needs.

The example covers authoring the various required AWS CloudFormation infrastructure as code (IaC) files and creating a required AWS Identity and Access Management (IAM) role. The example groups steps by the owning people roles.

Administrator steps

To enable developers to use components with a service

1. Create an AWS Identity and Access Management (IAM) role that scopes down the resources that directly defined components running in your environment can provision. AWS Proton assumes this role later to provision directly defined components in the environment.

   For this example, use the following policy:

   Example directly defined component role

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DescribeStacks",
```

```
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStackEvents",
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:UpdateStack",
        "cloudformation:DescribeChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket",
        "s3:DeleteBucket",
        "s3:GetBucket",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:GetPolicy",
        "iam:ListPolicyVersions",
        "iam:DeletePolicyVersion"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": "cloudformation.amazonaws.com"
        }
      }
    }
  ]
}
```

2. Provide the role you created in the previous step when you create or update the environment. In the AWS Proton console, specify a **Component role** on the **Configure environment** page. If you're using the AWS Proton API or AWS CLI, specify the `componentRoleArn` of the CreateEnvironment or UpdateEnvironment API actions.
3. Create a service template that refers to a directly defined component attached to the service instance.

The example shows how to write a robust service template that doesn't break if a component isn't attached to the service instance.

Example service CloudFormation IaC file using a component

```
# service/instance_infrastructure/cloudformation.yaml

Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      TaskRoleArn: !Ref TaskRole
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'
          # ...
          {% if service_instance.components.default.outputs | length > 0 %}
          Environment:
            {{ service_instance.components.default.outputs |
                proton_cfn_ecs_task_definition_formatted_env_vars }}
          {% endif %}

  # ...

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
        {{ service_instance.components.default.outputs
            | proton_cfn_iam_policy_arns }}

  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

4. Create a new service template minor version that declares directly defined components as supported.

- **Template bundle in Amazon S3** – In the AWS Proton console, when you create a service template version, for **Supported component sources**, choose **Directly defined**. If you're using the AWS Proton API or AWS CLI, specify `DIRECTLY_DEFINED` in the `supportedComponentSources` parameter of the CreateServiceTemplateVersion or UpdateServiceTemplateVersion API actions.
- **Template sync** – Commit a change to your service template bundle repository, where you specify `DIRECTLY_DEFINED` as an item of `supported_component_sources:` in the `.template-registration.yaml` file in the major version directory. For more information about this file, see the section called "Syncing service templates".

5. Publish the new service template minor version. For more information, see the section called "Publish".
6. Be sure to allow the `proton:CreateComponent` in the IAM role of developers that use this service template.

Developer steps

To use a directly defined component with a service instance

1. Create a service that uses the service template version that the administrator created with component support. Alternatively, update one of your existing service instances to use the latest template version.
2. Write a component IaC template file that provisions an Amazon S3 bucket and a related access policy and exposes these resources as outputs.

   Example component CloudFormation IaC file

```
# cloudformation.yaml

# A component that defines an S3 bucket and a policy for accessing the bucket.
Resources:
  S3Bucket:
    Type: 'AWS::S3::Bucket'
    Properties:
      BucketName: '{{environment.name}}-{{service.name}}-{{service_instance.name}}-
{{component.name}}'
  S3BucketAccessPolicy:
    Type: AWS::IAM::ManagedPolicy
```

```
    Properties:
      PolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Action:
              - 's3:Get*'
              - 's3:List*'
              - 's3:PutObject'
            Resource: !GetAtt S3Bucket.Arn
Outputs:
  BucketName:
    Description: "Bucket to access"
    Value: !GetAtt S3Bucket.Arn
  BucketAccessPolicyArn:
    Value: !Ref S3BucketAccessPolicy
```

3. If you're using the AWS Proton API or AWS CLI, write a manifest file for the component.

   Example directly defined component manifest

```
infrastructure:
  templates:
    - file: "cloudformation.yaml"
      rendering_engine: jinja
      template_language: cloudformation
```

4. Create a directly defined component. AWS Proton assumes the component role that the administrator defined to provision the component.

   In the AWS Proton console, on the Components page, choose **Create component**. For **Component settings**, enter a **Component name** and an optional **Component description**. For **Component attachment**, choose **Attach the component to a service instance.** Select your environment, service, and service instance. For **Component source**, choose **AWS CloudFormation**, and then choose the component IaC file.

   > **Note**
   >
   > You don't need to provide a manifest—the console creates one for you.

If you're using the AWS Proton API or AWS CLI, use the CreateComponent API action. Set a component `name` and optional `description`. Set `environmentName`, `serviceName`, and `serviceInstanceName`. Set `templateSource` and `manifest` to the paths of the files you created.

> **Note**
>
> Specifying an environment name is optional when you specify service and service instance names. The combination of these two is unique in your AWS account, and AWS Proton can determine the environment from the service instance.

5. Update your service instance to redeploy it. AWS Proton uses outputs from your component in the rendered service instance template, to enable your application to use the Amazon S3 bucket that the component provisioned.

Using git repositories with AWS Proton

AWS Proton uses git repositories for a variety of purposes. The following list categorizes the repository types associated with AWS Proton resources. For AWS Proton features that repeatedly connect to your repository to either push content to it or pull content from it, you have to register a *repository link* with AWS Proton in your AWS account. A repository link is a set of properties that AWS Proton can use when it connects to a repository. AWS Proton currently supports GitHub, GitHub Enterprise, and BitBucket.

Developer repositories

Code repository – A repository that developers use to store application code. Used for *code deployment*. AWS Proton doesn't interact directly with this repository. When a developer provisions a service that includes a pipeline, they provide the repository name and branch to read their application code from. AWS Proton passes this information to the pipeline that it provisions.

For more information, see the section called “Create”.

Administrator repositories

Template repository – A repository where administrators store AWS Proton template bundles. Used for *template sync*. When an administrator creates a template in AWS Proton, they can point to a template repository, and AWS Proton keeps the new template in sync with it. When the administrator updates the template bundle in the repository, AWS Proton automatically creates a new template version. Link a template repository to AWS Proton before you can use it for syncing.

For more information, see the section called “Template sync configurations”.

> **Note**
>
> A template repository isn't required if you continue to upload your templates to Amazon Simple Storage Service (Amazon S3) and call the AWS Proton template management APIs to create new templates or template versions.

Self-managed provisioning repositories

Infrastructure repository – A repository that hosts rendered infrastructure templates. Used for *self-managed provisioning* of *resource infrastructure*. When an administrator creates an environment for self-managed provisioning, they provide a repository. AWS Proton submits pull requests (PRs) to this repository to create the infrastructure for the environment and for any service instance deployed to the environment. Link an infrastructure repository to AWS Proton before you can use it for self-managed infrastructure provisioning.

Pipeline repository – A repository used to create pipelines. Used for *self-managed provisioning* of *pipelines*. Using an additional repository to provision pipelines allows AWS Proton to store pipeline configurations independently from any individual environment or service. You only need to provide a single pipeline repository for all your self-managed provisioning services. Link a pipeline repository to AWS Proton before you can use it for self-managed pipeline provisioning.

For more information, see the section called “AWS-managed provisioning”.

Topics

- Create a link to your repository
- View linked repository data
- Delete a repository link

Create a link to your repository

You can create a link to your repository using the console or CLI. When you create a repository link, AWS Proton creates a service linked role for you.

AWS Management Console

Create a link to your repository as shown in the following console steps.

1. In the AWS Proton console, choose **Repositories**.
2. Choose **Create repository**.
3. In the **Link new repository** page, in the **Repository details** section:

    a. Choose your repository provider.

   b. Choose one of your existing connections. If you don't have one, choose **Add a new CodeStar connection** to create a connection, and then go back to the AWS Proton console, refresh the connection list, and choose your new connection.
   c. Choose from your connected source code repositories.
4. *[optional]* In the **Tags** section, choose **Add new tag** one or more times, and enter **Key** and **Value** pairs.
5. Choose **Create repository**.
6. View the detail data for your linked repository.

AWS CLI

Create and register a link to your repository.

Run the following command:

```
$ aws proton create-repository \
    --name myrepos/environments \
    --connection-arn "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111" \
    --provider "GITHUB" \
    --encryption-key "arn:aws:kms:region-id:123456789012:key/bPxRfiCYEXAMPLEKEY" \
    --tags key=mytag1,value=value1 key=mytag2,value=value2
```

The last two parameters, **--encryption-key** and **--tags**, are optional.

Response:

```
{
    "repository": {
        "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
environments",
        "connectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/2ad03b28-a7c4-EXAMPLE11111",
        "encryptionKey": "arn:aws:kms:region-id:123456789012:key/
bPxRfiCYEXAMPLEKEY",
        "name": "myrepos/environments",
        "provider": "GITHUB"
    }
}
```

After you create a repository link, you can view a list of AWS and customer managed tags, as shown in the following example command. AWS Proton automatically generates AWS managed tags for you. You can also modify and create customer managed tags using the AWS CLI. For more information, see AWS Proton resources and tagging.

Command:

```
$ aws proton list-tags-for-resource \
    --resource-arn "arn:aws:proton:region-id:123456789012:repository/github:myrepos/environments"
```

View linked repository data

You can list and view linked repository details using the console or the AWS CLI. For repository links that are used to sync git repositories with AWS Proton, you can retrieve repository sync definition and status using the AWS CLI.

AWS Management Console

List and view linked repository details using the AWS Proton console.

1. To list of your linked repositories, choose **Repositories** in the navigation pane.
2. To view detail data, choose the name of a repository.

AWS CLI

List your linked repositories.

Run the following command:

```
$ aws proton list-repositories
```

Response:

```
{
    "repositories": [
        {
            "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/templates",
```

```
            "name": "myrepos/templates",
            "provider": "GITHUB"
        },
        {
            "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
environments",
            "name": "myrepos/environments",
            "provider": "GITHUB"
        }
    ]
}
```

View the details of a linked repository.

Run the following command:

```
$ aws proton get-repository \
    --name myrepos/templates \
    --provider "GITHUB"
```

Response:

```
{
    "repository": {
        "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
templates",
        "name": "myrepos/templates",
        "provider": "GITHUB"
    }
}
```

List your synced repositories.

The following example lists repositories that you configured for template sync.

Run the following command:

```
$ aws proton list-repository-sync-definitions \
    --branch "main" \
    --repository-name myrepos/templates \
    --repository-provider "GITHUB" \
    --sync-type "TEMPLATE_SYNC"
```

View repository sync status.

The following example retrieves sync status of a template sync repository.

Run the following command:

```
$ aws proton get-repository-sync-status \
    --branch "main" \
    --repository-name myrepos/templates \
    --repository-provider "GITHUB" \
    --sync-type "TEMPLATE_SYNC"
```

Response:

```
{
    "latestSync": {
        "events": [
            {
                "event": "Clone started",
                "time": "2021-11-21T00:26:35.883000+00:00",
                "type": "CLONE_STARTED"
            },
            {
                "event": "Updated configuration",
                "time": "2021-11-21T00:26:41.894000+00:00",
                "type": "CONFIG_UPDATED"
            },
            {
                "event": "Starting syncs for commit 62c03ff86eEXAMPLE1111111",
                "externalId": "62c03ff86eEXAMPLE1111111",
                "time": "2021-11-21T00:26:44.861000+00:00",
                "type": "STARTING_SYNC"
            }
        ],
        "startedAt": "2021-11-21T00:26:29.728000+00:00",
        "status": "SUCCEEDED"
    }
}
```

Delete a repository link

You can delete a repository link by using the console or the AWS CLI.

> **Note**
>
> Deleting a repository link only removes the registered link that AWS Proton has in your AWS account. It does not delete any information from your repository.

AWS Management Console

Delete a repository link using the console.

In the repository detail page.

1. In the AWS Proton console, choose **Repositories**.
2. In the list of repositories, choose the radio button to the left of the repository that you want to delete.
3. Choose **Delete**.
4. A modal prompts you to confirm the Delete action.
5. Follow the instructions and choose **Yes, delete**.

AWS CLI

Delete a repository link.

Run the following command:

```
$ aws proton delete-repository \
    --name myrepos/templates \
    --provider"GITHUB"
```

Response:

```
{
    "repository": {
        "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
templates",
        "name": "myrepos/templates",
        "provider": "GITHUB"
    }
}
```

Monitoring AWS Proton

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Proton and your AWS solutions. The following section describes monitoring tools that you can use with AWS Proton.

Automate AWS Proton with EventBridge

You can monitor AWS Proton events in Amazon EventBridge. EventBridge delivers a stream of real-time data from your own applications, software-as-a-service (SaaS) applications, and AWS services. You can configure events to respond to AWS resource state changes. EventBridge routes this data then to *target* services such as AWS Lambda and Amazon Simple Notification Service. These events are the same as those that appear in Amazon CloudWatch Events. CloudWatch Events delivers a near real-time stream of system events that describe changes in AWS resources. For more information, see What Is Amazon EventBridge? in the *Amazon EventBridge User Guide*.

Use EventBridge to be notified of state changes in the AWS Proton provisioning workflows.

Event types

Events are composed of rules that include an event pattern and targets. You configure a rule by choosing event pattern and target objects:

Event pattern

Each rule is expressed as an event pattern with the source and type of events to monitor and the event targets. To monitor events, you create a rule with the service that you're monitoring as the event source. For example, you can create a rule with an event pattern that uses AWS Proton as an event source to trigger a rule when there are changes in a deployment state.

Targets

The rule receives a selected service as the event target. You can set up a target service to send notifications, capture state information, take corrective action, initiate events, or take other actions.

Event objects contain standard fields of ID, account, AWS Region, detail-type, source, version, resource, time (optional). The detail field is a nested object containing custom fields for the event.

AWS Proton events are emitted on a best effort basis. Best effort delivery means that the service attempts to send all events to EventBridge, but in some rare cases an event might not be delivered.

For each AWS Proton resource that can emit events, the following table lists the detail-type value, detail fields, and (where available) a reference to a list of values for the `status` and `previousStatus` detail fields. When a resource is deleted, the `status` detail field value is `DELETED`.

Resource	Detail-type value	Detail fields

`EnvironmentTemplate`	AWS Proton Environment Template Status Change	`name`
		`status`
		`previousStatus`

`EnvironmentTemplateVersion`	AWS Proton Environment Template Version Status Change	`name`
		`majorVersion`
		`minorVersion`
		`status`
		`previousStatus`
		status values

`ServiceTemplate`	AWS Proton Service Template Status Change	`name`
		`status`
		`previousStatus`

`ServiceTemplateVersion`	AWS Proton Service Template Version Status Change	`name`

Resource	Detail-type value	Detail fields
		majorVersion minorVersion status previousStatus status values
Environment	AWS Proton Environment Status Change	name status previousStatus
Service	AWS Proton Service Status Change	name status previousStatus status values
ServiceInstance	AWS Proton Service Instance Status Change	name serviceName status previousStatus

Resource	Detail-type value	Detail fields
ServicePipeline	AWS Proton Service Pipeline Status Change	serviceName status previousStatus
EnvironmentAccountConnection	AWS Proton Environment Account Connection Status Change	id status previousStatus status values
Component	AWS Proton Component Status Change	name status previousStatus

AWS Proton event examples

The following examples show the ways that AWS Proton can send events to EventBridge.

Service template

```
{
    "source": "aws.proton",
    "detail-type": ["AWS Proton Service Template Status Change"],
    "time": "2021-03-22T23:21:40.734Z",
    "resources": ["arn:aws:proton:region_id:123456789012:service-template/sample-
service-template-name"],
    "detail": {
        "name": "sample-service-template-name",
        "status": "PUBLISHED",
        "previousStatus": "DRAFT"
```

```
    }
}
```

Service template version

```
{
    "source": "aws.proton",
    "detail-type": ["AWS Proton Service Template Version Status Change"],
    "time": "2021-03-22T23:21:40.734Z",
    "resources": ["arn:aws:proton:region_id:123456789012:service-template/sample-
service-template-name:1.0"],
    "detail": {
        "name": "sample-service-template-name",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "REGISTRATION_FAILED",
        "previousStatus": "REGISTRATION_IN_PROGRESS"
    }
}
```

Environment

```
{
    "source": "aws.proton",
    "detail-type": ["AWS Proton Environment Status Change"],
    "time": "2021-03-22T23:21:40.734Z",
    "resources": ["arn:aws:proton:region_id:123456789012:environment/sample-
environment"],
    "detail": {
        "name": "sample-environment",
        "status": "DELETE_FAILED",
        "previousStatus": "DELETE_IN_PROGRESS"
    }
}
```

EventBridgeTutorial: Send Amazon Simple Notification Service alerts for AWS Proton service status changes

In this tutorial, you use an AWS Proton pre-configured *event rule* that captures status changes for your AWS Proton service. EventBridge sends the status changes to an Amazon SNS topic. You

subscribe to the topic and Amazon SNS sends you status change emails for your AWS Proton service.

Prerequisites

You have an existing AWS Proton service with an `Active` status. As part of this tutorial, you can add service instances to this service, and then delete the instances.

If you need to create an AWS Proton service, see *Getting started*. For more information, see *AWS Proton quotas* and the section called "Edit".

Step 1: Create and subscribe to an Amazon SNS topic

Create an Amazon SNS topic to serve as an *event target* for the *event rule* that you create in Step 2.

Create an Amazon SNS topic

1. Log in and open the Amazon SNS console.
2. In the navigation pane, choose **Topics**, **Create topic**.
3. In **Create topic** page:
   a. Choose **Type Standard**.
   b. For **Name**, enter **`tutorial-service-status-change`** and choose **Create topic**.
4. In the **tutorial-service-status-change** detail page, choose **Create subscription**.
5. In the **Create subscription** page:
   a. For **Protocol**, choose **Email**.
   b. For **Endpoint**, enter an email address that you currently have access to and choose **Create subscription**.
6. Check your email account and wait to receive a subscription confirmation email message. When you receive it, open it and choose **Confirm subscription**.

Step 2: Register an event rule

Register an *event rule* that captures status changes for your AWS Proton service. For more information, see Prerequisites.

Create an event rule.

1. Open the Amazon EventBridge console.
2. In the navigation pane, choose **Events**, **Rules**.
3. In the **Rules** page, in the **Rules** section, choose **Create rule**.
4. In the **Create rule** page:
   a. In the **Name and description** section, for **Name**, enter **`tutorial-rule`**.
   b. In the **Define pattern** section, choose **Event pattern**.
      i. For **Event matching pattern**, choose **Pre-defined by service**.
      ii. For **Service provider**, choose **AWS**.
      iii. For **Service name**, choose **AWS Proton**.
      iv. For **Event type**, choose **AWS Proton Service Status Change**.

      The **Event pattern** appears in a text editor.
      v. Open the AWS Proton console.
      vi. In the navigation pane, choose **Services**.
      vii. In **Services** page, choose the name of your AWS Proton service.
      viii. In **Service details** page, copy the service Amazon Resource Name (ARN).
      ix. Navigate back to the *EventBridge console* and your tutorial rule and choose **Edit** at the text editor.
      x. In the text editor, for `"resources":`, enter the service ARN that you copied in step viii.

```
{
    "source": ["aws.proton"],
    "detail-type": ["AWS Proton Service Status Change"],
    "resources": ["arn:aws:proton:region-id:123456789012:service/your-
service"]
}
```

      xi. **Save** the event pattern.
   c. In the **Select targets** section:
      i. For **Target**, choose **SNS topic**.

ii. For **Topic**, choose **tutorial-service-status-change**.

d. Choose **Create**.

Step 3: Test your event rule

Verify that your *event rule* is working by adding an instance to your AWS Proton service.

1. Switch to the AWS Proton console.
2. In the navigation pane, choose **Services**.
3. In **Services** page, choose the name of your service.
4. In **Service details** page, choose **Edit**.
5. In **Configure service** page, choose **Next**.
6. In **Configure custom settings** page, in the **Service instances** section, choose **Add new instance**.
7. Complete the form for your **New instance**:

   a. Enter a **Name** for your new instance.

   b. Select the *same compatible environments* that you chose for your existing instances.

   c. Enter values for the required inputs.

   d. Choose **Next**.
8. Review your inputs and choose **Update**.
9. After the **Service status** is `Active`, check your email to verify you received AWS notifications that give status updates.

```
{
    "version": "0",
    "id": "af76c382-2b3c-7a0a-cf01-936dff228276",
    "detail-type": "AWS Proton Service Status Change",
    "source": "aws.proton",
    "account": "123456789012",
    "time": "2021-06-29T20:40:16Z",
    "region": "region-id",
    "resources": ["arn:aws:proton:region-id:123456789012:service/your-service"],
    "detail": {
        "previousStatus": "ACTIVE",
        "status": "UPDATE_IN_PROGRESS",
```

```
        "name": "your-service"
    }
}
```

```
{
    "version": "0",
    "id": "87131e29-ad95-bda2-cd30-0ce825dfb0cd",
    "detail-type": "AWS Proton Service Status Change",
    "source": "aws.proton",
    "account": "123456789012",
    "time": "2021-06-29T20:42:27Z",
    "region": "region-id",
    "resources": ["arn:aws:proton:region-id:123456789012:service/your-service"],
    "detail": {
        "previousStatus": "UPDATE_IN_PROGRESS",
        "status": "ACTIVE",
        "name": "your-service"
    }
}
```

Step 4: Clean up

Delete your Amazon SNS topic and subscription and delete your EventBridge rule.

Delete your Amazon SNS topic and subscription.

1. Navigate to the Amazon SNS console.
2. In the navigation panel, choose **Subscriptions**.
3. In the **Subscriptions** page, choose the subscription that you made to the topic named `tutorial-service-status-change` and then choose **Delete**.
4. In the navigation panel, choose **Topics**.
5. In the **Topics** page, choose the topic named `tutorial-service-status-change` and then choose **Delete**.
6. A modal prompts you to verify the deletion. Follow the instructions and choose **Delete**.

Delete your EventBridge rule.

1. Navigate to the Amazon EventBridge console.

2. In the navigation pane, choose **Events**, **Rules**.
3. In the **Rules** page, choose the rule named `tutorial-rule` and then choose **Delete**.
4. A modal prompts you to verify the deletion. Choose **Delete**.

Delete the added service instance.

1. Navigate to the AWS Proton console.
2. In the navigation pane, choose **Services**.
3. In the **Services** page, choose the name of your service.
4. In the **Service** detail page, choose **Edit** and then **Next**.
5. In **Configure custom settings** page, in the **Service instances** section, choose **Delete** for the service instance that you created as part of this tutorial and then choose **Next**.
6. Review your inputs and choose **Update**.
7. A modal prompts you to verify the deletion. Follow the instructions and choose **Yes, delete**.

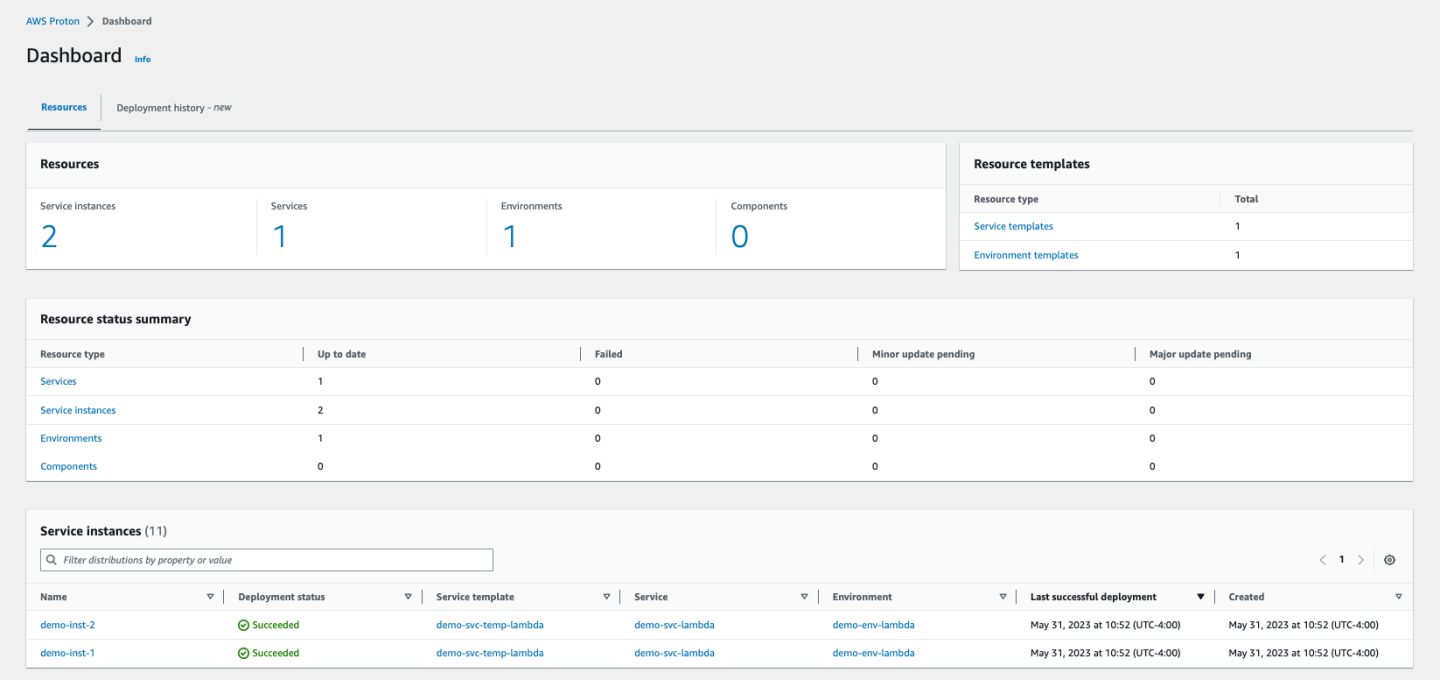
Keep infrastructure up to date with the AWS Proton dashboard

The AWS Proton dashboard provides a summary of AWS Proton resources in your AWS account, with a particular focus on *staleness*—how updated deployed resources are. A deployed resource is up to date when it uses the recommended version of its associated template. An out-of-date deployed resource might need a major or minor template version update.

View the dashboard in the AWS Proton console

To view the AWS Proton dashboard, open the AWS Proton console, and then, in the navigation pane, choose **Dashboard**.

Resources

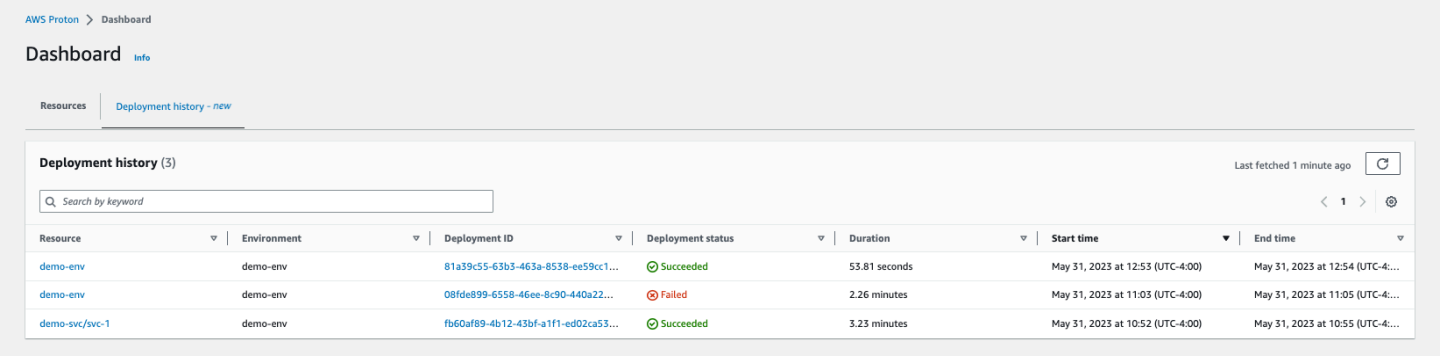


The first tab of the dashboard displays counts of all resources in your account. The resources tab shows the number of your service instances, services, environments, and components, as well as your resource templates. It also breaks down resource counts for each deployed resource type by the status of resources of that type. A service instance table shows details of each service instance—its deployment status, the AWS Proton resources that it's associated with, the updates that are available to it, and some time stamps.

You can filter the service instance list by any table property. For example, you can filter to see service instances with deployments within a specific time window, or service instances that are out of date relative to major or minor version recommendations.

Choose a service instance name to navigate to the service instance detail page, where you can act to make appropriate version updates. Choose any other AWS Proton resource name to navigate to its detail page, or choose a resource type to navigate to the respective resource list.

Deployment history



The deployment history tab lets you see details about your deployments. In the deployment history table, you can keep track of the deployment status, as well as environment and deployment ID. You can choose the resource name or the deployment ID to see even more details, such as a deployment status message and resource outputs. The table also allows you to filter on any table property.

Security in AWS Proton

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are 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 Proton, 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 Proton. The following topics show you how to configure AWS Proton to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS Proton resources.

Topics

- Identity and Access Management for AWS Proton
- Configuration and vulnerability analysis in AWS Proton
- Data protection in AWS Proton
- Infrastructure security in AWS Proton
- Logging and monitoring in AWS Proton
- Resilience in AWS Proton
- Security best practices for AWS Proton
- Cross-service confused deputy prevention
- CodeBuild provisioning custom Amazon VPC support

Identity and Access Management for AWS Proton

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS Proton resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How AWS Proton works with IAM
- Policy examples for AWS Proton
- AWS managed policies for AWS Proton
- Using service-linked roles for AWS Proton
- Troubleshooting AWS Proton identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS Proton.

Service user – If you use the AWS Proton service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS Proton features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS Proton, see Troubleshooting AWS Proton identity and access.

Service administrator – If you're in charge of AWS Proton resources at your company, you probably have full access to AWS Proton. It's your job to determine which AWS Proton features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS Proton, see How AWS Proton works with IAM.

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 Proton. To view example AWS Proton identity-based policies that you can use in IAM, see Identity-based policy examples for AWS Proton.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For

the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

- **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A

user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS Proton works with IAM

Before you use IAM to manage access to AWS Proton, learn what IAM features are available to use with AWS Proton.

IAM features you can use with AWS Proton

IAM feature	AWS Proton support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	Yes

To get a high-level view of how AWS Proton and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for AWS Proton

Supports identity-based policies	Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for AWS Proton

To view examples of AWS Proton identity-based policies, see Identity-based policy examples for AWS Proton.

Resource-based policies within AWS Proton

Supports resource-based policies	No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for AWS Proton

Supports policy actions	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of AWS Proton actions, see Actions defined by AWS Proton in the *Service Authorization Reference*.

Policy actions in AWS Proton use the following prefix before the action:

```
proton
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "proton:action1",
      "proton:action2"
         ]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `List`, include the following action:

```
"Action": "proton:List*"
```

To view examples of AWS Proton identity-based policies, see Identity-based policy examples for AWS Proton.

Policy resources for AWS Proton

Supports policy resources	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of AWS Proton resource types and their ARNs, see Resources defined by AWS Proton in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by AWS Proton.

To view examples of AWS Proton identity-based policies, see Identity-based policy examples for AWS Proton.

Policy condition keys for AWS Proton

Supports service-specific policy condition keys	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of AWS Proton condition keys, see Condition keys for AWS Proton in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by AWS Proton.

To view an example condition-key-based policy for limiting access to a resource, see Condition-key based policy examples for AWS Proton.

Access control lists (ACLs) in AWS Proton

Supports ACLs	No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Access control lists (ACLs) are lists of grantees that you can attach to resources. They grant accounts permissions to access the resource to which they are attached.

Attribute-based access control (ABAC) with AWS Proton

Supports ABAC (tags in policies)	Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

For more information about tagging AWS Proton resources, see AWS Proton resources and tagging.

Using Temporary credentials with AWS Proton

Supports temporary credentials	Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for AWS Proton

Supports forward access sessions (FAS)	Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for AWS Proton

Supports service roles	Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

For more information, see AWS Proton IAM service role policy examples.

> **Warning**
>
> Changing the permissions for a service role might break AWS Proton functionality. Edit service roles only when AWS Proton provides guidance to do so.

Service-linked roles for AWS Proton

Supports service-linked roles	Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Policy examples for AWS Proton

Find AWS Proton IAM policy examples in the following sections.

Topics

- Identity-based policy examples for AWS Proton
- AWS Proton IAM service role policy examples
- Condition-key based policy examples for AWS Proton

Identity-based policy examples for AWS Proton

By default, users and roles don't have permission to create or modify AWS Proton resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Creating IAM policies in the *IAM User Guide*.

For details about actions and resource types defined by AWS Proton, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for AWS Proton in the *Service Authorization Reference*.

Topics

- Policy best practices
- Links to Identity-based policy examples for AWS Proton

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS Proton resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We

recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.

- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Links to Identity-based policy examples for AWS Proton

Links to example identity-based policy examples for AWS Proton

- AWS managed policies for AWS Proton
- AWS Proton IAM service role policy examples
- Condition-key based policy examples for AWS Proton

AWS Proton IAM service role policy examples

Administrators own and manage the resources that AWS Proton creates as defined by the environment and service templates. They attach IAM service roles to their account that permit AWS Proton to create resources on their behalf. Administrators supply the IAM roles and AWS Key Management Service keys for resources that are later owned and managed by developers when AWS Proton deploys their application as an AWS Proton service in an AWS Proton environment. For more information about AWS KMS and data encryption, see Data protection in AWS Proton.

A service role is an Amazon Web Services (IAM) role that allows AWS Proton to make calls to resources on your behalf. If you specify a service role, AWS Proton uses that role's credentials. Use a service role to explicitly specify the actions that AWS Proton can perform.

You create the service role and its permission policy with the IAM service. For more information about creating a service role, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

AWS Proton service role for provisioning using AWS CloudFormation

As a member of the platform team, you can as an administrator create an AWS Proton service role and provide it to AWS Proton when you create an environment as the environment's CloudFormation service role (the `protonServiceRoleArn` parameter of the CreateEnvironment API action). This role allows AWS Proton to make API calls to other services on your behalf when the environment or any of the service instances running in it use AWS-managed provisioning and AWS CloudFormation to provision infrastructure.

We recommend that you use the following IAM role and trust policy for your AWS Proton service role. When you use the AWS Proton console to create an environment and choose to create a new role, this is the policy that AWS Proton adds to the service role it creates for you. When scoping down permission on this policy, keep in mind that AWS Proton fails on `Access Denied` errors.

> **Important**
>
> Be aware that the policies shown in the following examples grant administrator privileges to anyone that can register a template to your account. Because we don't know which resources you will define in your AWS Proton templates, these policies have broad permissions. We recommend that you scope down the permissions to the specific resources that will be deployed in your environments.

AWS Proton service role policy example for AWS CloudFormation

Replace *123456789012* with your AWS account ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:CreateChangeSet",
        "cloudformation:CreateStack",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DeleteStack",
        "cloudformation:DescribeChangeSet",
        "cloudformation:DescribeStackDriftDetectionStatus",
        "cloudformation:DescribeStackEvents",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStacks",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources",
        "cloudformation:UpdateStack"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "NotAction": [
        "organizations:*",
        "account:*"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": [
            "cloudformation.amazonaws.com"
          ]
        }
      }
    },
```

```
    {
      "Effect": "Allow",
      "Action": [
        "organizations:DescribeOrganization",
        "account:ListRegions"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": [
            "cloudformation.amazonaws.com"
          ]
        }
      }
    }
  ]
}
```

AWS Proton service trust policy

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ServiceTrustRelationshipWithConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "proton.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws::proton:*:123456789012:environment/*"
      }
    }
  }
}
```

Scoped down AWS-managed provisioning service role policy

The following is an example of a scoped down AWS Proton service role policy that you can use if you only need AWS Proton services to provision S3 resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:CreateChangeSet",
        "cloudformation:CreateStack",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DeleteStack",
        "cloudformation:DescribeChangeSet",
        "cloudformation:DescribeStackDriftDetectionStatus",
        "cloudformation:DescribeStackEvents",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStacks",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources",
        "cloudformation:UpdateStack"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": [
            "cloudformation.amazonaws.com"
          ]
        }
      }
    }
```

```
  ]
}
```

AWS Proton service role for CodeBuild provisioning

As a member of the platform team, you can as an administrator create an AWS Proton service role and provide it to AWS Proton when you create an environment as the environment's CodeBuild service role (the `codebuildRoleArn` parameter of the CreateEnvironment API action). This role allows AWS Proton to make API calls to other services on your behalf when the environment or any of the service instances running in it use CodeBuild provisioning to provision infrastructure.

When you use the AWS Proton console to create an environment and choose to create a new role, AWS Proton adds a policy with administrator privileges to the service role it creates for you. When you create your own role and scope down permissions, keep in mind that AWS Proton fails on `Access Denied` errors.

> **Important**
>
> Be aware that the policies that AWS Proton attaches to roles that it creates for you grant administrator privileges to anyone that can register a template to your account. Because we don't know which resources you will define in your AWS Proton templates, these policies have broad permissions. We recommend that you scope down the permissions to the specific resources that will be deployed in your environments.

AWS Proton service role policy example for CodeBuild

The following example provides permissions for CodeBuild to provision resources using the AWS Cloud Development Kit (AWS CDK).

Replace *123456789012* with your AWS account ID.

```
{
 "Version": "2012-10-17",
 "Statement": [
   {
     "Action": [
       "logs:CreateLogStream",
       "logs:CreateLogGroup",
       "logs:PutLogEvents"
```

```
      ],
      "Resource": [
        "arn:aws:logs:us-east-1:123456789012:log-group:/aws/codebuild/AWSProton-
 Shell-*",
        "arn:aws:logs:us-east-1:123456789012:log-group:/aws/codebuild/AWSProton-
 Shell-*:*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": "proton:NotifyResourceDeploymentStatusChange",
      "Resource": "arn:aws:proton:us-east-1:123456789012:*",
      "Effect": "Allow"
    },
    {
      "Action": "sts:AssumeRole",
      "Resource": [
        "arn:aws:iam::123456789012:role/cdk-*-deploy-role-*",
        "arn:aws:iam::123456789012:role/cdk-*-file-publishing-role-*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

AWS Proton CodeBuild trust policy

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "CodeBuildTrustRelationshipWithConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "codebuild.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws::proton:*:123456789012:environment/*"
      }
```

```
    }
  }
}
```

AWS Proton pipeline service roles

To provision service pipelines, AWS Proton needs permissions to make API calls to other services. The required service roles are similar to the service roles you provide when you create environments. However, the roles for creating pipelines are shared among all services in your AWS account, and you provide these roles as **Account settings** in the console, or through the UpdateAccountSettings API action.

When you use the AWS Proton console to update account settings and choose to create a new role for either the AWS CloudFormation or the CodeBuild service roles, the policies that AWS Proton adds to the service roles it creates for you are the same as the policies described in the previous sections, AWS-managed provisioning role and CodeBuild provisioning role. When scoping down permission on this policy, keep in mind that AWS Proton fails on `Access Denied` errors.

> **Important**
>
> Be aware that the example policies in the previous sections grant administrator privileges to anyone that can register a template to your account. Because we don't know which resources you will define in your AWS Proton templates, these policies have broad permissions. We recommend that you scope down the permissions to the specific resources that will be deployed in your pipelines.

AWS Proton component role

As a member of the platform team, you can as an administrator create an AWS Proton service role and provide it to AWS Proton when you create an environment as the environment's CloudFormation component role (the `componentRoleArn` parameter of the CreateEnvironment API action). This role scopes down the infrastructure that directly defined components can provision. For more information about components, see *Components*.

The following example policy supports creating a directly defined component that provisions an Amazon Simple Storage Service (Amazon S3) bucket and a related access policy.

AWS Proton component role policy example

Replace *123456789012* with your AWS account ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DescribeStacks",
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStackEvents",
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:UpdateStack",
        "cloudformation:DescribeChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket",
        "s3:DeleteBucket",
        "s3:GetBucket",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:GetPolicy",
        "iam:ListPolicyVersions",
        "iam:DeletePolicyVersion"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": "cloudformation.amazonaws.com"
```

```
        }
      }
    }
  ]
}
```

AWS Proton component trust policy

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ServiceTrustRelationshipWithConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "proton.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws::proton:*:123456789012:environment/*"
      }
    }
  }
}
```

Condition-key based policy examples for AWS Proton

The following example IAM policy denies access to AWS Proton actions that match the templates specified in the `Condition` block. Note that these condition keys are only supported by the actions listed at Actions, resources, and condition keys for AWS Proton. To manage permissions on other actions, such as `DeleteEnvironmentTemplate`, you must use Resource-level access control.

Example policy that denies AWS Proton template actions on a specific templates:

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

```
            "Effect": "Deny",
            "Action": ["proton:*"],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "proton:EnvironmentTemplate":
 ["arn:aws:proton:region_id:123456789012:environment-template/my-environment-template"]
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": ["proton:*"],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "proton:ServiceTemplate":
 ["arn:aws:proton:region_id:123456789012:service-template/my-service-template"]
                }
            }
        }
    ]
}
```

In the next example policy, the first Resource-level statement denies access to AWS Proton template actions, other than `ListServiceTemplates`, that match the service template listed in the `Resource` block. The second statement denies access to AWS Proton actions that match the template listed in the `Condition` block.

Example policy that denies AWS Proton actions that match a specific template:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "proton:*"
            ],
            "Resource": "arn:aws:region_id:123456789012:service-template/my-service-
template"
        },
        {
```

```
            "Effect": "Deny",
            "Action": [
                "proton:*"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "proton:ServiceTemplate": [
                        "arn:aws:proton:region_id:123456789012:service-template/my-
service-template"
                    ]
                }
            }
        }
    ]
}
```

The final policy example allows developer AWS Proton actions that match the specific service template listed in the `Condition` block.

Example policy to allow AWS Proton developer actions that match a specific template:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "proton:ListServiceTemplates",
                "proton:ListServiceTemplateVersions",
                "proton:ListServices",
                "proton:ListServiceInstances",
                "proton:ListEnvironments",
                "proton:GetServiceTemplate",
                "proton:GetServiceTemplateVersion",
                "proton:GetService",
                "proton:GetServiceInstance",
                "proton:GetEnvironment",
                "proton:CreateService",
                "proton:UpdateService",
                "proton:UpdateServiceInstance",
                "proton:UpdateServicePipeline",
                "proton:DeleteService",
```

```
                "codestar-connections:ListConnections"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "proton:ServiceTemplate":
 "arn:aws:proton:region_id:123456789012:service-template/my-service-template"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "codestar-connections:PassConnection"
            ],
            "Resource": "arn:aws:codestar-connections:*:*:connection/*",
            "Condition": {
                "StringEquals": {
                    "codestar-connections:PassedToService": "proton.amazonaws.com"
                }
            }
        }

    ]
}
```

AWS managed policies for AWS Proton

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS Proton provides managed IAM policies and trust relationships that you can attach to users, groups, or roles that allow differing levels of control over resources and API operations. You can apply these policies directly, or you can use them as starting points for creating your own policies.

The following trust relationship is used for each of the AWS Proton managed policies.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ExampleTrustRelationshipWithProtonConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "proton.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws::proton:*:123456789012:environment/*"
      }
    }
  }
}
```

AWS managed policy: AWSProtonFullAccess

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

This policy grants administrative permissions that allow full access to AWS Proton actions and limited access to other AWS service actions that AWS Proton depends on.

The policy includes the following key action namespaces:

- `proton` – Allows administrators full access to AWS Proton APIs.
- `iam` – Allows administrators to pass roles to AWS Proton. This is required so that AWS Proton can make API calls to other services on the administrator's behalf.
- `kms` – Allows administrators to add a grant to a customer managed key.
- `codeconnections` – Allows administrators to list and pass codeconnections so they can be used by AWS Proton.

Permissions details

This policy includes the following permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ProtonPermissions",
            "Effect": "Allow",
            "Action": [
                "proton:*",
                "codestar-connections:ListConnections",
                "kms:ListAliases",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CreateGrantPermissions",
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": "proton.*.amazonaws.com"
                }
            }
        },
        {
            "Sid": "PassRolePermissions",
            "Effect": "Allow",
```

```
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "proton.amazonaws.com"
                }
            }
        },
        {
            "Sid": "CreateServiceLinkedRolePermissions",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
sync.proton.amazonaws.com/AWSServiceRoleForProtonSync",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "sync.proton.amazonaws.com"
                }
            }
        },
        {
            "Sid": "CodeStarConnectionsPermissions",
            "Effect": "Allow",
            "Action": [
                "codestar-connections:PassConnection"
            ],
            "Resource": [
                "arn:aws:codestar-connections:*:*:connection/*",
                "arn:aws:codeconnections:*:*:connection/*"
            ],
            "Condition": {
                "StringEquals": {
                    "codestar-connections:PassedToService": "proton.amazonaws.com"
                }
            }
        },
        {
            "Sid": "CodeConnectionsPermissions",
            "Effect": "Allow",
            "Action": [
                "codeconnections:PassConnection"
            ],
```

```
            "Resource": [
                "arn:aws:codestar-connections:*:*:connection/*",
                "arn:aws:codeconnections:*:*:connection/*"
            ],
            "Condition": {
                "StringEquals": {
                    "codeconnections:PassedToService": "proton.amazonaws.com"
                }
            }
        }
    ]
}
```

AWS managed policy: AWSProtonDeveloperAccess

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

This policy grants permissions that allow limited access to AWS Proton actions and to other AWS actions that AWS Proton depends on. The scope of these permissions is designed to support the role of a developer who creates and deploys AWS Proton services.

This policy doesn't provide access to AWS Proton template and environment *create, delete and update* APIs. If developers need even more limited permissions than what this policy provides, we recommend creating a custom policy that is scoped down to grant the least privilege.

The policy includes the following key action namespaces:

- `proton` – Allows contributors access to a limited set of AWS Proton APIs.
- `codeconnections` – Allows contributors to list and pass codeconnections so they can be used by AWS Proton.

Permissions details

This policy includes the following permissions.

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

```
            "Action": [
                "codecommit:ListRepositories",
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineExecution",
                "codepipeline:GetPipelineState",
                "codepipeline:ListPipelineExecutions",
                "codepipeline:ListPipelines",
                "codestar-connections:ListConnections",
                "codestar-connections:UseConnection",
                "proton:CancelServiceInstanceDeployment",
                "proton:CancelServicePipelineDeployment",
                "proton:CreateService",
                "proton:DeleteService",
                "proton:GetAccountRoles",
                "proton:GetAccountSettings",
                "proton:GetEnvironment",
                "proton:GetEnvironmentAccountConnection",
                "proton:GetEnvironmentTemplate",
                "proton:GetEnvironmentTemplateMajorVersion",
                "proton:GetEnvironmentTemplateMinorVersion",
                "proton:GetEnvironmentTemplateVersion",
                "proton:GetRepository",
                "proton:GetRepositorySyncStatus",
                "proton:GetResourcesSummary",
                "proton:GetService",
                "proton:GetServiceInstance",
                "proton:GetServiceTemplate",
                "proton:GetServiceTemplateMajorVersion",
                "proton:GetServiceTemplateMinorVersion",
                "proton:GetServiceTemplateVersion",
                "proton:GetTemplateSyncConfig",
                "proton:GetTemplateSyncStatus",
                "proton:ListEnvironmentAccountConnections",
                "proton:ListEnvironmentOutputs",
                "proton:ListEnvironmentProvisionedResources",
                "proton:ListEnvironments",
                "proton:ListEnvironmentTemplateMajorVersions",
                "proton:ListEnvironmentTemplateMinorVersions",
                "proton:ListEnvironmentTemplates",
                "proton:ListEnvironmentTemplateVersions",
                "proton:ListRepositories",
                "proton:ListRepositorySyncDefinitions",
                "proton:ListServiceInstanceOutputs",
                "proton:ListServiceInstanceProvisionedResources",
```

```
                "proton:ListServiceInstances",
                "proton:ListServicePipelineOutputs",
                "proton:ListServicePipelineProvisionedResources",
                "proton:ListServices",
                "proton:ListServiceTemplateMajorVersions",
                "proton:ListServiceTemplateMinorVersions",
                "proton:ListServiceTemplates",
                "proton:ListServiceTemplateVersions",
                "proton:ListTagsForResource",
                "proton:UpdateService",
                "proton:UpdateServiceInstance",
                "proton:UpdateServicePipeline",
                "s3:ListAllMyBuckets",
                "s3:ListBucket"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CodeStarConnectionsPermissions",
            "Effect": "Allow",
            "Action": "codestar-connections:PassConnection",
            "Resource": [
                "arn:aws:codestar-connections:*:*:connection/*",
                "arn:aws:codeconnections:*:*:connection/*"
            ],
            "Condition": {
                "StringEquals": {
                    "codestar-connections:PassedToService": "proton.amazonaws.com"
                }
            }
        },
        {
            "Sid": "CodeConnectionsPermissions",
            "Effect": "Allow",
            "Action": "codeconnections:PassConnection",
            "Resource": [
                "arn:aws:codestar-connections:*:*:connection/*",
                "arn:aws:codeconnections:*:*:connection/*"
            ],
            "Condition": {
                "StringEquals": {
                    "codeconnections:PassedToService": "proton.amazonaws.com"
                }
            }
```

```
        }
    ]
}
```

AWS managed policy: AWSProtonReadOnlyAccess

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

This policy grants permissions that allow read-only access to AWS Proton actions and limited read-only access to other AWS service actions that AWS Proton depends on.

The policy includes the following key action namespaces:

- `proton` – Allows contributors read-only access to AWS Proton APIs.

Permissions details

This policy includes the following permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:ListPipelineExecutions",
        "codepipeline:ListPipelines",
        "codepipeline:GetPipeline",
        "codepipeline:GetPipelineState",
        "codepipeline:GetPipelineExecution",
        "proton:GetAccountRoles",
        "proton:GetAccountSettings",
        "proton:GetEnvironment",
        "proton:GetEnvironmentAccountConnection",
        "proton:GetEnvironmentTemplate",
        "proton:GetEnvironmentTemplateMajorVersion",
        "proton:GetEnvironmentTemplateMinorVersion",
        "proton:GetEnvironmentTemplateVersion",
        "proton:GetRepository",
        "proton:GetRepositorySyncStatus",
        "proton:GetResourcesSummary",
        "proton:GetService",
```

```
        "proton:GetServiceInstance",
        "proton:GetServiceTemplate",
        "proton:GetServiceTemplateMajorVersion",
        "proton:GetServiceTemplateMinorVersion",
        "proton:GetServiceTemplateVersion",
        "proton:GetTemplateSyncConfig",
        "proton:GetTemplateSyncStatus",
        "proton:ListEnvironmentAccountConnections",
        "proton:ListEnvironmentOutputs",
        "proton:ListEnvironmentProvisionedResources",
        "proton:ListEnvironments",
        "proton:ListEnvironmentTemplateMajorVersions",
        "proton:ListEnvironmentTemplateMinorVersions",
        "proton:ListEnvironmentTemplates",
        "proton:ListEnvironmentTemplateVersions",
        "proton:ListRepositories",
        "proton:ListRepositorySyncDefinitions",
        "proton:ListServiceInstanceOutputs",
        "proton:ListServiceInstanceProvisionedResources",
        "proton:ListServiceInstances",
        "proton:ListServicePipelineOutputs",
        "proton:ListServicePipelineProvisionedResources",
        "proton:ListServices",
        "proton:ListServiceTemplateMajorVersions",
        "proton:ListServiceTemplateMinorVersions",
        "proton:ListServiceTemplates",
        "proton:ListServiceTemplateVersions",
        "proton:ListTagsForResource"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS managed policy: AWSProtonSyncServiceRolePolicy

AWS Proton attaches this policy to the AWSServiceRoleForProtonSync service-linked role that allows AWS Proton to perform template sync.

This policy grants permissions that allow limited access to AWS Proton actions and to other AWS service actions that AWS Proton depends on.

The policy includes the following key action namespaces:

- `proton` – Allows AWS Proton sync limited access to AWS Proton APIs.
- `codeconnections` – Allows AWS Proton sync limited access to CodeConnections APIs.

For information on the permission details for the AWSProtonSyncServiceRolePolicy, see Service-linked role permissions for AWS Proton.

AWS managed policy: AWSProtonCodeBuildProvisioningBasicAccess

Permissions CodeBuild needs to run a build for AWS Proton CodeBuild Provisioning. You can attach `AWSProtonCodeBuildProvisioningBasicAccess` to your CodeBuild Provisioning Role.

This policy grants the minimum permissions for AWS Proton CodeBuild Provisioning to function. It grants permissions that allow CodeBuild to generate build logs. It also grants permission for Proton to make Infrastructure as Code (IaC) outputs available to AWS Proton users. It does not provide permissions needed by IaC tools to manage infrastructure.

The policy includes the following key action namespaces:

- `logs` - Allows CodeBuild to generate build logs. Without this permission, CodeBuild will fail to start.
- `proton` - Allows a CodeBuild Provisioning command to call `aws proton notify-resource-deployment-status-change` for updating the IaaC outputs for a given AWS Proton resource.

Permissions details

This policy includes the following permissions.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "logs:CreateLogStream",
    "logs:CreateLogGroup",
    "logs:PutLogEvents"
   ],
   "Resource": [
    "arn:aws:logs:*:*:log-group:/aws/codebuild/AWSProton-*"
   ]
```

```
  },
  {
   "Effect": "Allow",
   "Action": "proton:NotifyResourceDeploymentStatusChange",
   "Resource": "arn:aws:proton:*:*:*"
  }
 ]
}
```

AWS managed policy: AWSProtonCodeBuildProvisioningServiceRolePolicy

AWS Proton attaches this policy to the AWSServiceRoleForProtonCodeBuildProvisioning service-linked role that allows AWS Proton to perform CodeBuild-based provisioning.

This policy grants permissions that allow limited access to AWS service actions that AWS Proton depends on.

The policy includes the following key action namespaces:

- `cloudformation` – Allows AWS Proton CodeBuild-based provisioning limited access to AWS CloudFormation APIs.
- `codebuild` – Allows AWS Proton CodeBuild-based provisioning limited access to CodeBuild APIs.
- `iam` – Allows administrators to pass roles to AWS Proton. This is required so that AWS Proton can make API calls to other services on the administrator's behalf.
- `servicequotas` – Allows AWS Proton to check the CodeBuild concurrent build limit, which ensures proper build queuing.

Permissions details

This policy includes the following permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CreateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DeleteStack",
```

```
        "cloudformation:UpdateStack",
        "cloudformation:DescribeStacks",
        "cloudformation:DescribeStackEvents",
        "cloudformation:ListStackResources"
      ],
      "Resource": [
        "arn:aws:cloudformation:*:*:stack/AWSProton-CodeBuild-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:CreateProject",
        "codebuild:DeleteProject",
        "codebuild:UpdateProject",
        "codebuild:StartBuild",
        "codebuild:StopBuild",
        "codebuild:RetryBuild",
        "codebuild:BatchGetBuilds",
        "codebuild:BatchGetProjects"
      ],
      "Resource": "arn:aws:codebuild:*:*:project/AWSProton*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "*",
      "Condition": {
        "StringEqualsIfExists": {
          "iam:PassedToService": "codebuild.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
          "servicequotas:GetServiceQuota"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS managed policy: AwsProtonServiceGitSyncServiceRolePolicy

AWS Proton attaches this policy to the AwsProtonServiceGitSyncServiceRolePolicy service-linked role that allows AWS Proton to perform service sync.

This policy grants permissions that allow limited access to AWS Proton actions and to other AWS service actions that AWS Proton depends on.

The policy includes the following key action namespaces:

- `proton` – Allows AWS Proton sync limited access to AWS Proton APIs.

For information on the permission details for the AwsProtonServiceGitSyncServiceRolePolicy, see Service-linked role permissions for AWS Proton.

AWS Proton updates to AWS managed policies

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

Change	Description	Date
`AWSProtonFullAccess` – Update to an existing policy	The managed policy for the service-linked role to use Git sync with Git repositories has been updated for resources with both service prefixes. For more information, see Using service-linked roles for AWS CodeConnections and Managed policies.	April 25, 2024
`AWSProtonDeveloperAccess` – Update to an existing policy	The managed policy for the service-linked role to use Git sync with Git repositories has been updated for resources with both service prefixes. For more information, see	April 25, 2024

Change	Description	Date
	Using service-linked roles for AWS CodeConnections and Managed policies.	
AWSProtonSyncServiceRolePolicy – Update to an existing policy	The managed policy for the service-linked role to use Git sync with Git repositories has been updated for resources with both service prefixes. For more information, see Using service-linked roles for AWS CodeConnections and Managed policies.	April 25, 2024
AWSProtonCodeBuildProvisioningServiceRolePolicy – Update to an existing policy	AWS Proton updated this policy to add permissions to ensure accounts have the necessary CodeBuild concurrent build limit in order to use CodeBuild Provisioning.	May 12, 2023
AwsProtonServiceGitSyncServiceRolePolicy – New policy	AWS Proton added a new policy to allow AWS Proton to perform service syncing. The policy is used in the AWSServiceRoleForProtonServiceSync service-linked role.	March 31, 2023
AWSProtonDeveloperAccess – Update to an existing policy	AWS Proton added a new `GetResourcesSummary` action that allows you to view a summary of your templates, deployed template resources, and out of date resources.	November 18, 2022

Change	Description	Date
AWSProtonReadOnlyAccess – Update to an existing policy	AWS Proton added a new `GetResourcesSummary` action that allows you to view a summary of your templates , deployed template resources , and out of date resources.	November 18, 2022
AWSProtonCodeBuildProvisioningBasicAccess – New policy	AWS Proton added a new policy that gives CodeBuild the permissions it needs to run a build for AWS Proton CodeBuild Provisioning.	November 16, 2022
AWSProtonCodeBuildProvisioningServiceRolePolicy – New policy	AWS Proton added a new policy to allow AWS Proton to perform operations related to CodeBuild-based provision ing. The policy is used in the AWSServiceRoleForProtonCodeBuildProvisioning service-linked role.	September 02, 2022
AWSProtonFullAccess – Update to an existing policy	AWS Proton updated this policy to provide access to new AWS Proton API operations and to fix permission issues for some AWS Proton console operations.	March 30, 2022

Change	Description	Date
AWSProtonDeveloperAccess – Update to an existing policy	AWS Proton update this policy to provide access to new AWS Proton API operations and to fix permission issues for some AWS Proton console operations.	March 30, 2022
AWSProtonReadOnlyAccess – Update to an existing policy	AWS Proton update this policy to provide access to new AWS Proton API operations and to fix permission issues for some AWS Proton console operations.	March 30, 2022
AWSProtonSyncServiceRolePolicy – New policy	AWS Proton added a new policy to allow AWS Proton to perform operations related to template sync. The policy is used in the AWSServiceRoleForProtonSync service-linked role.	November 23, 2021
AWSProtonFullAccess – New policy	AWS Proton added a new policy to provide administrative role access to AWS Proton API operations and to the AWS Proton console.	June 09, 2021
AWSProtonDeveloperAccess – New policy	AWS Proton added a new policy to provide developer role access to AWS Proton API operations and to the AWS Proton console.	June 09, 2021

Change	Description	Date
AWSProtonReadOnlyAccess – New policy	AWS Proton added a new policy to provide read-only access to AWS Proton API operations and to the AWS Proton console.	June 09, 2021
AWS Proton started tracking changes.	AWS Proton started tracking changes for its AWS managed policies.	June 09, 2021

Using service-linked roles for AWS Proton

AWS Proton 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 Proton. Service-linked roles are predefined by AWS Proton and include all the permissions that the service requires to call other AWS services on your behalf.

Topics

- Using roles for AWS Proton sync
- Using roles for CodeBuild-based provisioning

Using roles for AWS Proton sync

AWS Proton 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 Proton. Service-linked roles are predefined by AWS Proton 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 Proton easier because you don't have to manually add the necessary permissions. AWS Proton defines the permissions of its service-linked roles, and unless defined otherwise, only AWS Proton 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 Proton 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 AWS Proton

AWS Proton uses two service-linked roles named **AWSServiceRoleForProtonSync** and **AWSServiceRoleForProtonServiceSync**.

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

- `sync.proton.amazonaws.com`

The role permissions policy named AWSProtonSyncServiceRolePolicy allows AWS Proton to complete the following actions on the specified resources:

- Action: *create, manage, and read* on *AWS Proton templates and template versions*
- Action: *use connection* on *CodeConnections*

AWSProtonSyncServiceRolePolicy

This policy includes the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SyncToProton",
            "Effect": "Allow",
            "Action": [
                "proton:UpdateServiceTemplateVersion",
                "proton:UpdateServiceTemplate",
                "proton:UpdateEnvironmentTemplateVersion",
                "proton:UpdateEnvironmentTemplate",
                "proton:GetServiceTemplateVersion",
                "proton:GetServiceTemplate",
```

```
                "proton:GetEnvironmentTemplateVersion",
                "proton:GetEnvironmentTemplate",
                "proton:DeleteServiceTemplateVersion",
                "proton:DeleteEnvironmentTemplateVersion",
                "proton:CreateServiceTemplateVersion",
                "proton:CreateServiceTemplate",
                "proton:CreateEnvironmentTemplateVersion",
                "proton:CreateEnvironmentTemplate",
                "proton:ListEnvironmentTemplateVersions",
                "proton:ListServiceTemplateVersions",
                "proton:CreateEnvironmentTemplateMajorVersion",
                "proton:CreateServiceTemplateMajorVersion"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AccessGitRepos",
            "Effect": "Allow",
            "Action": [
                "codestar-connections:UseConnection",
                "codeconnections:UseConnection"
             ],
            "Resource": [
                "arn:aws:codestar-connections:*:*:connection/*",
                "arn:aws:codeconnections:*:*:connection/*"
            ]
        }
    ]
}
```

For information on the the AWSProtonSyncServiceRolePolicy, see AWS managed policy: AWSProtonSyncServiceRolePolicy.

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

- `service-sync.proton.amazonaws.com`

The role permissions policy named AWSServiceRoleForProtonServiceSync allows AWS Proton to complete the following actions on the specified resources:

- Action: *create, manage, and read on AWS Proton services and service instances*

AwsProtonServiceGitSyncServiceRolePolicy

This policy includes the following permissions:

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "ProtonServiceSync",
   "Effect": "Allow",
   "Action": [
    "proton:GetService",
    "proton:UpdateService",
    "proton:UpdateServicePipeline",
    "proton:CreateServiceInstance",
    "proton:GetServiceInstance",
    "proton:UpdateServiceInstance",
    "proton:ListServiceInstances",
    "proton:GetComponent",
    "proton:CreateComponent",
    "proton:ListComponents",
    "proton:UpdateComponent",
    "proton:GetEnvironment",
    "proton:CreateEnvironment",
    "proton:ListEnvironments",
    "proton:UpdateEnvironment"
   ],
   "Resource": "*"
  }
 ]
}
```

For information on the the AwsProtonServiceSyncServiceRolePolicy, see AWS managed policy: AwsProtonServiceSyncServiceRolePolicy.

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 a service-linked role for AWS Proton

You don't need to manually create a service-linked role. When you configure a repository or service for sync in AWS Proton in the AWS Management Console, the AWS CLI, or the AWS API, AWS Proton creates the service-linked role for you.

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 configure a repository or service for sync in AWS Proton, AWS Proton creates the service-linked role for you again.

To recreate the AWSServiceRoleForProtonSync service-linked role, you would want to configure a repository for sync, and to recreate AWSServiceRoleForProtonServiceSync, you would want to configure a service for sync.

Editing a service-linked role for AWS Proton

AWS Proton doesn't allow you to edit the AWSServiceRoleForProtonSync service-linked role. After you create a service-linked role, you can't 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 Proton

You don't need to manually delete the AWSServiceRoleForProtonSync role. When you delete all AWS Proton linked repositories for repository sync in the AWS Management Console, the AWS CLI, or the AWS API, AWS Proton cleans up the resources and deletes the service-linked role for you.

Supported regions for AWS Proton service-linked roles

AWS Proton supports using service-linked roles in all of the AWS Regions where the service is available. For more information, see AWS Proton endpoints and quotas in the *AWS General Reference*.

Using roles for CodeBuild-based provisioning

AWS Proton 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 Proton. Service-linked roles are predefined by AWS Proton 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 Proton easier because you don’t have to manually add the necessary permissions. AWS Proton defines the permissions of its service-linked roles, and

unless defined otherwise, only AWS Proton 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 Proton 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 AWS Proton

AWS Proton uses the service-linked role named **AWSServiceRoleForProtonCodeBuildProvisioning** – A Service Linked Role for AWS Proton CodeBuild provisioning.

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

- `codebuild.proton.amazonaws.com`

The role permissions policy named AWSProtonCodeBuildProvisioningServiceRolePolicy allows AWS Proton to complete the following actions on the specified resources:

- Action: *create, manage, and read* on *AWS CloudFormation stacks and transforms*
- Action: *create, manage, and read* on *CodeBuild projects and builds*

AWSProtonCodeBuildProvisioningServiceRolePolicy

This policy includes the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CreateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
```

```
        "cloudformation:DeleteStack",
        "cloudformation:UpdateStack",
        "cloudformation:DescribeStacks",
        "cloudformation:DescribeStackEvents",
        "cloudformation:ListStackResources"
      ],
      "Resource": [
        "arn:aws:cloudformation:*:*:stack/AWSProton-CodeBuild-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:CreateProject",
        "codebuild:DeleteProject",
        "codebuild:UpdateProject",
        "codebuild:StartBuild",
        "codebuild:StopBuild",
        "codebuild:RetryBuild",
        "codebuild:BatchGetBuilds",
        "codebuild:BatchGetProjects"
      ],
      "Resource": "arn:aws:codebuild:*:*:project/AWSProton*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "*",
      "Condition": {
        "StringEqualsIfExists": {
          "iam:PassedToService": "codebuild.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
          "servicequotas:GetServiceQuota"
      ],
      "Resource": "*"
    }
  ]
}
```

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 a service-linked role for AWS Proton

You don't need to manually create a service-linked role. When you create an environment that uses CodeBuild-based provisioning in AWS Proton in the AWS Management Console, the AWS CLI, or the AWS API, AWS Proton creates the service-linked role for you.

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 create an environment that uses CodeBuild-based provisioning in AWS Proton, AWS Proton creates the service-linked role for you again.

Editing a service-linked role for AWS Proton

AWS Proton does not allow you to edit the AWSServiceRoleForProtonCodeBuildProvisioning service-linked role. 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 Proton

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don’t have an unused entity that is not actively monitored or maintained. However, you must delete all environments and services (instances and pipelines) that use CodeBuild-based provisioning in AWS Proton before you can manually delete it.

Manually delete the service-linked role

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForProtonCodeBuildProvisioning service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported regions for AWS Proton service-linked roles

AWS Proton supports using service-linked roles in all of the AWS Regions where the service is available. For more information, see AWS Proton endpoints and quotas in the *AWS General Reference*.

Troubleshooting AWS Proton identity and access

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

Topics

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

I am not authorized to perform an action in AWS Proton

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` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but does not have the fictional `proton:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 proton: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 `proton:`*`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, your policies must be updated to allow you to pass a role to AWS Proton.

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 an IAM user named `marymajor` tries to use the console to perform an action in AWS Proton. However, the action requires the service to have permissions that are 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's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS Proton 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 Proton supports these features, see How AWS Proton works with IAM.
- 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 Cross account resource access in IAM in the *IAM User Guide*.

Configuration and vulnerability analysis in AWS Proton

AWS Proton does not provide patches or updates for customer provided code. Customers are responsible for updating and applying patches to their own code, including the source code for their services and applications that are running on AWS Proton and the code provided in their service and environment template bundles.

Customers are responsible for updating and patching infrastructure resources in their environments and services. AWS Proton will not automatically update or patch any resources.

Customers should consult the documentation for the resources in their architecture to understand their respective patching policies.

Other than providing customer requested environment and service updates to minor versions of service and environment templates, AWS Proton does not provide patches or updates to the resources that customers define in their service and environment templates and template bundles.

For more details, see the following resources:

- Shared Responsibility Model
- Amazon Web Services: Overview of Security Processes

Data protection in AWS Proton

AWS Proton conforms to the AWS shared responsibility model which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud

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

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form text fields such as a **Name** field. This includes when you work with AWS Proton or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into free form text fields for resource identifiers or similar items related to the management of AWS resources might get picked up for inclusion in diagnostic logs. When you

provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog.*

Server side encryption at rest

If you choose to encrypt sensitive data in your template bundles at rest in the S3 bucket where you store your template bundles, you must use an SSE-S3 or SSE-KMS key to allow AWS Proton to retrieve the template bundles so they can be attached to a registered AWS Proton template.

Encryption in transit

All service to service communication is encrypted in transit using SSL/TLS.

AWS Proton encryption key management

Within AWS Proton, all customer data is encrypted by default using an AWS Proton owned key. If you supply a customer owned and managed AWS KMS key, all customer data is encrypted using the customer provided key as described in the following paragraphs.

When you create an AWS Proton template, you specify your key and AWS Proton uses your credentials to create a grant which allows AWS Proton to use your key.

If you manually retire the grant or, disable or delete your specified key, then AWS Proton is unable to read the data that was encrypted by the specified key and throws `ValidationException`.

AWS Proton encryption context

AWS Proton supports encryption context headers. An encryption context is an optional set of key-value pairs that can contain additional contextual information about the data. For general information about encryption context, see AWS Key Management Service Concepts - Encryption Context in the *AWS Key Management Service Developer Guide*.

An encryption context is a set of key–value pairs that contain arbitrary non-secret data. When including an encryption context in a request to encrypt data, AWS KMS cryptographically binds the encryption context to the encrypted data. To decrypt the data, you must pass in the same encryption context.

Customers can use the encryption context to identify use of their customer managed key in audit records and logs. It also appears in plaintext in logs, such as AWS CloudTrail and Amazon CloudWatch Logs.

AWS Proton does not take in any customer-specified or externally-specified encryption context.

AWS Proton adds the following encryption context.

```
{
  "aws:proton:template": "<proton-template-arn>",
  "aws:proton:resource": "<proton-resource-arn>"
}
```

The first encryption context identifies the AWS Proton template that the resource is associated with and also serves as a constraint for customer managed key permissions and grants.

The second encryption context identifies the AWS Proton resource that is encrypted.

The following examples show AWS Proton encryption context use.

Developer creating a service instance.

```
{
  "aws:proton:template": "arn:aws:proton:region_id:123456789012:service-template/my-
template",
  "aws:proton:resource": "arn:aws:proton:region_id:123456789012:service/my-service/
service-instance/my-service-instance"
}
```

An administrator creating a template.

```
{
  "aws:proton:template": "arn:aws:proton:region_id:123456789012:service-template/my-
template",
  "aws:proton:resource": "arn:aws:proton:region_id:123456789012:service-template/my-
template"
}
```

Infrastructure security in AWS Proton

As a managed service, AWS Proton 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 Proton 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.

To improve network isolation, you can use AWS PrivateLink as described in the following section.

AWS Proton and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and AWS Proton by creating an *interface VPC endpoint*. Interface endpoints are powered by AWS PrivateLink, a technology that enables you to privately access AWS Proton APIs without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with AWS Proton APIs. Traffic between your VPC and AWS Proton does not leave the Amazon network.

Each interface endpoint is represented by one or more Elastic Network Interfaces in your subnets.

For more information, see Interface VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

Considerations for AWS Proton VPC endpoints

Before you set up an interface VPC endpoint for AWS Proton, ensure that you review Interface endpoint properties and limitations in the *Amazon VPC User Guide*.

AWS Proton supports making calls to all of its API actions from your VPC.

VPC endpoint policies are supported for AWS Proton. By default, full access to AWS Proton is allowed through the endpoint. For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Creating an interface VPC endpoint for AWS Proton

You can create a VPC endpoint for the AWS Proton service using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

Create a VPC endpoint for AWS Proton using the following service name:

- com.amazonaws.*region*.proton

If you enable private DNS for the endpoint, you can make API requests to AWS Proton using its default DNS name for the Region, for example, `proton.`*`region`*`.amazonaws.com`.

For more information, see Accessing a service through an interface endpoint in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for AWS Proton

You can attach an endpoint policy to your VPC endpoint that controls access to AWS Proton. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for AWS Proton actions

The following is an example of an endpoint policy for AWS Proton. When attached to an endpoint, this policy grants access to the listed AWS Proton actions for all principals on all resources.

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Principal": "*",
      "Action": [
        "proton:ListServiceTemplates",
        "proton:ListServiceTemplateMajorVersions",
        "proton:ListServiceTemplateMinorVersions",
        "proton:ListServices",
        "proton:ListServiceInstances",
        "proton:ListEnvironments",
        "proton:GetServiceTemplate",
        "proton:GetServiceTemplateMajorVersion",
        "proton:GetServiceTemplateMinorVersion",
        "proton:GetService",
        "proton:GetServiceInstance",
        "proton:GetEnvironment",
        "proton:CreateService",
        "proton:UpdateService",
        "proton:UpdateServiceInstance",
        "proton:UpdateServicePipeline",
        "proton:DeleteService"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Logging and monitoring in AWS Proton

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Proton and your other AWS solutions. AWS provides the following monitoring tools to watch your instances running in AWS Proton, report when something is wrong, and take automatic actions when appropriate.

At this time, AWS Proton itself is not integrated with Amazon CloudWatch Logs or AWS Trusted Advisor. Administrators can configure and use CloudWatch to monitor other AWS services as defined in their service and environment templates. AWS Proton is integrated with AWS CloudTrail.

- *Amazon CloudWatch* monitors your AWS resources and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify

you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.

- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.
- *Amazon EventBridge* is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. EventBridge delivers a stream of real-time data from your own applications, Software-as-a-Service (SaaS) applications, and AWS services and routes that data to targets such as Lambda. This enables you to monitor events that happen in services, and build event-driven architectures. For more information, see Automate AWS Proton with EventBridge and the EventBridge User Guide.

Resilience in AWS Proton

The AWS global infrastructure is built around AWS Region 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 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 Proton offers features to help support your data resiliency and backup needs.

AWS Proton backups

AWS Proton maintains a backup of all customer data. In the case of a total outage, this backup can be used to restore AWS Proton and customer data from a previous valid state.

Security best practices for AWS Proton

AWS Proton provides 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.

Topics

- Use IAM to control access
- Do not embed credentials in your templates and template bundles
- Use encryption to protect sensitive data
- Use AWS CloudTrail to view and log API calls

Use IAM to control access

IAM is an AWS service that you can use to manage users and their permissions in AWS. You can use IAM with AWS Proton to specify which AWS Proton actions administrators and developers can perform, such as managing templates, environments or services. You can use IAM service roles to allow AWS Proton to make calls to other services on your behalf.

For more information on AWS Proton and IAM roles, see Identity and Access Management for AWS Proton.

Implement least privilege access. For more information, see Policies and permissions in IAM in the *AWS Identity and Access Management User Guide*.

Do not embed credentials in your templates and template bundles

Rather than embedding sensitive information in your AWS CloudFormation templates and template bundles, we recommend you use *dynamic references* in your stack template.

Dynamic references provide a compact, powerful way for you to reference external values that are stored and managed in other services, such as the AWS Systems Manager Parameter Store or AWS Secrets Manager. When you use a dynamic reference, CloudFormation retrieves the value of the specified reference when necessary during stack and change set operations, and passes the value to the appropriate resource. However, CloudFormation never stores the actual reference value. For more information, see Using Dynamic References to Specify Template Values in the *AWS CloudFormation User Guide*.

AWS Secrets Manager helps you to securely encrypt, store, and retrieve credentials for your databases and other services. The AWS Systems Manager Parameter Store provides secure, hierarchical storage for configuration data management.

For more information on defining template parameters, see https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/parameters-section-structure.html in the *AWS CloudFormation User Guide*.

Use encryption to protect sensitive data

Within AWS Proton, all customer data is encrypted by default using an AWS Proton owned key.

As a member of the platform team, you can provide a customer managed key to AWS Proton to encrypt and secure your sensitive data. Encrypt sensitive data at rest in your S3 bucket. For more information, see Data protection in AWS Proton.

Use AWS CloudTrail to view and log API calls

AWS CloudTrail tracks anyone making API calls in your AWS account. API calls are logged whenever anyone uses the AWS Proton API, the AWS Proton console or AWS Proton AWS CLI commands. Enable logging and specify an Amazon S3 bucket to store the logs. That way, if you need to, you can audit who made what AWS Proton call in your account. For more information, see Logging and monitoring in AWS Proton.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that AWS Proton gives another service to the resource. If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3

bucket ARN, you must use both global condition context keys to limit permissions. If you use both global condition context keys and the `aws:SourceArn` value contains the account ID, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The value of `aws:SourceArn` must be a resource that AWS Proton stores.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcards (`*`) for the unknown portions of the ARN. For example, `arn:aws::proton:*:123456789012:environment/*`.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in AWS Proton to prevent the confused deputy problem.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Sid": "ExampleProtonConfusedDeputyPreventionPolicy",
        "Effect": "Allow",
        "Principal": {"Service": "proton.amazonaws.com"},
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "123456789012"
            },
            "ArnLike": {
                "aws:SourceArn": "arn:aws::proton:*:123456789012:environment/*"
            }
        }
    }
}
```

CodeBuild provisioning custom Amazon VPC support

AWS Proton CodeBuild Provisioning executes arbitrary customer-supplied CLI commands in a CodeBuild project located in the AWS Proton Environment account. These commands typically

manage resource susing an Infrastructure as Code (IaC) tool, such as CDK. If you have resources in a Amazon VPC, CodeBuild may not be able to access them. To enable this, CodeBuild supports the ability to run within a specific Amazon VPC. A few example uses cases include:

- Retrieve dependencies from self-hosted, internal artifact repositories, such as `PyPI` for Python, `Maven` for Java, and `npm` for Node.js
- CodeBuild needs to access a Jenkins server in a particular Amazon VPC to register a pipeline.
- Access objects in an Amazon S3 bucket configured to allow access through an Amazon VPC endpoint only.
- Run integration tests from your build against data in an Amazon RDS database that's isolated on a private subnet.

For more information, see CodeBuild and VPC documentation.

If you want CodeBuild Provisioning to run in a custom VPC, AWS Proton provides a straightforward solution. First, you must add the VPC ID, subnets, and security groups to the environment template. Next, you enter those values into the environment spec. This will result in a CodeBuild project being created for you that targets a given VPC.

Updating the Environment Template

Schema

The VPC ID, subnets, and security groups need to be added to the template schema so they can exist in the environment spec.

An example `schema.yaml`:

```
schema:
  format:
    openapi: "3.0.0"
  environment_input_type: "EnvironmentInputType"
  types:
    EnvironmentInputType:
      type: object
      properties:
        codebuild_vpc_id:
          type: string
        codebuild_subnets:
```

```
          type: array
          items:
            type: string
        codebuild_security_groups:
          type: array
          items:
            type: string
```

This adds three new properties that will be used by the manifest:

- `codebuild_vpc_id`
- `codebuild_subnets`
- `codebuild_security_groups`

Manifest

To configure Amazon VPC settings in CodeBuild, an optional property called `project_properties` is available in the template manifest. Contents of `project_properties` are added to the AWS CloudFormation stack that creates the CodeBuild project. This makes it possible to add not only Amazon VPC AWS CloudFormation properties, but also any supported CodeBuild CloudFormation property, such as build timeout. The same data provided to `proton-inputs.json` is made available to the values of `project_properties`.

Add this section to your `manifest.yaml`:

```
project_properties:
          VpcConfig:
            VpcId: "{{ environment.inputs.codebuild_vpc_id }}"
            Subnets: "{{ environment.inputs.codebuild_subnets }}"
            SecurityGroupIds: "{{ environment.inputs.codebuild_security_groups }}"
```

The following is what the resulting `manifest.yaml` may look like:

```
infrastructure:
  templates:
    - rendering_engine: codebuild
      settings:
        image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
        runtimes:
          nodejs: 16
```

```
        provision:
          - npm install
          - npm run build
          - npm run cdk bootstrap
          - npm run cdk deploy -- --require-approval never
        deprovision:
          - npm install
          - npm run build
          - npm run cdk destroy -- --force
        project_properties:
          VpcConfig:
            VpcId: "{{ environment.inputs.codebuild_vpc_id }}"
            Subnets: "{{ environment.inputs.codebuild_subnets }}"
            SecurityGroupIds: "{{ environment.inputs.codebuild_security_groups }}"
```

Creating the environment

When you create an environment with your CodeBuild Provisioning VPC-enabled template, you must provide the Amazon VPC ID, subnets, and security groups.

To get a list of all Amazon VPC IDs in your Region, run the following command:

```
aws ec2 describe-vpcs
```

To get a list of all the subnet IDs, run:

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=vpc-id"
```

> **Important**
>
> Only include private subnets. CodeBuild will fail if you provide public subnets. Public subnets have a default route to an Internet Gateway, while private subnets don't.

Run the following command to obtain the security group IDs. These IDs can also be obtained through the AWS Management Console:

```
aws ec2 describe-security-groups --filters "Name=vpc-id,Values=vpc-id"
```

The values will resemble:

```
vpc-id: vpc-045ch35y28dec3a05
subnets:
  - subnet-04029a82e6ae46968
  - subnet-0f500a9294fc5f26a
 security-groups:
  - sg-03bc4c4ce32d67e8d
```

Ensuring CodeBuild permissions

Amazon VPC support requires certain permissions, such as the ability to create an Elastic Network Interface.

If the environment is being created in the console, enter these values during the environment creation wizard. If you want to programmatically create the environment, your `spec.yaml` looks like the following:

```
proton: EnvironmentSpec

spec:
  codebuild_vpc_id: vpc-045ch35y28dec3a05
  codebuild_subnets:
    - subnet-04029a82e6ae46968
    - subnet-0f500a9294fc5f26a
  codebuild_security_groups:
    - sg-03bc4c4ce32d67e8d
```

AWS Proton resources and tagging

AWS Proton resources that are assigned an Amazon Resource Name (ARN) include environment templates and their major and minor versions, service templates and their major and minor versions, environments, services, service instances, components, and repositories. You can tag these resources to help you organize and identify them. You can use tags to categorize resources by purpose, owner, environment, or other criteria. For more information, see Tagging Strategies. To track and manage your AWS Proton resources, you can use the tagging features described in the following sections.

AWS tagging

You can assign metadata to your AWS resources in the form of tags. Each tag consists of a customer defined key and optional value. Tags can help you manage, identify, organize, search for, and filter resources.

> **Important**
>
> Do not add personally identifiable information (PII) or other confidential or sensitive information in tags. Tags are accessible to many AWS services, including billing. Tags are not intended to be used for private or sensitive data.

Each tag has two parts.

- A tag key (for example, `CostCenter`, `Environment`, or `Project`). Tag keys are case sensitive.
- A tag value (optional) (for example, `111122223333` or `Production`). Like tag keys, tag values are case sensitive.

The following basic naming and usage requirements apply to tags.

- Each resource can have a maximum of 50 user created tags.

> **Note**
>
> System created tags that begin with the `aws:` prefix are reserved for AWS use, and do not count against this limit. You can't edit or delete a tag that begins with the `aws:` prefix.

- For each resource, each tag key must be unique, and each tag key can have only one value.
- The tag key must be a minimum of 1 and a maximum of 128 Unicode characters in UTF-8.
- The tag value must be a minimum of 1 and a maximum of 256 Unicode characters in UTF-8.
- Allowed characters in tags are letters, numbers, spaces representable in UTF-8, and the following characters:* _ . : / = + - @.

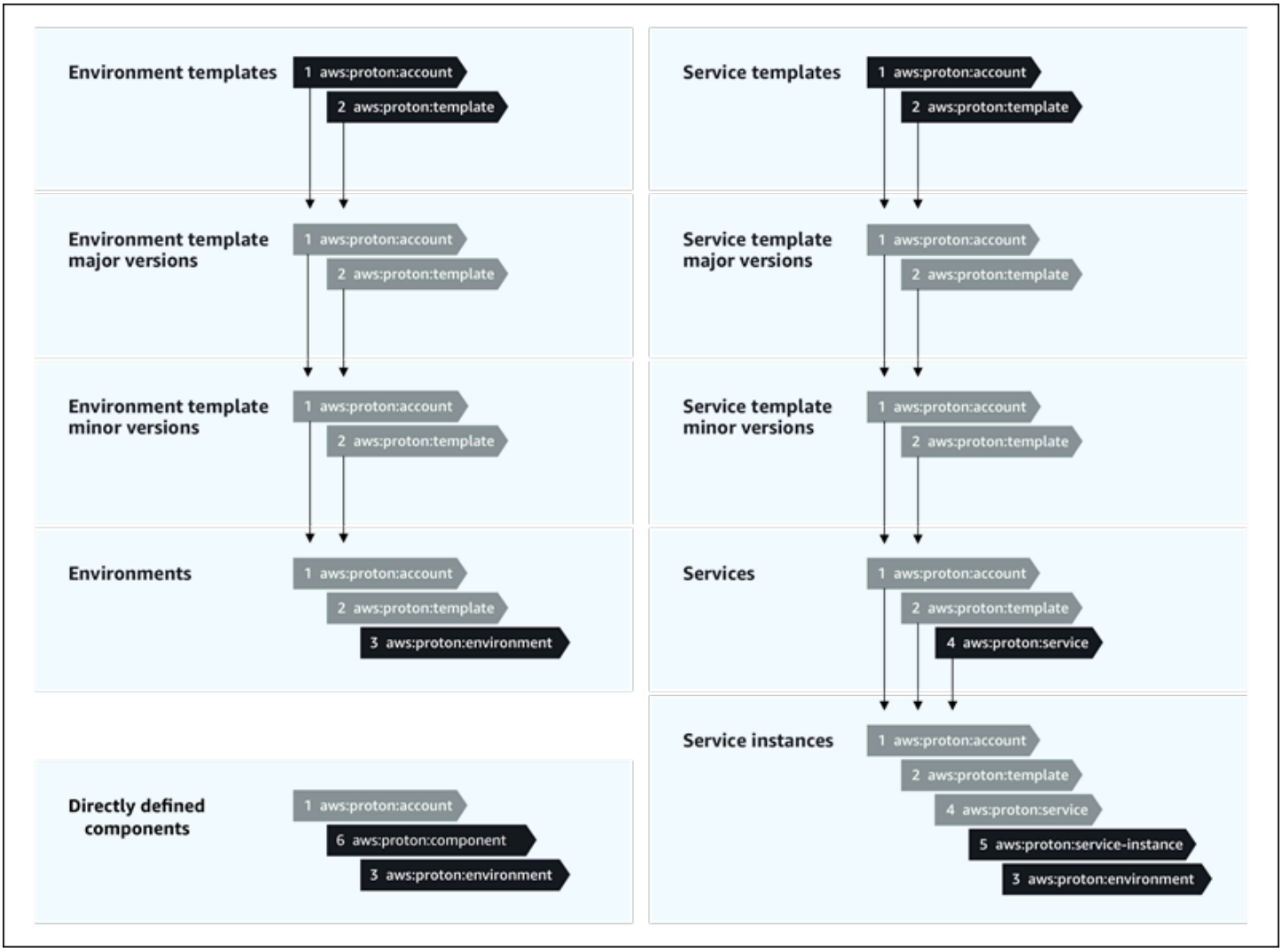
AWS Proton tagging

With AWS Proton, you can use both the tags that you create as well as the tags that AWS Proton automatically generates for you.

AWS Proton AWS managed tags

When you create an AWS Proton resource, AWS Proton automatically generates AWS managed tags for your new resource as shown in the following diagram. AWS managed tags later propagate to other AWS Proton resources that are based on your new resource. For example, managed tags from an environment template propagate to its versions, and managed tags from a service propagate to its service instances.

> **Note**
>
> AWS managed tags *aren't* generated for environment account connections. For more information, see the section called "Account connections".



Tag propagation to provisioned resources

If provisioned resources, such as those defined in service and environment templates, support AWS tagging, the AWS managed tags propagate as customer managed tags to provisioned resources. These tags won't propagate to a provisioned resource that doesn't support AWS tagging.

AWS Proton applies tags to your resources by AWS Proton accounts, registered templates and deployed environments, as well as services and service instances as described in the following table. You can use AWS managed tags to view and manage your AWS Proton resources, but you can't modify them.

AWS managed tag key	Propagated customer managed key	Description
`aws:proton:account`	`proton:account`	The AWS account that creates and deploys AWS Proton resources.
`aws:proton:template`	`proton:template`	The ARN of a selected template.
`aws:proton:environment`	`proton:environment`	The ARN of a selected environment.
`aws:proton:service`	`proton:service`	The ARN of a selected service.
`aws:proton:service-instance`	`proton:service-instance`	The ARN of a selected service instance.
`aws:proton:component`	`proton:component`	The ARN of a selected component.

The following is an example of an AWS managed tag for an AWS Proton resource.

```
"aws:proton:template" = "arn:aws:proton:region-id:account-id:environment-template/env-template"
```

The following is an example of a customer managed tag applied to a provisioned resource that was propagated from an AWS managed tag.

```
"proton:environment:database" = "arn:aws:proton:region-id:account-id:rds/env-db"
```

With AWS-managed provisioning, AWS Proton applies propagated tags directly to provisioned resources.

With self-managed provisioning, AWS Proton makes propagated tags available together with the rendered IaC files that it submits in the provisioning pull request (PR). Tags are provided in the string map variable `proton_tags`. We recommend that you make a reference to this variable in

your Terraform configuration to include AWS Proton tags in `default_tags`. This propagates AWS Proton tags to all provisioned resources.

The following example shows this method of tag propagation in an environment Terraform template.

Here's the `proton_tags` variable definition:

proton.environment.variables.tf:

```
variable "environment" {
  type = object({
    inputs = map(string)
    name = string
  })
}

variable "proton_tags" {
  type = map(string)
  default = null
}
```

Here's how tag values are assigned to this variable:

proton.auto.tfvars.json:

```
{
  "environment": {
    "name": "dev",
    "inputs": {
      "ssm_parameter_value": "MyNewParamValue"
    }
  }

  "proton_tags" : {
    "proton:account" : "123456789012",
    "proton:template" : "arn:aws:proton:us-east-1:123456789012:environment-template/
fargate-env",
    "proton:environment" : "arn:aws:proton:us-east-1:123456789012:environment/dev"
  }
}
```

And here's how you can add AWS Proton tags to your Terraform configuration so that they're added to provisioned resources:

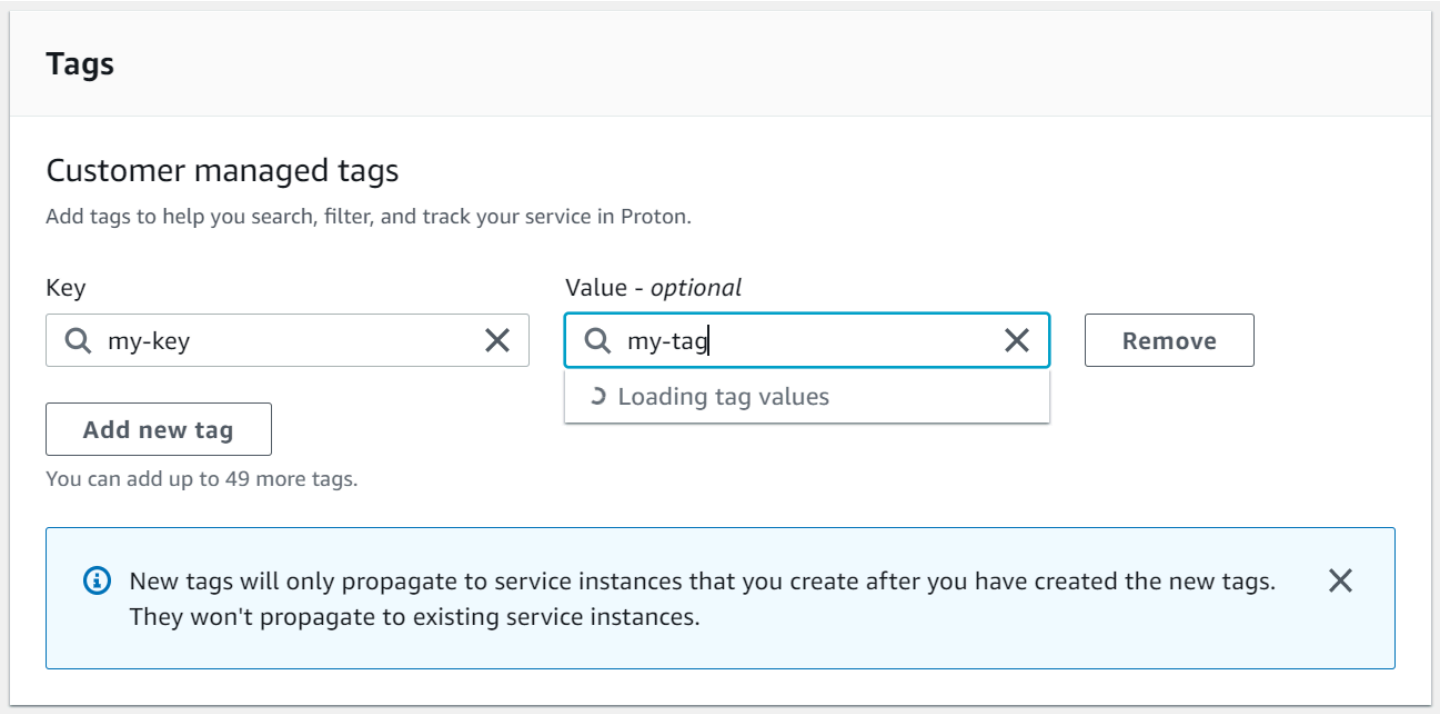
```
# Configure the AWS Provider
provider "aws" {
  region = var.aws_region
  default_tags {
    tags = var.proton_tags
  }
}
```

Customer managed tags

Each AWS Proton resource has a maximum quota of 50 customer managed tags. Customer managed tags propagate to child AWS Proton resources in the same way that AWS managed tags do, except they don't propagate to existing AWS Proton resources or to provisioned resources. If you apply a new tag to an AWS Proton resource with existing child resources and you want the existing child resources to be tagged with the new tag, you need to tag each existing child resource manually, using the console or AWS CLI.

Create tags using the console and CLI

When you create an AWS Proton resource using the console, you're given the opportunity to create customer managed tags either on the first or second page of the create procedure as shown in the following console snapshot. Choose **Add new tag**, enter the key and value and proceed.



After you create a new resource using the AWS Proton console, you can view its list of AWS managed and customer managed tags from the detail page.

Create or edit a tag

1. In the AWS Proton console, open an AWS Proton resource detail page where you can see a list of tags.
2. Choose **Manage tags**.
3. In the **Manage tags** page, you can view, create, remove and edit tags. You can't modify the AWS managed tags listed at the top. However, you can add to and modify the customer managed tags with editing fields, listed after the AWS managed tags.

   Choose **Add new tag** to create a new tag.
4. Enter a key and value for the new tag.
5. To edit a tag, enter a value in the tag value field for a selected key.
6. To delete a tag choose **Remove** for a selected tag.
7. When you have completed your changes, choose **Save changes**.

Create tags using the AWS Proton AWS CLI

You can view, create, remove and edit tags using the AWS Proton AWS CLI.

You can create or edit a tag for a resource as shown in the following example.

```
$ aws proton tag-resource \
    --resource-arn "arn:aws:proton:region-id:account-id:service-template/webservice" \
    --tags '[{"key":"mykey1","value":"myval1"},{"key":"mykey2","value":"myval2"}]'
```

You can remove a tag for a resource as shown in the next example.

```
$ aws proton untag-resource \
    --resource-arn "arn:aws:proton:region-id:account-id:service-template/webservice" \
    --tag-keys '["mykey1","mykey2"]'
```

You can list tags for a resource as shown in the final example.

```
$ aws proton list-tags-for-resource \
    --resource-arn "arn:aws:proton:region-id:account-id:service-template/webservice"
```

Troubleshooting AWS Proton

Learn to troubleshoot issues with AWS Proton.

Topics

- Deployment errors that reference AWS CloudFormation dynamic parameters

Deployment errors that reference AWS CloudFormation dynamic parameters

If you see deployment errors that reference your CloudFormation dynamic variables, verify that they are Jinja escaped. These errors can be caused by Jinja misinterpretation of your dynamic variables. The CloudFormation dynamic parameter syntax is very similar the Jinja syntax you use with your AWS Proton parameters.

Example CloudFormation dynamic variable syntax:

`'{{resolve:secretsmanager:MySecret:SecretString:password:EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE}}'`.

Example AWS Proton parameter Jinja syntax:

`'{{ service_instance.environment.outputs.env-outputs }}'`.

To avoid these misinterpretation errors, Jinja escape your CloudFormation dynamic parameters as shown in the following examples.

This example is from the AWS CloudFormation User Guide. The AWS Secrets Manager secret-name and json-key segments can be used to retrieve the sign-in credentials stored in the secret.

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

```
      MasterUserPassword:
 '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}'
```

To escape the CloudFormation dynamic parameters you can use two different methods:

- Enclose a block between `{% raw %}` and `{% endraw %}`:

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

- Enclose a parameter between `"{{ }}"`:

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

For information, see Jinja escaping.

AWS Proton quotas

The following table lists AWS Proton quotas. All values are per AWS account, per supported AWS Region.

Resource quota	Default limit	Adjustable?
Maximum size of template bundle	10 MB	× No
Maximum size of template manifest file	2 MB	× No
Maximum size of template schema file	2 MB	× No
Maximum size of each template file	2 MB	× No
Maximum length of each template name	100 characters	× No
Maximum number of CloudFormation template files per bundle	1	× No
Maximum number of registered templates per account, service and environment templates combined	1000	✓ Yes
Maximum number of template versions registered per template	1000	✓ Yes
Maximum number of files per CodeBuild Provisioning bundle	500	× No
Maximum number of environments per account	1000	✓ Yes
Maximum number of services per account	1000	✓ Yes
Maximum number of service instances per service	20	✓ Yes
Maximum number of components per account	1000	✓ Yes
Maximum number of environment account connections per environment account	1000	✓ Yes

Document history

The following table describes the important changes to the documentation related to the latest release of AWS Proton and customer feedback. For notification about updates to this documentation, you can subscribe to an RSS feed.

- **API version: 2020-07-20**

Change	Description	Date
Managed policy update	`AWSProtonDeveloperAccess` policy was updated.	April 25, 2024
Managed policy update	`AWSProtonFullAccess` policy was updated.	April 25, 2024
Managed policy update	`AWSProtonSyncServiceRolePolicy` policy was updated.	April 25, 2024
Managed policy update	`AWSProtonCodeBuildProvisioningServiceRolePolicy` policy was updated.	May 12, 2023
Service sync configurations.	AWS Proton adds support for service sync configurations.	March 31, 2023
CodeBuild	AWS Proton adds support for CodeBuild provisioning.	November 16, 2022
Managed policy update	Added `AWSProtonCodeBuildProvisioningBasicAccess` policy that gives CodeBuild the permissions it needs to	November 11, 2022

	run a build for AWS Proton CodeBuild Provisioning.	
Terraform tag propagation	Added Terraform tag propagation to the Tagging chapter.	September 16, 2022
API migration guide	Removed the pre-GA API migration guide.	August 12, 2022
AWS Proton objects	Added a topic about AWS Proton objects and their relationship to other AWS and third-party objects. See AWS Proton objects.	July 29, 2022
Linked repository clarifications	Clarified the purpose of linked (registered) repositories and their usage across the guide.	July 18, 2022
Guide merge	Merged the two separate administrator and user guides into a single guide, the *AWS Proton User Guide*.	June 30, 2022
Managed policy update	Updated managed policies to provide access to new AWS Proton API operations and to fix permission issues for some AWS Proton console operations. See AWS managed policies for AWS Proton.	June 20, 2022
Getting started with the CLI	Updated Getting started with the AWS CLI with a new tutorial that uses the new template library.	June 14, 2022

Directly defined components	Added the Components chapter and made related modifications throughout the guide.	June 1, 2022
AWS Proton template library	Added The AWS Proton template library topic.	May 6, 2022
Terraform general availability (GA)	Renamed *pull request provisioning* to *self-managed provisioning*. Added Provisioning methods topic.	March 23, 2022
Repository tagging	Added support for tagging Repository resources. See Create a link to your repository.	March 23, 2022
Documentation update	Added environment account connection tagging.	November 26, 2021
Template syncs and Terraform preview	Added automated template versioning with the template sync feature for general availability and pull request provisioning with Terraform in preview. API migration guide back in.	November 24, 2021
Documentation updates	Added EventBridge tutorial, Getting started workflow, How AWS Proton works, and Template bundle section enhancements.	September 17, 2021

AWS Proton console help panels release	Help panels added to the console. Console template version delete no longer deletes lower versions. The API migration guide has been removed.	September 8, 2021
AWS Proton general availability (GA) release	Added cross account environments, EventBridge monitoring, IAM condition keys, idempotency support, and increased quotas.	June 9, 2021
Add and delete service instances for a service and use existing external infrastructure for environments with AWS Proton	This public preview release includes updates that make it possible for you to add and delete service instances from a service, to use your existing external infrastructure in an AWS Proton environment and to cancel environment, service instance and pipeline deployments. AWS Proton now supports PrivateLink. An additional deletion validation has been added to prevent a minor version from being mistakenly deleted while a resource is using it.	April 27, 2021
Tagging with AWS Proton	Public preview release 2 includes AWS Proton tagging and the ability to launch services without a service pipeline.	March 5, 2021

Initial release	Public preview release is now available in selected regions.	December 1, 2020

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.