AWS Proton

User Guide

AWS Proton: User Guide

Table of Contents

What is AWS Proton?

AWS Proton is:

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

  The AWS Proton service is a two-pronged automation framework. Administrators create versioned *service templates* that define standardized infrastructure and deployment tooling for serverless and container-based applications. Then, 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  AWS Proton identifies all existing *service instances* that are using an outdated *template* version for you.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 a developer, you can request AWS Proton to update a *service instance* to the latest version of the *service template*.
- **Standardized infrastructure**

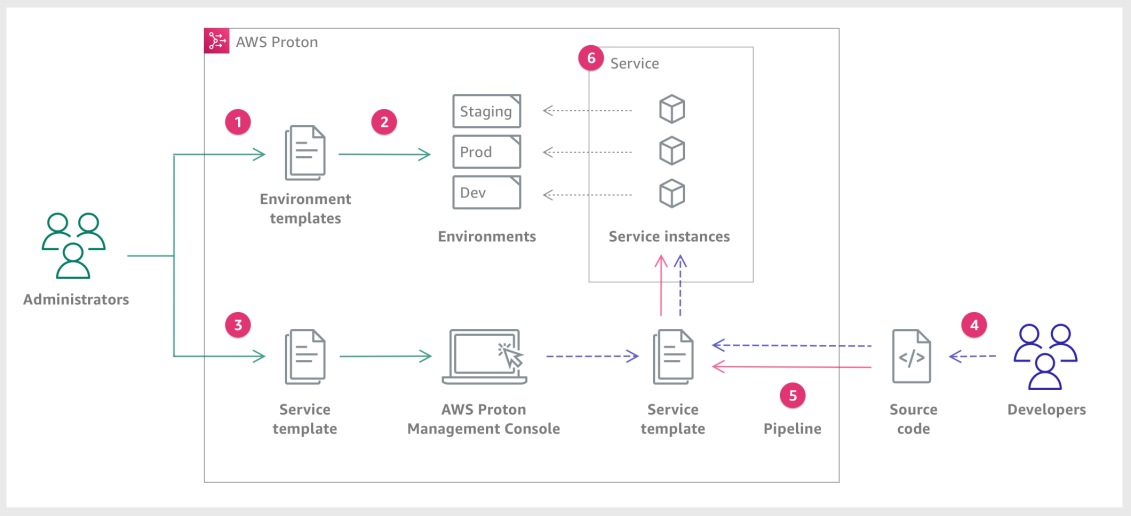
  Platform teams can use AWS Proton and versioned infrastructure-as-code templates to define and manage standard application stacks that contain the architecture, infrastructure resources and the CI/CD software deployment pipeline.
- **Deployments integrated with CI/CD**

  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 developers

As a 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*.

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 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 along with 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 deploys the **Service** with a **CI/CD Pipeline** for your **Service instances**.

6. AWS Proton deploys 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 AWS Proton services to deploy your applications.

Topics

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.

Create an IAM User

To create an administrator user for yourself and add the user to an administrators group (console)

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

   > **Note**
   > We strongly recommend that you adhere to the best practice of using the **`Administrator`** IAM user that follows and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.
2. In the navigation pane, choose **Users** and then choose **Add user**.
3. For **User name**, enter **`Administrator`**.
4. Select the check box next to **AWS Management Console access**. Then select **Custom password**, and then enter your new password in the text box.
5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.

6. Choose **Next: Permissions**.
7. Under **Set permissions**, choose **Add user to group**.
8. Choose **Create group**.
9. In the **Create group** dialog box, for **Group name** enter **`Administrators`**.
10. Choose **Filter policies**, and then select **AWS managed - job function** to filter the table contents.
11. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.

    Note
    You must activate IAM user and role access to Billing before you can use the `AdministratorAccess` permissions to access the AWS Billing and Cost Management console. To do this, follow the instructions in step 1 of the tutorial about delegating access to the billing console.
12. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
13. Choose **Next: Tags**.
14. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM entities in the *IAM User Guide*.
15. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users and to give your users access to your AWS account resources. To learn about using policies that restrict user permissions to specific AWS resources, see Access management and Example policies.

Setting up with AWS Proton

You need to set up a repository connection to create services with service templates that include a service pipeline.

Verify that you have installed the AWS CLI if you plan to use it to make AWS Proton API calls.

Setting up a repository connection

To create a service, you start by selecting a service template that defines your service infrastructure. In the console, you provide inputs to the template as you create a service from a service template. If you select a service template that includes a standard AWS Proton service pipeline, you must provide your code repository name and branch, along with a repository connection ARN. AWS Proton uses the repository connection to signal the service and pipeline in response to your source code commits.

This is one of the Prerequisites (p. 6) for following the steps in Getting started with the AWS Management Console (p. 6).

Set up a repository connection.

Connect AWS Proton to your code repository with AWS CodeStar connections to prompt an AWS Proton service pipeline when a new push is made to your repository. You can connect to Bitbucket, GitHub, GitHub Enterprise and GitHub Enterprise Server repositories with AWS CodeStar connections. For more information, see AWS CodeStar connections in the *AWS CodePipeline User Guide*.

1. Open the AWS Proton console.
2. In the navigation pane, select **Settings** and then **Source connections** to take you to the **Connections** page of **Developer Tools Settings**. The page displays a list of repository connections.

3. Choose **Create connection** and follow the instructions.

Setting up with 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 https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html

Getting started with AWS Proton

Get started with AWS Proton in the console or by using the AWS CLI.

Topics

Prerequisites

Before you begin *Step 1, Create a service*, make sure you have met the following prerequisites.

- You have an IAM account with administrator permissions. For more information, see Setting up (p. 3).
- You have a repository connection. For more information, see Setting up a repository connection (p. 4). AWS Proton uses the repository connection to signal the service and pipeline in response to your source code commits.
- You have working knowledge of AWS and with using the AWS Management Console.
- You're logged into your AWS account.
- You or your administrator has created the example fargate service template named **My Fargate Service**. For more information, see Getting started in the *AWS Proton Administrator Guide*.

Getting started with the AWS Management Console

Get started with AWS Proton

- Create and view a service.
- Deploy a service.
- Delete a service.

Step 1: Set up an example source repository

Set up repository.

In *Getting started*, you create a service from a service template that includes a service pipeline. You provide a path to your forked sample source code repository and your repository connection Amazon Resource Name (ARN) as inputs when you create your service. Your repository connection signals AWS Proton on new source code commits.

- Fork the example AWS sample repository that you use as source code to deploy with AWS Proton in *Getting started*.

Step 2: Create a service

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.

   If you don't see this template listed, contact your administrator or follow the Setting up and Getting started procedures in the *AWS Proton Administrator Guide*.
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, enter the name for the **Branch** and **Repository ID** (repo-username/repo-name) where you forked the source code (p. 4).
9. For **Repository Connection**, choose your repository connection from the list. Then, either proceed to the next step, or choose **Next**.
10. (Optional) In the **Tags** section, choose **Add new tag** and enter a key and value to create a customer managed tag. Then choose **Next**.
11. 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.
12. 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 (p. 64).
13. 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.
14. 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.
15. After the service is created, in the navigation pane, choose **Services** to view a list of your services.

Step 3: Edit a service

1. When your service status is **Active**, make a small update to your repository source code by changing 'AWS Proton' to 'AWS PROTON' for the graphic in `index.html`. Commit the update to prompt your pipeline to make the update to your service instance.

   The pipeline pulls the update. The update is included in the next deployment of your service instance.

2. Return to the AWS Proton console.
3. In the **Services** page, choose the service name **my-service**.

   You're now on the service detail page for **my-service**.
4. Choose the **Pipelines** tab.
5. In the **Outputs** section, choose the **PipelineEndpoint** link.

   This takes you to your service pipeline page.
6. Choose **Release** to force the update.
7. Navigate back to your service in the AWS Proton console to view the status of your service instance and application as the update is deployed.

Step 4: Delete a service

1. In the navigation pane, choose **Services**.
2. In the **Services** page, choose the service name **my-service**.

   You're now on the service detail page for **my-service**.
3. In the upper right-hand corner of the page, choose **Actions** and then **Delete**.
4. A modal prompts you to confirm the delete action.
5. Follow the instructions and choose **Yes, delete**.

Getting started with the AWS Proton AWS CLI

To get started with the AWS CLI, walk thru the following AWS CLI examples on GitHub.

- Multiple services share an AWS Proton environment
- Load-balanced web service using Amazon ECS and AWS Fargate
- Microservices using Amazon ECS and AWS Fargate

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. Environments represent 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, such as VPCs, clusters, and shared load balancers or API Gateways.

After your service instances are deployed, they can be updated by source code pushes that prompt the CI/CD pipeline or by the adoption of 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.

The following sections cover the use of AWS Proton service create, view, edit and delete operations. For additional information, see the *AWS Proton Service API Reference*.

Topics

Create a service

To deploy an application with AWS Proton, as a developer, you can either use the AWS Proton console or the AWS CLI. With either method, you only need to provide the following inputs.

1. The name of an AWS Proton service template that's published by your 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).

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 the latest `Recommended` minor version of the service template for the major version you specified.

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 your 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, your 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

When you create a service, you can choose from two different types of service templates as shown in the following list.

- A service template that includes a service pipeline (default).
- A service template that *doesn't* include a service pipeline.

Create a service using the console.

When you use the console, you can provide your service inputs directly to AWS Proton as described in Getting started with AWS Proton (p. 6). AWS Proton creates a YAML file from your inputs to the console. It is available for download in the service detail page **Actions** drop-down, after your service is created. When you *don't* want to use an AWS Proton service pipeline, choose a template marked with *Excludes pipeline*.

Create a service with an AWS Proton service pipeline using the CLI.

When you use the AWS CLI, you specify service inputs in a YAML formatted file, `.aws-proton/service.yaml`, located in your source code directory. 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. Use the `get-service-template` command to see if `pipelineProvisioning` is set to `"CUSTOMER_MANAGED"`.

The following example command and response shows how to use the CLI to view the schema `required` and `optional` parameters that you provide values for in your spec file. The schema is found in the response `schema` output.

Command:

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

Response:

```
{
    "serviceTemplateMinorVersion": {
        "arn": "arn:aws:proton:us-west-2:123456789012: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"
    }
}
```

The formatted schema from the response is shown in the next example, for your convenience.

Schema:

```
schema:
  format:
    openapi: "3.0.0"
  pipeline_input_type: "MyPipelineInputType"
  service_input_type: "MyServiceInstanceInputType"
  types:
    MyPipelineInputType:
      type: object
      description: "Pipeline input properties"
      required:
        - my_sample_pipeline_required_input
      properties:
        my_sample_pipeline_optional_input:          # parameter
          type: string
          description: "This is a sample input"
          default: "hello world"
        my_sample_pipeline_required_input:          # parameter
          type: string
          description: "Another sample input"
    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"
```

The following example spec includes parameter values for a service pipeline and one instance, based on the schema file.

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 as defined by a service template by specifying the name, repository connection ARN, repository ID, repository branch, template name, spec, the major and minor versions, description (optional) and tags (optional) as shown in the following command and response.

Command:

```
aws proton create-service --name "simple-svc" --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* an AWS Proton service pipeline by using the CLI.

The following example spec *doesn't* include inputs for a service pipeline.

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* an AWS Proton service pipeline, you provide the path to a `spec.yaml` and you *don't* include repository parameters as shown in the following 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",
        "templateName": "fargate-service-no-pipeline",
        "status": "CREATE_IN_PROGRESS"
    }
}
```

View service data

You can view service detail data using either the console or the AWS CLI.

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

1. To view a list of the services, choose **Services**.
2. To view detail data, choose the name of your service.

   View your service detail.

You can also use the AWS CLI for AWS Proton by using *get* or *list* operations as shown in the following example that includes a service pipeline. You can get or list services.

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"
    }
}
```

The next example command and response shows how to use the `get-service` command to view details for a service that *doesn't* have an AWS Proton service pipeline.

Command:

```
aws proton get-service --name "simple-svc-without-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 a service.

- Edit the service description.
- Edit a service by adding and removing service instances.

Edit service description

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, choose **Next**.
6. Review your edits and choose **Save changes**.

You can also use the AWS CLI to edit a description as shown in the next 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 by adding or removing service instances

For an AWS Proton service, you can add or delete service instances by submitting an edited `spec` (p. 16), if your request *isn't* made under the following conditions. AWS Proton fails your request under these conditions.

- Your service and pipeline is already being edited or deleted when you submit the edit request.
- Your edited `spec` includes edits that modify the service pipeline or existing service instances that *aren't* to be deleted.

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.

Add or remove service instances

For the quota of service instances for a service, see AWS Proton quotas (p. 68). 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 need to account for them when you edit your `spec`.

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 remediate 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 (p. 64).

Use the console or AWS CLI to edit a service

You can use the AWS Proton console and AWS CLI to edit a service by adding and removing instances.

Edit your service by adding or removing an instance using the console.

In the AWS Proton console

1. In the navigation pane, choose **Services**.
2. choose the service that you want to edit.

3. Choose **Edit**.
4. (Optional) On the **Configure service** page, edit the service name or description and, at the lower right corner of the page, 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. On the lower right-hand corner of the page, 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. In the service detail page, view the status details for your service.

The following is an example of using the AWS CLI with an edited `spec` to add and delete service instances. 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 `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` that's 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"
```

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"
        }
    ]
}
```

Delete

You can delete a service by using the console or the AWS CLI.

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. Choose **Actions** and then **Delete**.
4. A modal prompts you to confirm the delete action.
5. Follow the instructions and choose **Yes, delete**.

You can also use the AWS CLI for AWS Proton to delete a service as shown in the following 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

You can view service instance detail data using either the console or the AWS CLI.

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

1. To view a list of the service instances, choose **Services instances** in the navigation pane.
2. To view detail data, choose the name of a service instance.

   View the detail data of your service instance.

You can also use the AWS CLI for AWS Proton by using the get or list operations as shown in the following example commands and responses. You can *get* or *list* service instances.

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

There are four modes for updating a service instance as described in the following list. When you use the AWS CLI, the `deployment-type` field defines the mode. When you use the console, these modes map to the **Update spec**, **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 if 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`. When you do this, 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*.

Update a service instance using the console by following these steps.

1. From the AWS Proton console, choose **Service instances** in the left-hand menu.
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, **Update spec**, **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**.

The following example commands and responses show how to use the AWS CLI to update a service instance to a new minor version.

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"
    }
}
```

Use the console to cancel a service instance deployment 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**, on 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.

Use the AWS Proton AWS CLI to cancel an IN_PROGRESS service instance update deployment to a new minor version 2 as shown in the following 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

There are four modes for updating a service pipeline as described in the following. When you use the AWS CLI, the `deployment-type` field defines the mode. When using 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*.

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. To update the spec, choose **Edit Pipeline** and fill out each form and choose **Next** until you complete the final form. Then choose **Update pipeline**.

If there's an **information icon** that shows that a new version is available at **Pipeline template**, choose the name of the new template version to update the template version.

a. Choose **Update to recommended version**.

b. Fill out each form and choose **Next** until you complete the final form and then choose **Update**.

Use the AWS CLI to update a service pipeline to a new minor version as shown in the following example commands and responses.

Command: to update

```
aws proton update-service-pipeline --service-name --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"
    }
}
```


Use the console to cancel a service pipeline deployment 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.

Use the AWS Proton AWS CLI to cancel an IN_PROGRESS service pipeline update deployment to a new minor version 2 as shown in the following 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"
    }
}
```

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 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.

The following table lists the detail-type values, status values, and detail fields for each AWS Proton resource that can emit events. When a resource is deleted, the `"status"` detail field value is `"DELETED"`.

Resource	detail-type value	detail field: values
Environment Template	"AWS Proton Environment Template Status Change"	"name": *"myTemplate"* "status": EnvironmentTemplate "previousStatus": EnvironmentTemplate

Resource	detail-type value	detail field: values
Environment Template Version	"AWS Proton Environment Template Version Status Change"	"name": *"myTemplate"* "majorVersion": *"1"* "minorVersion": *"0"* "status": EnvironmentTemplateVersion "previousStatus": EnvironmentTemplateVersion
Service Template	"AWS Proton Service Template Status Change"	"name": *"myTemplate"* "status": ServiceTemplate "previousStatus": ServiceTemplate
Service Template Version	"AWS Proton Service Template Version Status Change"	"name": *"myTemplate"* "majorVersion": *"1"* "minorVersion": *"0"* "status": ServiceTemplateVersion "previousStatus": ServiceTemplateVersion
Environment	"AWS Proton Environment Status Change"	"name": *"myEnvironment"* "status": Environment "previousStatus": Environment
Service	"AWS Proton Service Status Change"	"name": *"myService"* "status": Service "previousStatus": Service
Service Instance	"AWS Proton Service Instance Status Change"	"name": *"myServiceInstance"* "serviceName": *"myService"* "status": ServiceInstance "previousStatus": ServiceInstance
Service Pipeline	"AWS Proton Service Pipeline Status Change"	"serviceName": *"myService"* "status": ServicePipeline "previousStatus": ServicePipeline

Resource	detail-type value	detail field: values
Environment Account Connection	"AWS Proton Environment Account Connection Status Change"	"id": *"myEnvironmentAccountConnection"* "status": EnvironmentAccountConnection "previousStatus": EnvironmentAccountConnection

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 with AWS Proton. For more information, see Quotas and Edit a service.

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 (p. 32).

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 **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**.

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

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

- AWS managed policies for AWS Proton (p. 52)
- Troubleshooting AWS Proton identity and access (p. 55)

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 (p. 55).

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 employees 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 (p. 40).

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 (p. 46).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also 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 Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single 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 do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then

securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

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:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role 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 How IAM roles differ from resource-based policies 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.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, resources, and condition keys for AWS Proton in the *Service Authorization Reference*.
  - **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 IAM 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 IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. 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**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

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 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 (p. 41)	Yes
Resource-based policies (p. 41)	No
Policy actions (p. 42)	Yes
Policy resources (p. 43)	Yes
Policy condition keys (p. 43)	Yes
ACLs (p. 44)	No
ABAC (tags in policies) (p. 44)	Yes
Temporary credentials (p. 44)	Yes
Principal permissions (p. 45)	Yes
Service roles (p. 45)	Yes
Service-linked roles (p. 45)	No

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 (p. 46).

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 How IAM roles differ from resource-based policies 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 `Describe`, include the following action:

```
"Action": "proton:List*"
```

To view examples of AWS Proton identity-based policies, see Identity-based policy examples for AWS Proton (p. 46).

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 (p. 46).

Policy condition keys for AWS Proton

Supports 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 (p. 46).

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.

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 (p. 64).

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 principal permissions	Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, resources, and condition keys for AWS Proton in the *Service Authorization Reference*.

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 (p. 48).

> **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	No

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 IAM 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 exampes in the following sections.

Topics

Identity-based policy examples for AWS Proton

By default, IAM users and roles don't have permission to create or modify AWS Proton resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform actions on the resources that they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies on the JSON tab in the *IAM User Guide*.

Topics

Policy best practices

Identity-based policies are very powerful. They 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 using AWS managed policies** – To start using AWS Proton quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Links to Identity-based policy examples for AWS Proton

Links to example identity-based policy examples for AWS Proton

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 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 (p. 57).

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

As a member of the platform team, you can as an administrator create an AWS Proton service role to allow AWS Proton to make API calls to other services, like CloudFormation, on your behalf.

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 your roles, this is the AWS Proton service role policy that AWS Proton 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.

IAM AWS Proton service role policy

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"]
                }
            }
        }
    ]
}
```

IAM AWS Proton service trust policy

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

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 pipeline service role

As a member of the platform team, you, as an administrator, can create an AWS Proton pipeline service role to allow AWS Proton to make CloudFormation API calls to deploy a pipeline CloudFormation stack on your behalf.

We recommend that you use the following IAM role and trust policy for your AWS Proton pipeline service role. When you use the AWS Proton console to create your roles, this is the AWS Proton pipeline service role policy that AWS Proton 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 pipelines.

IAM AWS Proton pipeline service role policy

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"]
                }
            }
        }
    ]
}
```

IAM AWS Proton pipeline service trust policy

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

To see an example of a scoped down policy, see AWS Proton service role (p. 49).

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 IAM 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": {
        "Effect": "Allow",
        "Principal": {"Service": "proton.amazonaws.com"},
        "Action": "sts:AssumeRole"
    }
}
```

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 and limited access to AWS Key Management Service, IAM and AWS CodeStar connections services.

Permissions details

This policy includes the following permissions.

This managed policy provides administrative access to the AWS Proton APIs and AWS Management Console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "proton:*",
                "codestar-connections:ListConnections",
                "kms:ListAliases",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": "proton.*.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "proton.amazonaws.com"
                }
            }
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "codestar-connections:PassConnection"
            ],
            "Resource": "arn:aws:codestar-connections:*:*:connection/*",
            "Condition": {
                "StringEquals": {
                    "codestar-connections:PassedToService": "proton.amazonaws.com"
                }
            }
        }
    ]
}
```

- `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.
- `codestar-connections` – Allows administrators to list and pass codestar-connections so they can be used by AWS Proton.

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 read-only permissions that allow read only access to AWS Proton APIs.

Permissions details

This policy includes the following permissions.

This managed policy provides read only access to the AWS Proton APIs and AWS Management Console.

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

- `proton` – Allows contributors read-only access to AWS Proton APIs.

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 (p. 53) – provides administrative access to the AWS Proton APIs and AWS Management Console.	AWS Proton added a new policy to provide administrative access to the AWS Proton APIs and AWS Management Console.	JUNE 09, 2021
AWSProtonReadOnlyAccess (p. 54) – added a new policy.	AWS Proton added a new policy to allow read-only access to AWS Proton APIs.	JUNE 09, 2021
AWS Proton started tracking changes.	AWS Proton started tracking changes for its AWS managed policies.	JUNE 09, 2021

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

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 user name and password.

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, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to AWS 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 granted by a service role. Mary does not have permissions to pass the role to the service.

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

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

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access AWS Proton

To allow others to access AWS Proton, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in AWS Proton.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

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 (p. 40).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.

- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

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 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 AWS Proton or other services 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 the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access AWS Proton through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key 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 (p. 29) and the EventBridge User Guide.

Resilience in AWS Proton

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between 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

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 (p. 36).

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 (p. 57).

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 (p. 61).

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 and service instances. 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 of as 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.

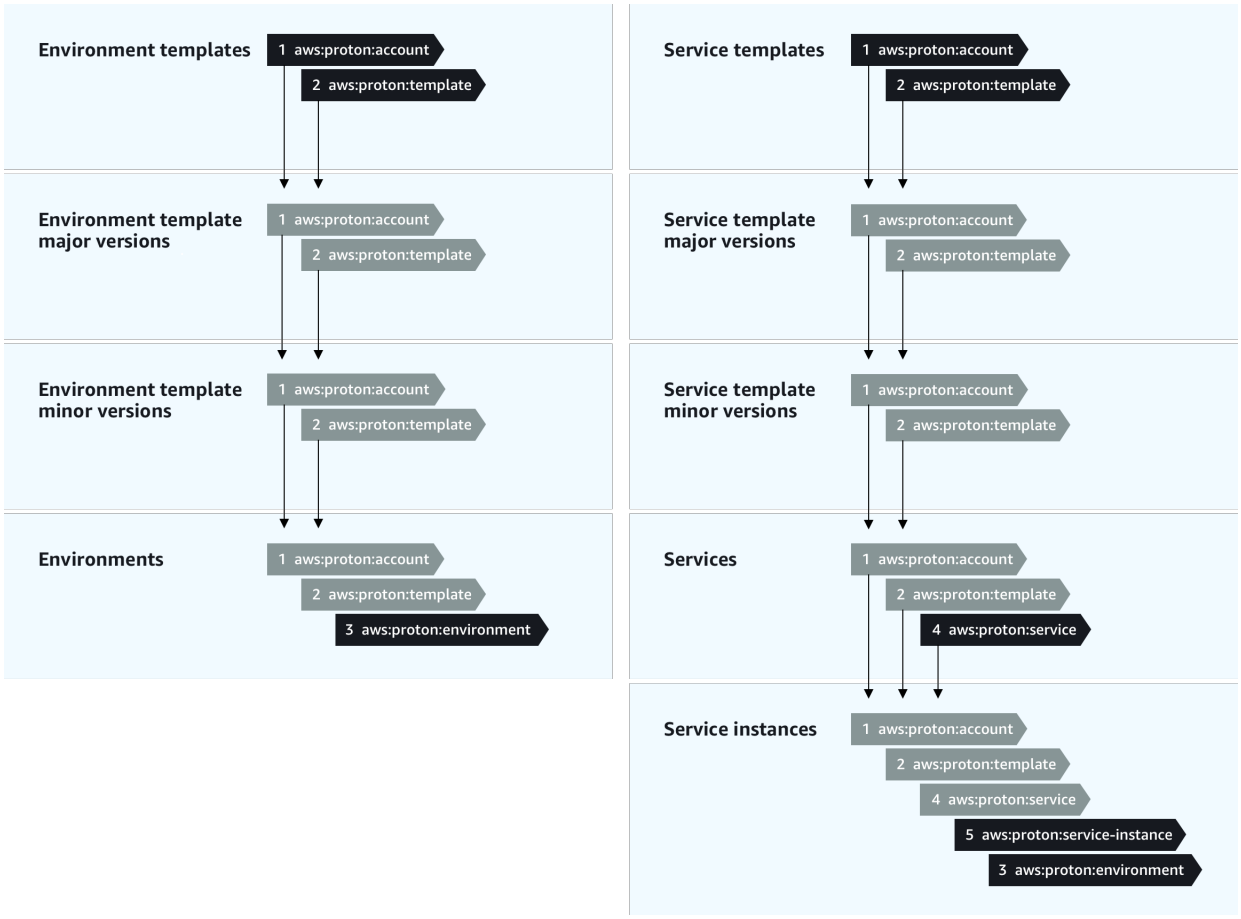
  > **Note**
  > System created tags that begin with the `aws:` prefix are reserved for AWS use, and do not count against this limit. You can't edit or delete a tag that begins with the `aws:` prefix.
- For each resource, each tag key must be unique, and each tag key can have only one value.
- The tag 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 propagate to the AWS Proton resources that your new resource creates or deploys.



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 ARN of the account that creates and deploys AWS Proton resources.
`aws:proton:template`	`proton:template`	The ARN of a selected template.
`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:environment`	`proton:environment`	The ARN of a selected environment.

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.

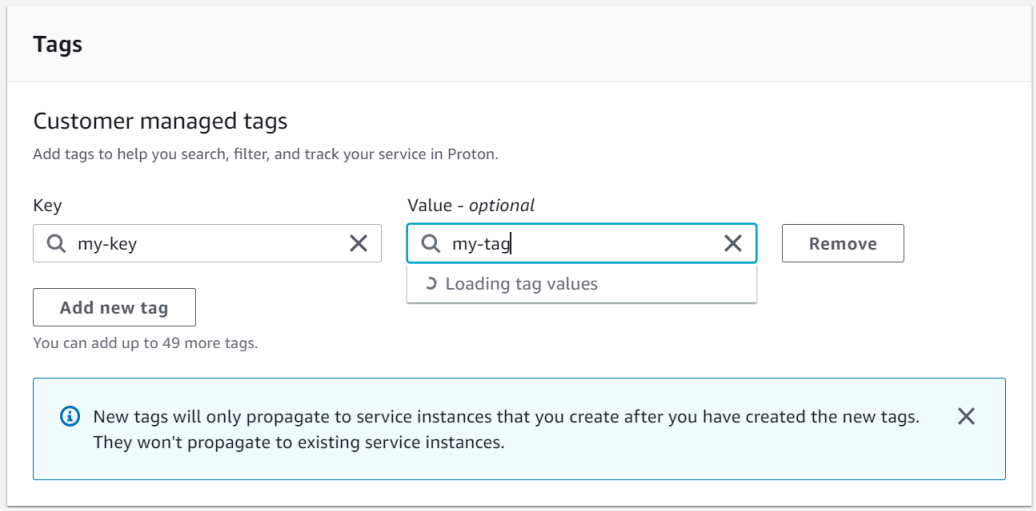
```
"proton:environment:database" = "arn:aws:proton:region-id:account-id:rds/env-db"
```

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

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"
```

AWS Proton quotas

The following table lists AWS Proton quotas.

Resource	Default limit
Maximum size of spec service.yaml file	200 KB
Maximum number of environments per account	100
Maximum number of services per account	1000
Maximum number of service instances per service	100

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**

update-history-change	update-history-description	update-history-date
Documentation update (p. 69)	Added EventBridge tutorial.	September 17, 2021
AWS Proton console help panels release (p. 69)	Help panels added to the console.	September 8, 2021
AWS Proton General availability (GA) release (p. 69)	Adds 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 (p. 69)	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 (p. 69)	Public preview release 2 includes AWS Proton tagging and the ability to launch services without a service pipeline.	March 5, 2021
Initial release (p. 69)	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 General Reference*.